A neural model of the motor control system

by

Travis DeWolf

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctorate of Philosophy
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2014

Abstract

In this thesis I present the Recurrent Error-driven Adaptive Control Hierarchy (REACH); a large-scale spiking neuron model of the motor cortices and cerebellum of the motor control system. The REACH model consists of anatomically organized spiking neurons that control a nonlinear three-link arm to perform reaching and handwriting, while being able to adapt to unknown changes in arm dynamics and structure. I show that the REACH model accounts for data across 19 clinical and experimental studies of the motor control system. These data includes a mix of behavioural and neural spiking activity, across normal and damaged subjects performing adaptive and static tasks.

The REACH model is a dynamical control system based on modern control theoretic methods, specifically operational space control, dynamic movement primitives, and nonlinear adaptive control. The model is implemented in spiking neurons using the Neural Engineering Framework (NEF).

The model plans trajectories in end-effector space, and transforms these commands into joint torques that can be sent to the arm simulation. Adaptive components of the model are able to compensate for unknown kinematic or dynamic system parameters, such as arm segment length or mass. Using the NEF the adaptive components of the system can be seeded with approximations of the system kinematics and dynamics, allowing faster convergence to stability. Stability proofs for nonlinear adaptation methods implemented in distributed systems with scalar output are presented.

By implementing the motor control model in spiking neurons, biological constraints such as neurotransmitter time-constants and anatomical connectivity can be imposed, allowing further comparison to experimental data for model validation. The REACH model is compared to clinical data from human patients as well as neural recording from monkeys performing reaching experiments. The REACH model represents a novel integration of control theoretic methods and neuroscientific constraints to specify a general, adaptive, biologically plausible motor control algorithm.

Acknowledgements

I would like to thank my supervisor, Dr. Chris Eliasmith, for all of his support and guidance throughout my time at Waterloo. I would like to thank my wife, Julie, for all her support throughout my degrees, as well as my parents and brother. I would also like to thank the other lab members that made the journey with me, Daniel, Xuan, Trevor, and Terry, and all the other members of the CNRG lab. I would also like to thank the whole potluck crew and all the great friends we've made there and throughout the years!

Dedication

I dedicate this to my amazing wife, Julie. If you read it with the right tone it's actually quite a romantic thesis.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Control of movement is arguably the single most important responsibility of the brain in an organism. It underwrites all interaction with the environment, allowing an organism to exert control over its circumstances: if it's hungry it can move towards and consume food; if it's cold it can seek out shelter and warmth; and if it needs to communicate it can perform interpretive dance to convey its feelings to other organisms.

Gaining an understanding of the motor control system of the brain has the potential to lead to large advancements in numerous fields. Insight into the kinds of algorithms the brain uses will directly affect the development of control algorithms in the field of robotics, as the brain remains a far more advanced, adaptable, and robust control system than any current technology. Having a functional model of the neural structures of the motor control system will allow for faster development and testing of treatments for neuro-degenerative motor disorders, such as Parkinson's or Huntington's disease. Similarly, such understanding could advance ideas into physiotherapy, suggesting more effective recovery or training methods for patients and athletes. Finally, such motor control models will allow for better brain-computer interface applications, including neural-prosthetics or the development of avatar surrogates.

Studying the motor system can also serve as a means of addressing more cognitive functions of the brain. Many higher-level processes use the same pathways and neural circuits as the motor system, and can be thought of as performing similar functions to the motor system with different input and output in a more abstract state space [49, 43].

Studying the motor control itself rather than these other systems has an advantage in that motor output is observable, whereas the output of cognitive systems can be much more difficult to measure directly.

Here I present and discuss the Recurrent Error-driven Adaptive Control Hierarchy (REACH) model, which is the first model of its kind: a large-scale, adaptive, fully spiking neural model of the motor cortices and cerebellum capable of generating and matching behavioural and experimental data from 19 separate clinical and experimental studies.

1.2 State of the field

In the field of computational or theoretical neuroscience models of motor control can be divided into one of two approaches: a neuro-scientific approach or a control theoretic approach.

Neuroscientific approaches to motor control are largely geared towards characterizing the basic anatomical elements and functions of the motor system. As such, they can be considered a largely 'bottom-up' approach to motor control – that is, an approach that attempts to identify the basic elements and functions of the system as a whole. Methods of this approach include tracing of interconnections between different areas [21, 90], recording single-cells to investigate neural sensitivity to different stimuli [61, 141], examining the effects of neural activity on motor output through stimulation experiments [157, 59], and observing lesioned animals or brain-damaged patients [85, 200]. This focus on specific, isolated functions has resulted in theoretical models that tend to perform poorly when tested in novel contexts [161, 105, 183]. As a result, such models often do not provide general, unified explanations of motor system activity.

Control theoretic methods, on the other hand, can be considered a more 'top-down' approach to motor control. These methods focus on more global aspects of motor control, such as movement optimization according to a cost-function [195], learning actions from observation [17], the generalization of movements such that controlled systems are robust to altered system dynamics or environmental changes [35], and so on. Much of the field, however, will take some 'biological inspiration', usually an observation of some unique phenomena of biological systems that is counter-intuitive in the context of current motor control methods, but gain little further inspiration from biological systems. As a result, mapping control-based behavioural models to biological systems proves difficult because the proposed controllers are typically not constrained by neuro-computational considerations.

Contemporary work on the motor system typically applies one of these approaches, focusing on either behavioural phenomena, such as stereotypical trajectories, velocity profiles [193], and motor learning ability [160, 132], or on neural phenomena, such as spiking neuron activity profiles [30] and neural data analysis [32]. Models that take control theoretic methods and implement them in biologically plausible neural structures are scarce. One reason for this apparent divide between behavioral and neural approaches is that it has not been clear how to implement large-scale models encompassing both low-level neural details and high-level behavioral characterizations.

Recent advances in large-scale neural modeling have focussed on how complex spiking neural networks give rise to a variety of sophisticated behaviors we observe in humans and other animals [50]. The largest such model is Spaun [50], which performs perceptual, cognitive, and motor tasks. However, the motor system of that model is minimal: it controls a two link linear arm, it can only draw digits from 0-9 (and does so poorly), it is not fully implemented in spiking neurons, and it is unable to adapt to changes in the environment or itself.

The methods used to develop the Spaun model, however, can be similarly used in the development of more detailed, biologically plausible models of the motor control system. These methods of implementing complex dynamical systems in spiking neural networks, i.e. the Neural Engineering Framework [49], form the necessary bridge between control theory and neuro-anatomically constrained implementations. While natural motor systems still remain far more robust, adaptive, and rapid than those that we are able to engineer, recent advances in control theory have suggested various means of building more advanced motor control systems [29, 97, 130]. Understanding how these control theoretic advances may relate to the algorithms employed by the brain, can help narrow the search for good approaches to motor control in engineering and robotics, while also improving our understanding of neural mechanisms in support of clinical applications.

The development and implementation of these algorithms in a spiking neural network is the focus of this thesis.

1.3 Thesis structure

The structure of this thesis is as follows. First, the relevant control theoretic algorithms and methods, which make up the basis of the REACH dynamical system, are reviewed in detail. Chapter 2 works through the development of operational space control algorithms, starting with how to characterize a system and calculate its Jacobian matrices, through to transforming control signals from operational space to generalized coordinates and how to develop a null-space filter to prevent secondary control signals from interfering with movement in the operational space. Chapter 3 discusses Dynamical Movement Primitives and their applications in trajectory control. Behavioural level results of the REACH model are used throughout this chapter to illustrate the effectiveness of the novel combination of operational space control and DMPs. Chapter 4 reviews nonlinear adaptive control which addresses the situation common to many control systems where some parameters of the system are unknown. The derivation of two types of adaptation laws, for unknown dynamics and kinematics, are worked through in detail to provide insight into how stability of these systems can be guaranteed.

Chapter 5 switches then from reviewing control theory to reviewing previous biological models of the motor control system. Several smaller models are discussed followed by a review of the larger-scale computational models of motor control. Most prominently the Neural Optimal Control Hierarchy framework is reviewed, which was the result of my master's thesis work, and serves as a foundation for the REACH model. Additionally, several novel results from an implementation of the NOCH are presented, taken from work done after my masters and presented in [50] and [40]. Chapter 6 works through the Neural Engineering Framework and presents the changes required to the previously discussed control theoretic methods for effective implementation in spiking neurons. Chapter 7 then presents the REACH model in its entirety, with non-neural implementation results, a detailed description of the neural implementation, and then neural implementation results and analysis, along with a comparison to the NOCH framework and discussion of future directions for the work.

There are several contributions of this thesis. The primary contribution is the design and implementation of a large-scale fully spiking neuron model of the motor control system. The amalgamation of the discussed control methods to form a behavioural level model of the motor control system is also a novel contribution of this thesis. Additionally, several novel proofs of stability for reformulations of the adaptive kinematics and dynamics control laws originally presented by [170, 158, 29] are provided.

Throughout this thesis sections and structure are borrowed from previous writing I've done throughout the course of my degree, drawing from posts on my research blog [1], a paper published in the Journal of Neural Engineering [41], and a recently filed patent based on a novel implementation of nonlinear adaptive control in distributed systems [42].

[1]http://www.studywolf.com

Chapter 2

Operational space control

In operational space control, one analytically derives the dynamics of the system, and then uses this knowledge to cancel out undesired forces and generate a control signal that moves the system through some task space, taking advantage of the passive dynamics of the system. In addition to cancelling out undesired forces, another major feature of operational space control is transforming force signals specified in operational space into forces in generalized coordinates, solving any configuration redundancies and generating a signal that can be applied to the plant.

Throughout this chapter the terms generalized coordinates and operational space will be used. ‘Generalized coordinates’ refer to a state space description of the system such that when given, all of the degrees of freedom (DOF) of the system are fully specified. ‘Operational space’, or ‘task space’, refers to a state space description of the system that does not necessarily define all of the DOF, but is where the current task is taking place. For example, in the control of a robot arm providing a state space description that specifies joint angles and velocities fully describes the current state of the system, and so this joint space description is considered to be in generalized coordinates. If the state space description of the robot arm instead specifies the position of the end-effector, there are potentially an infinite number of possible configurations for the arm to be in such that the end-effector is at this position, and so end-effector space descriptions are not generalized coordinates. Often we wish to plan trajectories in terms of end-effector space however, and so end-effector space is often the operational (or task) space.

The methods for calculating the forces acting on serial robots are reviewed in the appendix, along with a discussion of how to implement operational space control. This chapter starts with an example of control in generalized coordinates and operational space, and then discusses implementing secondary-controllers in the null-space of primary controllers, with some final considerations regarding handling complications like singularities in the Jacobian.

2.0.1 Controlling in generalized coordinates

Now that gravity and inertia have been accounted for, control is straightforward because the undesirable dynamics of the system can be cancelled out and trajectories can be designed as though the system behaved linearly. Cancelling inertia also means that the control of each of the joints can be done independently, since their movements no longer affect one another. The control system will then be comprised of a PD controller for each joint to generate the desired forces, which will then be incorporated with terms to compensate for inertia and gravity.

In the equation for system acceleration, Eq. 7.59, there are still some nonlinearities unaccounted for due to Coriolis and centrifugal effects. These terms require highly accurate models of the moments of inertia to properly cancel out, with measurements that are difficult to attain. If the moments are incorrect the attempted compensation can actually introduce instability into the system, so often these terms are ignored when building the controller. While not addressed in operational space control, they are accounted for in nonlinear adaptive control, which will be discussed in Chapter 4. For now however they will be ignored.

Rewriting the system dynamics, Eq. 7.59, in terms of acceleration gives

$$\ddot{\mathbf{q}} = \mathbf{M}^{-1}(\mathbf{q})(\mathbf{u} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) - \mathbf{g}(\mathbf{q})). \tag{2.1}$$

Ideally, the control signal would be constructed

$$\mathbf{u} = (\mathbf{M}(\mathbf{q})\, \ddot{\mathbf{q}}_{\text{des}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q})), \tag{2.2}$$

where $\ddot{\mathbf{q}}_{\text{des}}$ is the desired acceleration of the system. This would result in system acceleration

$$\begin{aligned} \ddot{\mathbf{q}} &= \mathbf{M}^{-1}(\mathbf{q})((\mathbf{M}(\mathbf{q})\, \ddot{\mathbf{q}}_{\text{des}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q})) - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) - \mathbf{g}(\mathbf{q})), && (2.3)\\ \ddot{\mathbf{q}} &= \mathbf{M}^{-1}(\mathbf{q})\mathbf{M}(\mathbf{q})\, \ddot{\mathbf{q}}_{\text{des}}, && (2.4)\\ \ddot{\mathbf{q}} &= \ddot{\mathbf{q}}_{\text{des}}, && (2.5) \end{aligned}$$

which would be ideal. The unmodeled Coriolis and centrifugal effects cannot be accounted for, so the instead the control signal is

$$\mathbf{u} = (\mathbf{M}(\mathbf{q})\, \ddot{\mathbf{q}}_{\text{des}} + \mathbf{g}(\mathbf{q})), \tag{2.6}$$

where $\mathbf{M}(\mathbf{q})$ is defined in Eq. 7.69 and $\mathbf{g}(\mathbf{q})$ is defined Eq. 7.86, using the Jacobians defined in Section 7.9.1.

Using a standard PD control formula to generate the desired acceleration:

$$\ddot{\mathbf{q}}_{\text{des}} = k_p \; (\mathbf{q}_{\text{des}} - \mathbf{q}) + k_v \; (\dot{\mathbf{q}}_{\text{des}} - \dot{\mathbf{q}}), \tag{2.7}$$

where k_p and k_v are our gain values, and the control signal has been fully defined:

$$\mathbf{u} = (\mathbf{M}(\mathbf{q}) \; (k_p \; (\mathbf{q}_{\text{des}} - \mathbf{q}) + k_v \; (\dot{\mathbf{q}}_{\text{des}} - \dot{\mathbf{q}})) + \mathbf{g}(\mathbf{q})). \tag{2.8}$$

2.0.2 Controlling in operational space

Given the robot arm shown in Figure 7.25 the goal in this example is to develop an operational space controller. The generalized coordinates for this example will be joint space, and the operational space is going to be the end-effector Cartesian coordinates relative to the base reference frame.

The Jacobians for the end-effector of this robot were previously defined in Section 7.8.2.3:

$$\mathbf{J}_{ee}(\mathbf{q}) = \begin{bmatrix} -L_0 sin(q_0) - L_1 sin(q_0 + q_1) & -L_1 sin(q_0 + q_1) \\ L_0 cos(q_0) + L_1 cos(q_0 + q_1) & L_1 cos(q_0 + q_1) \end{bmatrix}. \tag{2.9}$$

With $\mathbf{J}_{ee}(\mathbf{q})$ end-effector forces can be transformed into joint space as shown in Section 7.8 using Eq. 7.40:

$$\mathbf{u} = \mathbf{J}_{ee}^T(\mathbf{q}) \; \mathbf{F}_{\mathbf{x}}. \tag{2.10}$$

Rewriting $\mathbf{F}_{\mathbf{x}}$ as its component parts

$$\mathbf{F}_{\mathbf{x}} = \mathbf{M}_{\mathbf{x}_{ee}}(\mathbf{q}) \; \ddot{\mathbf{x}}_{\text{des}}, \tag{2.11}$$

where $\ddot{\mathbf{x}}$ is end-effector acceleration, and $\mathbf{M}_{\mathbf{x}_{ee(\mathbf{q})}}$ is defined in Eq. 7.79, using Jacobians from Section 7.9.1. Adding in a term to cancel the effects of gravity in joint space gives

$$\mathbf{u} = \mathbf{J}_{ee}^T(\mathbf{q})\mathbf{M}_{\mathbf{x}_{ee}}(\mathbf{q})\ddot{\mathbf{x}}_{\text{des}} + \mathbf{g}(\mathbf{q}), \tag{2.12}$$

where $\mathbf{g}(\mathbf{q})$ is the same as in the previous example, defined in Eq. 7.86 using Jacobians from Section 7.9.1. This controller converts desired end-effector acceleration into torque commands, and compensates for inertia and gravity.

Defining a basic PD controller in operational space

$$\ddot{\mathbf{x}}_{\text{des}} = k_p(\mathbf{x}_{\text{des}} - \mathbf{x}) + k_v(\dot{\mathbf{x}}_{\text{des}} - \dot{\mathbf{x}}), \tag{2.13}$$

and the full equation for the operational space control signal in joint space is:

$$\mathbf{u} = \mathbf{J}_{ee}^T(\mathbf{q}) \; \mathbf{M}_{\mathbf{x}_{ee}}(\mathbf{q})[k_p(\mathbf{x}_{\text{des}} - \mathbf{x}) + k_v(\dot{\mathbf{x}}_{\text{des}} - \dot{\mathbf{x}})] + \mathbf{g}(\mathbf{q}). \tag{2.14}$$

2.1 Controlling in the null space

The last example comprises the basics of operational space control; describe the system, calculate the system dynamics, transform desired forces from an operational space to the generalized coordinates, and build the control signal to cancel out the undesired system dynamics. Basic operational space control works quite well, but it is not uncommon to have several control goals at once; such as 'move the end-effector to this position' (primary goal), and 'keep the elbow raised high' (secondary goal) in the control of a robot arm.

If the operational space can also serve as generalized coordinates, i.e. if the system state specified in operational space constrains all of the degrees of freedom of the robot, then there is nothing that can be done without affecting the performance of the primary controller. In the case of controlling a two-link robot arm this is the case. The end-effector Cartesian space (chosen as the operational space) could also be a generalized coordinates system, because a specific (x, y) position fully constrains the position of the arm.

But often when using operational space control for more complex robots this is not the case. In these situations, the forces controlled in operational space have fewer dimensions than the robot has degrees of freedom, and so it is possible to accomplish the primary goal in a number of ways. The null space of this primary controller is the region of state space where there is a redundancy of solutions; the system can move in a number of ways and still not affect the completion of the goals of the primary controller. An example of this is all the different configurations the elbow can be in while a person moves their hand in a straight line. In these situations, a secondary controller can be created to operate in the null space of the primary controller, and the full control signal sent to the system is a sum of the primary control signal and a filtered version of the secondary control signal. In this section the derivation of the null-space filter will be worked through for a system with only a primary and secondary controller, but note that the process can be applied iteratively for systems with further controllers.

The filtering of the secondary control signal means that the secondary controller's goals will only be accomplished if it is possible to do so without affecting the performance of the first controller. In other words, the secondary controller must operate in the null space of the first controller. Denote the primary operational space control signal, e.g. the control signal defined in Eq. 2.8, as $\mathbf{u_x}$ and the control signal from the secondary controller $\mathbf{u}_{\text{null}}$. Define the force to apply to the system

$$\mathbf{u} = \mathbf{u_x} + (\mathbf{I} - \mathbf{J}_{ee}^T(\mathbf{q})\;\mathbf{J}_{ee}^{T+}(\mathbf{q}))\mathbf{u}_{\text{null}}, \tag{2.15}$$

where $\mathbf{J}_{ee}^{T+}(\mathbf{q})$ is the pseudo-inverse of $\mathbf{J}_{ee}^T(\mathbf{q})$.

Examining the filtering term that was added,

$$(\mathbf{I} - \mathbf{J}_{ee}^T(\mathbf{q})\mathbf{J}_{ee}^{T+}(\mathbf{q}))\mathbf{u}_{\text{null}}, \tag{2.16}$$

it can be seen that the Jacobian transpose multiplied by its pseudo-inverse will be 1's all along the diagonal, except in the null space. This means that $\mathbf{u}_{\text{null}}$ is subtracted from itself everywhere that affects the operational space movement and is left to apply any arbitrary control signal in the null space of the primary controller.

Unfortunately, this initial set up does not adequately filter out the effects of forces that might be generated by the secondary controller. The Jacobian is defined as a relationship between the velocities of two spaces, and so operating in the null space defined by the Jacobian ensures that no *velocities* are applied in operational space, but the required filter must also prevent any *accelerations* from affecting movement in operational space. The standard Jacobian pseudo-inverse null space is a velocity null space, and so a filter built using it will allow forces affecting the system's acceleration to still get through. What is required is a pseudo-inverse Jacobian defined to filter signals through an acceleration null space.

To acquire this acceleration filter, Eq. 2.15 will be substituted into the equation for acceleration in the operational space, Eq. 7.73, which, after cancelling out gravity effects with the control signal and removing the unmodeled dynamics, gives

$$\ddot{\mathbf{x}} = \mathbf{J}_{ee}(\mathbf{q})\mathbf{M}^{-1}(\mathbf{q})[\mathbf{J}_{ee}^T(\mathbf{q})\ \mathbf{M}_{\mathbf{x}_{ee}}(\mathbf{q})\ \ddot{\mathbf{x}}_{\text{des}} - (\mathbf{I} - \mathbf{J}_{ee}^T(\mathbf{q})\ \mathbf{J}_{ee}^{T+}(\mathbf{q}))\ \mathbf{u}_{\text{null}}]. \tag{2.17}$$

Rewriting this to separate the secondary controller into its own term

$$\ddot{\mathbf{x}} = \mathbf{J}_{ee}(\mathbf{q})\mathbf{M}^{-1}(\mathbf{q})\mathbf{J}_{ee}^T(\mathbf{q})\ \mathbf{M}_{\mathbf{x}_{ee}}(\mathbf{q})\ \ddot{\mathbf{x}}_{\text{des}} - \mathbf{J}_{ee}(\mathbf{q})\mathbf{M}^{-1}(\mathbf{q})[\mathbf{I} - \mathbf{J}_{ee}^T(\mathbf{q})\ \mathbf{J}_{ee}^{T+}(\mathbf{q})]\ \mathbf{u}_{\text{null}}, \tag{2.18}$$

it becomes clear that to not cause any unwanted movement in operational space the second term must be zero.

There is only one free term left in the second term, and that is the pseudo-inverse. There are numerous different pseudo-inverses that can be chosen for a given situation, and here what is required is to engineer a pseudo-inverse such that the term multiplying $\mathbf{u}_{\text{null}}$ in the above operational space acceleration equation is guaranteed to go to zero.

$$\mathbf{J}_{ee}(\mathbf{q})\mathbf{M}^{-1}(\mathbf{q})[\mathbf{I} - \mathbf{J}_{ee}^T(\mathbf{q})\mathbf{J}_{ee}^{T+}(\mathbf{q})]\mathbf{u}_{\text{null}} = \mathbf{0}, \tag{2.19}$$

this needs to be true for all $\mathbf{u}_{\text{null}}$, so it can be removed,

$$\begin{aligned} \mathbf{J}_{ee}(\mathbf{q})\ \mathbf{M}^{-1}(\mathbf{q})[\mathbf{I} - \mathbf{J}_{ee}^T(\mathbf{q})\ \mathbf{J}_{ee}^{T+}(\mathbf{q})] &= \mathbf{0}, & (2.20)\\ \mathbf{J}_{ee}(\mathbf{q})\ \mathbf{M}^{-1}(\mathbf{q}) &= \mathbf{J}_{ee}(\mathbf{q})\ \mathbf{M}^{-1}(\mathbf{q})\ \mathbf{J}_{ee}^T(\mathbf{q})\ \mathbf{J}_{ee}^{T+}(\mathbf{q}), & (2.21) \end{aligned}$$

substituting in using Eq. 7.79, which defines $\mathbf{M}(\mathbf{q})$,

$$\begin{aligned}\mathbf{J}_{ee}(\mathbf{q})\mathbf{M}^{-1}(\mathbf{q}) &= \mathbf{M}^{-1}_{\mathbf{x}_{ee}}(\mathbf{q})\mathbf{J}^{T+}_{ee}(\mathbf{q}), && (2.22)\\ \mathbf{J}^{T+}_{ee}(\mathbf{q}) &= \mathbf{M}_{\mathbf{x}_{ee}}(\mathbf{q})\;\mathbf{J}_{ee}(\mathbf{q})\;\mathbf{M}^{-1}(\mathbf{q}). && (2.23)\end{aligned}$$

This specific Jacobian inverse was presented in [112] and is called the ‘dynamically consistent generalized inverse’. Using this psuedo-inverse guarantees that any signal coming from the secondary controller will not affect movement in the primary controller’s operational space.

The null space filter cancels out the acceleration effects of forces in operational space from a signal that is being applied as part of the control system. But it can also be used to cancel out the effects of any unwanted signal that can be modeled. Given some undesirable force signal interfering with the system that can be effectively modeled, a null space filtering term can be implemented to cancel it out. The control signal in this case, with one primary operational space controller and a null space filter for the undesired force, looks like:

$$\mathbf{u} = \mathbf{J}^T_{ee}(\mathbf{q})\;\mathbf{M}_{\mathbf{x}}(\mathbf{q})\;\ddot{\mathbf{x}}_{\text{des}} - \mathbf{g}(\mathbf{q}) - \mathbf{J}^T_{ee}(\mathbf{q})\;\mathbf{J}^{T+}_{ee}(\mathbf{q})\;\mathbf{u}_{\text{undesired force}}. \qquad (2.24)$$

2.1.1 Example: Operational space control with secondary controller

Given a three link arm (revolute-revolute-revolute) operating in the (x, z) plane, shown in Figure 7.23, this example will construct the control system for a primary controller controlling the end-effector and a secondary controller working to keep the arm near its joint angles’ default resting positions.

Let the system state be $\mathbf{q} = [q_0, q_1, q_2]^T$ with default positions $\mathbf{q}^0 = \left[\frac{\pi}{3}, \frac{\pi}{4}, \frac{\pi}{4}\right]^T$. The control signal of the secondary controller is the difference between the target state and the current system state

$$\mathbf{u}_{\text{null}} = k_{p_{\text{null}}}(\mathbf{q}^0 - \mathbf{q}), \qquad (2.25)$$

where $k_{p_{\text{null}}}$ is a gain term.

Let the centres of mass be

$$\text{com}_0 = \begin{bmatrix} \frac{1}{2}cos(q_0) \\ 0 \\ \frac{1}{2}sin(q_0) \end{bmatrix}, \quad \text{com}_1 = \begin{bmatrix} \frac{1}{4}cos(q_1) \\ 0 \\ \frac{1}{4}sin(q_1) \end{bmatrix} \quad \text{com}_2 = \begin{bmatrix} \frac{1}{2}cos(q_2) \\ 0 \\ \frac{1}{4}sin(q_2) \end{bmatrix},$$

the Jacobians for the COMs are

$$\mathbf{J}_0(\mathbf{q}) \quad = \quad \begin{bmatrix} -\frac{1}{2}sin(q_0) & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{2}cos(q_0) & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

$$\mathbf{J}_1(\mathbf{q}) \quad = \quad \begin{bmatrix} -L_0sin(q_0) + \frac{1}{4}sin(q_{01}) & \frac{1}{4}sin(q_{01}) & 0 \\ 0 & 0 & 0 \\ L_0cos(q_0) + \frac{1}{4}cos(q_{01}) & \frac{1}{4}cos(q_{01}) & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

$$\mathbf{J}_1(\mathbf{q}) \quad = \quad \begin{bmatrix} -L_0sin(q_0) + L_1sin(q_{01}) - \frac{1}{2}sin(q_{012}) & L_1sin(q_{01}) - \frac{1}{2}sin(q_{012}) & -\frac{1}{2}sin(q_{012}) \\ 0 & 0 & 0 \\ L_0cos(q_0) + L_1cos(q_{01}) + \frac{1}{2}cos(q_{012}) & L_1cos(q_{01}) + \frac{1}{2}cos(q_{012}) & \frac{1}{2}cos(q_{012}) \\ 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}.$$

The Jacobian for the end-effector of this three link arm is

$$\mathbf{J}_{ee} = \begin{bmatrix} -L_0sin(q_0) - L_1sin(q_{01}) - L_2sin(q_{012}) & -L_1sin(q_{01}) - L_2sin(q_{012}) & -L_2sin(q_{012}) \\ L_0cos(q_0) + L_1cos(q_{01}) + L_2cos(q_{012}) & L_1cos(q_{01}) + L_2cos(q_{012}) & L_2cos(q_{012}), \end{bmatrix}$$

where $q_{01} = q_0 + q_1$ and $q_{012} = q_0 + q_1 + q_2$.

Taking the control signal developed in Section 2.0.2

$$\mathbf{u} = \mathbf{J}_{ee}^T(\mathbf{q})\ \mathbf{M}_{\mathbf{x}_{ee}}(\mathbf{q})[k_p(\mathbf{x}_{\text{des}} - \mathbf{x}) + k_v(\dot{\mathbf{x}}_{\text{des}} - \dot{\mathbf{x}})] - \mathbf{g}(\mathbf{q}), \tag{2.26}$$

where $\mathbf{M}_{ee}(\mathbf{q})$ is defined in Eq. 7.79 and $\mathbf{g}(\mathbf{q})$ is defined in Eq. 7.86 using the Jacobians above, and k_p and k_v are gain terms, usually set such that $k_v = \sqrt{k_p}$, and adding in the null space control signal and filter gives

$$\mathbf{u} = \mathbf{J}_{ee}^T(\mathbf{q})\ \mathbf{M}_{\mathbf{x}_{ee}}(\mathbf{q})[k_p(\mathbf{x}_{\text{des}} - \mathbf{x}) + k_v(\dot{\mathbf{x}}_{\text{des}} - \dot{\mathbf{x}})] - (\mathbf{I} - \mathbf{J}_{ee}^T(\mathbf{q})\mathbf{J}_{ee}^{T+}(\mathbf{q}))\mathbf{u}_{\text{null}} - \mathbf{g}(\mathbf{q}), \tag{2.27}$$

where $\mathbf{J}^{T+}(\mathbf{q})$ is the dynamically consistent generalized inverse defined in Eq. 2.23 and $\mathbf{u}_{\text{null}}$ is defined in Eq. 2.25.

2.2 Handling singularities

A final note for this chapter is on handling kinematic singularities, which quickly presents itself as a problem during implementation of operational space control.

2.2.1 What is a singularity?

In a two link arm, a singularity happens whenever the elbow angle nears 0 or π. Figure 2.1 shows a demonstration of the system behaviour when this occurs. What is happening at the point of instability is that the Jacobian for the end-effector has dropped rank and becomes singular (i.e. non-invertible), and when calculating the mass matrix for operational space:

$$\mathbf{M_x}(\mathbf{q}) = (\mathbf{J}(\mathbf{q})\ \mathbf{M}^{-1}(\mathbf{q})\mathbf{J}^T)^{-1}(\mathbf{q}) \tag{2.28}$$

the values explode in the inverse calculation. Dropping rank means that the rows of the Jacobian are no longer linearly independent, which means that the matrix can be rotated such that it gives a matrix with a row of zeros. This row of zeros is the degenerate direction, and the problems come from trying to send forces in that direction. To determine when the Jacobian becomes singular its determinant can be examined; if the determinant of the matrix is zero, then it is singular. Looking the Jacobian for the end-effector:

$$\mathbf{J}(\mathbf{q}) = \begin{bmatrix} -L_0 sin(q_0) - L_1 sin(q_0 + q_1) & -L_1 sin(q_0 + q_1) \\ L_0 cos(q_0) + L_1 cos(q_0 + q_1) & L_1 cos(q_0 + q_1) \end{bmatrix}. \tag{2.29}$$

When $q_1 = 0$ it can be that $sin(q_0 + 0) = sin(q_0)$, so the Jacobian becomes

$$\mathbf{J}(\mathbf{q}) = \begin{bmatrix} -(L_0 - L_1) sin(q_0) & -L_1 sin(q_0) \\ (L_0 + L_1) cos(q_0) & L_1 cos(q_0) \end{bmatrix} \tag{2.30}$$

which gives a determinant of

$$(L_0 - L_1)(-sin(q_0))(L_1)(cos(q_0)) - (L_1)(-sin(q_0))(L_0 + L_1)(cos(q_0)) = 0. \tag{2.31}$$

Similarly, when $q_1 = \pi$, where $sin(q_0 + \pi) = -sin(q_0)$ and $cos(q_0 + \pi) = -cos(q_0)$, the determinant of the Jacobian is

$$\mathbf{J}(\mathbf{q}) = \begin{bmatrix} -(L_0 - L_1) sin(q_0) & L_1 sin(q_0) \\ (L_0 + L_1) cos(q_0) & -L_1 cos(q_0) \end{bmatrix} \tag{2.32}$$

Calculating the determinant of this we get

$$(L_0 + L_1)(-sin(q_0))(L_1)(-cos(q_0)) - (L_1)(sin(q_0))(L_0 + L_1)(-cos(q_0)) = 0. \tag{2.33}$$

Note that while in these cases the Jacobian is a square matrix in the event that it is not a square matrix, the determinant of $\mathbf{J}(\mathbf{q})\ \mathbf{J}^T(\mathbf{q})$ can be found instead.

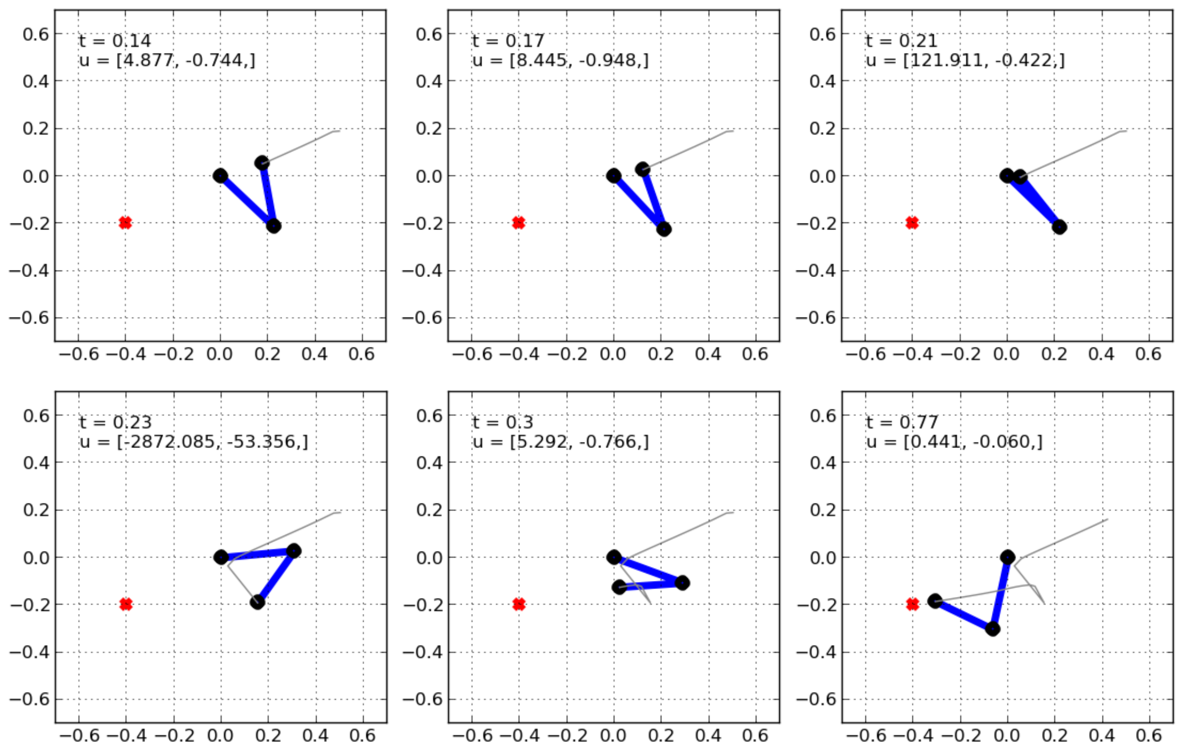


Figure 2.1: A two link arm moving to a target, encountering a kinematic singularity causing instability in the control signal and erratic performance. The red dot is the target position for the hand, the gray line is the path traced out by the line during movement. When the elbow joint angle is equal to π the Jacobian becomes singular and the control signal becomes unstable, causing the undesired spasm along the way to the target.

2.2.2 Fixing the problem

When a singularity is occurring it can be detected, but now it must be handled such that the controller behaves appropriately. This can be done by identifying the degenerate dimensions and setting the force in those directions to zero.

First the SVD decomposition of $\mathbf{M}_\mathbf{x}^{-1}(\mathbf{q}) = \mathbf{VSU}^T$ is found. To get the inverse of this matrix (i.e. to find $\mathbf{M}_\mathbf{x}(\mathbf{q})$) from the returned $\mathbf{V}, \mathbf{S}$ and $\mathbf{U}$ matrices is a matter of inverting the matrix $\mathbf{S}$:

$$\mathbf{M}_\mathbf{x}(\mathbf{q}) = \mathbf{VS}^{-1}\mathbf{U}^T, \tag{2.34}$$

where $\mathbf{S}$ is a diagonal matrix of singular values.

Because $\mathbf{S}$ is diagonal it is very easy to find its inverse, which is calculated by taking the reciprocal of each of the diagonal elements.

Whenever the system approaches a singularity some of the values of $\mathbf{S}$ will start to get very small, and when we take the reciprocal of them we start getting huge numbers, which is where the value explosion comes from. Instead of allowing this to happen, a check for approaching the singularity can be implemented, which then sets the singular values entries smaller than the threshold equal to zero, canceling out any forces that would be sent in that direction. A Python implementation of this algorithm follows:

```
Mx_inv = np.dot(JEE, np.dot(np.linalg.inv(Mq), JEE.T))
if abs(np.linalg.det(np.dot(JEE,JEE.T))) > .005**2:
    # if we’re not near a singularity
    Mx = np.linalg.inv(Mx_inv)
else:
    # in the case that the robot is entering near singularity
    u,s,v = np.linalg.svd(Mx_inv)
    for i in range(len(s)):
        if s[i] < .005: s[i] = 0
        else: s[i] = 1.0/float(s[i])
    Mx = np.dot(v, np.dot(np.diag(s), u.T))
```

A demonstration of the two-link arm moving through the same kinematic singularity with the above method is shown in Figure 2.2.

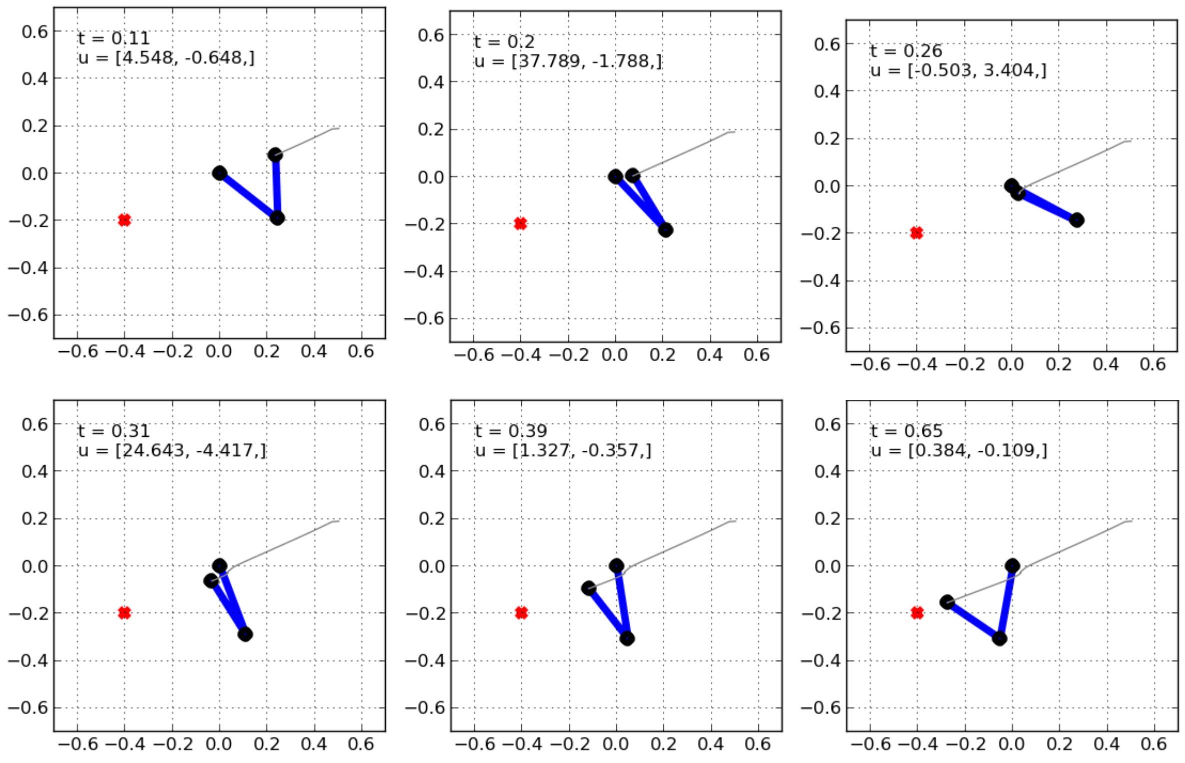


Figure 2.2: A two link arm moving to a target, encountering a kinematic singularity with the modified control signal, which filters unstable control signals in degenerate directions, allowing the arm to move smoothly to its target. The red dot is the target position for the hand, the gray line is the path traced out by the line during movement. When the elbow joint angle is equal to π the Jacobian becomes singular and the degenerate directions are set to 0 and the control signal remains stable.

Chapter 3

Dynamic movement primitives

The previous chapter developed methods for generating a control signal that was capable of moving the system with precision. The question of how to move the system, however, remained unaddressed. In this thesis trajectory generation will be performed through the employment of dynamical movement primitives.

Dynamic movement primitives (DMPs) are a method of trajectory control / planning developed by Dr. Stephan Schaal's lab, first presented in [159], and later reformulated to be more concise by Dr. Auke Ijspeert [97]. This work was motivated by the desire to find a way to represent complex motor actions that can be easily adjusted without tuning numerous parameters and with stability guarantees.

Complex movements have long been thought to be composed of sets of basic action 'building blocks' executed either in sequence or in parallel [23, 70] DMPs are a proposed mathematical formalization of these primitives. The difference between DMPs and action building blocks proposed elsewhere [37] is that DMPs are dynamical systems. The core idea behind DMPs is to take a dynamical system with well-specified, stable behaviour and add another term that pushes it to follow some interesting trajectory. There are two kinds of DMPs: discrete and rhythmic. For discrete movements the basic system is a point attractor, and for rhythmic movements the basic system is a limit cycle.

To picture DMPs as part of a full control system imagine two systems: an imaginary system where trajectories are planned, and a real system which works to follow these planned trajectories. DMPs are the imagined system which plan a trajectory for the real system to follow.

A DMP has its own set of dynamics, which may or may not reflect the dynamics of the real system. If the DMP system is planning a path for a hand to follow, then the current state of the system is compared to the target state as specified by the DMP system and a PD control signal is generated. This control signal specifies the forces for the system to apply to the hand to move along the path specified by the DMP. A separate component of the system must then take these hand forces and transform them into joint torques or muscle activations that can be directly applied to the real system's actuators. It is important to keep in mind that the whole DMP framework is for generating a trajectory to guide the real system, and does not directly interact with the real system.

Please also note that all of the code used to implement the systems described in this chapter can be found online on my Github account at `https://github.com/studywolf/blog/tree/master/DMPs`.

3.1 Discrete DMPs

As mentioned above, the basic dynamical system for the discrete DMP is a point attractor. A point attractor's dynamics evolve according to:

$$\ddot{y} = \alpha_y(\beta_y(g - y) - \dot{y}), \tag{3.1}$$

where y and $\dot{y}$ is the system state and velocity, respectively, g is the goal, and α and β are gain terms. This is also well known as the equation for proportional derivative (PD) control, which just drives the system in a straight line to the goal. To generate interesting behaviour along the way to the target a 'forcing' term, f, will be added, making the system dynamics:

$$\ddot{y} = \alpha_y(\beta_y(g - y) - \dot{y}) + f, \tag{3.2}$$

Defining the nonlinear function f such that a desired path is followed by the system is a non-trivial question. The crux of the DMP framework is that the method used to define the forcing function f over time will give the problem a well-defined and flexible structure. In addition, the function can be easily solved for and generalized both temporally and spatially. To accomplish this, a simple dynamical system is introduced into the framework, called the 'canonical system', denoted x. The canonical system evolves according to the dynamics:

$$\dot{x} = -\alpha_x x, \tag{3.3}$$

and is initialized to 1 at the beginning, decaying to 0 as specified by gain value α_x. The forcing function f is then defined as a function of the canonical system:

$$f(x, g) = \frac{\Sigma_{i=1}^{N} \psi_i w_i}{\Sigma_{i=1}^{N} \psi_i} x(g - y_0), \tag{3.4}$$

where y_0 is the initial position of the system, ψ_i is a Gaussian, defined

$$\psi_i = \exp\left(-h_i\left(x - c_i\right)^2\right), \tag{3.5}$$

and w_i is a weighting for a given basis function ψ_i. By having the forcing function not directly dependent on time, a temporal scaling can be achieved by altering the dynamics of the canonical system, rather than having to choose different center points for the basis functions, as discussed in Section 3.1.3.

The basis functions ψ_i are Gaussians centered along the canonical system state, c_i, with variance h_i. The forcing function is a set of Gaussians that are activated as the canonical system converges to 0. Their weighted summation is normalized, and then multiplied by the $x(g - y_0)$ term, which acts as both a 'diminishing' and spatial scaling term. Its role as a diminishing term ensures that the forcing function output will be reduced to 0 eventually, and the point attractor dynamics will be able to take over such that the system will always converge to the goal.

There are several important features of the DMP system as described to this point. The first is that the basis functions are activated as a function of x, shown in the top half of Figure 3.1. As the value of x decreases from 1 to 0, each of the Gaussians are activated (centered) at different x values. The second feature is that each of these basis functions are also assigned a weight, w_i. These weights are displayed in the lower half of Figure 3.1 in the bar plot. The output of the forcing function f is then the summation of the activations of these basis functions multiplied by their weight, also displayed in the lower half of Figure 3.1 in the blue line.

3.1.1 Spreading basis function centers

Because the canonical system does not converge linearly to the target, another step needs to be taken in placing the centres of the basis functions to make sure that they're activated evenly because the basis functions activation is dependent on x. If the system was linear then it would be easy to spread out the basis function activations as the system converged to the target. But, with the canonical system dynamics as described above, x is not a linear function of time. When the basis function activations are plotted against time, as shown in Figure 3.2, we see that the majority are activated immediately as x moves quickly at the beginning, and then the activations stretch out as the x slows down at the end.

In the interest of having the basis functions spaced out more evenly through time (so that the forcing function has resources to still move the system along interesting paths as

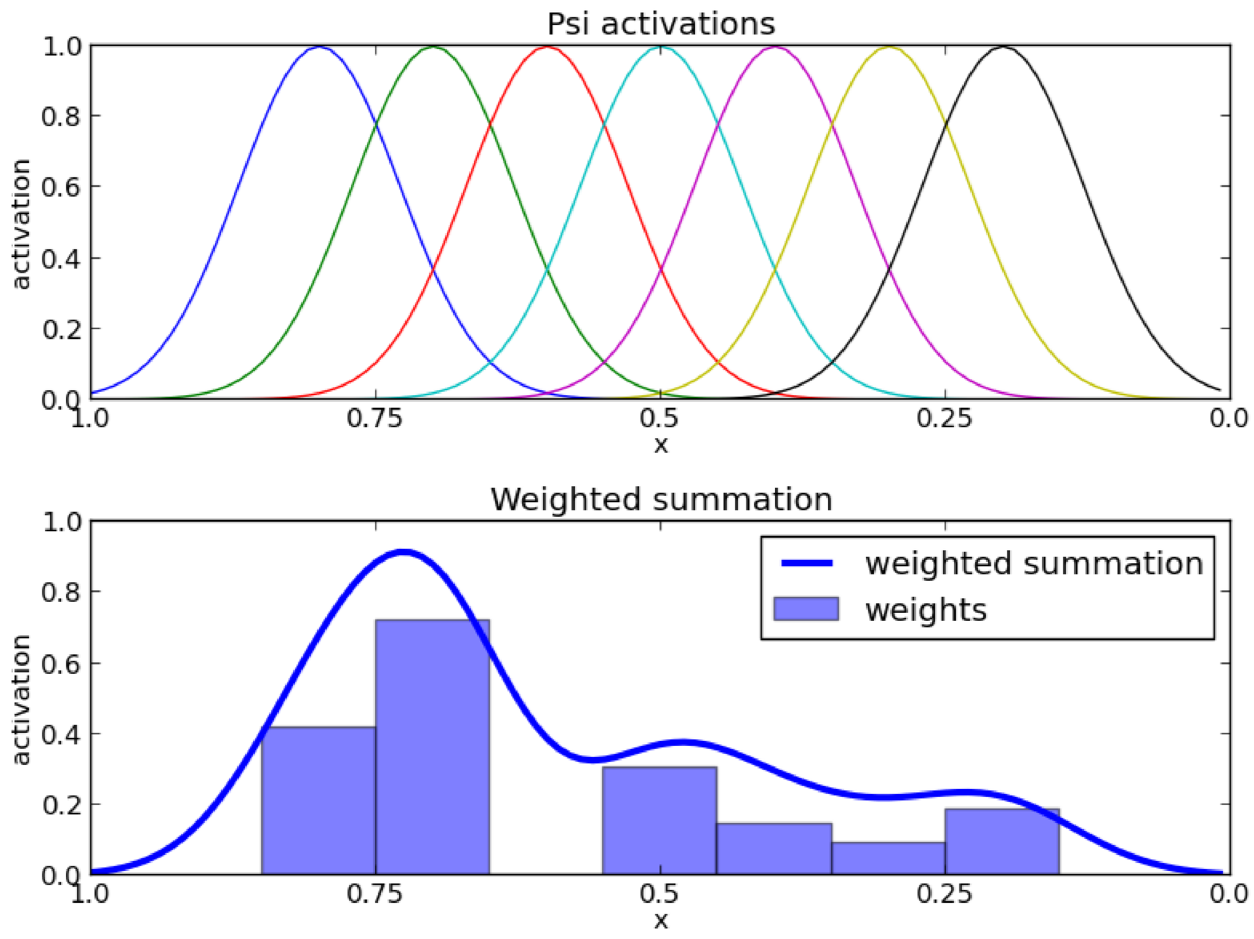


Figure 3.1: Top: The basis functions activations displayed as a function of the canonical system, x. Bottom: Bar plot - The weights assigned to each basis function. Blue line - The weighted summation of each of the basis functions as the canonical system moves from 1 to 0.

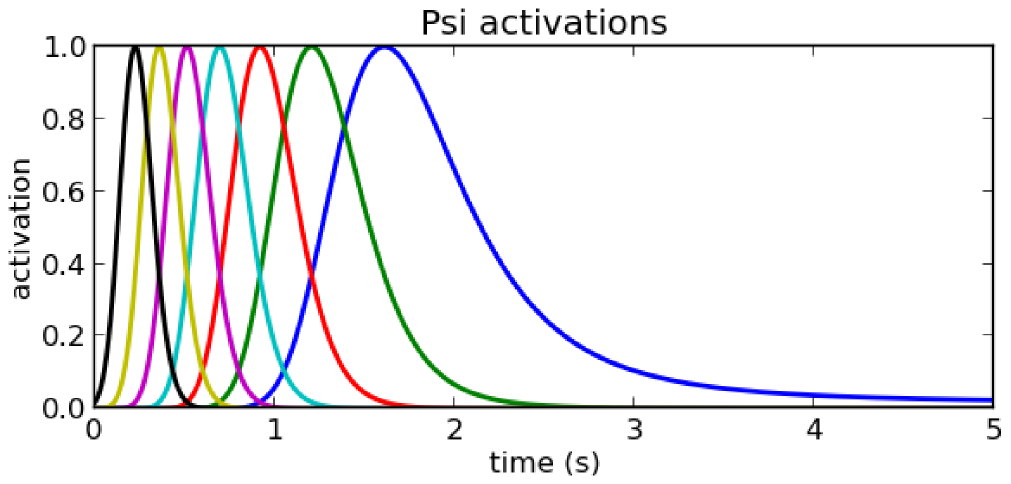


Figure 3.2: A plot of the activation of the Gaussian basis functions over time with an even spacing of the centre points along x. As can be seen, this spacing leads to a bunching of the activations early on.

it nears the target), the Gaussian center points must be chosen more carefully. As it is known that the system decays to 0 exponentially, the basis function centers can be spread evenly by first specify the desired centers, c_i^*, as if the system evolved linearly, then take its negative logarithm:

$$c_i = -\log(c_i^*) \tag{3.6}$$

Additionally, the width of each of the Gaussian needs to be set slightly differently, because the rate at which canonical system decreases slows over time. Consequently, those Gaussians activated later will be activated for longer periods of time given the same variance. To even it out the later basis function widths should be smaller. Through trial and error, a variance that works well is

$$h_i = \frac{\#BFs^{\frac{3}{2}}}{c_i}, \tag{3.7}$$

which reads as the variance of basis function ψ_i is equal to the number of basis functions raised to the power of $\frac{3}{2}$ divided by the center of that basis function. With these specifications, the activation of the basis functions plotted against time is now well spaced out, as shown in Figure 3.3.

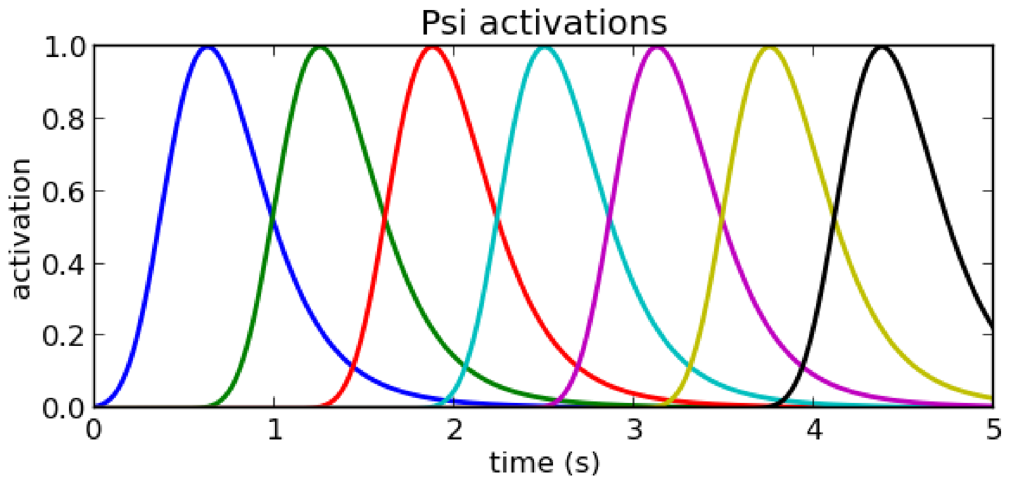


Figure 3.3: A plot of the activation of the Gaussian basis functions over time placing the centre points of the Gaussian basis functions using Eq. 3.6. This leads to a more evenly spread activation.

3.1.2 Spatial scaling

Spatial scaling for DMP systems means that once the system has been set up to follow a desired trajectory to a specific goal, that goal can be moved closer or farther away and the path that is followed will be a scaled version of trajectory originally specified. This is what the $(g - y_0)$ part of the forcing function permits. By scaling the activation of each of the basis functions to be relative to the distance to the target, the system will cover more or less distance.

For example, assume that there is a set of discrete DMPs set up to follow a given trajectory as specified in Figure 3.4 A. The goals have been specified as 1 and .5 for the two DMPs, which is where each DMP ends up, respectively. Everything has been specified for these particular goals (1 and .5), but assume that the goal has been changed to 2, and we would like to generalize the DMPs and get a scaled up version of this trajectory. Without appropriately scaling the forcing function (using the $(g - y_0)$ term) the DMPs traces a path as shown in Figure 3.4 B.

What has happened is that for these new goals the same weightings of the basis functions were too weak to get the system to follow or desired trajectory. Once the $(g - y_0)$ term

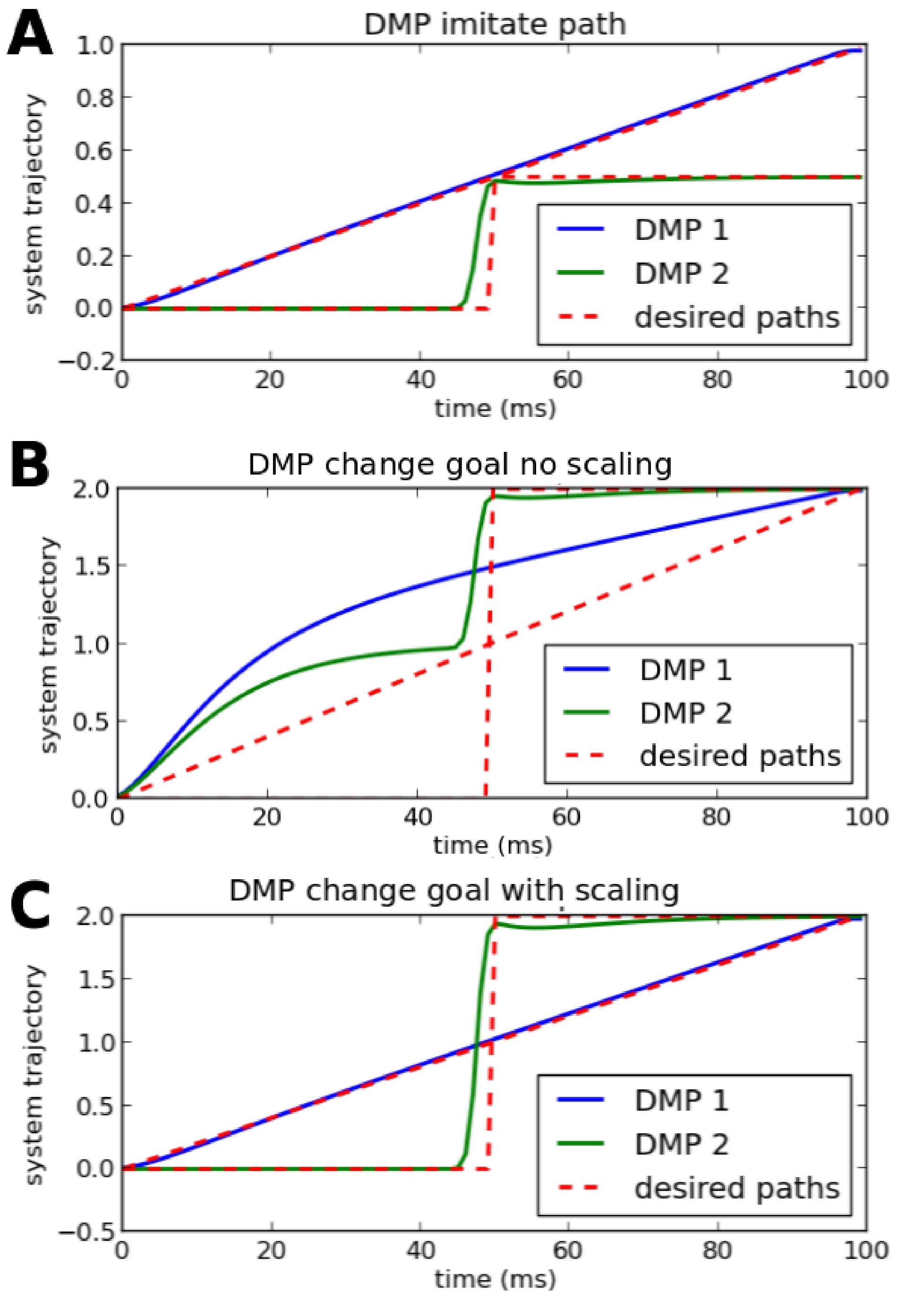


Figure 3.4: Path imitation with DMPs. A) Basic path imitation of a straight line and a step function. B) Changing the goals of the DMPs to be equal to 2, without scaling the forcing function appropriately. C) Changing the goals of the DMPs to be equal to 2, scaling the forcing function appropriately.

included in the forcing function, however, the DMP set traces a path as shown in Figure 3.4 C, which is the desired generalization.

3.1.3 Temporal scaling

For the system to generalize temporally it needs to be able to be run along the same trajectory at any desired speed. Sometimes quick, sometimes slow, but always tracing out the same path. To accomplish this another term needs to be added to the system dynamics, τ; a temporal scaling term. Given the system dynamics:

$$\ddot{y} = \alpha_y(\beta_y(g - y) - \dot{y}) + f, \tag{3.8}$$

$$\dot{y} = \ddot{y} * dt \tag{3.9}$$

$$\dot{x} = -\alpha_x x, \tag{3.10}$$

to achieve temporal flexibility with τ these dynamics are rewritten:

$$\tau\ddot{y} = (\alpha_y(\beta_y(g - y) - \tau\dot{y}) + f), \tag{3.11}$$

$$\tau\dot{y} = \ddot{y} * dt \tag{3.12}$$

$$\tau\dot{x} = (-\alpha_x x). \tag{3.13}$$

To slow down the system the temporal scaling term can be set τ to a value between 0 and 1, and to speed it up it can be set to be greater than 1.

3.2 Imitating a desired path

At this point the DMP system has been set up with a forcing term that can make the system take an interesting path as it converges to a target point, and can scale both temporally and spatially. An important step from this point is getting the system to follow a specific desired trajectory, rather than whatever arises from some random weighting over the basis functions. Ideally, this would work by providing the system a trajectory and have it be able to work backwards and figure out the required forces.

The forcing term affects system acceleration, and by changing the weights on the basis functions it's possible to generate the appropriate force to push the system along the desired trajectory. To figure out the required forces from the provided trajectory, $\mathbf{y}_d$ (where bold denotes a vector, in this case the time series of desired points in the trajectory), first differentiate it twice to get the accelerations:

$$\ddot{\mathbf{y}}_d = \frac{\partial}{\partial t}\dot{\mathbf{y}}_d = \frac{\partial}{\partial t}\frac{\partial}{\partial t}\mathbf{y}_d. \tag{3.14}$$

Once the required acceleration trajectory has been found, the effects of the base point attractor system need to be removed. Since the dynamics of the base system are known exactly, as specified in Eq. 3.11, these dynamics can be compensated for:

$$\mathbf{f}_d = \ddot{\mathbf{y}}_d - \alpha_y(\beta_y(g - \mathbf{y}_d) - \dot{\mathbf{y}}_d), \tag{3.15}$$

Because the forcing term is comprised of a weighted summation of basis functions which are activated through time, an optimization technique like locally weighted regression can be used to choose weights over our basis functions such that the output matches the desired force trajectory $\mathbf{f}_d$. Locally weighted regression sets up the weights to minimize:

$$\Sigma_t \psi_i(t)(f_d(t) - w_i(x(t)(g - y_0)))^2, \tag{3.16}$$

and the solution is

$$w_i = \frac{\mathbf{s}^T \boldsymbol{\psi}_i \mathbf{f}_d}{\mathbf{s}^T \boldsymbol{\psi}_i \mathbf{s}}, \tag{3.17}$$

where

$$\mathbf{s} = \begin{pmatrix} x_{t_0}(g - y_0) \\ \vdots \\ x_{t_N}(g - y_0) \end{pmatrix}, \quad \boldsymbol{\psi}_i = \begin{pmatrix} \psi_i(t_0) & \dots & 0 \\ 0 & \ddots & 0 \\ 0 & \dots & \psi_i(t_n) \end{pmatrix} \tag{3.18}$$

Figure 3.5 shows an example of the system imitating a sine wave and a piecewise function with different numbers of basis functions.

3.3 Interpolating trajectories for imitation

When imitating trajectories there can be some issues with having enough sample points and fitting them to the canonical system's timeframe. To get around these problems the system can be set up such that the canonical system always runs for 1 second, and whenever a trajectory is passed in to be imitated it gets scaled to be one second long. This can be done easily in code using interpolation:

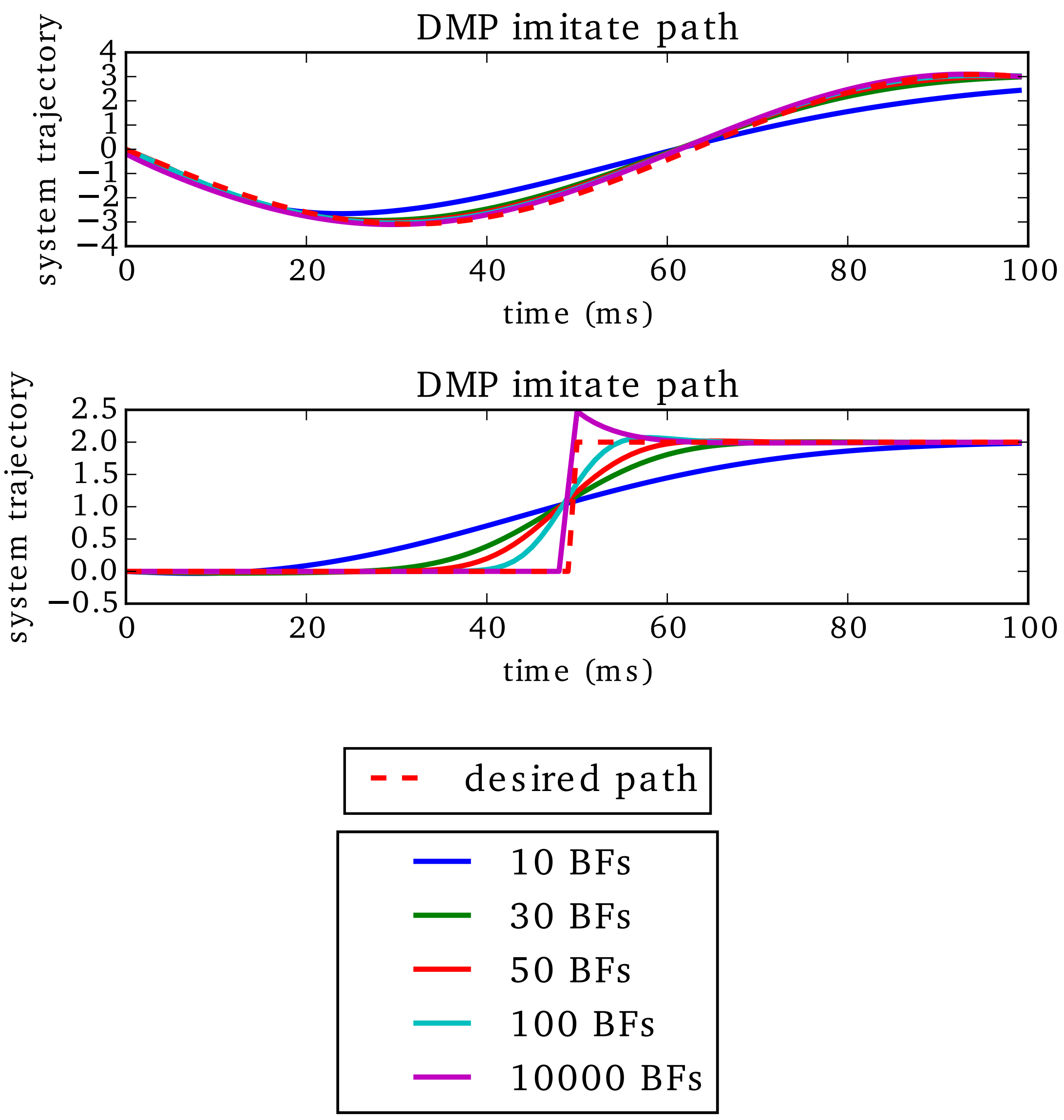


Figure 3.5: An example of using the DMP trajectory imitation to follow a set of specified trajectories with different numbers of basis functions. The more complex a trajectory, the more basis functions are required to be able to accurately follow it.

```
# generate function to interpolate the desired trajectory
import scipy.interpolate

path = np.zeros((self.dmps, self.timesteps))
x = np.linspace(0, self.cs.run_time, y_des.shape[1])

for d in range(self.dmps):

    # create interpolation function
    path_gen = scipy.interpolate.interp1d(x, y_des[d])

    # create evenly spaced sample points of desired trajectory
    for t in range(self.timesteps):
        path[d, t] = path_gen(t * self.dt)

y_des = path
```

The result is that now every trajectory is set up to run in the same time. This has the benefit of allowing temporal scaling to affect all movements the same way, regardless of the original length of the imitated path.

3.4 Controlling end-effector trajectories

As mentioned above, the dynamic movement primitive (DMP) framework was designed for trajectory control. In Chapter 2 an operational space controller (OSC) was developed, allowing forces to be specified in end-effector space. Here a set of DMPs is placed on top of the OSC to drive the system through complex trajectories. The 3 link arm and its OSC controller will be collectively referred to as 'the plant' throughout this section.

The DMPs here will be specifying the (x, y) trajectory of the hand, and the OSC will take care of transforming the desired hand forces into torques that drive the arm as desired. Additionally, a feedback signal with the (x, y) position of the hand will be sent back from the plant to the DMP system.

To generate a force signal from the target (x, y) positions is straight-forward. A basic control signal will be calculated by measuring the difference between the state of our DMP system and the plant state. Formally,

$$u = k_p(y_{\text{DMP}} - y), \tag{3.19}$$

where y_{DMP} is the state of the DMP system, y is the state of the plant, and k_p and is the position error gain term.

To have the system trace out the DMP trajectory then requires stepping the DMP system forward and setting gain values high enough that the plant is able to follow the DMP's trajectory.

Figure 3.6 shows a demonstration of a DMP controller following trajectories of digits, comparable with those drawn by motor control system for Spaun [50], shown later in Chapter 5, Figure 5.6. Both of the systems used to generate these numbers are not implemented in neurons. As can be seen, the combination of DMPs and operational space control is much more effective than the previous implementation [40].

3.5 Incorporating system feedback

One of the issues in implementing the control specified above is that some tuning has to be done regarding how quickly the DMP trajectory moves, because the DMP system isn't constrained by physical dynamics, but the plant can move reliably only up to a certain maximum velocity. Depending on the scaling of the movement, the DMP trajectory may be telling the system to move a metre a second or an centimetre a second.

Figure 3.7 shows the arm drawing the number 3 when the DMP system is moving too fast. The plant is unable to fully draw out the desired trajectories in places where the DMP system moved too quickly. The only way to remedy this without feedback is to have the DMP system move more slowly throughout the entire trajectory. It would be preferable to allow the system to adjust on-the-fly. Essentially, it's desirable to have the system to go as fast as possible as long as the plant state is within some threshold distance. This is how system feedback is used here.

It turns out to be straightforward to implement this: If the plant state falls behind the DMP state, slow down the execution speed of the DMP to allow the plant time to catch up. The do this a new term based on the system feedback is introduced into the canonical system:

$$\frac{1}{1 + \alpha_{\text{err}}(y_{\text{plant}} - y_{\text{DMP}})}. \tag{3.20}$$

This new term slows down the canonical system when there's an error. It can be thought of as an additional scaling on the time step. Additionally, the sensitivity of this term can be modulated using the scaling term α_{err} on the difference between the plant and DMP states.

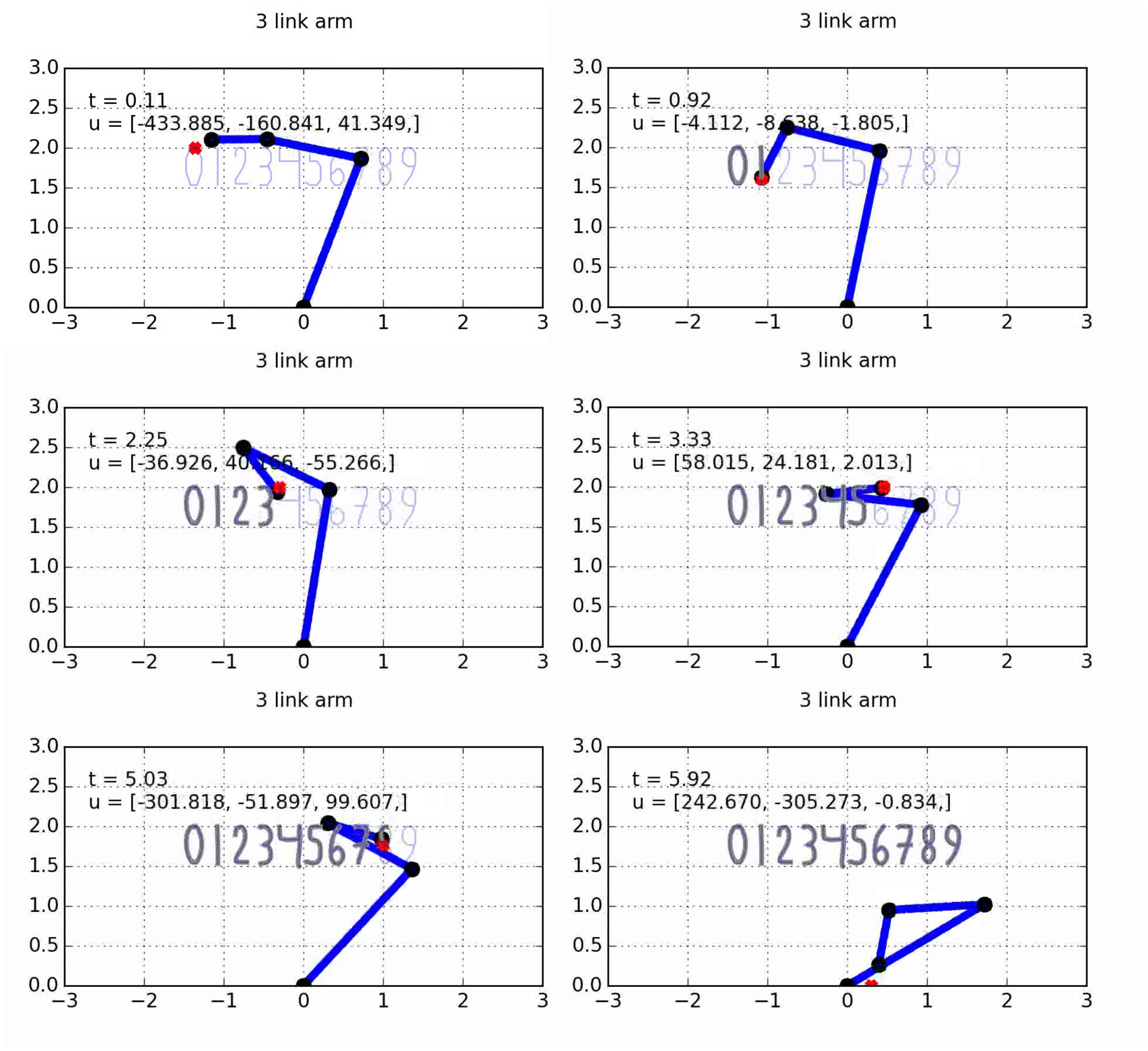


Figure 3.6: An example application of the DMP trajectory system for drawing the numbers 0-9. The light gray lines are the desired trajectories, the dark gray lines are the paths drawn by the arm. The red dot represents the desired position for the hand, specified by DMPs.

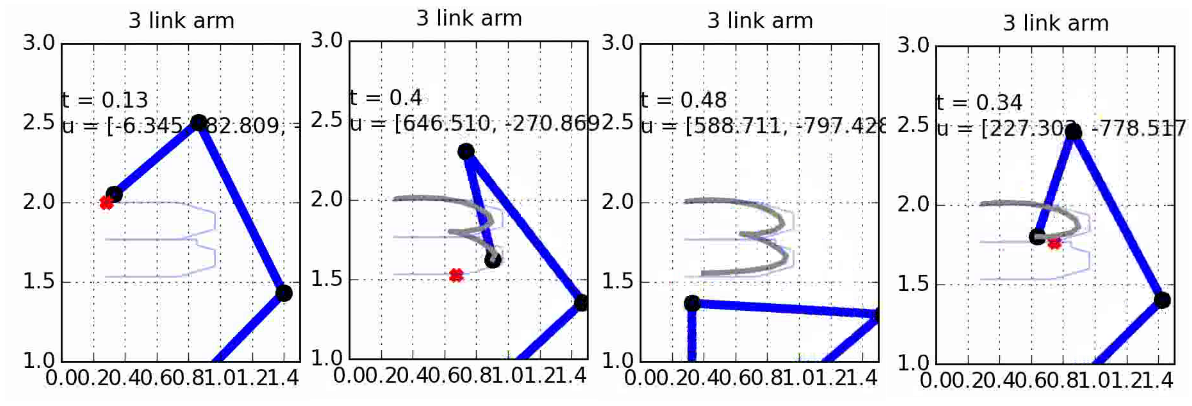


Figure 3.7: Drawing the number 3 when the DMP system is moving faster than the plant can follow. The plant fails to draw the sharp corners of the 3.

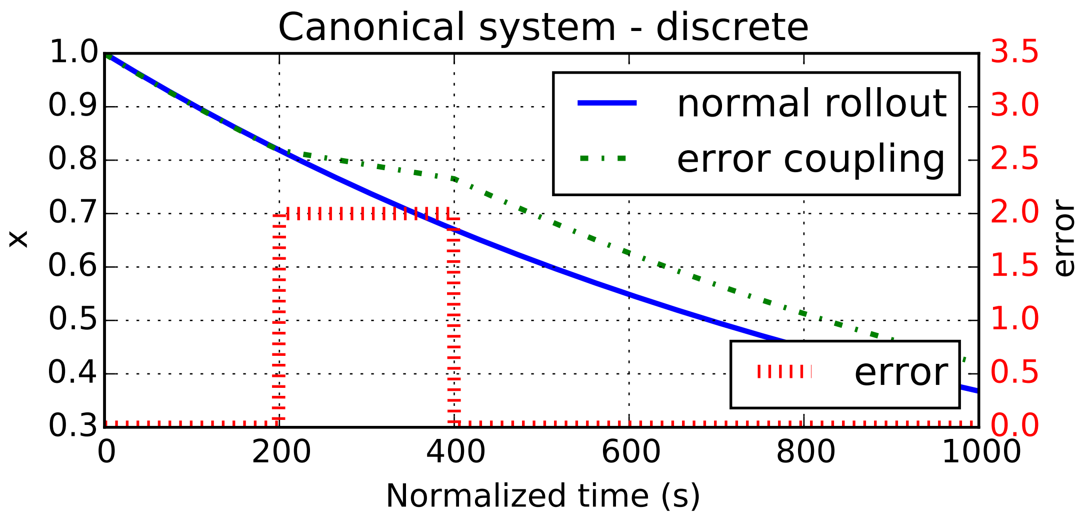


Figure 3.8: The blue line shows how the canonical system evolves when the error is 0, representative of the plant being within tolerance of the position of the DMP. The green line shows the dynamics of the canonical system when an error, shown in the red line with an axis along the right side of the figure, is introduced.

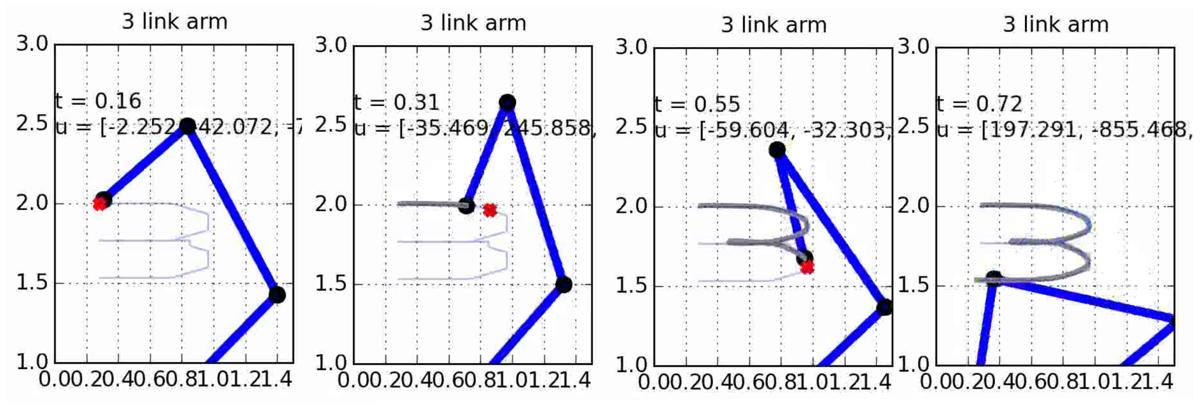


Figure 3.9: Drawing the number 3 using system feedback to slow down the canonical system execution speed whenever the target point specified by the DMP system gets too far ahead of the plant.

Figure 3.8 shows how this term affects the system by looking at the dynamics of the canonical system when an error is introduced mid-run.

As can be seen, when the error is introduced the dynamics of the canonical system slow down. Figure 3.9 shows an example of using this feedback term to slow down the execution of the canonical system when drawing the number 3. The specified trajectory still isn't traced out exactly, but this can be remedied by increasing the α_{err} term to make the DMP time-step decrease even further whenever the DMP gets ahead of the plant.

3.6 Direct trajectory control vs DMP based control

Using the above interpolation function it is possible to directly use the interpolation function's output to drive the plant. When a controller is set up in this fashion, very precise control of the end-effector is observed, more precise than the DMP control. The reason for this is that the DMP system is *approximating* the desired trajectory using a set of basis functions. Consequently some accuracy is lost in this approximation. Using the interpolation function to drive the plant directly the desired trajectory can be traced exactly.

The accuracy difference between the two approaches becomes more evident when the desired trajectories are especially complex. It's possible to improve the performance of the DMP guided system by choosing the centres of the basis functions dependent on the complexity of the trajectory [204], but for simply replicating a provided trajectory directly the direct trajectory control is simpler. Figure 3.10 shows a comparison of the two approaches for guiding the plant through drawing the word ‘kaplow’.

From this figure we can see that using the interpolation function to guide the plant gives the *exact* path that was specified, where using DMPs introduces additional error. Additionally, the error introduced by using DMPs increases with the size of the desired trajectory. Importantly, however, this is for comparing the drawing out of a given trajectory exactly as it was defined. Often this is not the desired behaviour, as the scale of the trajectory may need to be modulated to fit in some specific space. The strength of the DMP framework is that what guides the trajectory is a dynamical system. This allows interesting performance with simple changes to the parameters of the system. To generalize the scale of the trajectory, all that needs to be changed is the goal of the system. Figure 3.11 shows the DMP system guiding the plant through drawing the number ’3’ normally, scaled along the y axis, and then scaled along both axes, achieved by changing the goal for the x and y DMPs.

3.7 Rhythmic DMPs

The implementation of rhythmic DMPs only requires a few slight modifications of the discrete DMP case.

3.7.1 Obtaining consistent and repeatable basis function activation

First, consider the canonical system: In the discrete case the canonical system (which drives the activation of the basis functions) decayed from 1 to 0 and then stopped. In the rhythmic case the system should repeat indefinitely, so a reliable way of continuously activating the basis functions in the same order is needed. A simple, repeating function that can be used to accomplish this is the cosine function.

The idea is that the basis functions will be laid out along the range of 0 to 2π, and the canonical system will be set to forever increase linearly. The difference between the canonical system state and each of the center points will then be passed into the cosine function. Because the cosine function repeats at 2π, a repeating spread of basis function activations will be achieved. The cosine output can then be used in the Gaussian equation (with the gain and bias terms) and rhythmic activation of the basis function will be

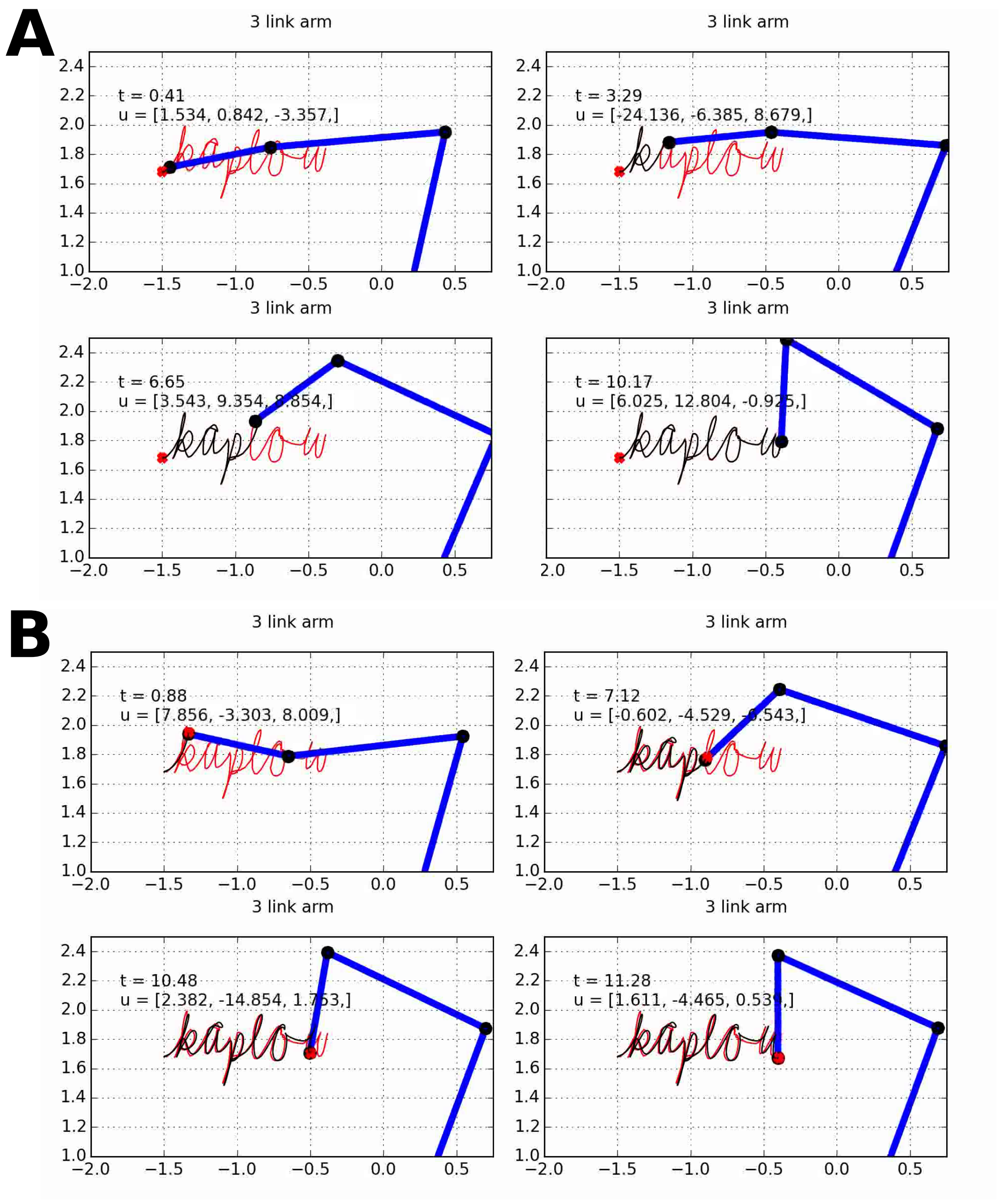


Figure 3.10: A comparison of direct trajectory control and DMP guided control for writing the word ‘kaplow’ as a single complex trajectory. The red line is the desired trajectory and the black line is the path drawn by the arm. A) Direct trajectory control. B) DMP guided control.

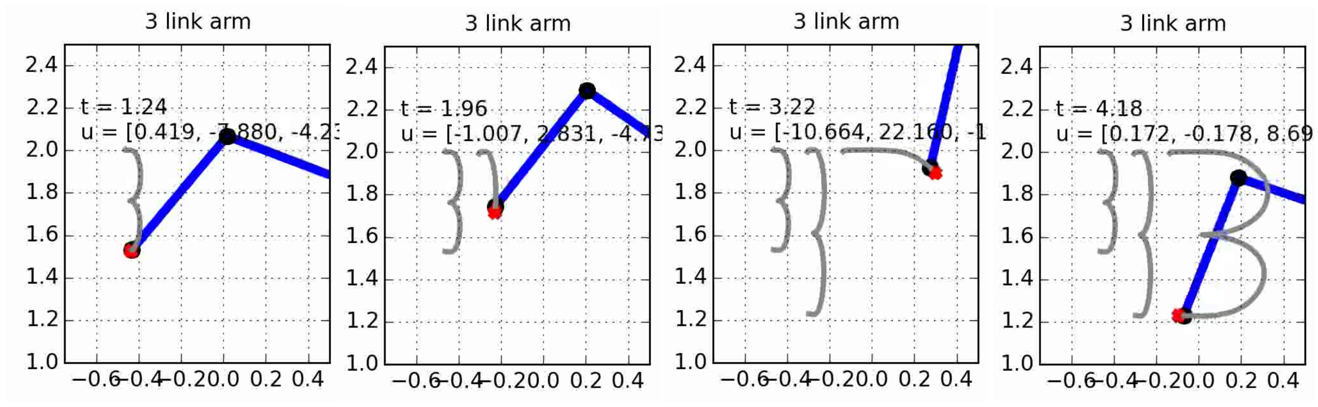


Figure 3.11: The DMP system guiding the plant through drawing the number '3' at three different scales. Normally, scaled along the y axis, and scaled along both axes. No changes were made to the system aside from modifying the goal state for the x and y DMPs.

achieved. Formally,

$$\dot{x} = 1 \tag{3.21}$$

$$\psi = \exp(h * \cos(x - c) - 1), \tag{3.22}$$

where x is the state of the canonical system, c is the basis function center point, and h is a gain term controlling the variance of the Gaussian. Figure 3.12 displays the activations of three basis functions with centers spread out evenly between 0 and 2π.

As long as our canonical system increases at a steady pace the basis functions will continue to activate in a reliable and repeatable way.

Additionally, the placement of the center points of the rhythmic system is easier than in the traditional discrete case because the rhythmic canonical system dynamics are linear.

3.8 Other differences between discrete and rhythmic

In the implementation of the rhythmic system, there are other differences that must also be accounted for. The first is that there is no diminishing term in the rhythmic system, because we wish the system to continuously activate the basis functions with the same strength. The second is that setting the goal states for path imitation changes. Assuming that the desired path is going to be a rhythmic pattern the goal state will be set as the

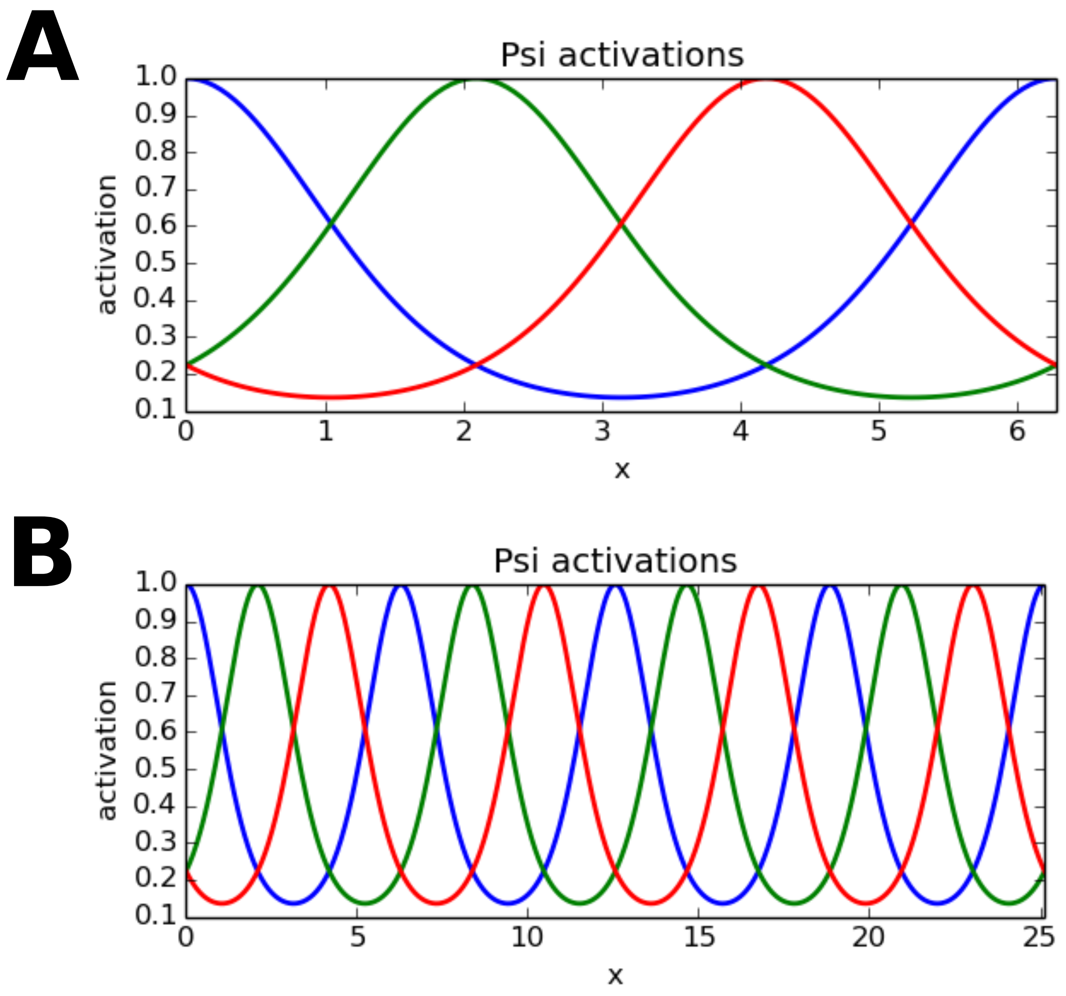


Figure 3.12: A) Three basis functions laid out evenly between 0 and 2π. B) Repeated activation of the basis functions.

center point of the desired trajectory, calculated by finding the average position of the trajectory along each dimension.

3.8.1 Example

As an example of rhythmic DMPs the locomotion pattern of humans during walking can be imitated. The desired trajectory has been taken from human kinematics data [1], tracking the position of the hip, knee, and ankle joint angles, and is shown in the top of Figure 3.13. This means that the DMPs will be directing the trajectories in joint space, in contrast to the discrete examples above where there was a DMP for the x and y positions of the end-effector. In this example there are three DMPs; one for each joint. The bottom of Figure 3.13 shows the implementation results.

3.9 Applications of DMPs

In this section, several of the applications of DMPs to different control problems are briefly surveyed, to give a sense of how these techniques can be generalized.

One common implementation is to observational learning [159]. In this application human movement was first tracked as a tennis racket was swung to generate the observed trajectory. Specifically, the joint angles were tracked and a DMP generated to guide a robotic system through a similar movement. The desired trajectory was followed by using inverse kinematics, rather than through operational space control, as is done in this thesis. Figure 3.14 shows an example of the robot swinging the tennis racket. Because of the strong generalizability of DMPs, a visual system was able to identify the position of the ball as it was thrown and update the goal position of the trajectory swing such that the robot was able to follow the ball and hit it as it was thrown to different locations [159].

Obstacle avoidance is also natural to add to the DMP framework [95]. Because the trajectory is generated from a dynamical system, all that is required is adding another term to the dynamics that pushes the system away from any identified obstacles. In [95] this is done by adding a force perpendicular to the object with a strength proportional to how close the system is to the target. Figure 3.15 shows an example set of trajectories reaching straight to a goal with an obstacle and obstacle avoidance term added to the DMP dynamics.

[1]found here: `http://www.sfu.ca/~leyland/Kin201%20Files/4%20Gait%20Analysis%20&amp;%20Angular%20Kinematics.pptx.pdf`

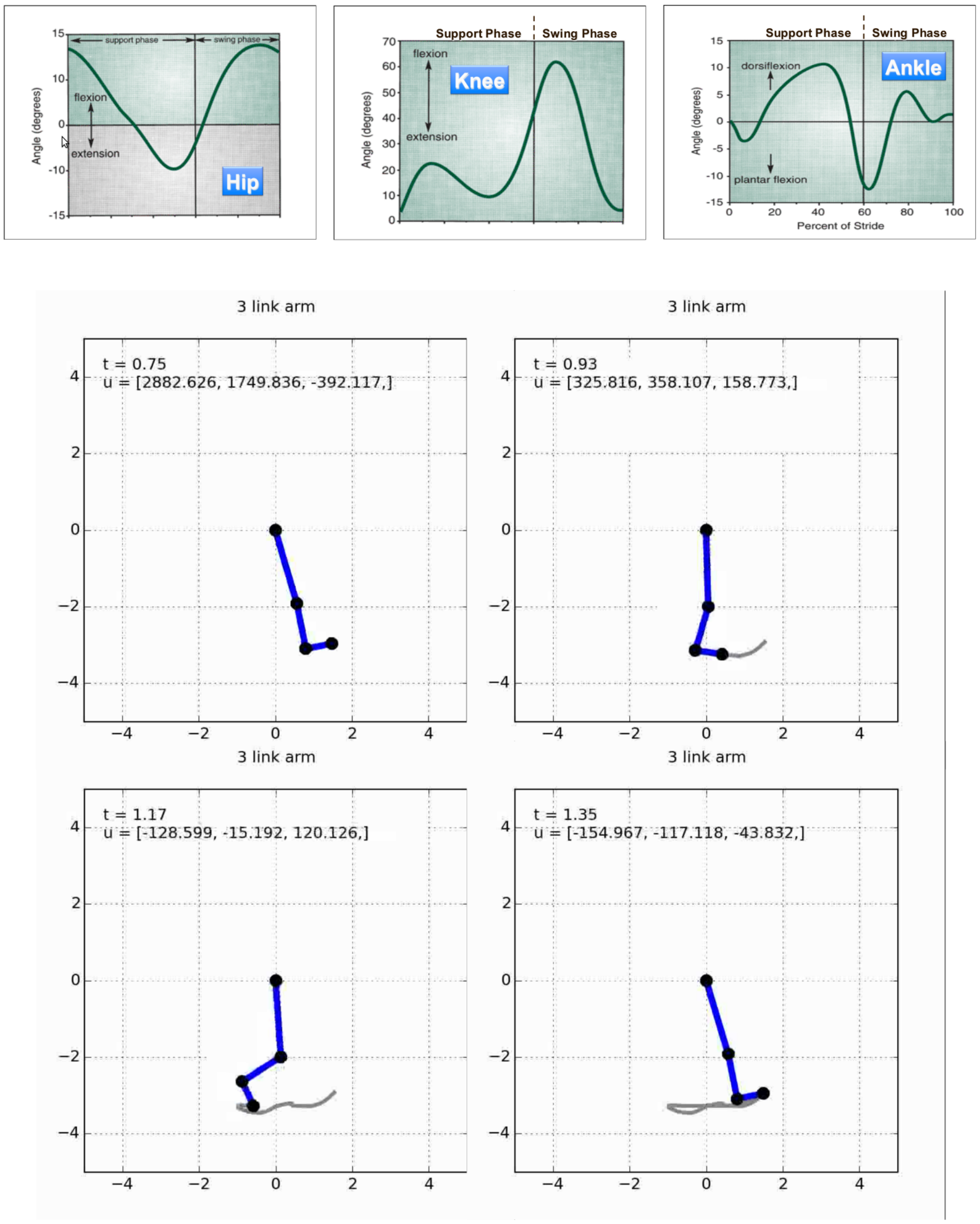


Figure 3.13: Top: The kinematics recorded from tracking the hip, knee, and ankle angles during human walking. Bottom: A DMP guided system directing a leg model through the same motions.

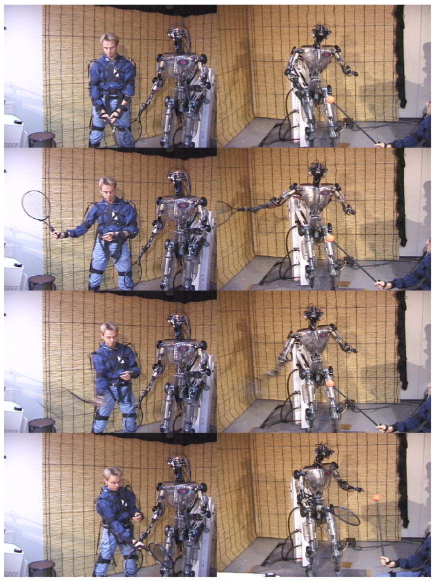


Figure 3.14: Left column: Teacher demonstration of a tennis swing. Right column: Imitation of a tennis swing by a humanoid robot, taken from [159].

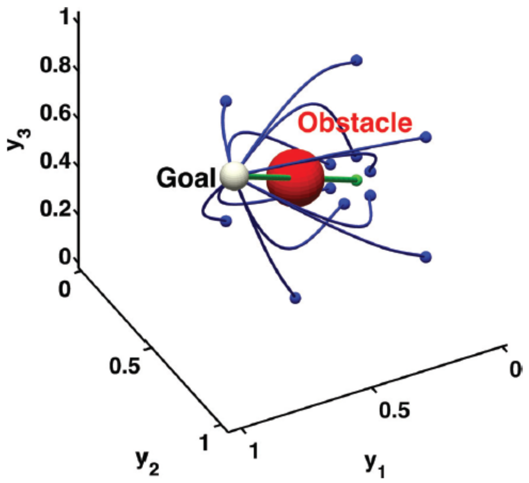


Figure 3.15: An example of moving from a goal to a target while avoiding an obstacle. Blue lines are the trajectories, the red ball is the obstacle, the white ball is the target, and the green line represents the path taken when no obstacle is present. Taken from [95].

Another application of DMPs is to sensory processing, providing a corrective signal based on current feedback compared to stereotypical sensory feedback on successful trials. Specifically, a set of sensors report their signals during success trials of a given task, such as grasping an object. These sensory trajectories are stored in a DMP and played out at the same time as future trials. When there is a discrepancy between the stored feedback signals and the current feedback trials the control signal is updated accordingly to try to bring the sensory feedback in line with the trajectory from the successful trial. For example, if a robotic manipulator is grasping a flashlight and a force sensor reports pressure feedback from one of the fingers before expected, the manipulator can stop moving that finger, or even move backwards, until the other fingers are in place, and then continue moving forward.

Reinforcement learning methods for learning to optimize DMP trajectories have also been developed, dubbed Policy Improvement through Path Integration (PI^2) [185]. PI^2 is a batch learning algorithm that tracks the immediate cost of the trajectory throughout movement, and then back-calculates the cost-to-go at each point in time with a reversed cumulative summation. Noise is added to each of the weights on the basis functions, and the differences in the cost-to-go across several trials are compared to approximate the value function manifold, at which point the weights over the basis functions are adjusted to follow the downward slope along the manifold. This method has proven highly efficient and been used to learn effective trajectories for grasping objects [128], opening door handles [104], impedance control [22], and simple games such as catching a ball on a string in a cup or bouncing a ball attached to a ping pong paddle [206].

Additionally, categorization of movements is possible with DMPs, as similar movements have similar weightings over the basis functions. Figure 3.16 shows a correlation plot of the similarity between trajectories trained to imitate different graffiti letters, 5 of each letter [95]. As can be seen, the trajectories for each letter corresponds most closely with other letters that are the same. By performing this correlation measure on uncategorized trajectories a reasonable categorization can by applied. Potential applications of this approach to categorization is discussed at the end of Chapter 7.

The literature surrounding applications and extensions to the DMP framework is substantial. This work, combined with the biological plausibility of DMPs, serves to make the incorporation of the DMP framework into neural models of the motor control system attractive. Previously, evidence for systems generating basic movements and trajectories has been based on the observations of ‘movement synergies’ in frogs and cats, where electrical stimulation results in the leg converging to a target location or a repeated pattern of movement [70, 37]. The argument for the biological plausibility of the DMP framework made here is based on its success capturing various behavioural phenomena of motor control (e.g.

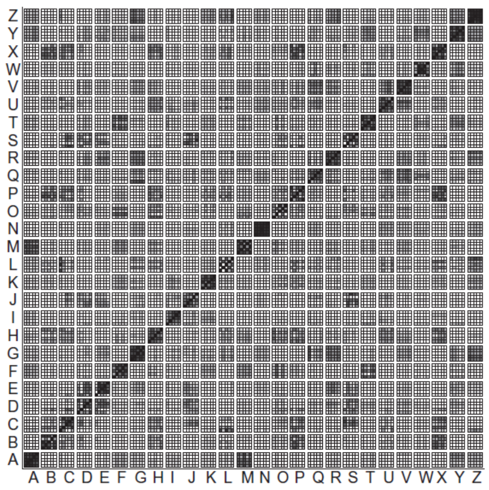


Figure 3.16: Correlations between the weight vectors of different graffiti characters (5 instances of each letter). Black represents a correlation of 1.0 and white represents a correlation of 0.0 or below. Taken from [95].

generalizability, planning in operational space, etc), along with the natural implementation of dynamical systems in recurrent neural networks (discussed in Chapter 6), and will be continued in Chapter 7.

Chapter 4

Nonlinear adaptive control

Effective control of a system is dependent on having an accurate model of the system's dynamics. As discussed in Chapter 2, obtaining an accurate model of the system dynamics is not an easy task; to the point that forces known to be affecting movement are not included and accounted for in the control signal because small errors in the correction signal can lead to system instability (e.g. Coriolis forces). Learning algorithms developed by Dr. Jean-Jacques Slotine [170] have addressed the problem of developing accurate system models by starting with a simple model with the known dynamics, and introducing nonlinear adaptive terms that learn to compensate for the errors introduced by the unmodeled dynamics. This chapter works through these algorithms and their derivations, showing how the learning rules were created through Lyapunov function stability analysis. In Chapter 6 these methods are extended to work in a neurally plausible setting.

There are two types of learning algorithms that will be discussed, which can be distinguished by how they are applied to the original control signal. The first is an adaptive 'bias' term, which compensates for unknown dynamic forces, and is added to the original control signal. The second is an adaptive 'transform' term, which compensates for unknown kinematic variables, and is multiplied by the original control signal.

4.1 Lyapunov's direct method

In the analysis of nonlinear systems Lyapunov's direct method provides a means of proving system stability at an equilibrium. An equilibrium is some state, $\mathbf{0}$, such that when the system arrives at this state it does not leave it. A stable system is one such that starting it near its desired operating point implies that it will remain around the point.

The framing of Lyapunov's direct method is based on creating and analyzing a characterization of the system such that it can be seen that the dynamics of the system will evolve such that the system can be guaranteed to be stable, and ideally converge to equilibrium. The idea behind this approach was based on the observation of physical systems and their energy: If a system continuously dissipates energy then it will eventually settle down to an equilibrium point. Proving a system to be stable can be accomplished by creating some scalar function of the system state analogous to the energy function of physical systems that is 0 at the equilibrium and positive everywhere else, and whose derivative is negative. Intuitively this works because if the 'energy' of a system is dissipated over time and lower bounded, it can be guaranteed to converge to the lower bound; which is in this case is an equilibrium point.

There are three different basic types of stability in Lyapunov's direct method: local stability, local asymptotic stability, and global asymptotic stability. Informally,

- If the candidate function is locally positive definite and the derivative is locally negative semi-definite, then the system is stable inside a given radius.
- If the candidate function is locally positive definite and the derivative is locally negative definite, then the system will always converge to the equilibrium state starting inside a given radius.
- If the candidate function is globally positive definite and the derivative is globally negative definite, then regardless of initial state the system will always converge to equilibrium.

Denoting the Lyapunov function $V(\mathbf{x})$, where $\mathbf{x}$ is the system state, these can be formally written as

- Given
$$V(\mathbf{x}) > 0 \;\; \forall \mathbf{x} \in \mathcal{B}, \tag{4.1}$$
$$\dot{V}(\mathbf{x}) \leq 0 \;\; \forall \mathbf{x} \in \mathcal{B}, \tag{4.2}$$
for some neighborhood $\mathcal{B}$ of the equilibrium, then the system is proven to be stable.
- Given
$$V(\mathbf{x}) > 0 \;\; \forall \mathbf{x} \in \mathcal{B}, \tag{4.3}$$

$$\dot{V}(\mathbf{x}) < 0 \ \ \forall \mathbf{x} \in \mathcal{B}, \tag{4.4}$$

for some neighborhood $\mathcal{B}$ of the equilibrium, then the system is proven to be locally asymptotically stable.

- Given

$$V(\mathbf{x}) > 0 \ \ \forall \mathbf{x} \in \mathbb{R}, \tag{4.5}$$

$$||\mathbf{x}|| \to \infty \implies V(\mathbf{x})(x) \to \infty, \tag{4.6}$$

$$\dot{V}(\mathbf{x}) < 0 \ \ \forall \mathbf{x} \in \mathbb{R}, \tag{4.7}$$

then the system is proven to be globally asymptotically stable. The second condition here is referred to as radial unboundedness, meaning that the radius of asymptotic stability is unbounded.

Asymptotic stability is a very desirable feature of a system, but can be difficult to prove with the above theorems because often $\dot{V}(\mathbf{x})$ is only negative semi-definite, rather than negative definite, and finding a Lyapunov function candidate with a negative definite derivative can be very difficult. To skirt the difficulty of finding a candidate function with a strictly negative derivative, the invariant set theorem can be used. An invariant set is any set of states of a dynamical system where once the set has been entered the system remains in that set. Some basic examples of invariant sets are equilibrium points or limit cycles.

The local invariant set theorem says that given a continuous system with continuous first partial derivatives and a scalar function $V(\mathbf{x})$, assume

$$\dot{V}(\mathbf{x}) \leq 0 \ \ \forall \mathbf{x} \in \Omega_l \tag{4.8}$$

where Ω_l is a bounded region where $V(\mathbf{x}) < l$ for some $l > 0$, and $\mathcal{M}$ is the union of all invariant sets (e.g. equilibrium points and limit cycles) in Ω_l. Then, as $t \to \infty$ the system will go to $\mathcal{M}$ for any trajectory started in Ω_l.

Using these tools asymptotic stability will be shown for autonomous adaptive systems.

4.2 Stability analysis of a robot arm

As discussed in Chapter 2, the dynamics of an n-link robotic arm are

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) = \mathbf{u}, \tag{4.9}$$

where $\mathbf{M}(\mathbf{q}) \in \mathbb{R}^{n \times n}$ is the inertia matrix, $\mathbf{u} \in \mathbb{R}^n$ is the joint torque applied to the robot, $\mathbf{g}(\mathbf{q})$ is the gravitational force, and $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ represents the centripetal and Coriolis forces. Note that the effects of viscosity have been left out for simplicity.

Importantly, $\dot{\mathbf{M}}(\mathbf{q}) - 2\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ is skew-symmetric [170] (i.e. $a_{ij} = -a_{ji}$ where a_{ij} is the element at row i column j). This is noteworthy because a skew-symmetric matrix $\mathbf{A}$ has the property that $\mathbf{v}^T \mathbf{A} \mathbf{v} = 0$ for any vector $\mathbf{v}$. The intuition behind the skew-symmetry of $\dot{\mathbf{M}}(\mathbf{q}) - 2\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ comes from the law of conservation of energy. This says that the time derivative of the robot arm's kinetic energy $\dot{\mathbf{q}}^T \mathbf{M}(\mathbf{q}) \dot{\mathbf{q}}$ must be equal to the external power input on the system. The fact that power equals velocity times force, and that the only external forces acting on the system are the input $\mathbf{u}$ and the gravity term $\mathbf{g}(\mathbf{q})$, means then that

$$\frac{1}{2} \frac{d}{dt} (\dot{\mathbf{q}}^T \mathbf{M}(\mathbf{q}) \dot{\mathbf{q}}) = \dot{\mathbf{q}}^T (\mathbf{u} - \mathbf{g}(\mathbf{q})). \tag{4.10}$$

Expanding the right hand side,

$$\frac{1}{2} \frac{d}{dt} (\dot{\mathbf{q}}^T \mathbf{M}(\mathbf{q}) \dot{\mathbf{q}}) = \frac{1}{2} \left(\ddot{\mathbf{q}}^T \mathbf{M}(\mathbf{q}) \dot{\mathbf{q}} + \dot{\mathbf{q}}^T \dot{\mathbf{M}}(\mathbf{q}) \dot{\mathbf{q}} + \dot{\mathbf{q}}^T \mathbf{M}(\mathbf{q}) \ddot{\mathbf{q}} \right). \tag{4.11}$$

Each of the terms reduce to scalar values, and so are equal to their own transpose, i.e.

$$\ddot{\mathbf{q}}^T \mathbf{M}(\mathbf{q}) \dot{\mathbf{q}} = (\ddot{\mathbf{q}}^T \mathbf{M}(\mathbf{q}) \dot{\mathbf{q}})^T = \dot{\mathbf{q}}^T \mathbf{M}(\mathbf{q}) \ddot{\mathbf{q}}, \tag{4.12}$$

so the left hand side of Eq. 4.11 may be rewritten

$$\dot{\mathbf{q}}^T \mathbf{M}(\mathbf{q}) \ddot{\mathbf{q}} + \frac{1}{2} \dot{\mathbf{q}}^T \dot{\mathbf{M}}(\mathbf{q}) \dot{\mathbf{q}}. \tag{4.13}$$

Using Eq. 4.9 to substitute into Eq. 4.13 for $\mathbf{M}(\mathbf{q}) \ddot{\mathbf{q}}$ gives

$$\dot{\mathbf{q}}^T (\mathbf{u} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \dot{\mathbf{q}} - \mathbf{g}(\mathbf{q})) + \frac{1}{2} \dot{\mathbf{q}}^T \dot{\mathbf{M}}(\mathbf{q}) \dot{\mathbf{q}}. \tag{4.14}$$

Rearranging terms gives

$$\dot{\mathbf{q}}^T (\mathbf{u} - \mathbf{g}(\mathbf{q})) + \dot{\mathbf{q}}^T \left(\frac{1}{2} \dot{\mathbf{M}}(\mathbf{q}) - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \right) \dot{\mathbf{q}}. \tag{4.15}$$

Setting Eq. 4.15 equal to the right hand side of Eq. 4.10 gives

$$\dot{\mathbf{q}}^T (\mathbf{u} - \mathbf{g}(\mathbf{q})) + \dot{\mathbf{q}}^T \left(\frac{1}{2} \dot{\mathbf{M}}(\mathbf{q}) - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \right) \dot{\mathbf{q}} = \dot{\mathbf{q}}^T (\mathbf{u} - \mathbf{g}(\mathbf{q})), \tag{4.16}$$

or

$$\dot{\mathbf{q}}^T \left(\frac{1}{2}\mathbf{M}(\mathbf{q}) - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \right) \dot{\mathbf{q}} = \mathbf{0}. \tag{4.17}$$

Additionally, this equality means that

$$\dot{\mathbf{M}}(\mathbf{q}) = 2\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}), \tag{4.18}$$

which will allow us to substitute in $2\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ for $\mathbf{M}(\mathbf{q})$ during analysis, which will also aid in proving stability.

4.2.1 Basic system stability

Choose for a Lyapunov function candidate the kinetic energy of the system

$$V(\mathbf{q}) = \frac{1}{2}\dot{\mathbf{q}}^T\mathbf{M}(\mathbf{q})\dot{\mathbf{q}}. \tag{4.19}$$

This is an appropriate seeming candidate function because $V(\mathbf{q})$ is a scalar function and positive semi-definite (because the inertia matrix $\mathbf{M}(\mathbf{q})$ is symmetric and uniformly positive definite for all $\mathbf{q} \in \mathbb{R}^n$), is radially unbounded (i.e. $V(\mathbf{q}) \to \infty$ as $||\mathbf{q}|| \to \infty$), and is lower bounded by 0, obtained when $\dot{\mathbf{q}} = \mathbf{0}$.

The derivative is as above:

$$\begin{aligned} \dot{V}(\mathbf{q}) &= \dot{\mathbf{q}}^T(\mathbf{u} - \mathbf{g}(\mathbf{q})) + \dot{\mathbf{q}}^T \left(\frac{1}{2}\dot{\mathbf{M}}(\mathbf{q}) - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \right) \dot{\mathbf{q}}, & (4.20) \\ \dot{V}(\mathbf{q}) &= \dot{\mathbf{q}}^T(\mathbf{u} - \mathbf{g}(\mathbf{q})), & (4.21) \end{aligned}$$

where Eq. 4.21 follows from Eq. 4.20 and Eq. 4.17.

Choosing the control signal

$$\mathbf{u} = -\mathbf{K}_v\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) \tag{4.22}$$

where $\mathbf{K}_v$ is some symmetric positive definite gain matrix, Eq. 4.21 becomes

$$\dot{V}(\mathbf{q}) = -\dot{\mathbf{q}}^T\mathbf{K}_v\dot{\mathbf{q}} \leq 0. \tag{4.23}$$

And so this system is globally asymptotically stable.

Clearly, though, this is an uninteresting system where the control signal simply cancels out any motion.

4.2.2 Stability moving to a target

More interesting and relevant to the desired tasks is to have the system move to a target location. To this end, define

$$\begin{aligned} \tilde{\mathbf{q}} &= \mathbf{q} - \mathbf{q}_{\text{des}}, & (4.24) \\ \dot{\tilde{\mathbf{q}}} &= \dot{\mathbf{q}} - \dot{\mathbf{q}}_{\text{des}}, & (4.25) \\ \ddot{\tilde{\mathbf{q}}} &= \ddot{\mathbf{q}} - \ddot{\mathbf{q}}_{\text{des}}. & (4.26) \end{aligned}$$

Choosing now for a Lyapunov function

$$V(\mathbf{q}) = \frac{1}{2}\dot{\tilde{\mathbf{q}}}^T \mathbf{M}(\mathbf{q})\dot{\tilde{\mathbf{q}}}, \tag{4.27}$$

the derivative is

$$\begin{aligned} \dot{V}(\mathbf{q}) &= \dot{\tilde{\mathbf{q}}}^T \mathbf{M}(\mathbf{q})\ddot{\tilde{\mathbf{q}}} + \frac{1}{2}\dot{\tilde{\mathbf{q}}}^T \dot{\mathbf{M}}(\mathbf{q})\dot{\tilde{\mathbf{q}}}, & (4.28) \\ \dot{V}(\mathbf{q}) &= \dot{\tilde{\mathbf{q}}}^T (\mathbf{u} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{g}(\mathbf{q}) - \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}_{\text{des}}) + \frac{1}{2}\dot{\tilde{\mathbf{q}}}^T \dot{\mathbf{M}}(\mathbf{q})\dot{\tilde{\mathbf{q}}}, & (4.29) \end{aligned}$$

substituting in for the last term using Eq. 4.18 gives

$$\begin{aligned} \dot{V}(\mathbf{q}) &= \dot{\tilde{\mathbf{q}}}^T (\mathbf{u} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{g}(\mathbf{q}) - \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}_{\text{des}}) + \dot{\tilde{\mathbf{q}}}^T \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\tilde{\mathbf{q}}}, & (4.30) \\ \dot{V}(\mathbf{q}) &= \dot{\tilde{\mathbf{q}}}^T (\mathbf{u} - \mathbf{g}(\mathbf{q}) - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}_{\text{des}} - \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}_{\text{des}}). & (4.31) \end{aligned}$$

Setting the control signal to

$$\mathbf{u} = -\mathbf{K}_v \dot{\tilde{\mathbf{q}}} + \mathbf{g} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}_{\text{des}} + \mathbf{M}(\mathbf{q})\dot{\mathbf{q}}_{\text{des}}, \tag{4.32}$$

and substituting in to Eq. 4.31 gives

$$\dot{V}(\mathbf{q}) = -\dot{\tilde{\mathbf{q}}}^T \mathbf{K}_v \dot{\tilde{\mathbf{q}}} \leq 0, \tag{4.33}$$

so once again the system is stable, and in this formulation the steady-state velocity error is guaranteed to be 0 since $\dot{\tilde{\mathbf{q}}}$ must be $\mathbf{0}$ inside the invariant set.

4.2.3 Zeroing steady-state position error

This doesn't guarantee that the system steady-state position error is zero, however, so the problem must be reformulated again. By reworking the Lyapunov function to include an extra term

$$V(\mathbf{q}) = \frac{1}{2}\dot{\tilde{\mathbf{q}}}^T \mathbf{M}(\mathbf{q})\dot{\tilde{\mathbf{q}}} + \frac{1}{2}\dot{\tilde{\mathbf{q}}}^T \mathbf{K}_p \dot{\tilde{\mathbf{q}}}, \tag{4.34}$$

the derivative becomes

$$\dot{V}(\mathbf{q}) = \dot{\tilde{\mathbf{q}}}^T(\mathbf{u} - \mathbf{g}(\mathbf{q}) - \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}}_{\text{des}} - \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}_{\text{des}} + \mathbf{K}_p\dot{\tilde{\mathbf{q}}}), \tag{4.35}$$

so now a position correction signal must be included in the control signal to make the system stable. Setting the control signal to

$$\mathbf{u} = -\mathbf{K}_p\tilde{\mathbf{q}} - \mathbf{K}_v\dot{\tilde{\mathbf{q}}} + \mathbf{g} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}}_{\text{des}} + \mathbf{M}(\mathbf{q})\dot{\mathbf{q}}_{\text{des}}, \tag{4.36}$$

and substituting into Eq. 4.35 gives once more

$$\dot{V}(\mathbf{q}) = -\dot{\tilde{\mathbf{q}}}^T\mathbf{K}_v\dot{\tilde{\mathbf{q}}} \leq 0, \tag{4.37}$$

so the system is once again globally asymptotically stable. But even though the control signal drives the system towards the target position now, the steady-state position error is not guaranteed to be zero. This is because being at an equilibrium point (where $\dot{V}(\mathbf{q}) = 0$) only guarantees that the velocity error is $\mathbf{0}$, because Eq. 4.37 contains $\dot{\tilde{\mathbf{q}}} = \dot{\mathbf{q}} - \dot{\mathbf{q}}_{\text{des}}$, and does not reflect anything about the position error of the system.

Define a new state representation

$$\begin{aligned} \mathbf{s} &= \dot{\tilde{\mathbf{q}}} - \lambda\tilde{\mathbf{q}}, &(4.38)\\ \dot{\mathbf{s}} &= \ddot{\tilde{\mathbf{q}}} - \lambda\dot{\tilde{\mathbf{q}}}, &(4.39)\end{aligned}$$

where λ is some positive constant. This can be rewritten

$$\begin{aligned} \mathbf{s} &= \dot{\mathbf{q}} - \dot{\mathbf{q}}_r, &(4.40)\\ \dot{\mathbf{s}} &= \ddot{\mathbf{q}} - \ddot{\mathbf{q}}_r, &(4.41)\end{aligned}$$

where $\dot{\mathbf{q}}_r = \dot{\mathbf{q}}_{\text{des}} - \lambda\tilde{\mathbf{q}}$, and $\ddot{\mathbf{q}}_r = \ddot{\mathbf{q}}_{\text{des}} - \lambda\dot{\tilde{\mathbf{q}}}$.

Choosing now the Lyapunov function to be a function of $\mathbf{s}$ gives

$$V(\mathbf{s}) = \frac{1}{2}\mathbf{s}^T\mathbf{M}(\mathbf{q})\mathbf{s}, \tag{4.42}$$

and the derivative is now

$$\begin{aligned} \dot{V}(\mathbf{s}) &= \mathbf{s}^T\mathbf{M}(\mathbf{q})\dot{\mathbf{s}} + \frac{1}{2}\mathbf{s}^T\dot{\mathbf{M}}(\mathbf{q})\mathbf{s}, &(4.43)\\ &= \mathbf{s}^T\mathbf{M}(\mathbf{q})\dot{\mathbf{s}} + \mathbf{s}^T\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\mathbf{s}, &(4.44)\\ &= \mathbf{s}^T(\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} - \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}_r + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}}_r), &(4.45)\\ &= \mathbf{s}^T(\mathbf{u} - \mathbf{g}(\mathbf{q}) - \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}_r - \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}}_r). &(4.46)\end{aligned}$$

Setting the control signal equal to

$$\mathbf{u} = -\mathbf{K}_s\mathbf{s} + \mathbf{g}(\mathbf{q}) + \mathbf{M}(\mathbf{q})\mathbf{q}_r + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\mathbf{q}_r, \tag{4.47}$$

where $\mathbf{K}_s$ is a symmetric positive definite matrix, and substituting into Eq. 4.46 gives

$$\dot{V}(s) = -\mathbf{s}^T\mathbf{K}_s\mathbf{s} \leq \mathbf{0}. \tag{4.48}$$

The system is globally asymptotically stable, and now it is guaranteed to have zero position and velocity error when it enters the steady-state because $\dot{V}$ contains $\mathbf{s} = \dot{\mathbf{q}} - \dot{\mathbf{q}}_r$, which must be zero when the system is at equilibrium.

4.3 Bias adaptation for unknown dynamics

In the design of the control signal in Eq. 4.47 it is assumed that a perfect model of the system dynamics is known, specifically the effect of gravity $\mathbf{g}(\mathbf{q})$, the inertia matrix $\mathbf{M}(\mathbf{q})$, and the effects of Coriolis and centripetal forces $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$. As discussed in Chapter 2 this is often unrealistic, and most times the effects of the $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ term is just ignored due to complexity of modeling. Rather than attempting to analytically derive the dynamics, it can be more convenient to learn them.

This turns out to be possible because of an interesting property of the dynamics first presented in [113, 123]. Here the observation was made that the dynamics of a system can be rewritten linearly in a set of select parameters, which are multiplied by a set of known functions of the system state.

For example, given a basic pendulum system, as shown in Figure 4.1, the dynamics of the system are

$$m\ddot{q} + b\dot{q}|\dot{q}| + mgl\sin(q) = u, \tag{4.49}$$

where m is mass, $b\dot{q}|\dot{q}|$ models drag on the system, $mgl\sin(q) = g(q)$ is the gravitational torque, and u is the motor torque.

The Lyapunov candidate function

$$V(s) = \frac{1}{2}ms^2, \tag{4.50}$$

has derivative

$$\dot{V}(s) = \dot{s}(u - m\ddot{q}_r - b\dot{q}|\dot{q}| - mgl\sin(q)). \tag{4.51}$$

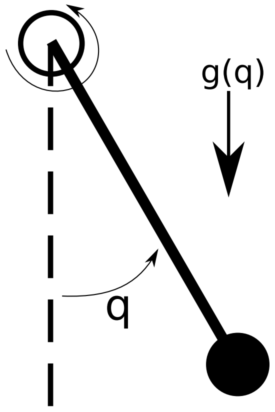


Figure 4.1: A single link pendulum with gravity.

The form of each of the different system dynamics terms here is a certain known function of the system state multiplied by a constant value specific to the system. Rewriting this in vector form

$$\dot{V}(s) = \dot{s}(u - \mathbf{Y}_d(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}_r)\boldsymbol{\theta}_d), \tag{4.52}$$

where $\mathbf{Y}_d(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}_r) = \begin{bmatrix} \ddot{q}_r & \dot{q}|\dot{q}| & \sin(q) \end{bmatrix}$, and $\boldsymbol{\theta}_d = \begin{bmatrix} m & b & mgl \end{bmatrix}^T$, $\mathbf{Y}_d(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}_r)$ is the set of known functions of the system state generic to all systems, and $\boldsymbol{\theta}_d$ is the set of constants specific to the system.

When the actual parameters of the system are known, the control signal of Eq. 4.47 is

$$\mathbf{u} = -K_s s + \mathbf{Y}_d(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}_r)\boldsymbol{\theta}_d, \tag{4.53}$$

and $\dot{V}$ becomes

$$\dot{V}(s) = -K_s s^2 \leq 0. \tag{4.54}$$

If, however, the constant parameters of the system are unknown, or can only be roughly approximated, the control signal becomes

$$\mathbf{u} = -K_s s + \mathbf{Y}_d(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}_r)\hat{\boldsymbol{\theta}}_d, \tag{4.55}$$

where $\hat{\boldsymbol{\theta}}_d$ is an approximation of $\boldsymbol{\theta}_d$. This then makes the derivative of V

$$\dot{V}(s) = -K_s s^2 + s\mathbf{Y}_d(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}_r)\tilde{\boldsymbol{\theta}}_d, \tag{4.56}$$

where $\tilde{\boldsymbol{\theta}}_d = \boldsymbol{\theta}_d - \hat{\boldsymbol{\theta}}_d$. This means that $\dot{V}$ is no longer guaranteed to be negative, and so the system is not globally asymptotically stable. To address this, another term must be added to the Lyapunov candidate function that will allow the derivative to be manipulated such that the unknown error disappears.

4.3.1 Choosing an adaptation law for stability

Let the new Lyapunov candidate function be

$$V(s) = \frac{1}{2}ms^2 + \frac{1}{2}\tilde{\boldsymbol{\theta}}_d^T \mathbf{L}_d^{-1} \tilde{\boldsymbol{\theta}}_d, \tag{4.57}$$

where $\mathbf{L}_d^{-1}$ is a symmetric positive definite matrix. The derivative of the second term is

$$\frac{d}{dt}(\frac{1}{2}\tilde{\boldsymbol{\theta}}_d^T \mathbf{L}_d^{-1}) = \frac{1}{2}(\dot{\boldsymbol{\theta}}_d - \dot{\hat{\boldsymbol{\theta}}}_d)^T \mathbf{L}_d^{-1} \tilde{\boldsymbol{\theta}}_d + \frac{1}{2}\tilde{\boldsymbol{\theta}}_d^T \mathbf{L}_d^{-1} (\dot{\boldsymbol{\theta}}_d - \dot{\hat{\boldsymbol{\theta}}}_d). \tag{4.58}$$

Noting that $\dot{\boldsymbol{\theta}}_d = 0$, because it is a vector of constants, and that each term is equal to its own transpose because they work out to scalar values, this can be rewritten

$$\frac{d}{dt}(\frac{1}{2}\tilde{\boldsymbol{\theta}}_d^T \mathbf{L}_d^{-1}) = -\dot{\hat{\boldsymbol{\theta}}}_d^T \mathbf{L}_d^{-1} \tilde{\boldsymbol{\theta}}_d, \tag{4.59}$$

where $\dot{\hat{\boldsymbol{\theta}}}_d$ specifies how $\hat{\boldsymbol{\theta}}_d$ changes over time. The $\dot{\hat{\boldsymbol{\theta}}}_d$ term is free to be specified as desired, and the goal is to specify it such that it cancels out the error present in $\dot{V}(s)$.

By choosing

$$\dot{\hat{\boldsymbol{\theta}}}_d = \mathbf{L}_d \mathbf{Y}_d^T(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}_r)s, \tag{4.60}$$

the derivative of $V(s)$ becomes

$$\begin{aligned}
\dot{V}(s) &= -K_s s^2 + s\mathbf{Y}_d(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}_r)\tilde{\boldsymbol{\theta}}_d - \dot{\hat{\boldsymbol{\theta}}}_d^T \mathbf{L}_d^{-1} \tilde{\boldsymbol{\theta}}_d, &(4.61)\\
&= -K_s s^2 + s\mathbf{Y}_d(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}_r)\tilde{\boldsymbol{\theta}}_d - (\mathbf{L}_d \mathbf{Y}_d^T(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}_r)s)^T \mathbf{L}_d^{-1} \tilde{\boldsymbol{\theta}}_d, &(4.62)\\
&= -K_s s^2 + s\mathbf{Y}_d(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}_r)\tilde{\boldsymbol{\theta}}_d - s\mathbf{Y}_d(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}_r)\mathbf{L}_d \mathbf{L}_d^{-1} \tilde{\boldsymbol{\theta}}_d, &(4.63)\\
&= -K_s s^2 + s\mathbf{Y}_d(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}_r)\tilde{\boldsymbol{\theta}}_d - s\mathbf{Y}_d(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}_r)\tilde{\boldsymbol{\theta}}_d, &(4.64)\\
&= -K_s s^2 \leq \mathbf{0}, &(4.65)
\end{aligned}$$

and the system is proven to be globally asymptotically stable. This means that the combination of the control law in Eq. 4.53 and the adaptation low specified in Eq. 4.60 guarantees that the system will arrive at the target position and velocity even starting with imperfect knowledge of the system dynamics.

4.4 Transform adaptation for unknown kinematics

As also discussed in Chapter 2, specifying the control signal in terms of movement of the end-effector is desirable. Again, preferable to exact measurement of system parameters is having a system that learns the constant parameters, and is able to adapt to any kinematic changes that may occur in the system (for example picking up a tool that extends the distance of the end-effector from the wrist). As in the case with bias adaptation, the idea is to rewrite the kinematics of the system to be linear in some set of unknown system parameters.

For example, given a basic two-link robot arm system with arm segment lengths l_0 and l_1 for the first and second links, respectively, where $\mathbf{x}$ denotes the position of the end-effector in Cartesian coordinates and $\mathbf{q}$ denotes joint angles, the Jacobian for this system is

$$\mathbf{J}(\mathbf{q}) = \begin{bmatrix} -l_0\sin(q_0) & -l_1\sin(q_0+q_1) \\ l_0\cos(q_0) & l_1\cos(q_0+q_1) \end{bmatrix}. \tag{4.66}$$

Using the basic equation for a Jacobian matrix then gives

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}, & (4.67) \\ &= \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} = \begin{bmatrix} -l_0\sin(q_0) & -l_1\sin(q_0+q_1) \\ l_0\cos(q_0) & l_1\cos(q_0+q_1) \end{bmatrix} \begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \end{bmatrix}, & (4.68) \\ &= \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} = \begin{bmatrix} -l_0\sin(q_0)\dot{q}_0 - l_1\sin(q_0+q_1)\dot{q}_1 \\ l_0\cos(q_0)\dot{q}_0 + l_1\cos(q_0+q_1)\dot{q}_1 \end{bmatrix}. & (4.69) \end{aligned}$$

Rewriting this by separating functions of the system state and system parameters gives

$$\begin{aligned} \dot{\mathbf{x}} &= \begin{bmatrix} -\sin(q_0)\dot{q}_0 & -\sin(q_0+q_1)\dot{q}_1 & 0 & 0 \\ 0 & 0 & \cos(q_0)\dot{q}_0 & \cos(q_0+q_1)\dot{q}_1 \end{bmatrix} \begin{bmatrix} l_0 \\ l_1 \\ l_0 \\ l_1 \end{bmatrix}, & (4.70) \\ \dot{\mathbf{x}} &= \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} = \mathbf{Y}_k(\mathbf{q},\dot{\mathbf{q}})\boldsymbol{\theta}_k, & (4.71) \end{aligned}$$

where $\mathbf{Y}_k(\mathbf{q},\dot{\mathbf{q}})$ is the set of known functions of the system state and $\boldsymbol{\theta}_k$ is the set of kinematic parameters specific to this system.

The Lyapunov candidate function

$$V(\mathbf{s}) = \frac{1}{2}\dot{\mathbf{q}}^T\mathbf{M}(\mathbf{q})\dot{\mathbf{q}} \tag{4.72}$$

has derivative

$$\dot{V}(\mathbf{q}) = \dot{\mathbf{q}}^T(\mathbf{u} - \mathbf{g}(\mathbf{q})), \tag{4.73}$$

by letting the control signal be

$$\mathbf{u} = -\hat{\mathbf{J}}^T(\mathbf{q}, \hat{\boldsymbol{\theta}}_k)\mathbf{K}_p\tilde{\mathbf{x}} - \mathbf{K}_v\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}), \tag{4.74}$$

where $\tilde{\mathbf{x}} = (\mathbf{x} - \mathbf{x}_{\text{des}})$, and the Jacobian approximation $\hat{\mathbf{J}}(\mathbf{q}, \hat{\boldsymbol{\theta}}_k)$ is found by extracting the relevant terms from the approximation of $\dot{\mathbf{x}}$ made with $\mathbf{Y}_k(\mathbf{q}, \dot{\mathbf{q}})\hat{\boldsymbol{\theta}}_k$. Substituting the control signal into Eq. 4.73, the derivative becomes

$$\dot{V}(\mathbf{q}) = -\dot{\mathbf{q}}^T\mathbf{K}_v\dot{\mathbf{q}} - \dot{\mathbf{q}}^T(\hat{\mathbf{J}}^T(\mathbf{q}, \hat{\boldsymbol{\theta}}_k)\mathbf{K}_p\tilde{\mathbf{x}}), \tag{4.75}$$

which is not guaranteed to be less than or equal to $\mathbf{0}$, so the candidate function must be modified.

Ideally, both the end-effector position error and the kinematics approximation error will be in the derivative of the candidate function, so they are guaranteed to go to zero. Choose a new Lyapunov candidate function

$$V(\mathbf{q}) = \frac{1}{2}\dot{\mathbf{q}}^T\mathbf{M}(\mathbf{q})\dot{\mathbf{q}} + \frac{1}{2}\tilde{\boldsymbol{\theta}_k}^T\mathbf{L}_k^{-1}\tilde{\boldsymbol{\theta}_k} + \frac{1}{2}\tilde{\mathbf{x}}^T\mathbf{K}_p\tilde{\mathbf{x}}, \tag{4.76}$$

where $\mathbf{L}_k$ and $\mathbf{K}_p$ are symmetric positive definite matrices, $\tilde{\boldsymbol{\theta}_k} = (\boldsymbol{\theta}_k - \hat{\boldsymbol{\theta}}_k)$, and $\tilde{\mathbf{x}} = (\mathbf{x} - \mathbf{x}_{\text{des}})$.

The derivative of the second term is

$$\frac{d}{dt}(\frac{1}{2}\tilde{\boldsymbol{\theta}_k}^T\mathbf{L}_k^{-1}\tilde{\boldsymbol{\theta}_k}) = \dot{\hat{\boldsymbol{\theta}}}_k^T\mathbf{L}_k^{-1}\tilde{\boldsymbol{\theta}_k}, \tag{4.77}$$

and the derivative of the third term is

$$\begin{aligned} \frac{d}{dt}(\frac{1}{2}\tilde{\mathbf{x}}^T\mathbf{K}_p\tilde{\mathbf{x}}) &= \dot{\mathbf{x}}^T\mathbf{K}_p\tilde{\mathbf{x}}, && (4.78) \\ &= (\mathbf{J}(\mathbf{q})\dot{\mathbf{q}})^T\mathbf{K}_p\tilde{\mathbf{x}}, && (4.79) \\ &= \dot{\mathbf{q}}^T\mathbf{J}_T(\mathbf{q})\mathbf{K}_p\tilde{\mathbf{x}}. && (4.80) \end{aligned}$$

The derivative of the full Lyapunov candidate function is then

$$\dot{V}(\mathbf{q}) = \dot{\mathbf{q}}^T(\mathbf{u} - \mathbf{g}(\mathbf{q})) + \dot{\hat{\boldsymbol{\theta}}}_k^T\mathbf{L}_k^{-1}\tilde{\boldsymbol{\theta}_k} + \dot{\mathbf{q}}^T\mathbf{J}_T(\mathbf{q})\mathbf{K}_p\tilde{\mathbf{x}}. \tag{4.81}$$

Substituting in the control signal specified in Eq. 4.74 into Eq. 4.81 gives

$$\begin{aligned} \dot{V}(\mathbf{q}) &= -\dot{\mathbf{q}}^T\mathbf{K}_v\dot{\mathbf{q}} - \dot{\mathbf{q}}^T\hat{\mathbf{J}}^T(\mathbf{q}, \hat{\boldsymbol{\theta}}_k)\mathbf{K}_p\tilde{\mathbf{x}} + \dot{\hat{\boldsymbol{\theta}}}_k^T\mathbf{L}_k^{-1}\tilde{\boldsymbol{\theta}_k} + \dot{\mathbf{q}}^T\mathbf{J}_T(\mathbf{q})\mathbf{K}_p\tilde{\mathbf{x}}, && (4.82) \\ &= -\dot{\mathbf{q}}^T\mathbf{K}_v\dot{\mathbf{q}} + \dot{\mathbf{q}}^T(\mathbf{J}_T(\mathbf{q}) - \hat{\mathbf{J}}^T(\mathbf{q}, \hat{\boldsymbol{\theta}}_k))\mathbf{K}_p\tilde{\mathbf{x}} + \dot{\hat{\boldsymbol{\theta}}}_k^T\mathbf{L}_k^{-1}\tilde{\boldsymbol{\theta}_k}. && (4.83) \end{aligned}$$

Once again now the rate of change of the kinematics parameters is free to be determined. Choose

$$\dot{\hat{\boldsymbol{\theta}}}_k = \mathbf{L}_k \mathbf{Y}_k^T(\mathbf{q}, \dot{\mathbf{q}})\mathbf{K}_p\tilde{\mathbf{x}}. \tag{4.84}$$

Substituting this learning rule in Eq. 4.83 becomes

$$\begin{aligned}
\dot{V}(\mathbf{q}) &= -\dot{\mathbf{q}}^T\mathbf{K}_v\dot{\mathbf{q}} + \dot{\mathbf{q}}^T(\mathbf{J}_T(\mathbf{q}) - \hat{\mathbf{J}}^T(\mathbf{q}, \hat{\boldsymbol{\theta}}_k))\mathbf{K}_p\tilde{\mathbf{x}} + (\mathbf{L}_k\mathbf{Y}_k^T(\mathbf{q}, \dot{\mathbf{q}})\mathbf{K}_p\tilde{\mathbf{x}})^T\mathbf{L}_k^{-1}\tilde{\boldsymbol{\theta}}_k, && (4.85)\\
&= -\dot{\mathbf{q}}^T\mathbf{K}_v\dot{\mathbf{q}} + \dot{\mathbf{q}}^T(\mathbf{J}_T(\mathbf{q}) - \hat{\mathbf{J}}^T(\mathbf{q}, \hat{\boldsymbol{\theta}}_k))\mathbf{K}_p\tilde{\mathbf{x}} + \tilde{\mathbf{x}}^T\mathbf{K}_p\mathbf{Y}_k(\mathbf{q}, \dot{\mathbf{q}})\mathbf{L}_k\mathbf{L}_k^{-1}\tilde{\boldsymbol{\theta}}_k, && (4.86)\\
&= -\dot{\mathbf{q}}^T\mathbf{K}_v\dot{\mathbf{q}} + \dot{\mathbf{q}}^T(\mathbf{J}_T(\mathbf{q}) - \hat{\mathbf{J}}^T(\mathbf{q}, \hat{\boldsymbol{\theta}}_k))\mathbf{K}_p\tilde{\mathbf{x}} + \tilde{\mathbf{x}}^T\mathbf{K}_p\mathbf{Y}_k(\mathbf{q}, \dot{\mathbf{q}})\tilde{\boldsymbol{\theta}}_k, && (4.87)\\
&= -\dot{\mathbf{q}}^T\mathbf{K}_v\dot{\mathbf{q}} + \dot{\mathbf{q}}^T(\mathbf{J}_T(\mathbf{q}) - \hat{\mathbf{J}}^T(\mathbf{q}, \hat{\boldsymbol{\theta}}_k))\mathbf{K}_p\tilde{\mathbf{x}} + \tilde{\mathbf{x}}^T\mathbf{K}_p(\mathbf{J}(\mathbf{q}) - \hat{\mathbf{J}}(\mathbf{q}, \hat{\boldsymbol{\theta}}_k))\dot{\mathbf{q}}, && (4.88)
\end{aligned}$$

where the step between Eq. 4.87 and Eq. 4.88 comes from applying Eq. 4.71. Again the last term can be rewritten as its own transpose because it reduces to a scalar term, which allows Eq. 4.87 to be rewritten as

$$\begin{aligned}
\dot{V}(\mathbf{q}) &= -\dot{\mathbf{q}}^T\mathbf{K}_v\dot{\mathbf{q}} + \dot{\mathbf{q}}^T(\mathbf{J}_T(\mathbf{q}) - \hat{\mathbf{J}}^T(\mathbf{q}, \hat{\boldsymbol{\theta}}_k))\mathbf{K}_p\tilde{\mathbf{x}} + \dot{\mathbf{q}}^T(\mathbf{J}_T(\mathbf{q}) - \hat{\mathbf{J}}^T(\mathbf{q}, \hat{\boldsymbol{\theta}}_k))\mathbf{K}_p\tilde{\mathbf{x}}, && (4.89)\\
\dot{V}(\mathbf{q}) &= -\dot{\mathbf{q}}^T\mathbf{K}_v\dot{\mathbf{q}} \leq \mathbf{0}, && (4.90)
\end{aligned}$$

where the last two terms of Eq. 4.89 are now equivalent and so cancel out. This shows that the combination of the learning rule specified in Eq. 4.83 and control law Eq. 4.74 guarantees that $\dot{V}(\mathbf{q})$ will be negative semi-definite. Applying the invariant set theorem this system is guaranteed to be globally asymptotically stable.

4.5 Discussion

The combination of the control methods presented in these last three chapters, operational space control, dynamical movement primitives, and the nonlinear adaptive control methods form the basis of the dynamical system for the REACH model, presented in Chapter 7.

The next two chapters will now focus on the biological side of computational models of the motor control system, reviewing past models and methods for neural implementation.

Chapter 5

Computational neuroscience background

In this chapter previous work related to modeling the motor control system or parts thereof are reviewed. The chapter finishes with a discussion of the Neural Optimal Control Hierarchy (NOCH) framework, which was developed as part of my master's thesis. Novel results matching to experimental data generated in the first year of my PhD will be presented at the end of the chapter. The discussion of the NOCH framework sets a context for the REACH model, presented in Chapter 7, which is an update to and novel implementation of a subset of the NOCH framework.

5.1 Previous models

This review begins by discussing models that focus on specific parts of the motor control system, rather than the entire system. Of interest are those models that relate to the work presented in Chapter 7, and so the review focuses on models of the cerebellum and motor cortices, as well as models that address adaptive control or complex trajectory generation.

[125] presents a model based on an attractor network implementation that was able to capture the stereotyped sudden transitions (STs) in the dorsal pre-motor cortex (PMd) predicting forthcoming actions on a single-trial basis. The idea behind the model was that there is a cascade of STs throughout the areas of the brain involved in motor control starting with those areas that plan the movement (PMd) and down to areas executing the movement, i.e. the primary motor cortex. While this model captured some of the

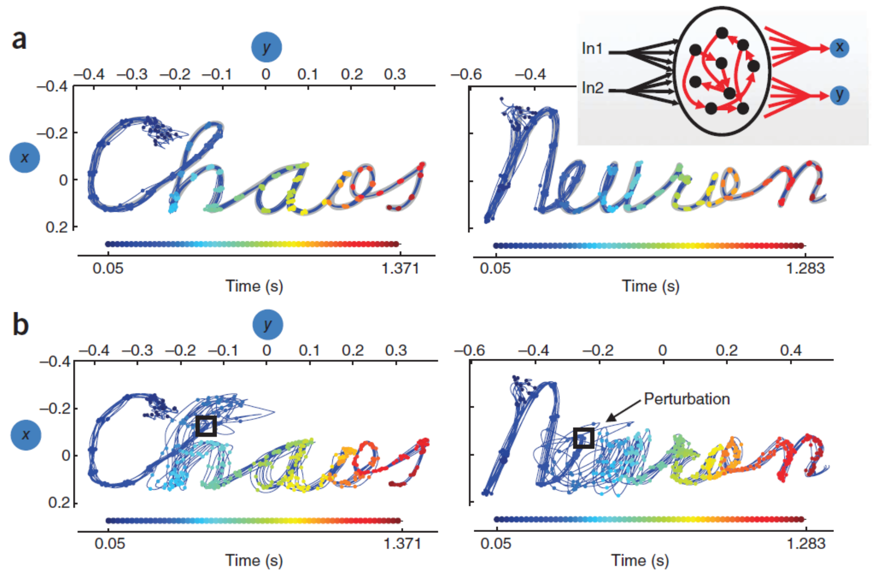


Figure 5.1: An example of some of the trajectories generated using the model presented in [116]. a) Ten test trials after training, where a target circle is placed every 18ms throughout the trajectory. b) The same trials after training, but where a perturbation is introduced after 300ms, showing the system is able to recover after being disturbed.

neurophysiological phenomena of the motor system, it offers little in the way of mechanistic explanation of the function of these areas.

In [116] the authors present a model able to achieve repeatable activation patterns by training networks of rate neurons to converge to a specific path through the state space. The authors suggest that a similar training mechanism and network could serve as a basis for trajectory generation in the brain, by decoding the neural activity at different parts of the path into a sequence of target positions. An example trajectory the system produced is shown in Figure 5.1. While this model is able to perform some of the functionality required by trajectory planning areas, it was implemented in rate neurons as opposed to spiking neurons (the authors noted explicitly that how to implement their network in spiking neurons was unclear), and lacks the generalizability of a trajectory generation system based on dynamic movement primitives.

Recently, in [130] the authors implemented a spiking neuron system that performs the transformation between task-space and low-level control signals, similar to our primary motor cortex model, shown in Figure 5.2. The focus of the work here was the implemen-

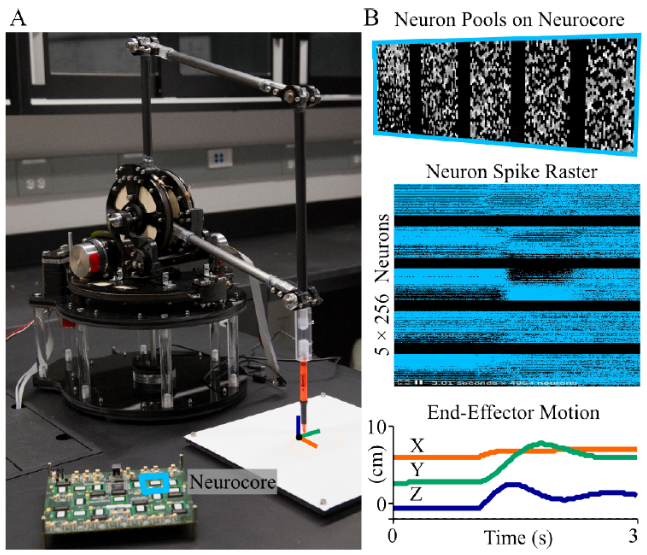


Figure 5.2: Robot control using the Neurogrid neuromorphic hardware and a spiking implementation of operational space control.

tation on neuro-morphic hardware and the controlling of a physical robotic arm, so no comparisons were made to the motor control system of the brain. Additionally, the neural network does not involve any secondary null-space controllers, as discussed in Chapter 2, nor does it feature any of the nonlinear adaptation discussed in Chapter 4 which both significantly extend the functionality of the REACH model.

The model presented in [164] is a partially-neural model that encompasses the motor cortex and cerebellum. The authors employ internal models of the dynamics of the system, learned in an anatomically grounded cerebellar circuit during fast movements to allow the arm to move more precisely during ballistic movements. [164] demonstrates how internal models associated with the cerebellum allow an arm to move precisely during ballistic movements. The model is an extension to the virtual trajectory control hypothesis, which states that rather than controlling muscle activation as an output of the system a set of desired equilibrium muscle lengths and velocities is sent out to the body, which then generates motion as a result of the muscle stiffnesses and differences between desired and actual equilibrium lengths of each muscle. The authors acknowledge several limitations of their model, however, including the lack of a biologically plausible learning rule, lack of ability to model non-linear functions effectively, and the use of a simple two-link planar arm with no redundant configuration space. Additionally, only the adaptive portion of the cerebellum is modeled with neurons, without any discussion on how the rest of the system functionality could be implemented in a neural network.

There are also a large number of models that focus less on behavioral data, and instead attempt to capture specific neural observations. Many of the computational models of the cerebellum fall into this category. They address the cerebellum's role in activities ranging from supervised learning [109, 132] to ballistic movement control [36, 74] to locomotive and balance control [175, 139] suggesting a variety of neural mechanisms that would allow these structures to perform different functions. In the REACH model the scope of the cerebellum model is limited to supervised learning and ballistic movement control.

The primary motor cortex model presented in [189] demonstrated how output from the motor cortex in the terms of activation of muscles can account for the high-level movement parameters that have been previously correlated with neural activity during movement recorded during experiments. This work serves as one of the inspirations for the low-level output from the primary motor cortex in the model presented in this thesis. This model, however, only shows how these correlations could be explained given a system that outputs muscle activation signals, and does not explain how control systems could be implemented in neural networks.

There is also more directly experimental research [30, 32], which has examined the activity of neurons in motor cortex and argued for the existence of an underlying dynamical system responsible for the activity and correlations seen. The methods in this and related work are largely developed and used for the purposes of data analysis rather than hypothesizing mechanisms that give rise to the data and resulting behaviours. These analyses will be described further at the end of both this chapter and the next, where results from the simulated neural activity of the model presented here is compared to the experimental data collected in monkey studies.

There are two main previous large-scale models of the motor control system in the literature that overlap with the work presented here. Both of these models are at a behavioural level, focused on capturing some of the central aspects of human movement and not concerned with a biologically plausible implementation. The first, MOSAIC and its extensions, focuses primarily on modeling the cerebellum. The second, the optimal feedback control framework, focuses primarily on modeling the output of the primary motor cortex.

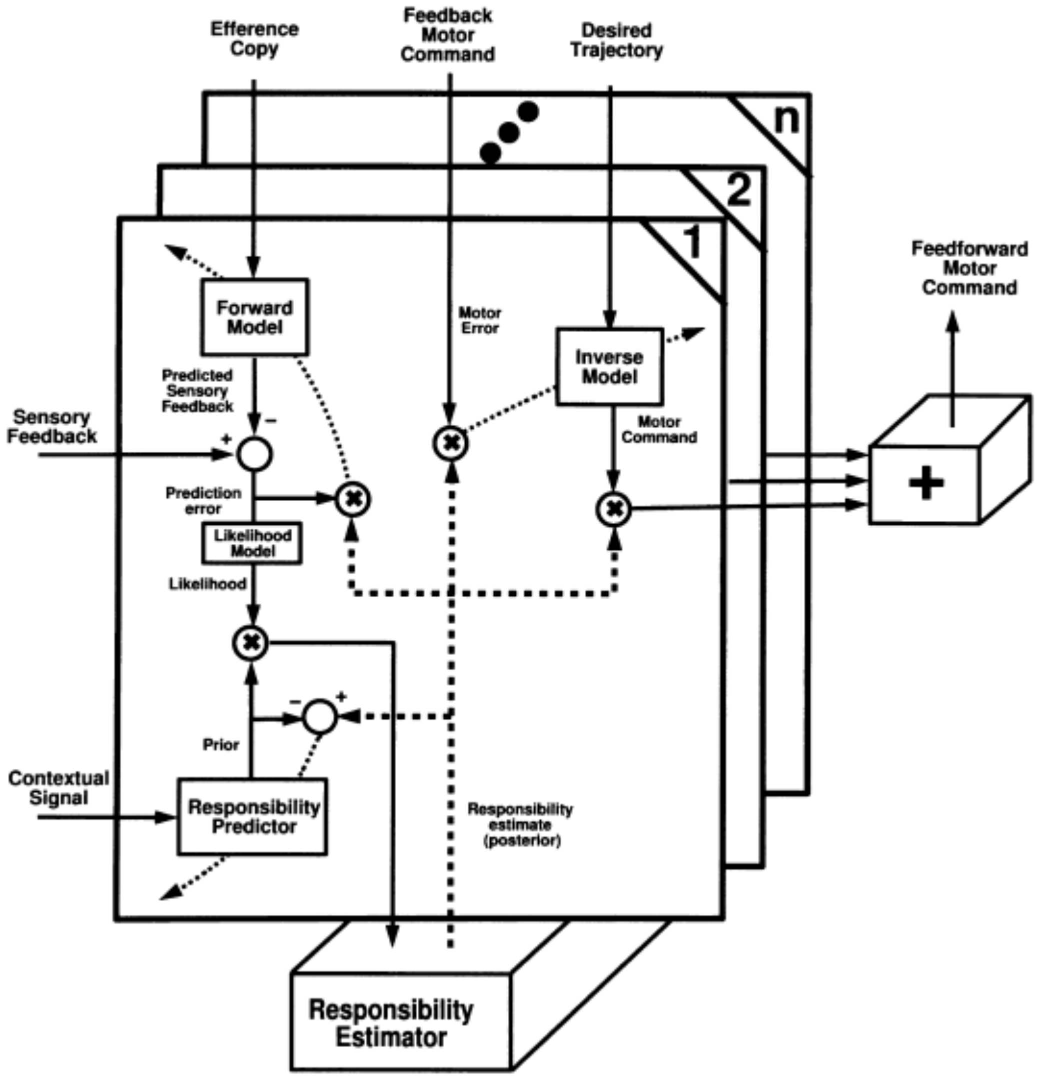


Figure 5.3: An overview of the MOSAIC system. An efference copy of the outgoing motor command is sent to the forward modeling modules, which each focus on learning the dynamics of the system in a different area of state space. Given the system state the Responsibility Estimator assigns a weighting to each of the Inverse Models, which each generate a proposed motor command given the current and target states of the system. Each of the Inverse Models is tied to a Forward Model in the area of state space that they optimize over, and the Responsibility Estimator uses the accuracy of the Forward Model state prediction to determine which Inverse Models should be in control of movement in a specific area of state space.

5.1.1 MOSAIC and extensions

The MOSAIC model [210] is a well-known large-scale computational model of the motor control system, presented first in 1998 by Dr. Daniel Wolpert and Dr. Mitsuo Kawato. The basic idea of a MOSAIC system is to exploit a mixture-of-experts structure to predict the forward dynamics of the system. Each of the experts is trained to a specific area of state space and they are coupled to an inverse dynamics model trained for the same area of state space. These inverse models all receive the abstract control signal to be implemented and output their estimate of the low-level control signal that will accomplish this task. The output of the inverse models are then all weighted by an estimate of how well that controller performs in that part of state space and summed. The 'responsibility' of each controller for an area of state space is updated based on the accuracy of their counterpart forward model prediction. An outline of the general system structure is shown in Figure 5.3.

Since its original presentation, the MOSAIC model has spawned a number of extensions, including hierarchical MOSAIC [82], MOSAIC-MR [179] for multiple reward scenarios, MACOP [205] which looks at having multiple controllers for each part of state space where each outputs control signals for different primitive actions and learns how to weight these different actions such that the overall desired trajectory is carried out, and MODEM [6] which reframed the idea of coupled forward and inverse models in the context of a modularized hierarchical system structure.

Neuroanatomically, the original MOSAIC model proposed that the forward and inverse models match well to the structure and assumed functionality of the cerebellum. Further models have extended this mapping. In MODEM a sensorimotor hierarchy is divided into a low and high level, with a 'gate selector' component that chooses among the high level features to carry out a given task. The high level is proposed to be a basic model of the motor cortex, and the function of the gate selector is proposed as an appropriate role for the basal ganglia. All of these models are strictly behavioural, however, with no extensions involving a neural implementation.

5.1.2 Optimal feedback control theory as a framework for motor control

In [165] a framing of the motor control system in terms of optimal feedback control theory is presented. The main ideas of optimal feedback control theory are the rejoining of trajectory planning and execution, in that the trajectory is updated as the system moves in order to take into account perturbations and obstacles, and the 'minimum intervention principle' which states that movement through the redundant solution space is not of concern, and only movement that affects task-space trajectories is corrected. The basic assignment of functions to anatomy in this proposed structure is that the primary motor cortex generates the optimal control signal, and the cerebellum is involved in online feedback based correction.

This is a very similar framing of the problem as the one presented here developed from operational space control methods. In terms of minimizing the energy of the system, the methods discussed in Chapter 2 are optimal, and the control signal to move to the current target is determined online using system feedback. The minimum intervention principle is analogous to the idea of allowing movement in the null-space of the primary controller, discussed in Chapter 2. But in the case of optimal feedback control the prevention of movement in the operational space is based either on system feedback or model-prediction rather than by calculating a filter to explicitly prevent any unwanted signals from affecting task-space movement. Also, the incorporation of additional controllers is slightly trickier, as these secondary goals are incorporated into the cost function and cannot be guaranteed not to interfere with the completion of primary goals.

The main obstacle in using optimal feedback control theory as a model of the motor control system is that it's based on solving the Bellman equation, which is defined as the cost of moving from the current state to the target optimally. Solving the Bellman equation is a very difficult and computationally expensive task, and it is unclear how this problem could be solved by biologically plausible algorithms and processes analogous to those used in optimal feedback control [165].

Analytic solutions to the Bellman equation for nonlinear systems involves performing singular value decomposition across a matrix representing all possible states weighted by the desirability of being in those states given the current task [191]. Other possibly more biologically plausible approaches to this solution, such as the iLQG method [194], involve an iterative process using forward models, where an initial control policy is proposed and the predicted effects are simulated. The results of this simulation are then used to modify the control signal such that a more optimal outcome is achieved. This process is then repeated until an optimal control policy is converged upon given the current model of the system, and this process is repeated at the next time step, using the previously optimal control signal as a seed for the next round of iterations. This process is extremely computationally intensive, requiring many large matrix inverses, which it is not clear how to perform efficiently in a neural implementation.

The use of DMPs for creating a non-autonomous system, i.e. one that is able to track a desired trajectory, represents a subtle difference between optimal feedback control theory and the model presented in this thesis. In optimal feedback control, trajectory tracking can be accomplished by identifying a cost function that penalizes the system for not being in the desired state at different points in time. Because the full trajectory and control signal is recalculated at every time step in optimal feedback control theory this allows for robust handling of perturbations. However, with the incorporation of a system feedback signal into the DMP framework the same situations may be similarly handled. Thus one of the main differences between iterative optimal feedback control methods and the methods used herein is simply the computational cost incurred by recalculating the control signal at every time step.

The use of forward models for updating the control signal, also known as model-predictive control, to predict and compensate for future disturbances or plan trajectories still represents an obvious advantage for feedback control systems, however. Such an ability has been used to solve a number of interesting problems, such as cooperation between agents [138], contact dynamics optimization [137], and control of humanoids in basic stepping and standing actions [51]. The human brain similarly possesses the ability to use internal models of the environment to develop and optimize trajectories prior to execution, but this functionality lies outside the scope of the model presented in this thesis. The possibility of extending the REACH model to incorporate such functionality is discussed at the end of Chapter 7.

5.2 The Neural Optimal Control Hierarchy (NOCH) framework

In this section the Neural Optimal Control Hierarchy (NOCH) framework, which was the product of my master's thesis, is reviewed along with a novel result comparing premovement and movement activity to experimental data. The NOCH framework was an initial attempt at bridging the gap between control theoretic methods and experimental neuroscience. While the specific control theoretic methods used as a basis for the neural circuits in the NOCH are entirely different to the model presented in this thesis, the overall architecture and functional assignment is largely the same. For this reason the focus of the review will be on the neuroanatomical mapping of function and support for these views rather than the mathematics of any specific implementation.

5.2.1 NOCH: A brief summary

A block diagram of the NOCH framework is displayed in Figure 5.4. The numbering on this figure is used to aid description, and does not indicate sequential information flow. For additional details see [40].

1 - Premotor cortex (PM) and the Supplementary Motor Area (SMA)

The premotor cortex (PM) and the supplementary motor area (SMA) integrate sensory information and specify target(s) in a low-dimensional task-space, which is an abstraction of the system state convenient for efficient planning of trajectories [112].

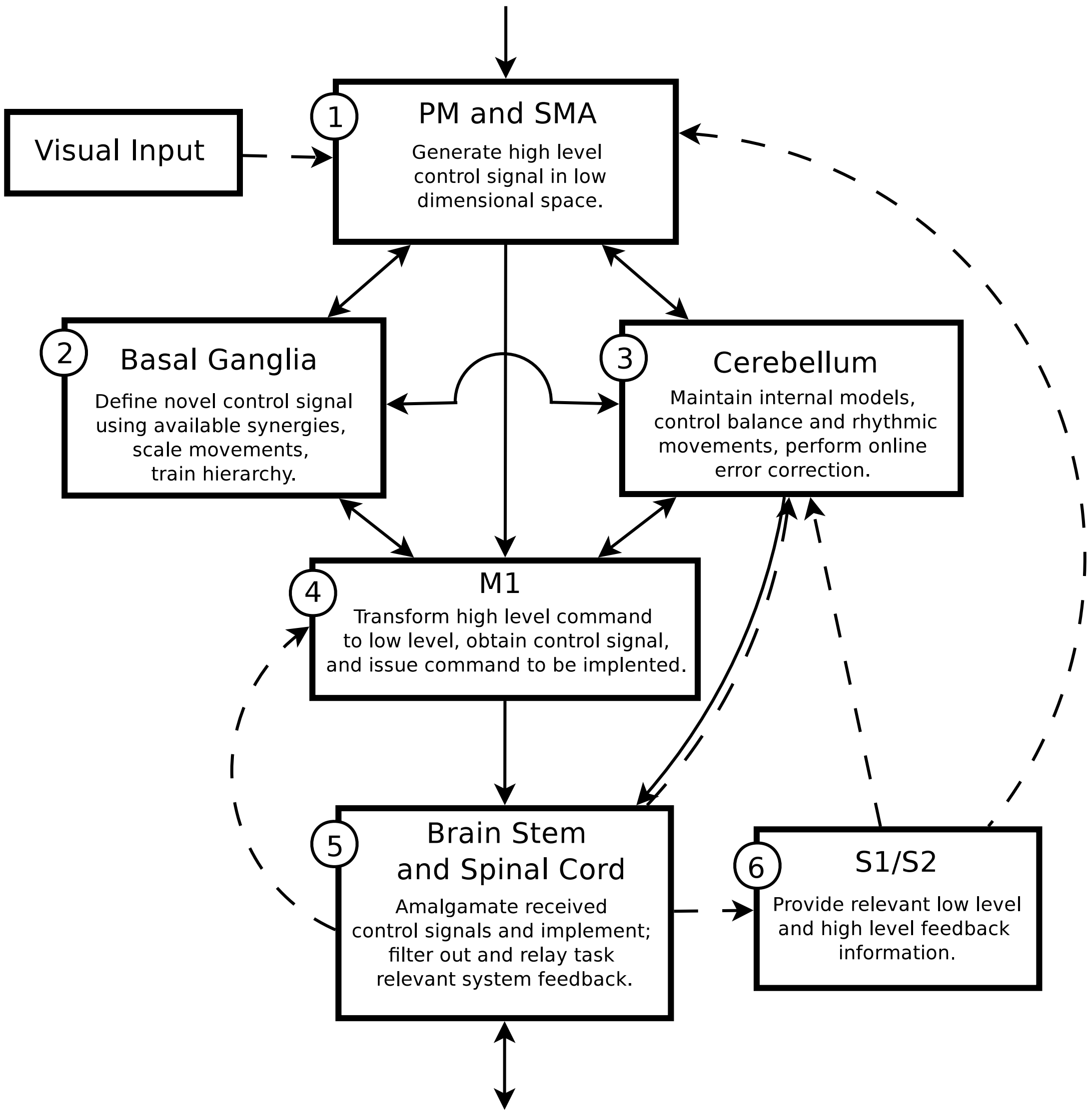
An example of PM/SMA function in arm reaching begins with the planning of a path from current hand position to a target, which incorporates information from the environment, such as obstacle position, as described in Chapter 3. These areas act as the highest levels in a motor control hierarchy that proceeds through M1 and eventually to muscle activations.

2 - Basal Ganglia

The basal ganglia has been characterized as a winner-take-all (WTA) circuit [76], as responsible for scaling movements or providing an 'energy vigor' term [201], and as performing dimension reduction [10]. Spiking neuron implementations have been used to construct a WTA circuit model that has strong matches to neural timing data [178], and can incorporate both movement scaling and dimension reduction [198]. In the NOCH, the basal ganglia is taken to weight movement synergies for the generation of novel actions, perform dimension reduction to extract the salient features of a space for training cortical hierarchies and developing new synergies, and scale movement during directed action.

3 - Cerebellum

The cerebellum is widely regarded as an adaptation device, performing online error correction, and is thus often taken as the site for the storage of internal models [44, 11, 211, 107]. In the NOCH the cerebellum plays a similar role. While in the cortex the movement synergies that are stored are for well-learned environments and situations operating under normal system dynamics, the cerebellum stores synergies that allow the system to adapt to new environments and dynamics, and learn or recall situations to adapt current movements appropriately based on sensory feedback. Additionally, the cerebellum plays a central role in correcting for noise and other perturbations, and correcting movement errors to bring the system to target states as specified by higher-level controllers, as described in Chapter 4. The cerebellum is also responsible for control of automatic balance and rhythmic movements, such as locomotion.



Visual Input
1
PM and SMA
Generate high level control signal in low dimensional space.
2
Basal Ganglia
Define novel control signal using available synergies, scale movements, train hierarchy.
3
Cerebellum
Maintain internal models, control balance and rhythmic movements, perform online error correction.
4
M1
Transform high level command to low level, obtain control signal, and issue command to be implented.
5
Brain Stem and Spinal Cord
Amalgamate received control signals and implement; filter out and relay task relevant system feedback.
6
S1/S2
Provide relevant low level and high level feedback information.

Figure 5.4: The NOCH framework. This diagram embodies a high-level description of the central hypotheses that comprise the Neural Optimal Control Hierarchy (NOCH). The numbering on this figure is used to aid description, and does not indicate sequential information flow. See text for details.

4 - Primary Motor Cortex

In the NOCH, the primary motor cortex (M1) is understood as containing the lower-levels of a hierarchical control system. The primary motor cortex thus acts as a hierarchy that accepts high-level commands, such as end-effector force in 3D space, from the PM and SMA, and translates them through hierarchical levels to muscle activation signals, as described in Chapter 2. The main functional role of this hierarchy is to map high-level control signals to low-level muscle activations in an efficient manner, allowing the PM and SMA to develop control signals in a reduced, lower-dimensional space. The synergies at each level are assumed to change over time, as skills are developed or lost.

5 - Brain Stem & Spinal Cord

For our purposes, the brain stem acts as a gateway for efferent motor command and afferent sensory input pathways. Here, descending commands from different neural systems in the NOCH can be combined and passed on to the spinal neural circuits and motor neurons for execution.

6 - Sensory Cortices (S1/S2)

The primary sensory cortex (S1) is used for sensory feedback amalgamation and processing in the NOCH, serving to produce multi-modal feedback which is then relayed to the motor areas such as M1 and the cerebellum. Both of these systems are taken to work in lower-level, higher-dimensional spaces, and are thus in a position to incorporate appropriate feedback signals into the working motor plan, as demonstrated in Chapter 3. The secondary sensory cortex (S2) is responsible for transforming the information from the primary sensory cortex into high-level sensory feedback information which is then relayed to the high-levels of the M1 hierarchy, as well as PM and SMA. Additionally, S1 and S2 perform a noise filtering on sensory feedback, combining different types of feedback to arrive at the most reliable prediction of body and environment state, analogous to the function of a Kalman filter.

The remaining subsections provide further discussion and justification for the characterizations presented in the preceding brief outline.

5.2.2 The motor cortices: M1, PM, and SMA

Numerous studies have linked the neural activity of the motor cortices to a vast array of different movement parameters, ranging from arm position to visual target location to joint configuration [190]. In the NOCH framework, the descending output signals generated through the cortical hierarchy are in terms of muscle activation, operating either directly on motor neurons, or by driving or modulating the inter-neurons and neural circuits of the spinal cord. Muscle activation as cortical output gives rise to all of the correlations observed in the above mentioned studies, and can be justified from an evolutionary vantage as well [190].

The assumed hierarchical structure of the motor cortices allows for abstraction away from this intrinsically high-dimensional space to a lower-dimensional representation such that the ability to perform effective, efficient control is not lost, and a mapping down to muscle activations is still available. The hierarchical structure of the motor cortices is divided into two main elements in the NOCH, the premotor cortex (PM) and supplementary motor areas (SMA), which generate an end-effector agnostic, high-level control signal (such as the s, and the primary motor cortex (M1), which receives the output from the PM/SMA, and transforms the signal into a limb-specific set of muscle activation commands to carry out the specified high-level control signal.

The functional division of the motor cortices in this manner allows for the generation of a high-level control signal that can be executed by any available or desired end-effector without reformulating the high-level trajectory. This type of high-level control matches experimental observations noting that high level path planning such as handwriting style, are preserved regardless of the end-effector chosen for implementation [207].

5.2.3 The cerebellum

In the NOCH, the cerebellum is modeled as three anatomical areas, each responsible for a function: the intermediate spino-cerebellum for error correction, the lateral cerebro-cerebellum for maintenance of internal models, and the vermis for control of automatic and rhythmic action. This functional assignment arises from examining neuroscientific pathway tracing studies of the input and output cerebellar connections, the neuroanatomical structure of the cerebellum, and the required control theoretic functionality. The cerebellum constitutes only 10% of the total volume of the brain, but contains more than half of its neurons [68]. Additionally, the neural architecture of the cerebellum is remarkably uniform, with a very regular structuring repeated throughout the area [71]. These features suggest the cerebellum is ideal for massively parallel computations, and hence the NOCH employs it to meet the demands of robustness, generalizability, and feedforward predictive modeling.

When the execution of a motor command results in unexpected movement, or otherwise does not effect the desired outcome, the cerebellum is responsible for providing corrective signals to the system, using methods described in Chapter 4. These errors break down into two main categories, caused by either unexpected external forces, or by inaccurate models of the system dynamics. In both cases, a short term solution can be enacted through error integration. Indeed, biological modeling simulations support this system as a site for a neural implementation of error correction [71, 115]. The intermediate zones of the cerebellum are part of the spino-cerebellum, named for the area's strong connectivity with the spinal cord sensory feedback pathways. Neural activity is relayed from the intermediate cerebellum through the interposed nuclei out to the motor cortices, specifically to areas which influence the neural activity of the lateral spinal tracts, which house motor neurons responsible for distal muscle control, suggesting it as a likely candidate for motor error correction based on system feedback.

This functionality is supplemented by the second main responsibility of the cerebellum, the storage and maintenance of internal models. These include predictive models that can be used for feedforward control, which are important for explicit high-level control, and can be implemented using the methods described in Chapter 2. In addition, such models can be employed to adapt motor actions under a given set of environmental conditions, as described in Chapter 4. To perform these functions, the cerebellum requires an efferent copy of the motor plans that are sent to descending pathway for execution. In combination with the massive sensory feedback projections the cerebellum receives, corrective signals can be generated, and tested on the working system. These can then be used to correct cortical motor synergies or develop internal models for context-dependent adaptation, in the case where system error is consistent and predictable. These functional requirements match the input/output structure of the cerebro-cerebellum, located in the lateral cerebellum, which has massive reciprocal connections with the cerebral cortex, making it an ideal site for the integration of internal model signals into generated motor commands, as well as the training of motor synergies for persistent errors.

The third major functional responsibility assigned to the cerebellum is the control of balance and rhythmic movements. Damage to the cerebellum has been associated with impaired locomotor performance and balance, and it is proposed to play a role in the generation of rhythmic movements, dynamic regulation of equilibrium, and adaptation of posture and locomotion through practice [140]. The assigned biological correlate of this functionality is the vermis in the cerebellum, which maintains strong spinal feedback connections and projects through the fastigial nuclei to systems influencing the activity of the medial tracts of the spinal cord. Additionally, experimental studies have confirmed vermis activity to be strongly associated with automatic and rhythmic movements, such as balance and locomotion.

5.2.4 The relation between cerebellar and cortical control

The motor cortices became renowned for being deeply involved in the control of motor actions from early stimulation experiments presented in [148] in 1950. While it is indisputable that the motor cortices play an integral part in motor control, it is also important to consider that they appeared relatively late in the evolutionary development of the brain. There are many species that have little to no cortex, but are very skilled at carrying out motor actions arising from seemingly simple interactions of spinal neural circuits [103, 98, 87].

For example, animals such as the salamander or lamprey are capable of performing very efficient and effective locomotive movements, which can be characterized solely through the activity of spinal motor nuclei and circuits [34]. As the ability of animals to produce more complex, dexterous movements increases across species, the respective size of their cortices, and cortical areas dedicated to motor control, as well as the number of corticospinal projections, increases [122, 20]. These observations provide some clues about the respective roles of the long-standing cerebellum versus the relatively recent motor cortices. The characterization of cerebellar and cortical motor function in the NOCH framework is based on the above observations, the connectivity of each to spinal neurons and circuitry, biological modeling results, and localized damage/lesion studies.

For instance, pathways from the cerebellum relay to the spinal cord through the fastigial nucleus and project to medial spinal tracts, where motor neurons and interneuron circuits with broad projections to proximal muscle groups are housed [19]. The resulting co-activation of motor neurons offers a type of intrinsic muscle synergy structure to the cerebellum, coordinating the activation of sets of proximal muscles [69], and acts to provide a base set of features in muscle space that are useful for simplifying the creation of complex actions. Dense connectivity with the brain stem [55], strong sensory feedback from ascending spinal pathways [19], a recognized error correction functionality [68], and results from damage studies [140] suggest the cerebellum is a centre for control of larger proximal actions, specifically balance and rhythmic movements such as locomotion. These features integrate well with the characterization of cortical-cerebellar interactions as being one of more controlled and volitional action from cortex being corrected and supported by cerebellum.

The motor cortex holds strong connections to spinal neurons controlling proximal muscles, but is also responsible for the vast majority of projections to the lateral spinal tracts [154], where motor neurons controlling distal muscles are housed. Notably, the control effected by these neurons is far more localized than that of the medial motor neurons controlling proximal muscle activation [69]. The information processing ability of cortical hierarchies discussed in the last section, along with the projections to more individually actuated distal muscles, makes the motor cortices a natural compliment to the cerebellar processes, allowing controllers to operate in abstracted muscle spaces, enabling efficient, dexterous control of high-level system parameters.

In the NOCH framework, the control provided by the motor cortices focuses on operational space control and transforming operational space control signals to low-level signals that can be sent out to the body. This is also reflected in the motor cortices being responsible for learning the kinematic parameters of the system, as discussed in Chapter 4, necessary for performing transformation from operational space to low-level signals. The negotiation of external forces and load or system parameter changes is provided by additional processing centers in the cerebellum. The cerebellum has long been assumed to provide storage of internal models used for feedforward/predictive control [211, 107, 44], ideal for the adaptation of motor commands trained for general control problems. When the system dynamics or environment are altered and performance declines, the control signals from the motor cortices are dynamically adapted through the incorporation of cerebellar output, and the resulting amalgamation is sent to the descending pathways.

5.2.5 The basal ganglia

The basal ganglia have long been implicated in the processes of action selection [63, 1, 18], with clear application to motor control. A recent spiking neuron implementation [178] of a biologically plausible model of the basal ganglia [76] provides a neural implementation of a winner-take-all circuit, able to quickly determine a winner among large numbers of input, while scaling input values and matching a wide variety of neural data. Results from this implementation have shown that with slightly altered dynamics this circuit can perform thresholding and similarity evaluation. In the NOCH, the action selection functionality of such a circuit is employed for composing novel actions from movement synergies. Specifically, the basal ganglia is assumed to compute the similarity between a desired action and the available synergies, and generate a set of normalized weights that determines the composition of the synergies and scales the movement.

Interestingly, these same models can be understood as performing dimensionality reduction [198]. Optimal control is more efficient to implement in low dimensional spaces [112]. However, to move from the high-dimensional muscle space to a lower-dimensional synergy space for efficient control, a hierarchy must be established during development. Sets of muscles commonly activated together in specific patterns can be selected as a likely 'building block', or synergy, for more complex movements; by then often selecting sets of synergies more abstract and complicated synergies can be developed. The availability of these synergies reduces the amount of specification required to perform more complex actions, effectively letting control signals be planned in lower dimensional spaces. Understanding the basal ganglia as performing dimensionality reduction lends it a natural role to play in identifying prominent features of a space. Its broad cortical connectivity, central location, and known role in training cortical connections [201] places it in an ideal position to train muscle synergies on the various cortical hierarchical levels, by identifying salient features of a space, and feeding them back to cortex.

Considerations consistent with this functionality are found in infant studies of 'motor babbling', which is thought to perform a similar function to the verbal babbling; it supports building up a set of movement components that can be combined to create more complex actions [174, 149]. Initial motor actions would be in a very high-dimensional muscle space, though controlling a much lower-dimensional action state space. Dimension reduction in the basal ganglia is highly useful for developing a hierarchy, training a lower-dimensional analog of the higher-dimensional space for efficient control.

This dimension reduction through development is taken to be employed until a hierarchy has been created where movements can be efficiently planned without leaving any critical degrees-of-freedom (DOFs) unspecified on lower levels. The definition of a 'critical DOF' is task-dependent, as some movements may be concerned only with hand position, while others may also constrain joint angles or muscle activation. This variety suggests that control may influence various levels of the hierarchy, depending on the type of action being carried out.

Notably, the proposed functions the basal ganglia are consistent across levels of the motor cortex hierarchy. As well, this functionality is important for development, but not critical to the system's real-time operation. That is, basal ganglia under this characterization is not explicitly involved in action selection much of the time. This view is supported by lesion/damage studies in which subjects have had a bilateral pallidectomy (i.e., there are no output pathways from basal ganglia), but show no marked motor deficiencies, except those relating to the proper scaling of movement and performance of novel movement sequences [201].

In sum, the basal ganglia is proposed to be responsible for composing novel actions from motor synergies, scaling movements, and training synergies for the cortical hierarchy. While speculative, this characterization is supported by both experimental studies and model simulation results that suggest the basal ganglia is largely responsible for these processes.

5.2.6 Novel results from the NOCH

The mathematical implementation of the NOCH framework is detailed in [40] and [41]. Here, several of the novel results created in the first year of my PhD are presented. While the NOCH implementation as presented in [40] was strictly non-neural, the results first presented in [41] involved comparisons with neural data by implementing the primary motor cortex of the model in neurons.

5.2.6.1 Spiking neuron model of M1

The arm model employed here is a standard two-link arm model[1]. To implement the primary motor cortex in neurons the NEF (described in Chapter 6) was employed. Specifically, using the neural modeling software Nengo a population of 400 spiking neurons was generated that represents a two-dimensional control space that sent the control signal to the two-link arm model.

A single simulation experiment consists of generating the control network and reaching to 7 targets placed in a circle around the initial starting point of the hand. The neural spiking activity and arm trajectories were recorded, and the data was then exported to Matlab. The neural spikes were filtered by an 80ms Gaussian filter to match the filtering applied to the spike trains gathered from monkeys performing the same task [31]. The activity of single neurons for the 7 different reaches were then amalgamated, and preferred reaching movements identified by selecting the 4 trials with the highest overall activity. Additionally, a Gaussian filter of 70ms was applied to the arm trajectories to account for the unmodeled movement smoothing implicit in muscle-based movements.

5.2.6.2 Single cell tuning in motor cortex during reach

The results from the spiking cell model of motor cortex are displayed in Figure 5.5. Notably, the salient features of the data we discuss are identified in the original data paper [31]. As a result, while the classification of cells is qualitative, it is not model driven, but rather independently determined by the original data analysis. Consequently, to the extent the model is able to generate cells that have the features noted by this analysis (e.g. being biphasic, or switching preferred directions), it can be considered to be a potential explanation of those features.

[1]The model can be found online at `http://compneuro.uwaterloo.ca/research/motor-control/2-link-arm-models.html`

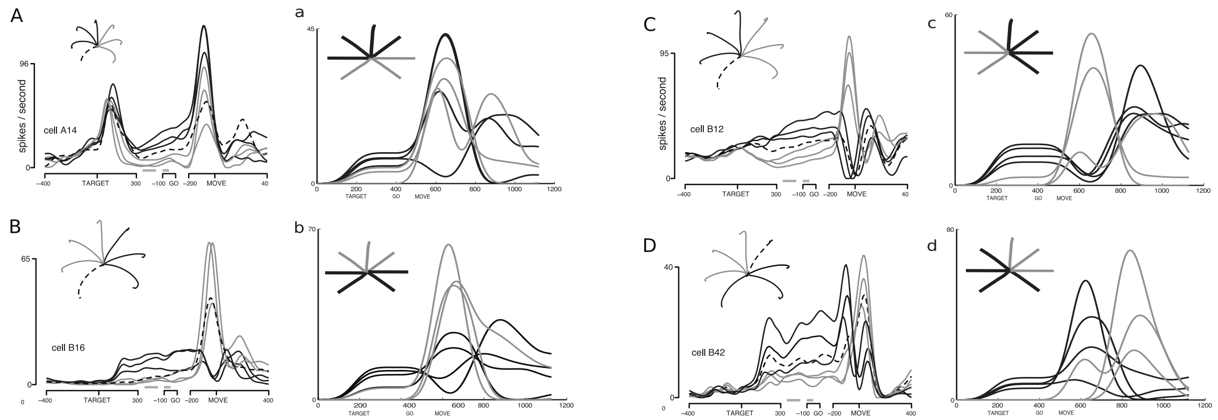


Figure 5.5: A comparison of spiking neuron activity from single cell recordings in monkeys (uppercase letters) and the model (lowercase letters). The activity in monkeys was recorded from dorsal pre-motor and primary motor cortex. It includes both pre-movement activity and activity during the execution of an arm reach. Both sets of data stabilize during the pre-movement period, and then display highly nonlinear activity during movement. The model contained cells that displayed many of the salient properties of recorded cells. Black movements are in the preferred direction of the cell and grey are in the non-preferred direction. Each pair shows the model capturing a salient feature of classes of cells identified by the experimentalists. For instance, A/a show cells that generally respond well for movement initiation regardless of direction. B/b show cells whose preferred direction reverses during movement (i.e., black and grey lines cross). C/c show cells with a reversal and a biphasic response (i.e., black cells are 'on', 'off', 'on'). D/d show cells with a preservation of preferred direction but a strong phase-shifted response related to the preferred direction (i.e. black cells fire first and grey cells second). Data from [31], with permission.

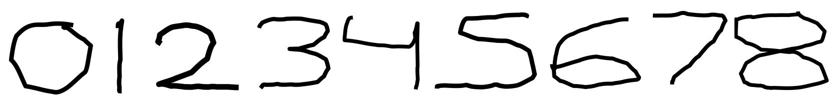


Figure 5.6: An example of the handwriting of SPAUN [50], created by an implementation of the NOCH framework (detailed in [40]).

Overall, the results of this model show that there is a pre-movement convergence to a reasonably constant firing rate, followed by highly nonlinear neural activity during the execution of the movement, as is seen in the recorded neural activity. There are clearly some features of the premovement data not captured by the model (e.g., the pre-delay burst in A).

Nevertheless, the major movement-related features of the data neurons are found in the model neurons. For example, in A/a pre-movement relations and magnitude are preserved during the movement (i.e., black is higher than grey before and during movement), while in B/b and C/c there is a reversal of pre-movement relations during the movement (i.e., black starts higher than grey, but grey becomes higher during movement). In addition, the bi-phasic relationships of the cells in B/b and C/c are clear in both the data and the model. Finally, in D/d the neurons with different preferred directions are clearly phase-shifted with respect to one another. These properties are those identified by the experimentalists in their examination of the complete data set, so it is important that they are captured by the model.

Overall, perhaps the most important feature of the neural responses that is captured by the model is that single cell responses are highly nonlinear during the movement, despite linear arm movements towards the target. It has been suggested that these kinds of nonlinear responses can only be captured in terms of non-plant related parameters [31]. However, this model maps the control signals generated in the plant parameter space directly to neural responses, and finds many of the major nonlinear responses observed in the experimental setting.

5.2.6.3 Number writing

At the same time as the NOCH implementation is able to match neuro-physiological data it is also able to generate meaningful behavioural data. As part of the SPAUN model presented in 2012 [50] the NOCH was used to draw out responses to questions, which are shown in Figure 5.6. To do this a series of target points were presented in the appropriate order for each number, and the control system moved the arm through each of the points.

5.3 Discussion

In this chapter related computational models of the motor control system have been reviewed. While each of these models are able to match some behavioural or neurophysiological phenomena seen in biological systems, none are able to provide a comprehensive account of the possible neural mechanisms involved in a variety of motor control situations, as well as reproduce a wide variety of high-level behaviour.

The REACH model, presented in Chapter 7 is proposed in order to fill this gap in the field of computational models of the motor control system. While evolved as a subset of the NOCH framework, there are a number of noteworthy differences between the NOCH and the REACH, most of which came from the alternate control methods used as a foundation of the dynamical model and issues that arose from implementation the system entirely in spiking neurons. These are discussed in depth at the end of Chapter 7, after the REACH model has been presented. Next, the main method used to build a neural model from the previous, purely control-base characterization, is described.

Chapter 6

Neural implementation

This chapter focuses on describing the process involved in implementing the control algorithms discussed in Chapters 2, 3, and 4 in a biologically plausible network of spiking neurons, modeling the function, architecture, and behaviour of the motor control system in the brain. First a review of the Neural Engineering Framework is given, followed by a description of the reformulations of DMPs and the nonlinear adaptive control methods for implementation in neurons.

6.1 The Neural Engineering Framework

This section is based on a description of the Neural Engineering Framework from [176].

The Neural Engineering Framework (NEF) is a method for implementing algorithms in high-dimensional distributed systems, such as neural networks. It acts as a kind of 'neural compiler', taking a high-level description of the desired dynamics of a system and imposing them upon the neural system. The NEF can be used for any set of nonlinearities or neuron models, allowing biological plausibility and computational requirements to dictate the dynamics of the individual components of the distributed system.

A core premise of the NEF is that the activity of neurons can be decoded into a vector space representation. The high-level description of the desired dynamics must be in terms of vectors and functions on those vectors, i.e. ordinary differential equations (ODEs). The NEF then sets up a neural network such that the neurons approximate these vectors, and the connections between neurons approximate some function over the vectors. The error of these representations can be made arbitrarily small by increasing the number of neurons in the population. This ability to implement ODEs makes the NEF ideal for implementing algorithms from control theory. Biological constraints can be included by specifying the details of neurons, such as their maximum firing rate or the time constants of the neurotransmitters used in the area of the brain being modeled. This allows the plausibility of different algorithms and their implementations to be tested by comparing to data across several levels from behavior down to single cell spiking activity.

The NEF is composed of three main principles; representation, transformation, and dynamics.

6.1.1 Principle 1: Representation

The idea of neurons representing vectors and population decoding in general comes from experimental observations such as those made by Georgopoulos is his canonical papers on decoding the direction of movement of a monkey's arm from the neural activity in the primary motor cortex [66]. In this paradigm, each neuron has a 'preferred direction', which specifies the kinds of input signals it is most responsive to. By taking a weighted summation across all of the neurons of a population it is possible to retrieve the original input vector. In the NEF this idea is captured by 'encoders' and 'decoders'.

Formally, the input current to neuron, $\mathbf{I}$, in response to a given signal, $\mathbf{x}$, is equal to the dot product between that signal and the neurons 'encoding vector', $\mathbf{e}$, which specifies its preferred direction, multiplied by a randomly chosen gain term α plus a background bias current, $\mathbf{I}^{bias}$. The response of a neuron i to an input signal is then defined as

$$\delta_i = G_i \left[\alpha_i \mathbf{e}_i \mathbf{x} + \mathbf{I}_i^{bias} \right], \tag{6.1}$$

where δ_i is the spiking output of the neuron, and G_i is the neuron model. In the simulations performed here, the standard leaky-integrate-and-fire neuron model was used [99].

The vector $\mathbf{e}$ that denotes the preferred direction is referred to as the encoder because it captures the transformation of the input signal $\mathbf{x}$ from a vector space into a neuron space, i.e. input current. Figure 6.1 shows an example of a population of 4 neurons representing a 2D input signal, where each of their preferred directions points in a different direction in 2D space.

Where the input signal moves from vector space to neuron space using the encoder $\mathbf{e}$, a decoder, $\mathbf{d}$, can also be defined to transform the output from neurons back into a vector space. To generate a continuous estimate of the signal being represented by a population of neurons given their output the NEF makes use of the profile of post-synaptic activity

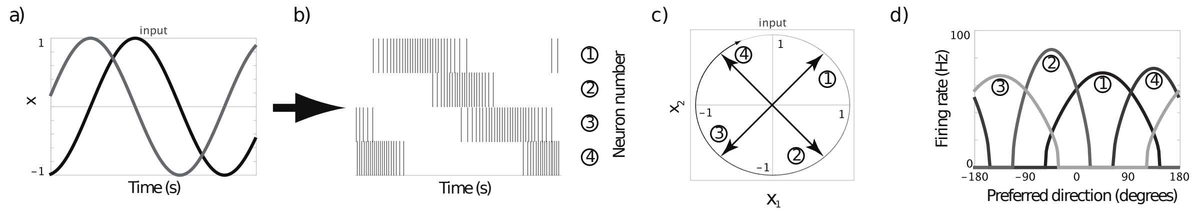


Figure 6.1: An example of using the NEF to encode a 2D signal in a population of four neurons. (a) The input signal, $\mathbf{x} = [\sin(6t), \cos(6t)]^T$. (b) The spikes generated from each of the four neurons with input signal a, generated using Eq. 6.1. (c) The input signal, which is a unit circle, shown in vector space where the signal closer to the current time is darker. The encoders for each of the neurons is also shown in vector space. (d) The firing rate responses of the four neurons as a function of the degree of the input signal along the unit circle. The different shapes of the responses are a result of the randomly chosen gains, α_i, and bias input current, $\mathbf{I}_i^{bias}$, for each neuron i. Figure reproduced from [50].

generated by the reception of a spike across a synapse as a filter. Specifically, this activity is taken to be a linear filter applied to the spiking activity, calculated by convolving the synaptic response function with the spike train:

$$a_i(\mathbf{x}) = \sum_j h_i(t) * \delta_j(t - t_j(\mathbf{x})), \tag{6.2}$$

where a_i is the activity of neuron i, h_i is the spiking response function (usually a decaying exponential weighted by a time constant), $*$ is the convolution operator, and δ_i is the spike train generated in response to input signal $\mathbf{x}$, with spike times indexed by j. This continuous characterization of the neural activity defined as a function of the discontinuous spiking events allows a decoding operation for estimating the input signal $\mathbf{x}$ to be specified across the population of neurons.

This decoding operation will be defined as a linear summation of the neuron activities of the population

$$\hat{\mathbf{x}} = \sum_i^N a_i(\mathbf{x})\mathbf{d}_i, \tag{6.3}$$

where $\hat{\mathbf{x}}$ is the estimate of the original input signal $\mathbf{x}$, N is the number of neurons in the population, and $\mathbf{d}_i$ is the decoders for neuron i.

To find the decoders that give this estimate of the original signal any optimization

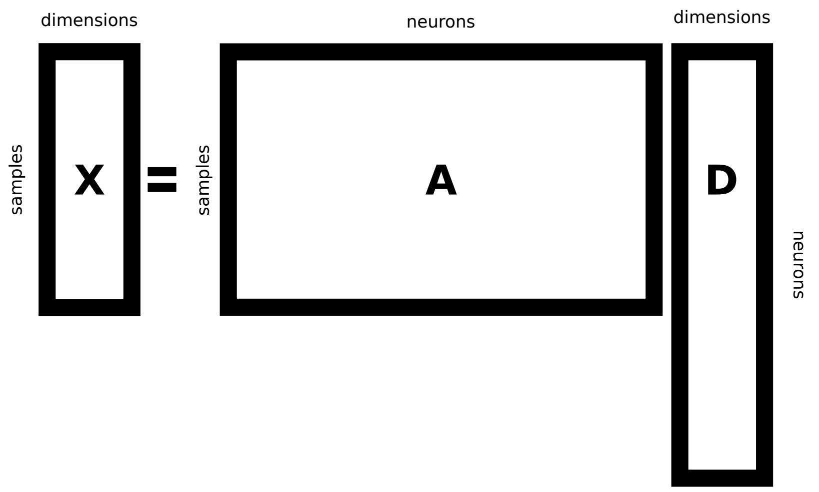


Figure 6.2: **X** is a matrix with samples of the desired output signal in the rows and columns representing the different dimensions of the input signal, **A** is the matrix of neural responses where the rows indicate the different samples in **X** and columns are the different neurons, and **D** is a matrix of decoders where the rows represent the different neurons and the columns are the decoders for each of the different dimensions of the output signal for that neuron.

method can be used, the simplest is least-squares optimization

$$\arg\min_{\mathbf{d}_i} \left[\mathbf{x} - \sum_i^N a_i \mathbf{d}_i \right]^2, \tag{6.4}$$

where the error is minimized and the integral is over all values of $\mathbf{x}$. Rewriting Eq. 6.3 in matrix form

$$\mathbf{X} = \mathbf{A}\mathbf{D}, \tag{6.5}$$

where the form of each of these matrices is shown in Figure 6.2, the decoders can be computed by solving for $\mathbf{D}$:

$$\mathbf{D} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{X}. \tag{6.6}$$

Figure 6.3 shows a network with 20 neurons decoding the neural activity to generate an approximation of the original input signal.

The use of linear decoders allows the NEF to directly generate the connection weights between neurons in the network, which is the main strength of the NEF relative to more

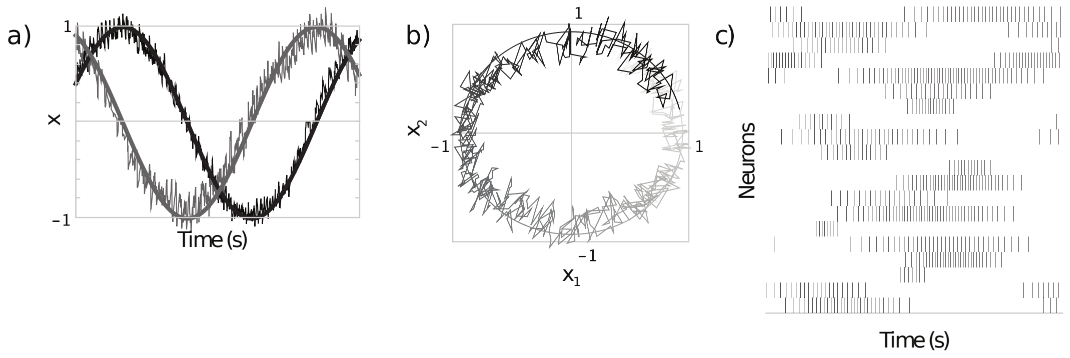


Figure 6.3: An example of decoding neural activity using the NEF for a 2D signal and a population of 20 neurons, simulations performed with $h_i = .005$. (a) The input signal $\mathbf{x} = [\sin(6t), \cos(6t)]^T$ in black and grey, respectively, and its decoded estimate as a function of time. (b) The input signal and its decoded estimate in vector space. The signal fades from black to white as a function of time. (c) The spikes of the neural population over time, used to generate the decoded signal shown in the first two plots. Figure reproduced from [50].

traditional methods of training neural networks. Rather than using a learning rule to search the entire space of all possible connection weights, the NEF solves the simpler problem of optimizing over the significantly reduced space of possible linear decoders, and uses that result to then generate the connection weights. Because of the characterization of neural activity as a function of the preferred directions there is no difference between optimizing over this reduced space and the full connection weight matrix space.

To understand how a weight matrix can be computed from this characterization of encoding and decoding, suppose a connection between two neural populations is used to compute the identity function, i.e.

$$\mathbf{y} = \mathbf{x}, \tag{6.7}$$

where $\mathbf{y}$ denotes the vector represented by the post-synaptic population, B, and $\mathbf{x}$ denotes the vector represented by the pre-synaptic population, A. The connection weights, ω, between the individual neurons of A and B are calculated by combining the decoder of the pre-synaptic neuron and the gain term and encoder of the post-synaptic neuron, giving the equation

$$\omega_{ij} = \alpha_j \mathbf{d}_i \mathbf{e}_j, \tag{6.8}$$

where i indexes the neurons in population A and j indexes the neurons in population B. Figure 6.4 a) shows an example of an implementation of this network.

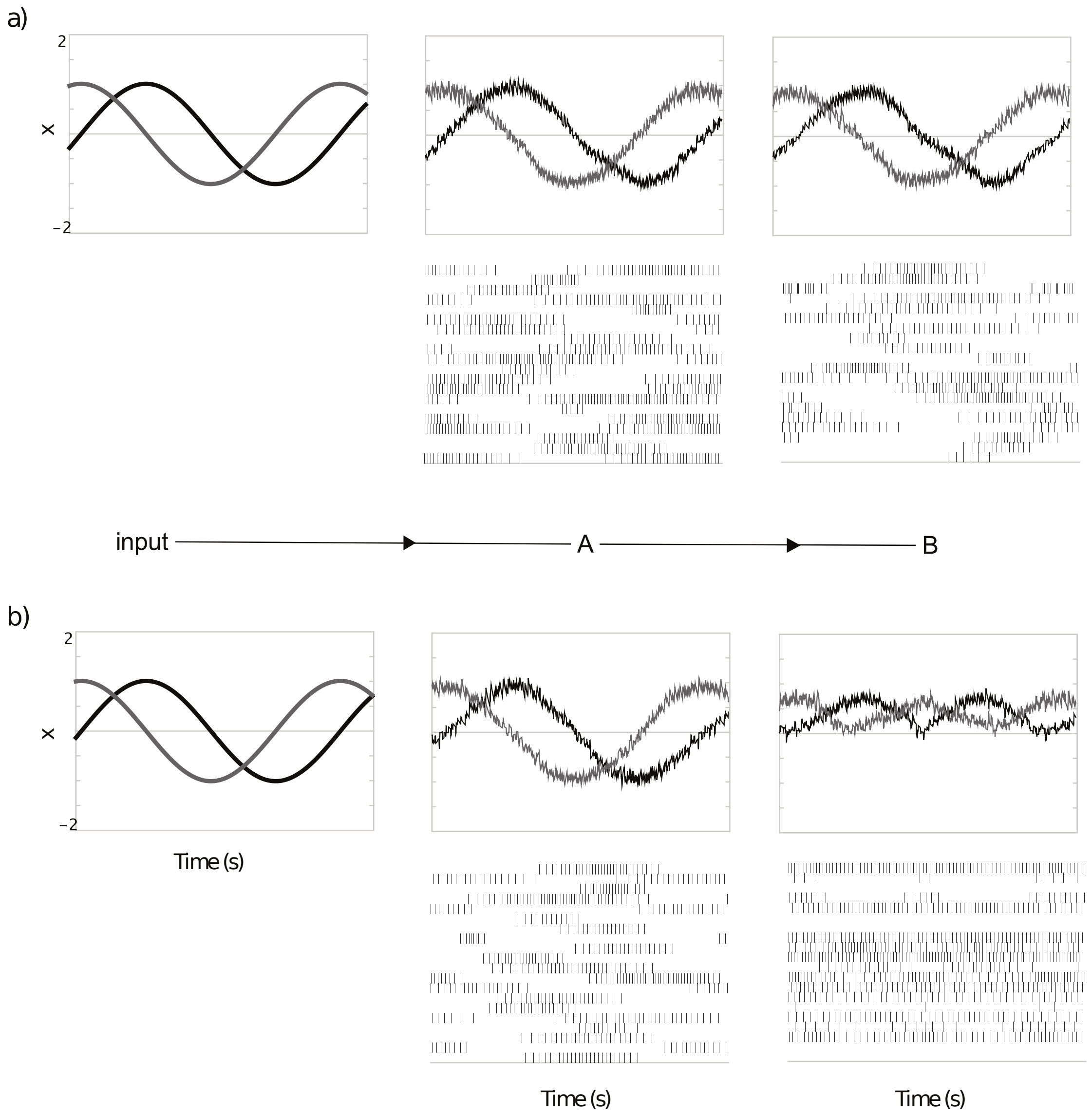


Figure 6.4: Connecting two populations of 20 neurons each using the NEF. (a) The connection weights are specified between A and B such that population B represents the same signal as A. (b) The connection weights between A and B are specified such that B represents the element-wise square of the signal in A. Simulations are 1.2 seconds long, and all neurons have randomly chosen encoder, gain, and bias terms. Figure reproduced from [50].

It should also be noted that while the least-squares method of optimizing the connection weights is not biologically plausible, learning rules that are biologically plausible have been developed for the NEF that can account for the development of these connection weights [121].

The learning rule for these neurons can be phrased both in terms of decoders and in the more common form of connection weight updates. The decoder form of the learning rule is:

$$\dot{\mathbf{d}}_i = L\ a_i\ \mathbf{err}, \tag{6.9}$$

where L is the learning rate, and **err** is the error signal. The equivalent learning rule for adjusting the connection weights is defined:

$$\dot{\omega}_{ij} = L\ \alpha_j\ \mathbf{e}_j \cdot a_i\ \mathbf{err}. \tag{6.10}$$

This is known as the prescribed error sensitivity (PES) learning rule [121]. This is only one example of a learning rule that effectively learns the same circuit. Extensions to the PES rule (such as the hPES rule [13]) or alternatives (such as Oja's rule [143]) may also be used. However, only the learning formulation can be plausibly used for on-line adaptation.

There is a similar level of precision in both the learning and least-squares optimization methods, but the learning is several orders of magnitude slower and more computationally expensive.

6.1.2 Principle 2: Transformation

Just as the decoders can be specified to minimize the error between the original signal and the decoded signal, they can also be specified to minimize the error between some function of the original signal and the decoded signal. By changing the minimization such that $f(\mathbf{x})$ is used instead of $\mathbf{x}$, the problem becomes

$$\underset{\mathbf{d}_i}{\arg\min} \left[f(\mathbf{x}) - \sum_i^N a_i \mathbf{d}_i^f \right]^2, \tag{6.11}$$

where $\mathbf{d}^f$ denotes decoders for the function $f(\mathbf{x})$. With these decoders $\mathbf{y} = f(\mathbf{x})$ is now implemented across the connection weights between population A and B rather than $\mathbf{y} = \mathbf{x}$.

When $f(\mathbf{x})$ is a linear function, i.e. $\mathbf{y} = \mathbf{T}\mathbf{x}$, the linear transform $\mathbf{T}$ can simply be incorporated into the connection weights specified for the identity function

$$\omega_{ij} = \alpha_j \mathbf{d}_i \mathbf{T} \mathbf{e}_j. \tag{6.12}$$

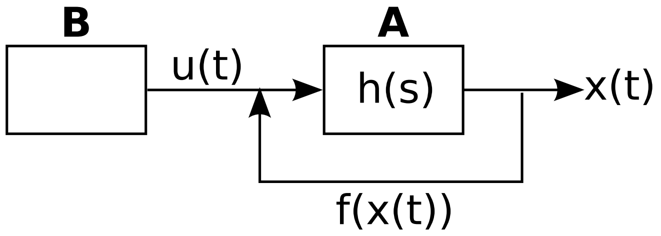


Figure 6.5: A diagram of a generic neural system for implementing ordinary differential equations using a recurrent connection on population A. System input from neural population B, $\mathbf{u}(t)$, system state and output, $\mathbf{x}(t)$, and the equation for the dynamics on the recurrent connection, $f(\mathbf{x}(t))$, with the transfer function, $h(s)$, equal to the Laplace transform of the synaptic dynamics.

These two approaches can be combined such that $\mathbf{y} = \mathbf{T}f(\mathbf{x})$ can be implemented with the connection weights

$$\omega_{ij} = \alpha_j \mathbf{d}_i^f \mathbf{T} \mathbf{e}_j. \tag{6.13}$$

An example of element-wise squaring of the signal between populations A and B is shown in Figure 6.4 b).

6.1.3 Principle 3: Dynamics

The third principle of the NEF is dynamics, and it arises from the ability to compute functions of the form

$$\frac{d\mathbf{x}}{dt} = f(\mathbf{x}, \mathbf{u}), \tag{6.14}$$

where $\mathbf{u}$ is a signal projected in from some other population. This is done by exploiting the same spiking response function, h, used in Eq. 6.1, which represents the current flowing into the post-synaptic neuron when a input spike is received. As mentioned, this post-synaptic current is well-approximated by a decaying exponential

$$h(t) = \beta(t) e^{-t/\tau}, \tag{6.15}$$

where β is the step function and τ is the time constant of the neurotransmitter at this synapse. This spiking response function acts as a low pass filter on the incoming signal, so that a connection between populations of neurons doesn't actually compute $\mathbf{y}(t) = f(\mathbf{x}(t))$ but rather $\mathbf{y}(t) = f(\mathbf{x}(t)) * h(t)$.

Implementing ordinary differential equations can be done using the NEF by setting up a neural population A with a recurrent connection, and a connection from an outside population B to A, as shown in Figure 6.5. The techniques described above can be used to set up the functions on these connections such that the desired dynamics $\dot{\mathbf{x}}(t) = f(\mathbf{x}(t)) + \mathbf{u}(t)$ is implemented. Determining which specific functions need to be computed can be done by analyzing the equation describing the dynamics in Figure 6.5:

$$\mathbf{x}(t) = (f(\mathbf{x}(t)) + \mathbf{u}(t)) * h(t). \tag{6.16}$$

Transforming this equation into Laplace space gives

$$\mathbf{X}(s) = (F(s) + \mathbf{U}(s))H(s). \tag{6.17}$$

Eq. 6.15 gives the equation for $h(t)$, and ignoring the time shift of the step function the Laplace transform of $h(t)$ gives

$$H(s) = \frac{1}{1 + s\tau}. \tag{6.18}$$

Substituting Eq. 6.18 into Eq. 6.17 gives

$$\begin{aligned} \mathbf{X}(s) &= \frac{F(s) + \mathbf{U}(s)}{1 + s\tau}, & (6.19)\\ (1 + s\tau)\mathbf{X}(s) &= F(s) + \mathbf{U}(s), & (6.20)\\ s\mathbf{X}(s) &= \frac{F(s) - \mathbf{X}}{\tau} + \frac{\mathbf{U}(s)}{\tau}, & (6.21) \end{aligned}$$

which, translated back into the time domain, becomes

$$\dot{\mathbf{x}}(t) = \frac{f(\mathbf{x}(t)) - \mathbf{x}(t)}{\tau} + \frac{\mathbf{u}(t)}{\tau}. \tag{6.22}$$

Now to get the dynamics of this system to be equal to the desired dynamics

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t)) + \mathbf{u}(t), \tag{6.23}$$

the recurrent connection needs to be set up to compute $\tau(f(\mathbf{x}(t)) + \mathbf{x}(t))$ and the connection from B to A needs to be set up to compute $\tau\mathbf{u}(t)$. This can be accomplished using the first two principles of the NEF.

These three principles allow for the implementation of a wide variety of systems, and are the basis of the neural models presented in the next chapter implementing the previously discussed control algorithms. Before these algorithms can be fully implemented, however, there are several modifications that need to be made to permit efficient implementation in neurons. These modifications, along with several tips for developing more effective models in the NEF, will be the focus of the remainder of the chapter.

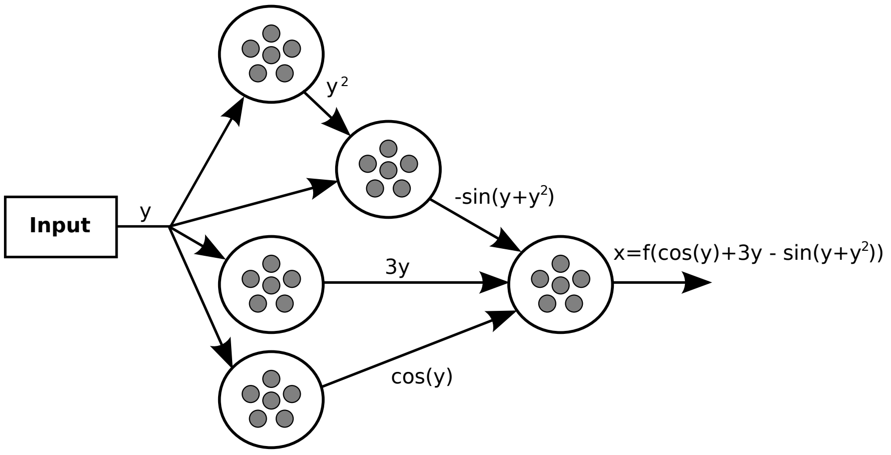


Figure 6.6: Network 1: A first attempt at implementing the equation $x = \cos(y) + 3y - \sin(y + y^2)$ in the NEF, by using separate neural populations to approximate each of the nonlinearities in the equation.

6.2 Modeling complex functions in the NEF

6.2.1 Collapsing functions

There are usually a number of ways to implement any dynamical system in the NEF. The most common approach for someone new to modeling with the NEF is to break all of the calculations up in to separate populations, one for each nonlinearity. For example, given the equation:

$$x = f(\cos(y) + 3y - \sin(y + y^2)), \tag{6.24}$$

where y is a scalar value, and f is some nonlinear function, the natural first attempt would be to break this up into four populations, one for each of $\cos(y)$, $3y$, y^2, $\sin(y + y^2)$, and finally $f(cos(y) + 3y - \sin(y + y^2))$, as shown in Figure 6.6.

Assuming that no special measures are taken during the implementation of these populations, this is a less than ideal implementation for several reasons. First, noise gets added through each connection and function approximation, and it compounds as the equation is split up into many different populations. Each of these populations introduces their own representation error of the variable y. Second, time delays such as the extra time step in passing y^2 to the population to compute $\sin(y + y^2)$ can be introduced and throw off the precision of the answer, as what's actually being computed here is approximately

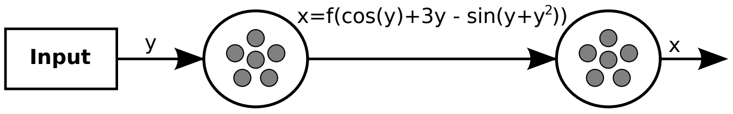


Figure 6.7: Network 2: A more efficient implementation of the equation $x = \cos(y) + 3y + \sin(y + y^2)$ in the NEF, by reducing all of the calculations into a single connection / function approximation.

$\sin(y(t) + y(t-1)^2)$. Third, more neurons are required here to achieve the same precision possible with other implementations because each population needs to have enough neurons to guarantee their output isn't reducing the accuracy of the system output.

Because this whole system is a function of a single variable, a better way to go about this is instead to decode it all from one population, as shown in Figure 6.7. This works because the decoders are calculated to represent a function of the vector represented by the neural activity, and instead of representing all of the intermediate steps along the way it's possible to skip right to the end and only approximate the overall function. A comparison of the two networks running is shown in Figure 6.8, where each population in Network 1 is implemented with 100 neurons, and the single population in Network 2 is implemented with 100, 200, and 400 neurons. As can be seen, the precision in Network 2 for the same number of neurons is significantly improved over the Network 1 implementation.

As mentioned above, these advantages are conditioned on all neural populations being implemented using the default methods for randomly choosing parameters. For example, if the function to compute instead $x = f(h(y) + 3y + \sin(y + y^2)$ where h is a piecewise function that returns 0 when y is negative and y otherwise, it is much more efficient to implement $h(y)$ in its own neural population where the neuron parameters are chosen such that they don't respond to negative input, rather than trying to approximate such a nonlinear function with decoders alone. The results of two neural populations implementing h are shown in Figure 6.9, where the second network has been generated with neurons that don't respond to negative signals. Another example situation where sending the computation off to a specialized population would be when the function being approximated is highly nonlinear for a small range of values of y then it is more efficient to specify the neurons such that they respond most nonlinearly in this range. These specific examples help to illustrate that each dynamical system holds its own nuances which must be considered when implementing in neurons.

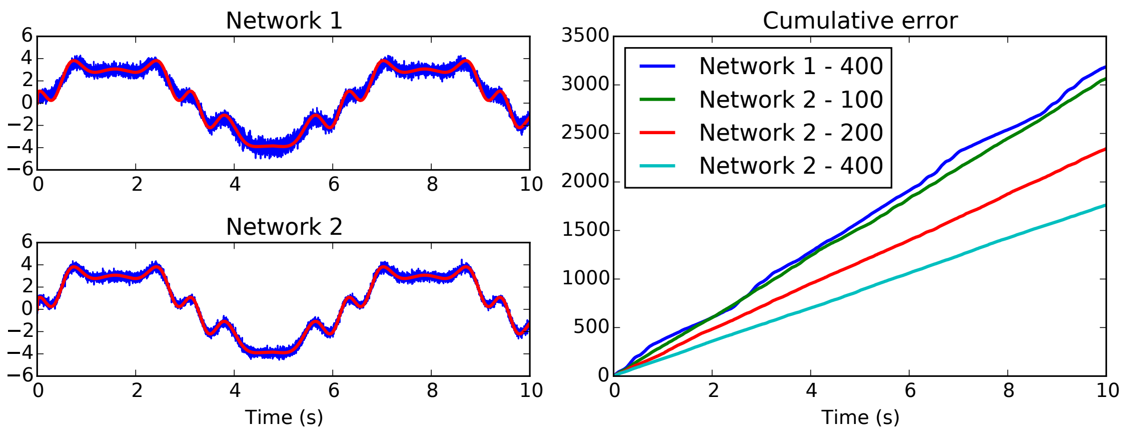


Figure 6.8: A comparison of the network implementations shown in Figure 6.6 and Figure 6.7. On the left shows the results of each network (blue) compared to the answer (red), with each population in each network implemented with a total of 400 neurons. On the right the cumulative error of the signal from each network is plotted, also looking at implementations of Network 2 with different neuron numbers.

6.2.2 Independent variables in separate ensembles

Another common naive approach in neural modeling is representing unrelated variables in the same population. For example, imagine a system to compute the element-wise squaring of the vector $\mathbf{y}$, i.e. $\mathbf{x} = \mathbf{y}^2$. It's very natural to go about implementing this in a single population which represents all of the dimensions of $\mathbf{y}$, as shown in Figure 6.10. But looking at the desired function, element-wise squaring, it can be seen that the calculation for any element is independent of all the other elements. In these situations greater precision can be achieved by separating the independent variables into separate neural populations for representation.

The only time that multiple variables need to be represented in the same population is when they interact nonlinearly. When this is not the case, it is usually preferred to separate the variables into different populations of neurons because representing multiple dimensions introduces a number of subtle issues. Most of these issues boil down to the fact that the number of evaluation points (samples of the signal when generating the decoders) required for a given precision increases exponentially with the number of dimensions represented. Having more dimensions in a population than absolutely necessary results in a less accurate

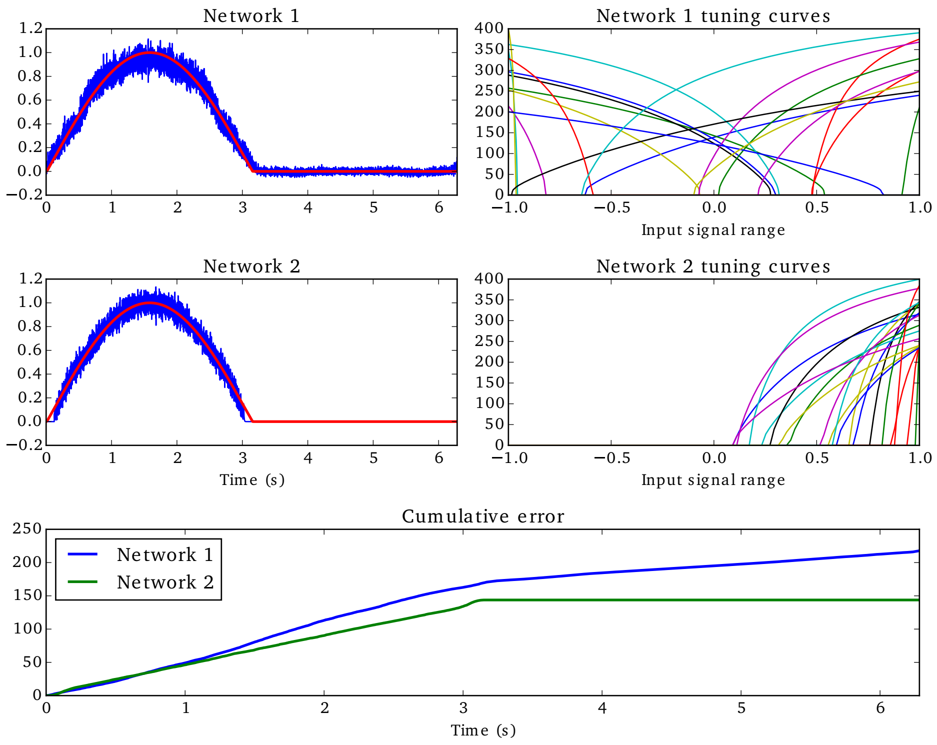


Figure 6.9: An example of how specific properties of a neural population can be exploited for efficient computation of certain functions. In this case, a piecewise function $h(y)$ is being computed in both network, where $h = y$ when $y > 0$ and $h = 0$ otherwise. The first network is implemented with randomly chosen parameters. The second network is specified such that the neurons do not respond to any negative signals, making it better able to accurately compute $h(y)$. The cumulative error of the two networks is shown on the bottom.

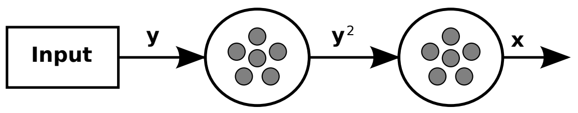


Figure 6.10: Network 1: An intuitive implementation of element-wise squaring of a vector, using a single neural ensemble to represent all of the elements of the vector.

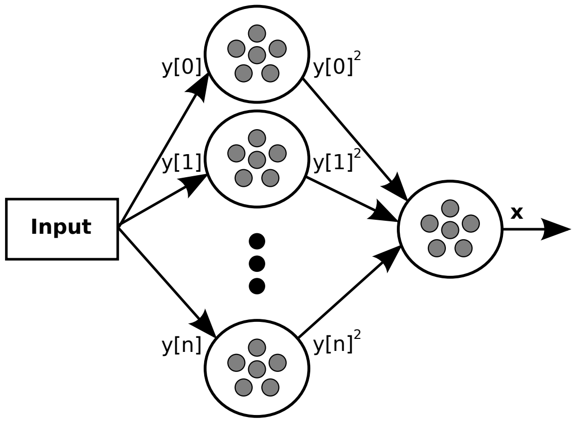


Figure 6.11: Network 2: A more efficient implementation of element-wise squaring of a vector, by representing all of the variables in separate populations.

set of representation and transformation decoders because the state space is harder to sample thoroughly.

A more effective way to implement the element-wise squaring network is shown in Figure 6.11. Figure 6.12 shows a comparison of the two networks. As can be seen, the implementation of Network 2 using a fourth of the number of neurons as the Network 1 implementation generates comparable results, and quickly becomes far more precise as neurons are added, ending up with roughly one fourth of the cumulative error when the same total number of neurons are used in both networks.

Once again, this is a rule-of-thumb when engineering systems with the NEF using the default methods for randomly generating neural parameters, and does not reflect some inherent limitation of the NEF. It is possible to specify a single population of neurons capable of representing multi-dimensional input signals as well as an array of populations. Essentially, the encoders of the single large population can be chosen such that a given neuron is sensitive to only one of the dimensions of the input signal, and the samples for calculating the decoder can be similarly specified such that the same performance is given as in the case with a single population for each dimension. This approach is complicated, however, and requires a detailed specification of the neural parameters, and is avoided when possible.

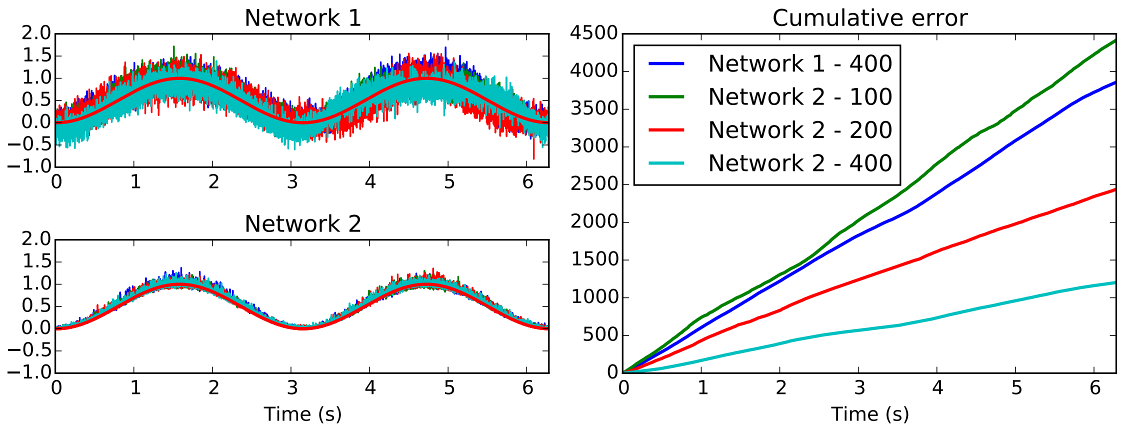


Figure 6.12: A comparison of the implementations in Figure 6.10 and Figure 6.11 for implementing a 4-dimensional element-wise squaring. On the left shows the results of each network compared to the answer (red); Network 1 has 400 neurons and Network 2 has 100 neurons in each population. On the right the cumulative error from each network is plotted, also looking at implementations of Network 2 with different total neuron numbers.

6.3 Representing large values in the NEF

When required to use multiple dimensions in a single population another issue can arise that involves the range of state space to sample from when generating the decoders. Often the ranges of the different variables represented are different, and sometimes significantly larger than the default sampling range of -1 to 1.

A naive approach in these situations is to change the sampling range to include the maximum of any of the represented variables, but this can cause sampling across ranges that are not of relevance for other variables, reducing the effectiveness of the decoders for these variables. In these situations a better solution is to identify the ranges for each of the variables and scale the input to each dimension by their respective maximum values, and then account for this scaling on any connection output from that neural population.

For example, in a system calculating $\mathbf{x} = y[0] \times y[1]$ where the range of $y[0]$ is between -100 and 100 and the range of $y[1]$ is between -1 and 1, it is effective to scale the input to $y[0]$ by .01 so the function on the connection out becomes $\mathbf{x} = 100 \times y[0] \times y[1]$. The results of these two implementations are shown in Figure 6.13, as can be seen, the latter

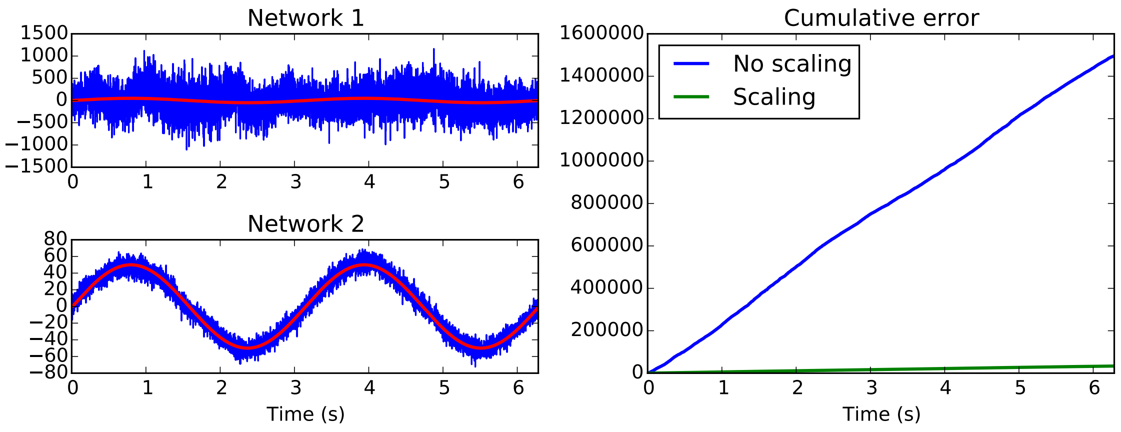


Figure 6.13: A comparison of the implementations with increasing the range of sampling to be equal to the maximum value of the signals represented in the ensemble versus scaling each signal to be between -1 and 1. On the left shows the results of each network (blue) compared to the answer (red); each network was implemented with 200 neurons. On the right the cumulative error from each network is plotted.

implementation is far more precise. This scaling becomes especially important in situations where there is a great disparity between the dynamic range of the signals represented.

Another important consideration when working with large input signals is to consider the mean value. While the minimum and the maximum values may be -100 and 100, the mean value could be around 80, with an average range of 70-90. Without accounting for the mean value before scaling, the full range of firing rates of the neural population won't be taken advantage of. In the case where the mean is 80 and the input signal is scaled by 100, the input signal will hover between .7 and .9. This means that the majority of the time the decoders have much less variety in neural firing rates to use for approximating the desired function of the input. It is important to take into account the importance of representing the full range of the input signal. If it's the case that -100 only rarely appears and does not affect system performance, it is much more effective to first subtract 80 from the input signal, then scale by 10. This will ensure that the full range of the population's firing rates will be used.

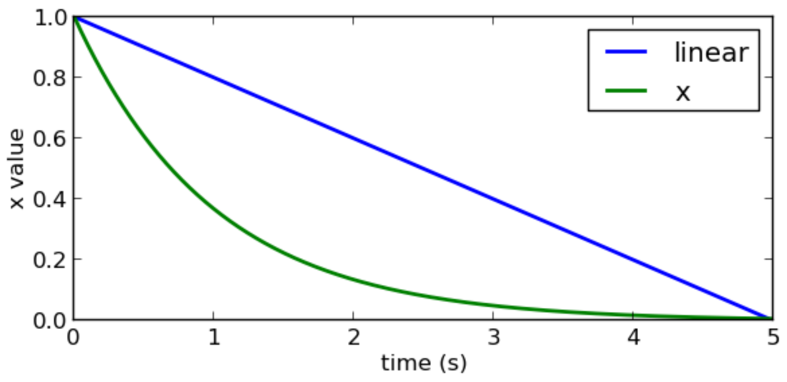


Figure 6.14: The original canonical system dynamics compared with a linearly decaying system that will be used for the neural implementation, here $\alpha = -.2$.

6.4 Simplifying discrete DMPs

The original formulation of DMPs is difficult to implement in neurons for two reasons. First, the dynamics of the canonical system are exponential, resulting in a wide dynamic range being necessary to equally well represent many states of the system. Second, an exponential decay imposes more complex constraints on centering the basis functions. To simplify things, a linear canonical system can be used with dynamics that evolve from 0 to 1:

$$\dot{x} = \begin{cases} \alpha & x < 1 \\ 0 & x \geq 1 \end{cases} \tag{6.25}$$

Figure 6.14 shows the difference between the two canonical systems.

With this change, the placement of the basis function centers is simply linearly spaced in x space.

6.5 Reworking nonlinear adaptive control for neural implementation

Using the NEF, it's possible to implement the nonlinear adaptive control methods discussed in Chapter 4 directly, by having a population of neurons decode specific functions of the system state into a matrix $\mathbf{Y}$, and then learn an appropriate weighting across these functions. But this approach requires specific knowledge of the functions that form the basis of the unknown forces that are being approximated. Instead, it is more desirable to not require this knowledge and use some multitude of simple basis functions that cover a state space sufficiently that any desired function can be approximated.

In [158] the authors use sets of Gaussian functions with multi-dimensional output to tile the state space, and learn a weighting over these functions that approximates the unknown dynamics of a system, such that a mapping from joint-torques to kinematics is identified. Reworking this approach with NEF methods such that neurons can be used as the basis functions requires reformulating the problem such that the set of basis functions output are all scalar rather than multi-dimensional and that the set of decoders are multi-dimensional rather than scalar.

This reformulation provides several benefits. First, using the methods of the NEF it is possible to seed the neural network with an approximation of the answer, when prior knowledge is available, greatly speeding time to convergence during learning. Additionally, when tiling a multi-dimensional space the curse of dimensionality is encountered; the number of basis functions required to adequately cover state space increases exponentially with the number of degrees of freedom of the system. In the NEF implementation the different dimensions are all approximated independently, avoiding this coupling and the curse of dimensionality, which also means that the same set of basis functions can be used with different decoders to approximate all of the different dimensions of the output signal. Note that being able to efficiently tile the state-space does not simultaneously address the problem of learning in higher-dimensional state-spaces.

This reformulation is presented formally below, along with proofs of global asymptotic stability of the system using Lyapunov's direct method.

6.5.1 Bias adaptation for unknown dynamics

First the equations for a neural network to directly implement the nonlinear bias adaptation to account for unknown dynamics using the NEF are shown. Following this a reworking of the equations for scalar basis function output and decoders as a tensor is shown and analyzed.

In the system, shown in Figure 6.15, there is a basic control signal, $\mathbf{u}$, and a set of basis functions that takes the system state, $\mathbf{q}$ as input. As in Section 4.3, there are assumed to be some forces acting on the system that are unaccounted for by the control signal, denoted

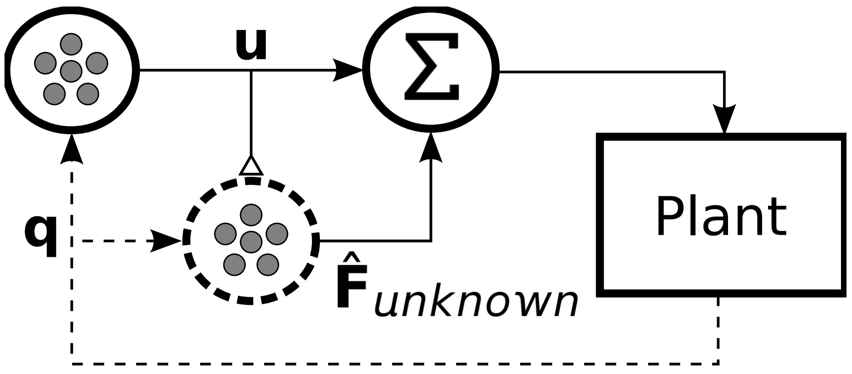


Figure 6.15: A diagram of the basic setup of a control system using a bias adaptation component. The bias adaptation component (inside the dashed circle) takes the system state $\mathbf{q}$ as input and the control signal $\mathbf{u}$ as a learning signal, and outputs an approximation of the Jacobian, $\hat{\mathbf{J}}$. This signal is then summed with $\mathbf{u}$ and sent out to the system under control. The hollow triangle denotes a modulatory connection.

$\mathbf{F}_{unknown}$. The output of the basis functions is weighted by a set of learned parameters $\hat{\boldsymbol{\theta}}_d$ to give an approximation of the known forces, $\hat{\mathbf{F}}_{unknown}$

The goal of the bias adaptation term is to learn a set of parameters $\hat{\boldsymbol{\theta}}_d$ that can accurately approximate these unknown forces such that they can be compensated for in the control signal and removed from the system dynamics. The learning rule for updating the learned parameters, $\dot{\hat{\boldsymbol{\theta}}}_d$, is responsible for minimizing the approximation error of $\hat{\mathbf{F}}_{unknown}$.

6.5.1.1 Control with adaptive bias in the vector space

To directly implement the algorithms for nonlinear adaptive control, as presented in [170], in a neural network, the methods presented in Section 6.1 can be employed. Given desired functions of the input to the neural population for each of the elements of the basis functions matrix, $\mathbf{Y}_d$, a set of decoders can be generated using the methods described in Section 6.1 to approximate each of these functions from the neural activity. Projecting these into a second neural population, the decoders on the output from this second population can then serve as the learned parameters $\hat{\boldsymbol{\theta}}_d$ and be modified based on the learning rules described in Chapter 4. The structure of this network is shown in Figure 6.16.

As mentioned above, this can be a useful approach when the form of the disturbances is known. For example, if perturbations are known to occur at frequencies determined as

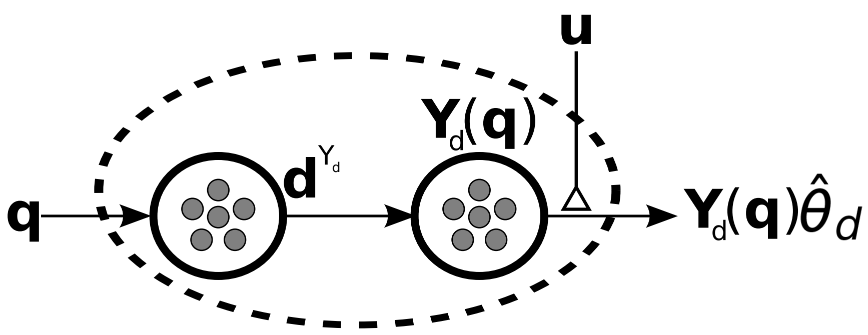

u
$\mathbf{Y}_d(\mathbf{q})$
$\mathbf{d}^{Y_d}$
q
$\mathbf{Y}_d(\mathbf{q})\hat{\theta}_d$

Figure 6.16: The structure of a neural network set up to implement the original formulation of the nonlinear adaptive bias component. The matrix of basis functions $\mathbf{Y}_d$ is decoded from the neural activity of the first population, where each of the elements of $\mathbf{Y}_d$ is approximated through a set of decoders determined with the methods described in Section 6.1. This matrix is passed to a second population of neurons which applies the set of learned system parameters $\hat{\boldsymbol{\theta}}_d$ stored in its decoders. These decoders are updated through a learning signal that is a function of the control signal $\mathbf{u}$.

a function of some parameter of the system input $\mathbf{q}$ then the set basis functions $\mathbf{Y}_d(\mathbf{q})$ can be created to approximate these functions based on the output of the neural population. The system output with this adaptive compensation term is then written:

$$\mathbf{u} + \mathbf{Y}_d(\mathbf{q})\hat{\boldsymbol{\theta}}_d, \tag{6.26}$$

with the learning rule

$$\dot{\hat{\boldsymbol{\theta}}}_d = L_d \mathbf{Y}_d^T(\mathbf{q})\mathbf{u}, \tag{6.27}$$

where L_d is a scalar gain term. Equivalently, this can be rewritten in terms of the decoders of the neural population as:

$$\dot{\hat{\boldsymbol{\theta}}}_d = L_d(\mathbf{a}\,\mathbf{d}^{Y_d})^T\mathbf{u}, \tag{6.28}$$

where $\mathbf{a}$ is the activity of the neurons, and $\mathbf{d}^{Y_d}$ is the set of decoders calculated to generate the $\mathbf{Y}_d$ matrix from this activity. The learning can be implemented in neurons using Eq. 6.9. The same proof presented in Chapter 4 for bias adaptation holds here.

6.5.1.2 Control with adaptive bias in neuron space

As mentioned above, it is possible to implement the algorithms using known functions of the system state in neurons using the NEF, but it requires knowledge of the form of the

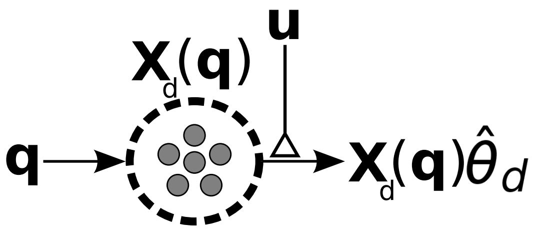


Figure 6.17: The structure of a neural network set up to implement the reworked formulation of the nonlinear adaptive bias component. The basis functions vector $\mathbf{X}_d$ is the neural activity of the population in the dashed line, and the decoders of this population are the learned parameters $\hat{\boldsymbol{\theta}}_d$. These parameters are updated as a function of the control signal $\mathbf{u}$. The hollow triangle denotes a modulatory connection.

unknown dynamics affecting the system. Instead, it would be more desirable to use the neurons themselves as the basis functions and learn a set of decoders over their activity profiles that approximates the unknown dynamics, such that forces of unknown form could be compensated for. Again, the critical difference between the past work of Sanner and Slotine [158] and that presented here is that in their work each of the nonlinear components making up their basis function set output a multi-dimensional signal which was then given a weight to approximate the multi-dimensional force affecting the system, whereas in the system here each basis function component outputs a scalar term, and the set of learned parameters are multi-dimensional.

Going back to the original formulation presented in [170], the unknown forces affecting the system are approximated

$$\hat{\mathbf{F}}_{unknown} = \mathbf{Y}_d(\mathbf{q})\hat{\boldsymbol{\theta}}_d \tag{6.29}$$

where $\mathbf{Y}_d \in \mathbb{R}^{n\times d}$ is the set of basis functions and $\boldsymbol{\theta}_d \in \mathbb{R}^{d\times 1}$ are the learned parameters, where n is the number of basis functions and d is the dimensionality of the control signal, $\mathbf{u}$.

Here this is reformulated such that the set of basis functions is now a vector and the learned parameters a matrix. The basis functions in vector form will be denoted $\mathbf{X}_d \in \mathbb{R}^{1\times n}$, and the corresponding set of decoders is of the form $\hat{\boldsymbol{\theta}}_d \in \mathbb{R}^{n\times d}$. The unknown forces approximation looks the same,

$$\hat{\mathbf{F}}_{unknown} = \mathbf{X}_d(\mathbf{q})\hat{\boldsymbol{\theta}}_d, \tag{6.30}$$

but now the learning rule is done with an outer product rather than an inner product:

$$\dot{\hat{\boldsymbol{\theta}}}_d = \mathbf{L}_d\mathbf{X}_d \otimes \mathbf{s}, \tag{6.31}$$

where $\otimes$ denotes the outer product operation, and $\mathbf{s}$ is the matrix version of the system state representation defined in Chapter 4,

$$\mathbf{s} = (\dot{\mathbf{q}} - \dot{\mathbf{q}}_{\text{des}}) + \Lambda(\mathbf{q} - \mathbf{q}_{\text{des}}), \tag{6.32}$$

where Λ is some symmetric positive definite matrix. The structure of the bias adaptation component is shown in Figure 6.20.

6.5.1.3 Einstein summation notation

At this point Einstein summation notation will be used, where subscripts now represent dimensions of variables, and identical indices between variables indicates a summation along that dimensions [89]. For example, given a vector

$$\mathbf{a} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} \tag{6.33}$$

and matrices

$$\mathbf{B} = \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \\ b_{20} & b_{21} \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \\ c_{20} & c_{21} \end{bmatrix} \tag{6.34}$$

then define $\mathbf{a}_i\mathbf{a}_i$ to be the dot product operation, equal to a scalar value

$$\sum_i \mathbf{a}_i\mathbf{a}_i = a_0a_0 + a_1a_1 + a_2a_2, \tag{6.35}$$

$\mathbf{a}_i\mathbf{a}_j$ to be the outer product operation, equal to the matrix $\mathbf{A}$, with element values

$$\mathbf{A}_{ij} = \mathbf{a}_i\mathbf{a}_j, \tag{6.36}$$

$\mathbf{a}_i\mathbf{B}_{ij}$ to be the dot product operation between a vector and a matrix, such that a vector $\mathbf{h}$ results, with element values

$$\mathbf{h}_i = \sum_i \mathbf{a}_i\mathbf{B}_{ij} \tag{6.37}$$

giving the vector

$$\mathbf{a}_i\mathbf{B}_{ij} = \mathbf{h} = \begin{bmatrix} a_0b_{00} + a_1b_{10} + a_2b_{20} & a_0b_{01} + a_1b_{11} + a_2b_{21} \end{bmatrix}, \tag{6.38}$$

$\mathbf{C}_{ij}\mathbf{B}_{ij}$ to be a double dot product operation, such that a scalar value results

$$\begin{aligned} \mathbf{C}_{ij}\mathbf{B}_{ij} &= \sum_j \sum_i \mathbf{C}_{ij}\mathbf{B}_{ij} && (6.39) \\ &= c_{00}b_{00} + c_{10}b_{10} + c_{20}b_{20} + c_{01}b_{01} + c_{11}b_{11} + c_{21}b_{21}. && (6.40) \end{aligned}$$

6.5.1.4 Stability proof of reformulation of bias adaptation

Because subscripts are no longer available for denoting specific variables in Einstein summation notation, the notational subscripts will be suppressed for the duration of the proof. Additionally, the gain on the learning term, L, will be a scalar value.

Stability of this system can be shown by choosing Lyapunov-candidate function

$$V(\mathbf{s}) = \frac{1}{2}\mathbf{s}_i\mathbf{M}_{ij}(\mathbf{q})\mathbf{s}_j + \frac{1}{2}\tilde{\boldsymbol{\theta}}_{ij}L^{-1}\tilde{\boldsymbol{\theta}}_{ij}, \tag{6.41}$$

where the second term reduces to a scalar value, as shown in Eq. 6.39, and the derivative is

$$\dot{V}(\mathbf{s}) = \mathbf{s}_i(\mathbf{u}_i - \mathbf{X}_j(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}}_r)\boldsymbol{\theta}_{ij}) + \dot{\hat{\boldsymbol{\theta}}}_{ij}L^{-1}\tilde{\boldsymbol{\theta}}_{ij}. \tag{6.42}$$

Choose the control signal

$$\mathbf{u}_i = -\mathbf{K}_{ij}\mathbf{s}_j + \mathbf{X}_j(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}}_r)\hat{\boldsymbol{\theta}}_{ij}, \tag{6.43}$$

and learning rule

$$\dot{\hat{\boldsymbol{\theta}}} = -L\mathbf{X}_j\mathbf{s}_i, \tag{6.44}$$

which is the same as Eq. 6.31, written in Einstein summation notation. Substituting these into the derivative of the Lyapunov-candidate function gives

$$\begin{aligned}
\dot{V}(\mathbf{s}) &= \mathbf{s}_i(-\mathbf{K}_{ij}\mathbf{s}_j + \mathbf{X}_j(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}}_r)\hat{\boldsymbol{\theta}}_{ij} - \mathbf{X}_j(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}}_r)\boldsymbol{\theta}_{ij}) - \mathbf{s}_i\mathbf{X}_j(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}}_r)LL^{-1}\tilde{\boldsymbol{\theta}}_{ij}, && (6.45)\\
\dot{V}(\mathbf{s}) &= -\mathbf{s}_i\mathbf{K}_{ij}\mathbf{s}_j + \mathbf{s}_i\mathbf{X}_j(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}}_r)\tilde{\boldsymbol{\theta}}_{ij} - \mathbf{s}_i\mathbf{X}_j(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}}_r)\tilde{\boldsymbol{\theta}}_{ij}, && (6.46)\\
\dot{V}(\mathbf{s}) &= -\mathbf{s}_i\mathbf{K}_{ij}\mathbf{s}_j \leq \mathbf{0}, && (6.47)
\end{aligned}$$

and so the system is globally asymptotically stable.

6.5.2 Transformation adaptation for unknown kinematics

In this section we revert back to normal matrix notation.

When attempting to implement transform adaptation to learn unknown kinematics in a neural network, several obstacles appear. The first obstacle comes from attempting to implement this adaptation in a distributed system. In the original formulation of transform adaptation, reviewed in Chapter 4, the learned parameters approximate the end-effector velocity

$$\dot{\mathbf{x}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} = \mathbf{Y}_k(\mathbf{q},\dot{\mathbf{q}})\hat{\boldsymbol{\theta}}_k, \tag{6.48}$$

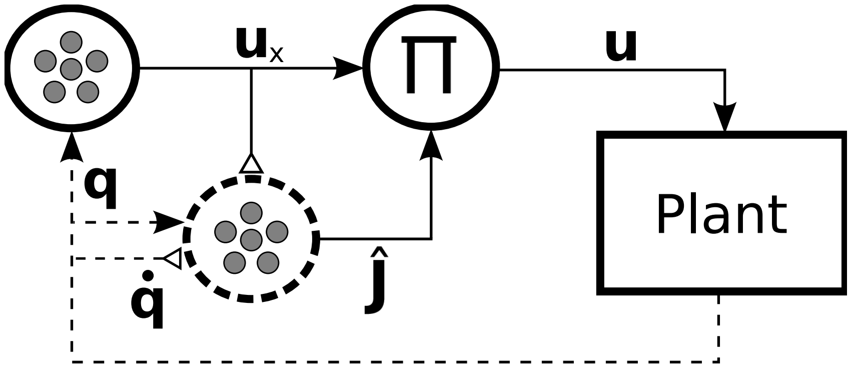


Figure 6.18: A diagram of the basic setup of a control system using a bias adaptation component. The bias adaptation component (inside the dashed circle) takes the system state $\mathbf{q}$ as input and the control signal $\mathbf{u}$ as a learning signal, and outputs an approximation of the unknown forces, $\hat{\mathbf{F}}_{unknown}$. This signal is then sent to be summed with $\mathbf{u}$ and sent out to the system under control. The hollow triangle denotes a modulatory connection.

and then the $\dot{\mathbf{q}}$ parameters are removed artificially from $\mathbf{Y}_k(\mathbf{q}, \dot{\mathbf{q}})\hat{\boldsymbol{\theta}}_k$ to get the approximation of the Jacobian $\hat{\mathbf{J}}(\mathbf{q}, \hat{\boldsymbol{\theta}}_k)$. This artificial removing of parameters from the signal, in addition to being biologically implausible, can't be done in a distributed system, because the approximation of each of the elements aren't cleanly separated into distinct system parameters. Rather, a weighted summation across all of the nonlinear components of the distributed system must be taken to get a meaningful signal. The result of this weighted summation then is the signal $\hat{\mathbf{J}}(\mathbf{q}, \hat{\boldsymbol{\theta}}_k)\dot{\mathbf{q}}$, and the Jacobian approximation can't be cleanly extracted.

To get around this the adaptive system instead approximates the Jacobian directly, which then gives rise to the second implementation obstacle: the learning rule used in the original formulation is no longer appropriate. In this section the reformulation of the problem to approximate the Jacobian will be presented, including a stability proof, followed by a second reformulation into an equation that can be implemented in a distributed system with scalar output, as in the previous section.

In both cases the general system setup is as shown in Figure 6.18. There is a task-space control signal, $\mathbf{u}_\mathbf{x}$ that takes the system state $\mathbf{q}$ as input to the neurons, and the task-space control signal and system velocity $\dot{\mathbf{q}}$ (whose inclusion will be analytically justified in the next section) as input to the learning signal. The adaptive system is attempting to approximate the Jacobian of the system, to transform the task-space control signal into a signal that can interact with the system, i.e. the low-level control signal $\mathbf{u}$. To this end, the output of the adaptive system is the Jacobian approximation, $\hat{\mathbf{J}}(\mathbf{q}, \hat{\boldsymbol{\theta}}_k)$, which is then multiplied by the task-space control signal. The full control signal will be of the form

$$\mathbf{u} = \hat{\mathbf{J}}^T(\mathbf{q}, \hat{\boldsymbol{\theta}}_k)\mathbf{K}_p\tilde{\mathbf{x}} - \mathbf{K}_v\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}), \tag{6.49}$$

where $\tilde{\mathbf{x}} = (\mathbf{x} - \mathbf{x}_{\text{des}})$ is the task-space position error signal, $\mathbf{K}_p$ and $\mathbf{K}_v$ are symmetric positive definite gain matrices, and $\mathbf{g}(\mathbf{q})$ is the gravity compensation term.

The goal of the transform adaptation term is to learn a set of parameters $\hat{\boldsymbol{\theta}}_k$ that accurately approximates the Jacobian of the system. This setup allows the system to learn how to transform a control signal from one space, such as end-effector space, to another, such as joint space. The learning rule for updating the learned parameters $\dot{\hat{\boldsymbol{\theta}}}_k$ is responsible for minimizing the approximation error of $\hat{\mathbf{J}}(\mathbf{q}, \hat{\boldsymbol{\theta}}_k)$.

6.5.2.1 Control with adaptive transformation in the vector space

As mentioned above, in the typical formulation [29] the Jacobian is not directly solved for. Instead, the learned parameters, $\hat{\boldsymbol{\theta}}_k$, are solved for in the equation:

$$\mathbf{Y}_k(\mathbf{q}, \dot{\mathbf{q}})\,\hat{\boldsymbol{\theta}}_k = \hat{\mathbf{J}}(\mathbf{q}, \hat{\boldsymbol{\theta}}_k)\dot{\mathbf{q}}, \tag{6.50}$$

and then the Jacobian approximation, $\hat{\mathbf{J}}$, is artificially extracted afterwards from $\mathbf{Y}_k(\mathbf{q}, \dot{\mathbf{q}})\hat{\boldsymbol{\theta}}_k$ by removing the $\dot{\mathbf{q}}$ terms. A novel reformulation of the problem such that $\hat{\boldsymbol{\theta}}_k$ is solved for can be found in this expression:

$$\mathbf{Z}_k(\mathbf{q})\hat{\boldsymbol{\theta}}_k = \hat{\mathbf{J}}(\mathbf{q}, \hat{\boldsymbol{\theta}}_k), \tag{6.51}$$

where $\mathbf{Z}_k(\mathbf{q})$ is a third-rank tensor. For example, when $\mathbf{J} \in \mathbb{R}^{2\times3}$ and $\hat{\boldsymbol{\theta}}_k \in \mathbb{R}^{n\times1}$, then $\mathbf{Z}_k(\mathbf{q}) \in \mathbb{R}^{3\times2\times n}$.

The system state drives the activation of the neurons in the multiplicative basis set, $\mathbf{Z}_k(\mathbf{q})$. The output is again weighted by the learned parameters $\hat{\boldsymbol{\theta}}_k$, which sum and decode it into the vector space. The result of this weighted summation, $\hat{\mathbf{J}}$, is sent to be multiplied by the control signal $\mathbf{u}_x$ to transform it from a high-level signal to a lower level signal. The system velocities, $\dot{\mathbf{q}}$, are separately multiplied by the output of $\mathbf{Z}_k(\mathbf{q})$ to generate the learning signal, $\dot{\hat{\boldsymbol{\theta}}}_k$.

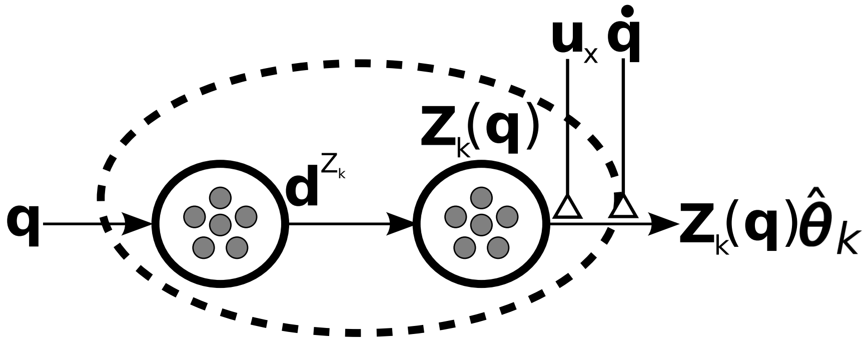

$\mathbf{u}_x$
$\dot{\mathbf{q}}$
$\mathbf{Z}_k(\mathbf{q})$
$\mathbf{d}^{Z_k}$
$\mathbf{q}$
$\mathbf{Z}_k(\mathbf{q})\hat{\boldsymbol{\theta}}_k$

Figure 6.19: The structure of a neural network set up to implement the modified formulation of the nonlinear adaptive transform component. The matrix of basis functions $\mathbf{Z}_k$ is decoded from the neural activity of the first population, where each of the elements of $\mathbf{Z}_k$ is approximated through a set of decoders determined with the methods presented in Section 6.1. This matrix is passed to a second population of neurons which applies the set of learned system parameters $\hat{\boldsymbol{\theta}}_k$ stored in its decoders. These decoders are updated through a learning signal that is a function of the task-space control signal $\mathbf{u}_x$ and system velocity $\dot{\mathbf{q}}$.

The learning update for the parameters $\hat{\boldsymbol{\theta}}_k$ is calculated through the following novel variant of Slotine's original update rule:

$$\dot{\hat{\boldsymbol{\theta}}}_k = -\mathbf{L}_k \dot{\mathbf{q}}^T \mathbf{Z}_k(\mathbf{q})^T \mathbf{K}_p \tilde{\mathbf{x}}, \tag{6.52}$$

where $\mathbf{L}_k$ is the learning rate for the adaptive transform.

The system structure of the implementation of this transformation adaptation component in a neural network is shown in Figure 6.19.

A proof of the stability of the system under this formulation with this learning rule is now presented.

6.5.2.2 Stability proof of vector space adaptive transform reformulation

Redefine the function that is approximated by the transform adaptation component as

$$\mathbf{J}(\mathbf{q}) = \mathbf{Z}_k(\mathbf{q})\boldsymbol{\theta}_k \tag{6.53}$$

where $\mathbf{J}(\mathbf{q}) \in \mathbb{R}^{l \times m}$, $\mathbf{Z}_k(\mathbf{q}) \in \mathbb{R}^{l \times m \times n}$, and $\boldsymbol{\theta}_k \in \mathbb{R}^{n \times 1}$.

6.5.2.2.1 Attempt 1

In the original formulation the learning rule was defined with basis matrix $\mathbf{Y}_k$ that incorporated $\dot{\mathbf{q}}$:

$$\mathbf{Y}_k(\mathbf{q}, \dot{\mathbf{q}})\boldsymbol{\theta}_k = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} = \mathbf{Z}_k(\mathbf{q})\boldsymbol{\theta}_k\dot{\mathbf{q}}, \tag{6.54}$$

so an approach might be to attempt to build a similar $\mathbf{Y}_z$ matrix

$$\mathbf{Y}_z(\mathbf{q}, \dot{\mathbf{q}}, \hat{\boldsymbol{\theta}}_k) = \mathbf{Z}_k(\mathbf{q})\boldsymbol{\theta}_k\dot{\mathbf{q}}, \tag{6.55}$$

and continue using the learning rule of the same form:

$$\dot{\hat{\boldsymbol{\theta}}}_k = \mathbf{L}_k\mathbf{Y}_z^T(\mathbf{q}, \dot{\mathbf{q}}, \hat{\boldsymbol{\theta}}_k)\mathbf{K}_p\tilde{\mathbf{x}}. \tag{6.56}$$

Using the same Lyapunov-like function candidate as in the original proof (detailed in Section 4.4)

$$V(\mathbf{q}) = \frac{1}{2}\dot{\mathbf{q}}^T\mathbf{M}(\mathbf{q})\dot{\mathbf{q}} + \frac{1}{2}\tilde{\hat{\boldsymbol{\theta}}}_k\mathbf{L}_k^{-1}\tilde{\hat{\boldsymbol{\theta}}}_k + \frac{1}{2}\tilde{\mathbf{x}}^T\mathbf{K}_p\tilde{\mathbf{x}}, \tag{6.57}$$

to be negative semi-definite the derivative $\dot{V}(\mathbf{q})$ requires that

$$\dot{\mathbf{q}}^T(\mathbf{J}_T(\mathbf{q}) - \hat{\mathbf{J}}^T(\mathbf{q}, \hat{\boldsymbol{\theta}}_k))\mathbf{K}_p\tilde{\mathbf{x}} = \dot{\hat{\boldsymbol{\theta}}}_k^T\mathbf{L}_k^{-1}\tilde{\boldsymbol{\theta}}_k. \tag{6.58}$$

Substituting in Eq. 6.56 to the right hand side of Eq. 6.58 gives

$$\begin{aligned}
\dot{\mathbf{q}}^T(\mathbf{J}_T(\mathbf{q}) - \hat{\mathbf{J}}^T(\mathbf{q}, \hat{\boldsymbol{\theta}}_k))\mathbf{K}_p\tilde{\mathbf{x}} &= \tilde{\mathbf{x}}^T\mathbf{K}_p\mathbf{Y}_z(\mathbf{q}, \dot{\mathbf{q}}, \hat{\boldsymbol{\theta}}_k)\mathbf{L}_z\mathbf{L}_z^{-1}\tilde{\boldsymbol{\theta}}_k, && (6.59)\\
\dot{\mathbf{q}}^T(\mathbf{J}_T(\mathbf{q}) - \hat{\mathbf{J}}^T(\mathbf{q}, \hat{\boldsymbol{\theta}}_k))\mathbf{K}_p\tilde{\mathbf{x}} &= \tilde{\boldsymbol{\theta}_k}^T\mathbf{Y}_z^T(\mathbf{q}, \dot{\mathbf{q}}, \hat{\boldsymbol{\theta}}_k)\mathbf{K}_p\tilde{\mathbf{x}}. && (6.60)
\end{aligned}$$

Where previously this worked out because $\mathbf{Y}_k(\mathbf{q}, \dot{\mathbf{q}})\boldsymbol{\theta}_k = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}$, this is no longer the case, as $\mathbf{Y}_z$ and $\boldsymbol{\theta}_k$ have no such relationship. As a result this learning rule won't work to cancel out the left hand side of Eq. 6.58, and $\dot{V}$ cannot be proven to be negative semi-definite.

6.5.2.2.2 Attempt 2

Another approach is to define the learning rule directly with $\mathbf{Z}$, i.e.

$$\dot{\hat{\boldsymbol{\theta}}}_k = \mathbf{L}_z\mathbf{Z}_k^T(\mathbf{q})\mathbf{K}_p\tilde{\mathbf{x}}. \tag{6.61}$$

Immediately problems are encountered in that the dimensionality of the left hand side is not the same as the right.

Ignoring this for the moment, some intuition about how to change this equation to be appropriate can be found by following the same Lyapunuv-candidate function proof as in the first attempt. Substituting Eq. 6.61 into Eq. 6.58 gives

$$\begin{aligned}
\dot{\mathbf{q}}^T(\mathbf{J}_T(\mathbf{q}) - \hat{\mathbf{J}}^T(\mathbf{q}, \hat{\boldsymbol{\theta}}_k))\mathbf{K}_p\tilde{\mathbf{x}} &= \tilde{\mathbf{x}}^T\mathbf{K}_p\mathbf{Z}_k(\mathbf{q})\mathbf{L}_z\mathbf{L}_z^{-1}\tilde{\boldsymbol{\theta}}_k, && (6.62)\\
\dot{\mathbf{q}}^T(\mathbf{J}_T(\mathbf{q}) - \hat{\mathbf{J}}^T(\mathbf{q}, \hat{\boldsymbol{\theta}}_k))\mathbf{K}_p\tilde{\mathbf{x}} &= \tilde{\boldsymbol{\theta}}_k^{\ T}\mathbf{Z}_k^T(\mathbf{q})\mathbf{K}_p\tilde{\mathbf{x}} && (6.63)\\
\dot{\mathbf{q}}^T(\mathbf{J}_T(\mathbf{q}) - \hat{\mathbf{J}}^T(\mathbf{q}, \hat{\boldsymbol{\theta}}_k))\mathbf{K}_p\tilde{\mathbf{x}} &= (\mathbf{J}(\mathbf{q}) - \hat{\mathbf{J}}(\mathbf{q}, \hat{\boldsymbol{\theta}}_k))^T\mathbf{K}_p\tilde{\mathbf{x}}, && (6.64)
\end{aligned}$$

and it can be seen that a $\dot{\mathbf{q}}$ term is missing from the beginning of the right hand side in Eq. 6.64.

6.5.2.2.3 Attempt 3

Rewriting the learning rule to incorporate the $\dot{\mathbf{q}}$ term appropriately suggests the learning rule

$$\dot{\hat{\boldsymbol{\theta}}}_k = \mathbf{L}_z\dot{\mathbf{q}}^T\mathbf{Z}_k^T(\mathbf{q})\mathbf{K}_p\tilde{\mathbf{x}}, \tag{6.65}$$

where the left and right hand sides are both now the same dimensionality. Using the same Lyapunov-like function candidate as in the first two attempts and substituting in Eq. 6.65 to Eq. 6.58 gives

$$\begin{aligned}
\dot{\mathbf{q}}^T(\mathbf{J}_T(\mathbf{q}) - \hat{\mathbf{J}}^T(\mathbf{q}, \hat{\boldsymbol{\theta}}_k))\mathbf{K}_p\tilde{\mathbf{x}} &= \tilde{\mathbf{x}}^T\mathbf{K}_p\mathbf{Z}_k(\mathbf{q})\mathbf{q}\mathbf{L}_k\mathbf{L}_k^{-1}\tilde{\hat{\boldsymbol{\theta}}}_k, && (6.66)\\
\dot{\mathbf{q}}^T(\mathbf{J}_T(\mathbf{q}) - \hat{\mathbf{J}}^T(\mathbf{q}, \hat{\boldsymbol{\theta}}_k))\mathbf{K}_p\tilde{\mathbf{x}} &= \tilde{\hat{\boldsymbol{\theta}}}_k^T\dot{\mathbf{q}}^T\mathbf{Z}_k^T(\mathbf{q})\mathbf{K}_p\tilde{\mathbf{x}}. && (6.67)
\end{aligned}$$

Because $\mathbf{Z}_k(\mathbf{q})$ is a third rank tensor, the math at this point is a bit unintuitive, but it can be shown that

$$(\mathbf{Z}_k(\mathbf{q})\hat{\boldsymbol{\theta}}_k\dot{\mathbf{q}})^T = (\mathbf{J}(\mathbf{q})\dot{\mathbf{q}})^T = \hat{\boldsymbol{\theta}}_k^T\dot{\mathbf{q}}^T\mathbf{Z}_k^T(\mathbf{q}) = \dot{\mathbf{q}}^T\mathbf{J}_T(\mathbf{q}). \tag{6.68}$$

Using this equality Eq. 6.67 can be rewritten

$$\dot{\mathbf{q}}^T(\mathbf{J}_T(\mathbf{q}) - \hat{\mathbf{J}}^T(\mathbf{q}, \hat{\boldsymbol{\theta}}_k))\mathbf{K}_p\tilde{\mathbf{x}} = \dot{\mathbf{q}}^T(\mathbf{J}_T(\mathbf{q}) - \hat{\mathbf{J}}^T(\mathbf{q}, \hat{\boldsymbol{\theta}}_k))\mathbf{K}_p\tilde{\mathbf{x}}, \tag{6.69}$$

and so the derivative of $V(\mathbf{q})$ works out to

$$\dot{V}(\mathbf{q}) = -\dot{\mathbf{q}}^T\mathbf{K}_v\dot{\mathbf{q}} \leq \mathbf{0}, \tag{6.70}$$

and so the system is globally asymptotically stable.

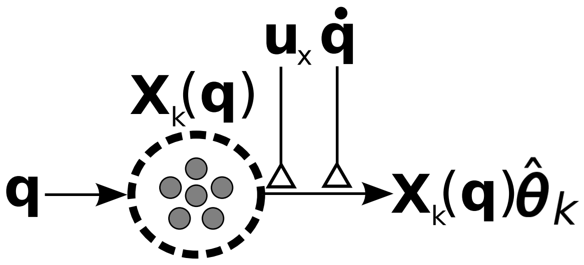

$\mathbf{u}_x$
$\dot{\mathbf{q}}$
$\mathbf{X}_k(\mathbf{q})$
$\mathbf{q}$
$\mathbf{X}_k(\mathbf{q})\hat{\boldsymbol{\theta}}_k$

Figure 6.20: The structure of a neural network set up to implement the reworked formulation of the nonlinear adaptive transform component. The basis functions vector $\mathbf{X}_k$ is the neural activity of the population in the dashed line, and the decoders of this population are the learned parameters $\hat{\boldsymbol{\theta}}_k$. These parameters are updated as a function of the control signal $\mathbf{u}$ and the system velocity $\dot{\mathbf{q}}$. The hollow triangle denotes a modulatory connection.

6.5.2.3 Control with adaptive transformation in neuron space

In the formulation of nonlinear adaptation in the previous section a large set of basis functions, $\mathbf{Z}_k(\mathbf{q})$, are used, and a single set of learned parameters, $\hat{\boldsymbol{\theta}}_k$, weights multiple basis functions for the different decoded dimensions. As in the bias adaptation the algorithm above with multi-dimensional output from the basis functions cannot be implemented using the neurons directly as the basis functions. This calls for an alternate implementation with scalar basis functions and multi-dimensional learned parameters that lends itself to neural implementation.

To implement this, define a set of basis functions $\mathbf{X}_k$ to be the activity of a population of neurons with the system state, $\mathbf{q}$, as input. Instead of having the basis functions be a tensor as in the first reformulation of transform adaptation, the learned parameters, $\hat{\boldsymbol{\theta}}_k \in \mathbb{R}^{i \times j \times k}$, will be a tensor, and define $\mathbf{X}_k(\mathbf{q}) \in \mathbb{R}^{1 \times i}$ such that

$$\mathbf{X}_k(\mathbf{q})\hat{\boldsymbol{\theta}}_k = \mathbf{J}(\mathbf{q}), \tag{6.71}$$

where the Jacobian $\mathbf{J}(\mathbf{q}) \in \mathbb{R}^{j \times k}$.

6.5.2.3.1 Stability proof of neuron space adaptive transform reformulation

Once again Einstein summation notation will be used, and so for the duration of this proof all notational subscripts will be suppressed. This notation switch is especially useful here with three-tensors because it greatly simplifies denoting the operation to square and sum all the elements. Given a vector $\mathbf{a} \in \mathbb{R}^i$ and tensor $\mathbf{C} \in \mathbb{R}^{i \times j \times k}$, the tensor equivalent of $\mathbf{a}^T\mathbf{a}$, which squares and sums all of the elements (dot product with itself), can be written as $\mathbf{C}_{ijk}\mathbf{C}_{ijk}$. Additionally, the learning gain term, L, will once again be chosen as a scalar value.

Choose the Lyapunov candidate function

$$V(\mathbf{q}) = \frac{1}{2}\dot{\mathbf{q}}_i\mathbf{M}_{ij}(\mathbf{q})\dot{\mathbf{q}}_j + \frac{1}{2}\tilde{\boldsymbol{\theta}}_{ijk}L^{-1}\tilde{\boldsymbol{\theta}}_{ijk} + \frac{1}{2}\tilde{\mathbf{x}}_i\mathbf{K}_{ij}\tilde{\mathbf{x}}_j, \tag{6.72}$$

whose time derivative is

$$\dot{V}(\mathbf{q}) = -\dot{\mathbf{q}}_i\mathbf{K}_{ij}\dot{\mathbf{q}}_j + \dot{\mathbf{q}}_k(\mathbf{J}_{jk}(\mathbf{q}) - \hat{\mathbf{J}}_{jk}(\mathbf{q}, \hat{\boldsymbol{\theta}}))\mathbf{K}_{ji}\tilde{\mathbf{x}}_i - \dot{\hat{\boldsymbol{\theta}}}_{ijk}L^{-1}\tilde{\boldsymbol{\theta}}_{ijk}. \tag{6.73}$$

Choosing a learning rule candidate:

$$\dot{\hat{\boldsymbol{\theta}}}_{ijk} = L\mathbf{X}_i(\mathbf{q})\dot{\mathbf{q}}_k\mathbf{K}_{jl}\tilde{\mathbf{x}}_l, \tag{6.74}$$

gives

$$\dot{\hat{\boldsymbol{\theta}}}_{ijk}L^{-1}\tilde{\boldsymbol{\theta}}_{ijk} = \tilde{\mathbf{x}}_l\mathbf{K}_{lj}\dot{\mathbf{q}}_k\mathbf{X}_i(\mathbf{q})\tilde{\boldsymbol{\theta}}_{ijk} = \tilde{\mathbf{x}}_l\mathbf{K}_{lj}\dot{\mathbf{q}}_k(\mathbf{J}_{jk}(\mathbf{q}) - \hat{\mathbf{J}}_{jk}(\mathbf{q}, \hat{\boldsymbol{\theta}}_k)). \tag{6.75}$$

Because this term reduces to a scalar, it's equal to its own transpose and can be rewritten

$$(\mathbf{J}_{jk}(\mathbf{q}) - \hat{\mathbf{J}}_{jk}(\mathbf{q}, \hat{\boldsymbol{\theta}}))\dot{\mathbf{q}}_k\mathbf{K}_{jl}\tilde{\mathbf{x}}_l. \tag{6.76}$$

Substituting this into Eq. 6.73 gives

$$\dot{V}(\mathbf{q}) = -\dot{\mathbf{q}}_i\mathbf{K}_{ik}\dot{\mathbf{q}}_k + \dot{\mathbf{q}}_k(\mathbf{J}_{jk}(\mathbf{q}) - \hat{\mathbf{J}}_{jk}(\mathbf{q}, \hat{\boldsymbol{\theta}}))\mathbf{K}_{ji}\tilde{\mathbf{x}}_i - (\mathbf{J}_{jk}(\mathbf{q}) - \hat{\mathbf{J}}_{jk}(\mathbf{q}, \hat{\boldsymbol{\theta}}))\dot{\mathbf{q}}_k\mathbf{K}_{ji}\tilde{\mathbf{x}}_i. \tag{6.77}$$

For $\dot{V}(\mathbf{q})$ to be negative semi-definite it needs to be shown that

$$\mathbf{q}_k(\mathbf{J}_{jk}(\mathbf{q}) - \hat{\mathbf{J}}_{jk}(\mathbf{q}, \hat{\boldsymbol{\theta}}))\mathbf{K}_{ji}\tilde{\mathbf{x}}_i = (\mathbf{J}_{jk}(\mathbf{q}) - \hat{\mathbf{J}}_{jk}(\mathbf{q}, \hat{\boldsymbol{\theta}}))\mathbf{q}_k\mathbf{K}_{ji}\tilde{\mathbf{x}}_i, \tag{6.78}$$

which reduces to showing that

$$\begin{aligned} \dot{\mathbf{q}}_k(\mathbf{J}_{jk}(\mathbf{q}) - \hat{\mathbf{J}}_{jk}(\mathbf{q}, \hat{\boldsymbol{\theta}})) &= (\mathbf{J}_{jk}(\mathbf{q}) - \hat{\mathbf{J}}_{jk}(\mathbf{q}, \hat{\boldsymbol{\theta}}))\dot{\mathbf{q}}_k, & (6.79) \\ \dot{\mathbf{q}}_k\tilde{\mathbf{J}}_{jk} &= \tilde{\mathbf{J}}_{jk}\dot{\mathbf{q}}_k, & (6.80) \end{aligned}$$

where $\tilde{\mathbf{J}}_{jk} = \mathbf{J}_{jk}(\mathbf{q}) - \hat{\mathbf{J}}_{jk}(\mathbf{q}, \hat{\boldsymbol{\theta}})$. Both of these sides say the same thing, where the vector $\dot{\mathbf{q}}$ is multiplied by the columns of $\tilde{\mathbf{J}}$, so they are equal. This equality then reduces Eq. 6.77 to

$$\dot{V}(\mathbf{q}) = -\dot{\mathbf{q}}_i\mathbf{K}_{ik}\dot{\mathbf{q}}_k \leq \mathbf{0}, \tag{6.81}$$

and global stability is proven.

6.6 Discussion

The reformulations of the nonlinear adaptive algorithms are the result of work with Terry Stewart, Chris Eliasmith, Jean-Jacques Slotine, and myself, and a patent has been filed [42]. Notably, this reformulation allows the application of nonlinear adaptive control methods to neuromorphic hardware, where the basis function output is necessarily scalar. Neuromorphic hardware represents an opportunity for extremely low-energy efficient implementations of neural networks and will be a focus of future work.

At this point it is now possible to implement the control theoretic methods described in the first half of this thesis in a neural network as part of a model of the motor control system, which is the focus of the next chapter.

Chapter 7

The REACH model

In this chapter the Recurrent Error-driven Adaptive Control Hierarchy (REACH) model, a large-scale spiking neuron model of the motor cortices and cerebellum of the motor control system, is presented. The REACH model is an implementation and update of a subset of the NOCH framework, described in Chapter 5. The model consists of anatomically organized spiking neurons that control a nonlinear three-link arm to perform reaching and handwriting, while being able to adapt to unknown changes in arm dynamics and structure (i.e. kinematics). I demonstrate the model's ability to control a 3 link, nonlinear arm through complex paths including handwritten words and numbers. I also demonstrate its ability to adapt to environmental changes (e.g. an unknown force field) and changes to the physical properties of the arm (e.g. from growth). I employ the same methods as used in Spaun, so this model also serves to update the previous non-neural motor control system presented with Spaun originally in [50].

The dynamical system that forms the basis of the spiking neuron implementation of the REACH model is a combination of the operational space control, nonlinear adaptive control, and dynamic movement primitive methods discussed in previous chapters. This combination in itself is novel. Previous DMP models use inverse kinematics for translating desired end-effector trajectories into robotic movement while operational space control and nonlinear adaptive control models have their trajectories explicitly programmed in. While such a behavioural-level model can be used to compare to high-level movement phenomena, of particular interest is a spiking neuron model that can be compared to both high and low-level data from the motor system, such as single-cell activity profiles, neural correlations with high-level movement parameters, and neural analysis from experimental data. As shown below, the REACH model accounts for data across 19 clinical and experimental studies of the motor control system. These data includes a mix of behavioural and neural spiking activity, across normal and damaged subjects performing adaptive and static tasks. I provide a variety of comparisons to behavioural performance during static and adaptive tasks, as well as detailed neural response analyses compared to classic and recent results in experimental neuroscience.

In short, the REACH model represents a novel integration of control theoretic methods and neuroscientific constraints to specify a general, adaptive, biologically plausible motor control algorithm. Such algorithms can provide critical clinical insights for developing motor system interventions, and are of particular interest given recent developments in neuromorphic hardware [131, 146, 14], that requires advanced algorithms employing spiking neural networks.

In the remainder of the chapter the REACH model is presented, with a brief review of the control theory that forms the basis of the relevant dynamical system. Following this, results from the non-neural implementation of the REACH model are presented. A detailed description of the neural network implementations of the pre-motor cortex, primary motor cortex, and cerebellum models is then given. Finally, results from the neural implementation of the REACH model are presented along with a discussion of the limitations of the model, directions for future work, and a comparison to previous NOCH models.

7.1 REACH overview

The left hand side of Figure 7.1 identifies the major neural and physical components of the model. The arm itself, shown on the right hand side of Figure 7.1, is a three-link arm operating in the horizontal plane, physically modeled after the human arm using length and inertia parameters as described in [181]. The arm simulation was built using MapleSim, based on the arm model presented in [182], and is available online.[1].

The neural components of the model capture major systems in the motor hierarchy and are shown at the top of Figure 7.1. As described in Chapter 5, the premotor cortex (PM) is known to play an important role in the preparation and planning of movements, as well as the control of timing during execution [73, 3, 209]. The neural activity recorded from PM during tasks where a path is traced out using different limbs suggests that it is largely responsible for abstract trajectory generation [155, 33]. The premotor cortex's trajectory generation functionality is modeled using dynamical movement primitives [159, 97] (DMPs), as presented in Chapter 3. The resulting trajectories are typically specified

[1]`https://github.com/studywolf/blog/tree/master/Control/Arms/three_link`

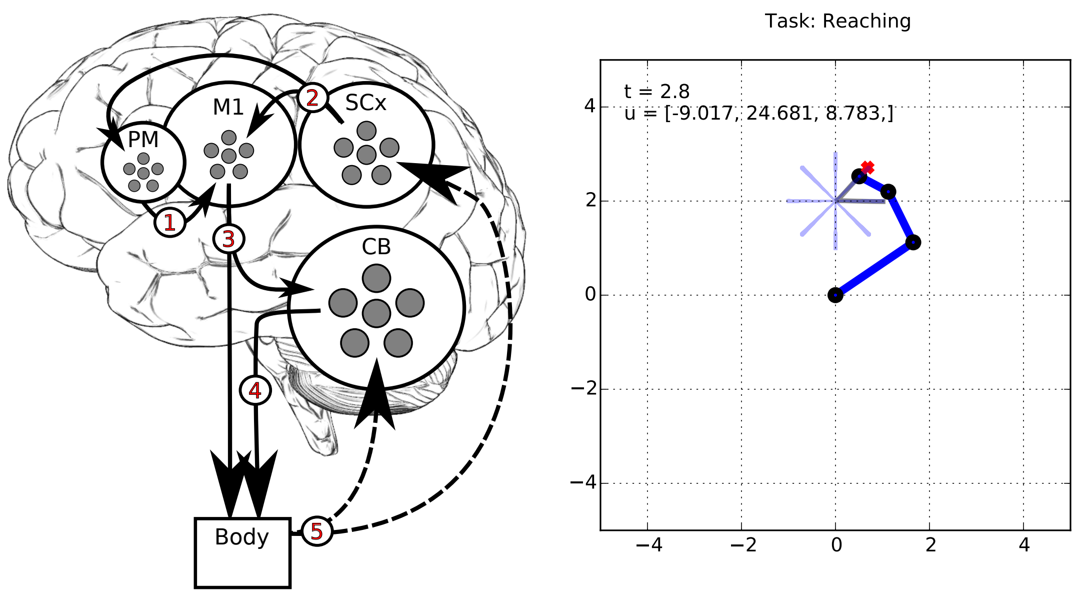


Figure 7.1: Left: An overview of the model. Numbers identify major communication pathways. Dashed lines indicate closed-loop feedback signals generated from the senses. The premotor cortex (PM) generates a trajectory for the system to follow with a sequence of (x, y) coordinates. The primary motor cortex (M1) receives these target positions (**1**) from the sensory cortices (SCx) and compares them to the current system state, received through (**2**). M1 combines this signal with locally stored Jacobians to transform the desired hand movement commands into a low-level signal that is sent to the arm and cerebellum (CB) along (**3**). The cerebellum (CB) projects an adaptive signal to the body along (**4**) that compensates for velocity and movement errors. The senses generate a closed-loop feedback signal along (**5**) that is provided to the CB and SCx. Right: A picture of the three-link arm controlled by the REACH model during a centre-out reaching task.

in an abstract space, such as the Cartesian coordinates of the end-effector in the case of arm control. Because of this abstraction, the trajectories are easily generalized through spatial transformations (e.g. scaling, rotation, etc.), temporal transformations (e.g. rapid approximate, or slow accurate movements), and in terms of which limb carries out the command. The use of DMPs thus naturally accounts for the observation that handwriting style is typically transfered between hands and even to feet and other body parts [208].

The kinds of generalization supported by DMPs are much easier in the abstract space, provided there is a method for transforming the abstract trajectory commands to low-level commands that can drive the complex dynamics of the system, in this case the arm. In our model, PM generates trajectories in 2D (x, y) space and sends those to the primary motor cortex (M1) for execution. It is M1 that effects the transformation from the abstract space to the lower-level to drive the physical arm, using the operational space control methods discussed in Chapter 2.

Graziano [73] presents results that show that M1 is not a simple fixed mapping to muscles, but rather a complex function of the state of the system. This mapping can be described in control theoretic terms as determining the low-level system state $\mathbf{q}$ from the high-level abstract representation of the system $\mathbf{x}$ using a Jacobian matrix that is a function of the system position.

Recall that the Jacobian also defines a relationship between the forces in these two spaces:

$$\mathbf{u} = \mathbf{J}^T \mathbf{u}_\mathbf{x} \tag{7.1}$$

where $\mathbf{u}$ is the joint torque motor command sent to the arm, and $\mathbf{u}_\mathbf{x}$ is the high-level motor command generated in PM. Functional dependencies are suppressed in this chapter for clarity.

To accurately control the system, the controller must compensate for the inertia generated by the arm in task space. Accounting for this, the motor command sent out by M1 becomes

$$\mathbf{u} = \mathbf{J}^T \mathbf{M}_\mathbf{x} \mathbf{u}_\mathbf{x}, \tag{7.2}$$

where $\mathbf{M}_\mathbf{x}$ is the task-space inertia matrix decoded from system state information, and is a function of the system position and velocity, whose derivation is worked through is Section 7.9.1.2.

Additionally, M1 needs to be capable of adapting to changes in the physical properties of the arm (e.g. those caused by growth). The nonlinear adaptive control [29, 42] discussed in Chapter 4 allow the Jacobian to be learned on-the-fly . This adaptability is indicated by replacing the standard Jacobian $\mathbf{J}$ with an adaptive Jacobian, $\hat{\mathbf{J}}$.

Furthermore, animals are able to optimize movements with respect to several goals simultaneously [16]. For instance, monkeys can reach to a target while staying as close to a default resting position as possible, to minimize energy usage. Recent work on operational space control [112] provides a natural framework for including these kinds of additional constraints. To account for the contributions of this kind of secondary controller, an additional force term, $\mathbf{u}_{\text{null}}$, is added. The signal $\mathbf{u}_{\text{null}}$ represents a filtered control signal from the secondary controller, modified to guarantee that movement in the operational space is not affected using the methods discussed in Section 2.1.

As a result of these two extensions, the motor command becomes (see Figure 7.9 for the neural circuit diagram):

$$\mathbf{u} = \hat{\mathbf{J}}^T \mathbf{M}_{\mathbf{x}} \mathbf{u}_{\mathbf{x}} + \mathbf{u}_{\text{null}}. \tag{7.3}$$

The cerebellum is thought to maintain internal models of the system dynamics as well as provide error correction signals [212, 108, 110]. Motor control researchers generally identify two kinds of internal models: forward models, which map potential actions to their outcomes; and inverse models, which map desired outcomes to actions. The REACH model includes both forward and inverse models by focusing on the error correction ability of the cerebellum. Specifically, the cerebellum receives the M1 generated control signal and information regarding the current state of the body, which it uses to learn the result of an action given the resulting error. This allows it to build up a forward and inverse model that it then uses to generate a corrective command that is added to the M1 generated command to reduce movement error. Support for the cerebellum providing error correction based on predicted error is found in numerous studies [172, 199, 57]. As discussed in Chapter 5, the corrective signals are divided into two terms, one that accounts for the internal dynamics of the system, and one that accounts for external and temporary changes in the system dynamics, representing contributions from the lateral and intermediate cerebellum, respectively. With the addition of the cerebellar signals, the full command sent to the body becomes:

$$\mathbf{u} = \hat{\mathbf{J}}^T \mathbf{M}_{\mathbf{x}} \mathbf{u}_{\mathbf{x}} + \mathbf{u}_{\text{null}} - \mathbf{M}\dot{\mathbf{q}} + \mathbf{u}_{adapt}, \tag{7.4}$$

where the $-\mathbf{M}\dot{\mathbf{q}}$ term accounts for and cancels out the internal forces of the system, and $\mathbf{u}_{adapt}$ is an adaptive function of $\mathbf{q}$ and $\dot{\mathbf{q}}$ that learns to correct for control errors (see Figure 7.10 for the neural circuit diagram).

While not included in this model, the basal ganglia are generally considered part of the motor system. The basal ganglia is not included here for two reasons. First, the basal ganglia is considered critical for reinforcement learning [45, 102] and the volitional control of movement [134, 77]. Neither of these elements are important for the variety of tasks and data that are explored here. For instance, the kind of learning relevant to our model is believed to occur in the cerebellum [47, 45]. Studies have shown that patients with basal ganglia lesions, but not those with cerebellar lesions, are capable of adapting to reach errors [172, 12]. As well, the basal ganglia are important for determining which actions are most appropriate for a given situation, but the tasks performed by the REACH model include explicit direction at the beginning of each simulation as to which action to perform, making flexible action selection outside the scope of the model. Second, the Spaun model includes a basal ganglia that is able to account for flexible action selection and simple reinforcement learning [50]. Consequently, its inclusion here would be somewhat redundant.

7.2 Non-neural implementation results

Prior to a neural implementation, the REACH model was built without neurons to make sure that behavioural level phenomena were successfully captured by the chosen control methods. Many of these results have already been seen, in the figures of Chapter 2 and Chapter 3. This included tracing paths, performing handwriting, and implementing null-space controllers to keep the arm near a default resting position during movement. Here several further results involving the reformulation of nonlinear adaptation, presented in Chapter 6, are shown. Note that while the adaptive components of the system are implemented in neurons, the rest of the system has been implemented in standard Python. This control system can be found online on github.[2]

7.2.1 Bias adaptation example

The bias adaptation component receives joint positions and angular velocities as an input signal, and the low-level control signal **u** as a learning signal, as described in Chapter 6.

Here results for adapting to internal and external forces that were not sufficiently modeled and accounted for by the initial controller are shown. First, in Figure 7.2 the system is attempting to move to a set of targets from a center starting point while a perturbing force field is being applied to the hand. This force field is taken from a similar experiment performed on human subjects [168]. The size of the applied force is based on the velocity of the hand. On the left is the system, with no adaptive bias component. On the right the same system is presented, but with the adaptive bias component added in, consisting of

[2]https://github.com/studywolf/blog/tree/master/Control

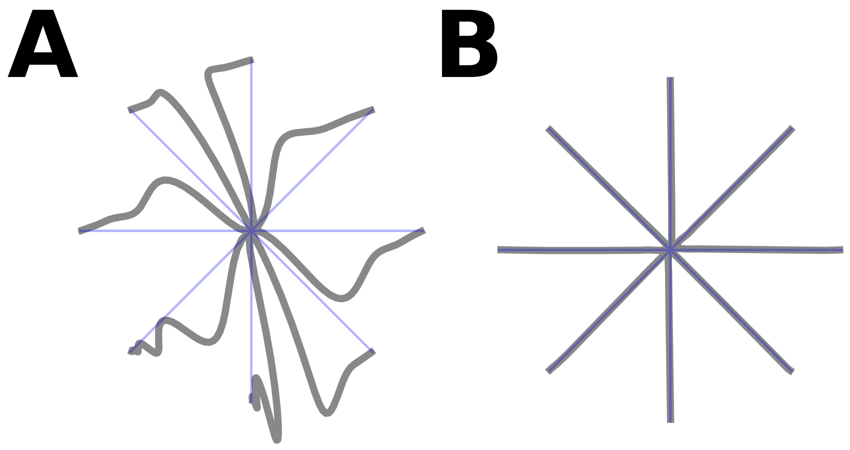


Figure 7.2: A diagram of reaching from a center point outward to eight equidistant targets with an end-effector velocity based force field applied. The desired path is the thin line, simulation reach is the thick line. A) Movement in a force field before adaptation. B) Simulation results immediately after adaptation with 1,000 neurons on the first trial.

1,000 rate based neurons. These results are from the first trial with the additive adaptation component applied, showing immediate error correction.

In Figure 7.3 the same task is shown while a perturbing force field is being applied based on the angular velocity of the joints. On the left is performance without any adaptive bias component. On the right is the same system, but with the adaptive bias component added in, consisting of 1,000 rate based neurons. These results are from the first trial with the additive adaptation applied, showing immediate error correction.

In Figure 7.4 the same task is performed with an inaccurate internal model of the inertia matrices of each of the arm segments. On the left is performance without any adaptive bias component. On the right we present the same system, but with the adaptive bias component added in, consisting of 20,000 rate based neurons. These results are from the first trial with the additive adaptation applied, showing immediate error compensation.

In all cases, when the adaptive bias component is employed, the system is able to immediately compensate for the errors in movement, introduced by either external forces or inaccurate internal models, and move to the targets in a straight or nearly straight lines.

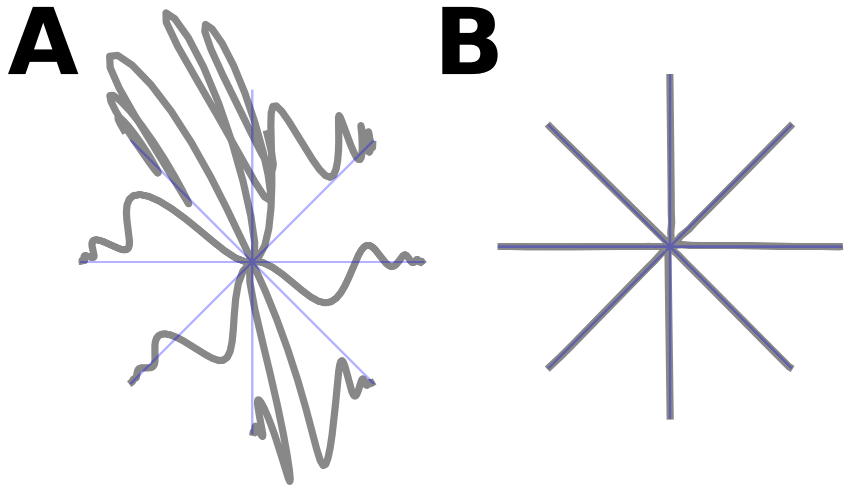


Figure 7.3: A diagram of reaching from a center point outward to eight equidistant targets with a joint velocity based force field applied. The desired path is the thin line, simulation reach is the thick line. A) Movement in a force field before adaptation. B) Simulation results immediately after adaptation with 1,000 neurons on the first trial.

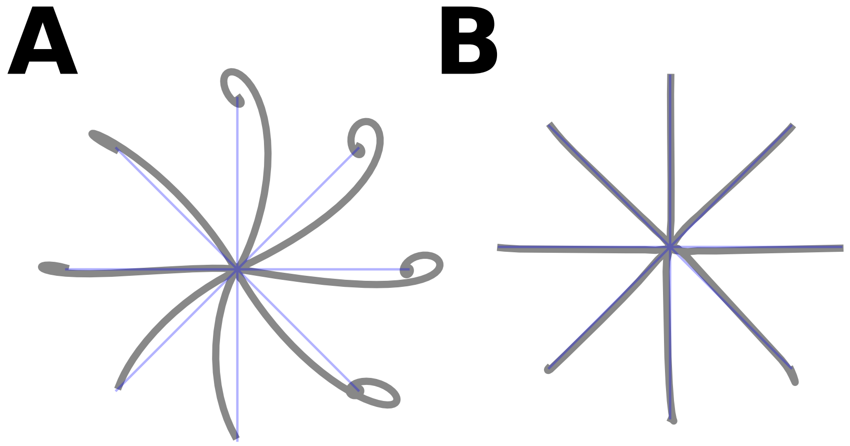


Figure 7.4: A diagram of reaching from a center point outward to eight equidistant targets with inaccurate inertia matrices for limb segments. The desired path is the thin line, simulation reach is the thick line. A) Movement in a force field before adaptation. B) Simulation results immediately after adaptation with 20,000 neurons on the first trial.

7.2.2 Transformation adaptation example

Figure 7.5 shows results from the application of transformation adaptation. Specifically, the arm segment lengths are all set to 3.0 for calculating the Jacobian, instead of their actual values of 2.0, 1.2, and .7. While the model repeatedly attempts to trace an ellipse, it correctly adapts to this change and eventually accurately reproduces the trajectory. The path of the hand as the system learned to move correctly is shown in the top portion of Figure 7.5. The root-squared error of the arm, measuring the difference from perfect tracing for each lap around the ellipse is shown on the bottom of Figure 7.5. As can be seen, the adaptive transform implemented in a population of neurons is able to quickly learn an approximation of the transformation between the spaces with the mass matrix included, bringing the system to an operational state, and then continually refine its approximation to further reduce error as practice continues.

7.3 Neural implementation

In this section the neural networks used to model the pre-motor cortex, primary motor cortex, and cerebellum are presented. These networks together form the whole of the REACH model implementation in spiking neurons.

7.3.1 The pre-motor cortex

The pre-motor cortex is modeled using the dynamical movement primitives (DMPs) system structure [159, 97] as a means of specifying desired trajectories in task space. A basic implementation of DMPs have been implemented in spiking neurons, capable of generating complex trajectories after a single demonstration of the desired path. This path is sent to the primary motor cortex where it is used to guide the arm using operational space control.

Figure 7.6 shows a diagram of the pre-motor cortex model. The PM is provided a 'start pulse' signal, which is a step function that transitions an oscillator from a default resting state into its limit cycle activity pattern. Figure 7.7 shows the step function, goal signals, and the activity of the oscillator and forcing function's ramp signal. While the start pulse function is active the goal states are the starting positions for the arm, driving the x and y populations to the initial end-effector position. The start pulse also inhibits output from the forcing function, allowing the x and y point attractor systems to converge quickly to their targets. These initial trajectory positions are projected from x and y into

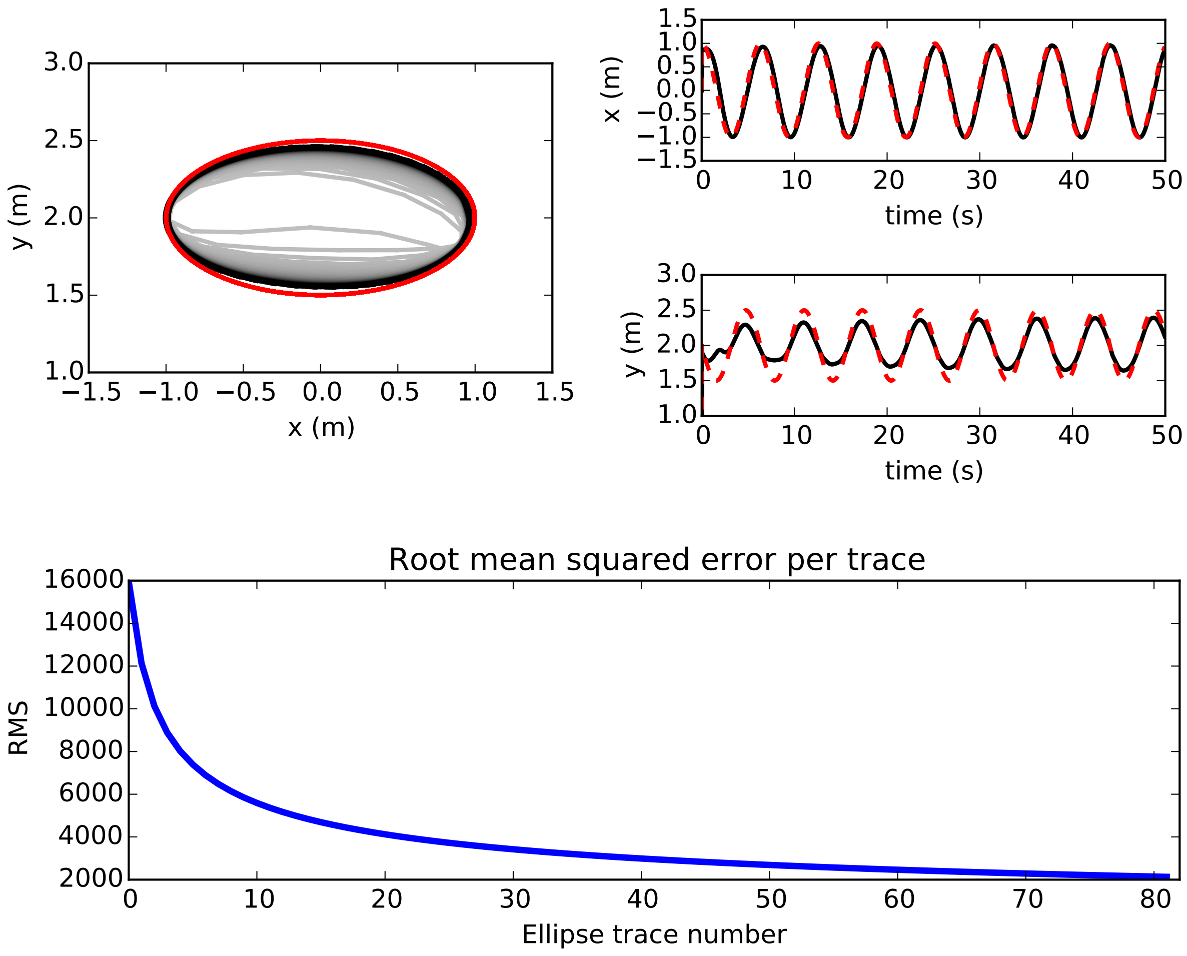


Figure 7.5: Top: The path traced out by the arm during correction of incorrect Jacobian kinematic parameters. On the left is the path as drawn out in Cartesian coordinates, where points farther back in time are lighter. The red ellipse is the target path. On the right the x and y dimensions of the trajectory are plotted against time in black, with the target values plotted in red. Bottom: A diagram of the root-squared error while learning to trace an ellipse using the adaptive transform component. The error is calculated as the summed root-squared distance from the target values during over each full trace of the ellipse.

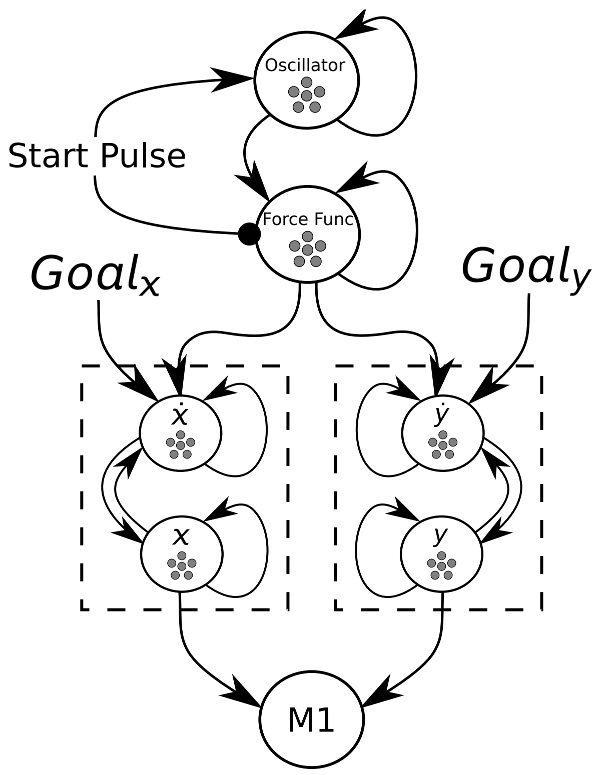


Figure 7.6: A model of the pre-motor cortex based on implementing dynamic movement primitives. In this diagram the two DMPs are for the x and y paths followed by a single trajectory. To generate different trajectories the system needs to specify a different output from the forcing function into $\dot{x}$ and $\dot{y}$. The solid arrow is a normal connection, the solid circle is inhibitory.

M1, which then moves the arm to the starting location for the trajectory. When the start pulse finishes the goal input to x and y will change and move the system towards the end-points of the trajectories.

At the same time, the oscillator is providing a high-frequency signal with a non-zero mean to the forcing function (as shown in Figure 7.7), which takes the value represented in the forcing function from 0 to 1. The reason for using an oscillator as opposed to just a constant signal is that the small fluctuations of the oscillating signal help to prevent the forcing function population from getting stuck in small pockets of representation (which have an effect similar to local minima) when integrating. While little difference is noticeable when the ramp integrates quickly to 1, the use of the oscillator becomes important when slowing down the speed of the trajectory, i.e. reducing the size of the input signal to the

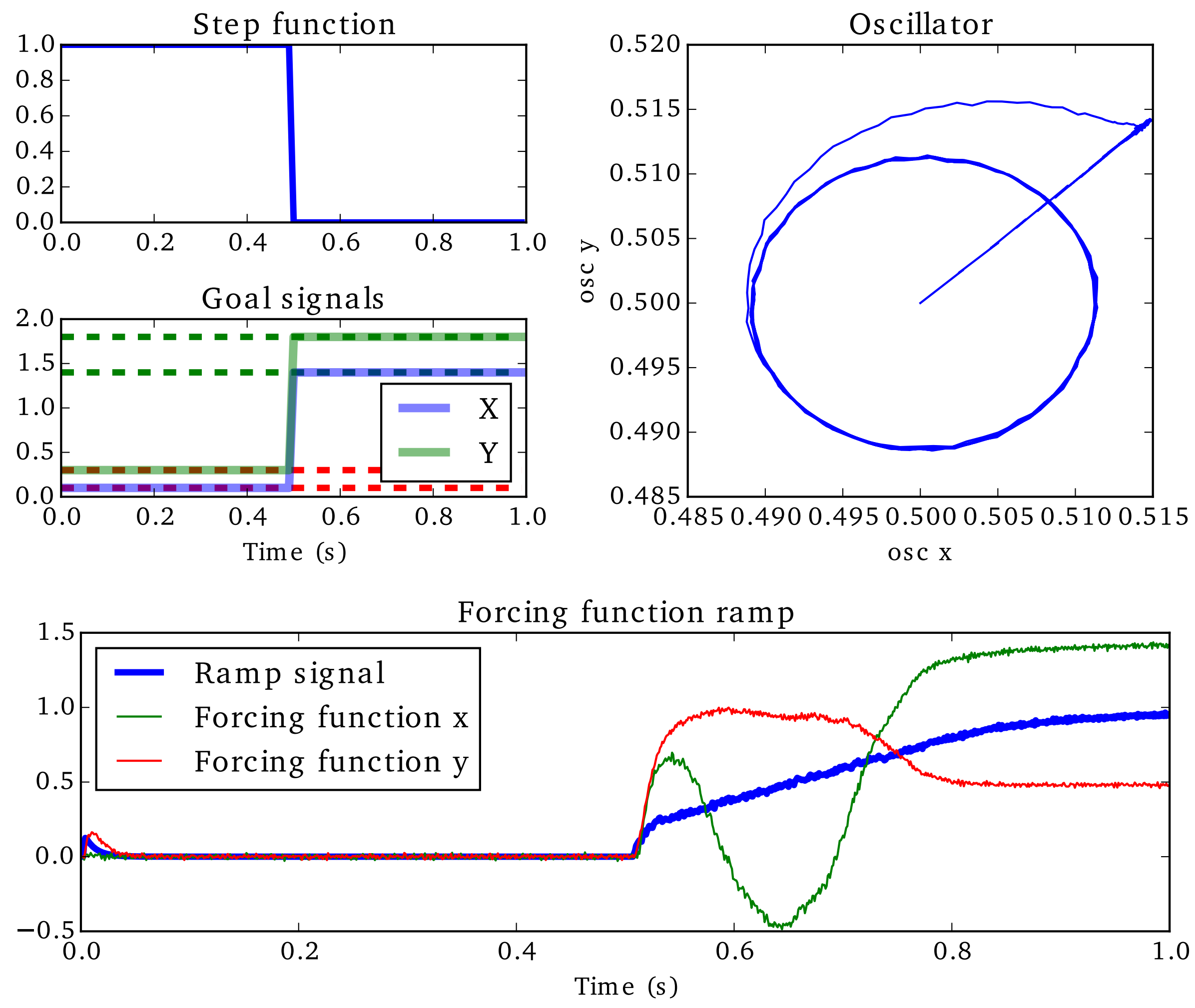


Figure 7.7: The step function (start pulse) and dependent signals. Goal signals: The input to the x and y point attractors, driving them to the initial points of the trajectory while the step function is active and to the goal points of the trajectory when the step function goes to zero. Oscillator: The step function pushes the oscillator from its resting state at $(.5, .5)$ to the top right of the plot. Once the step function goes to zero the oscillator enters its limit cycle activity pattern. Forcing function ramp: The step function inhibits this output; when the step function goes to zero the forcing function integrates one of the oscillator output dimensions. From this ramp signal both the x (green) and y (red) forcing function are decoded.

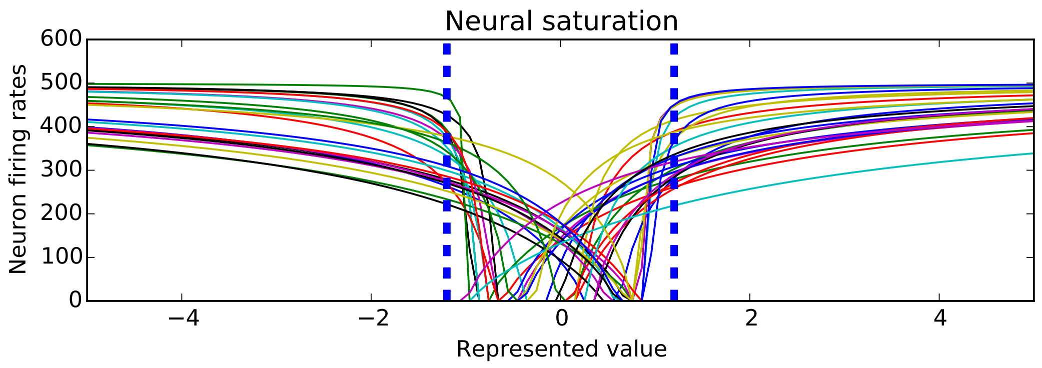


Figure 7.8: The firing rates of the neurons as the represented signal changes. Saturation of the neurons refers to a point (approximated by the dashed blue lines) where differences in the represented value do not cause sufficient change in the neural activity to accurately decode the signal. The decoded value increases to the point of saturation and then grows no further, even with a constantly growing input.

forcing function population. As the value in the forcing function population moves from 0 to 1, a function is decoded from the neural activity that is projected into the $\dot{x}$ and $\dot{y}$ populations, altering their dynamics of x and y. As a result, the forcing function moves the system state to the target along the expected trajectory.

Saturation of the neurons is used to ensure that the ramping signal stops at 1 and doesn't continue growing forever. Intuitively, at some point all of the neurons will either be firing at their maximum rate or not at all, as shown in Figure 7.8. By setting up the decoders such that this point represents 1, it can be guaranteed that 1 is the maximum value possible to be represented by this population. This is referred to as neural saturation.

There has been much work extending the DMP architecture [97], allowing the system to generalize both spatially and temporally, incorporate system feedback, and execute independently or couple multiple DMPs to each other or other signals from the environment. I have implemented a basic model of the full DMP architecture in spiking neurons here, which lays the groundwork for the development of neural models that include such extensions.

7.3.2 The primary motor cortex

The primary motor cortex (M1) is used to map high-level control signals defined in an abstract space to a low-level control signal that can be sent to the body; in this implementation end-effector forces are mapped to joint torques. It accounts for some of the effects of inertia on the system and helps to cancel them out, and generates a secondary control signal in the null space of the primary control signal which serves to keep the system near a comfortable 'resting state', or default position. In addition to these functions it also learns to correct for inaccuracies in the Jacobian, minimizing performance errors using only the high-level control signal $\mathbf{u_x}$ and system velocity $\dot{\mathbf{q}}$ as training signals.

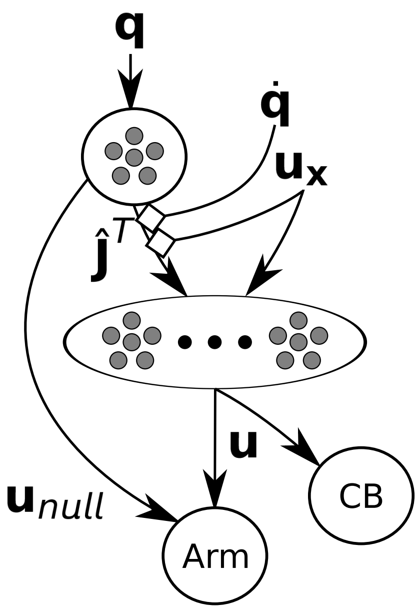


Figure 7.9: A model of the primary motor cortex (M1). From a population representing the system state, the adaptive Jacobian is projected into a population that performs a dot product with the high-level control signal, $\mathbf{u_x}$. At the same time, $\mathbf{u_x}$, along with $\dot{\mathbf{q}}$, is used to adapt the Jacobian, $\hat{\mathbf{J}}^T$, which is projected into the dot product population. The result of this dot product is the low level control signal $\mathbf{u}$, which is projected both to the cerebellum (CB) as a teaching signal, and the arm as a control signal. The secondary control signal, $\mathbf{u}_{null}$ which keeps the system close to it's resting position during movement, is also projected to the arm as well. The solid arrow is a normal connection, the hollow diamond is a modulatory connection used for learning.

The neural network implementation of these computations can be broken down into two parts. The first is a population representing the system state, from which the adaptive Jacobian, $\hat{\mathbf{J}}^T$ is decoded. Note that the adaptive Jacobian also includes the inertia matrix $\mathbf{M}_\mathbf{x}$. The connection along which the Jacobian is generated is subject to adaptation, using the high-level control signal $\mathbf{u}_\mathbf{x}$ and system velocity $\dot{\mathbf{q}}$ as training signals, implementing the learning algorithm described in Chapter 4. This Jacobian is projected into an array of ensembles along with the high-level control signal $\mathbf{u}_\mathbf{x}$, allowing the ensembles to perform the dot-product operation, as described in Eq. 7.3. As shown in Figure 7.9, the resulting low-level control signal, $\mathbf{u}$, is sent to the cerebellum as a training signal and to the arm along with the secondary control signal $\mathbf{u}_{null}$ to implement Eq. 7.3.

7.3.3 The cerebellum

The cerebellum provides a corrective signal to the arm that cancels out the effects of inertia and learns to account for and minimize errors due to unmodeled dynamics that arise during movement.

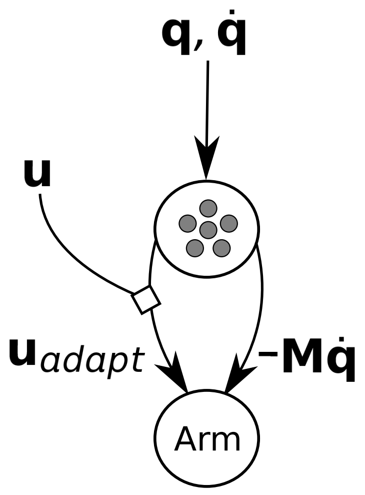


Figure 7.10: A model of the cerebellum. System state information arrives from the sensory cortices and spinal cord, and is then used to generate corrective control signals sent to the arm. $\mathbf{M}\dot{\mathbf{q}}$ accounts for and cancels out the effects of inertia on the system, and $\mathbf{u}_{adapt}$ is an adaptive signal that corrects for unmodeled dynamics. The teaching signal, $\mathbf{u}$, comes from M1. The solid arrow is a normal connection, the hollow diamond is a modulatory connection used for learning.

To do this, the cerebellum calculates the forces due to the current motions of the system, $\mathbf{M}\dot{\mathbf{q}}$ and projects a compensatory signal to the arm. At the same time, an adaptive signal is also projected to the arm, $\mathbf{u}_{adapt}$, which learns to correct for unmodeled forces using the low-level control signal as a training signal, through the algorithms described in Section 4.3.

Although the model presented here is in spiking neurons and functionally detailed, there is a wealth of neuroanatomical details of the cerebellar structure that are not included in the REACH model [164]. Important future work will be to determine how the known anatomical structure of the cerebellum can be exploited to perform the proposed computations.

7.4 Neural implementation results

The combination of the pre-motor cortex, primary motor cortex, and cerebellar neural models provides a spiking neuron implementation of the REACH model. By selecting a given trajectory to follow or providing a target location the model can generate appropriate control signals to move a three-link non-linear arm. In this sections results are presented from simulations carried out to replicate experimental studies, and demonstrate the overall performance of the model.

7.4.1 Reaching and handwriting

As a standard behavioral test of the model, it is perform to control 8 center-out reaches. Figure 7.11 shows the resulting arm trajectories and velocity profiles from the model compared to human data collected by [168]. To account for the force response profiles of muscles (which are not included in the arm model), a simple model (i.e., a skewed Gaussian response) has been used to mimic the effects of the force response profiles of muscles [127]. The velocity profiles match with a correlation of .96 in the mean and .703 in the variance.

To generate more complex trajectories, DMPs that have been trained to output specific (x, y) Cartesian coordinates for the system to follow are activated. Figure 7.12 demonstrates the ability of the model to move the arm through a specific set of complex trajectories. In this case, the digits from 0-9 and the words "hello world" were learned from a single presentation of the author's handwriting. This kind of learning is very rapid because we directly optimize neural connection weights in PM to generate the desired paths. However, the model also exhibits slow adaptation based on observation of its own errors during performance.

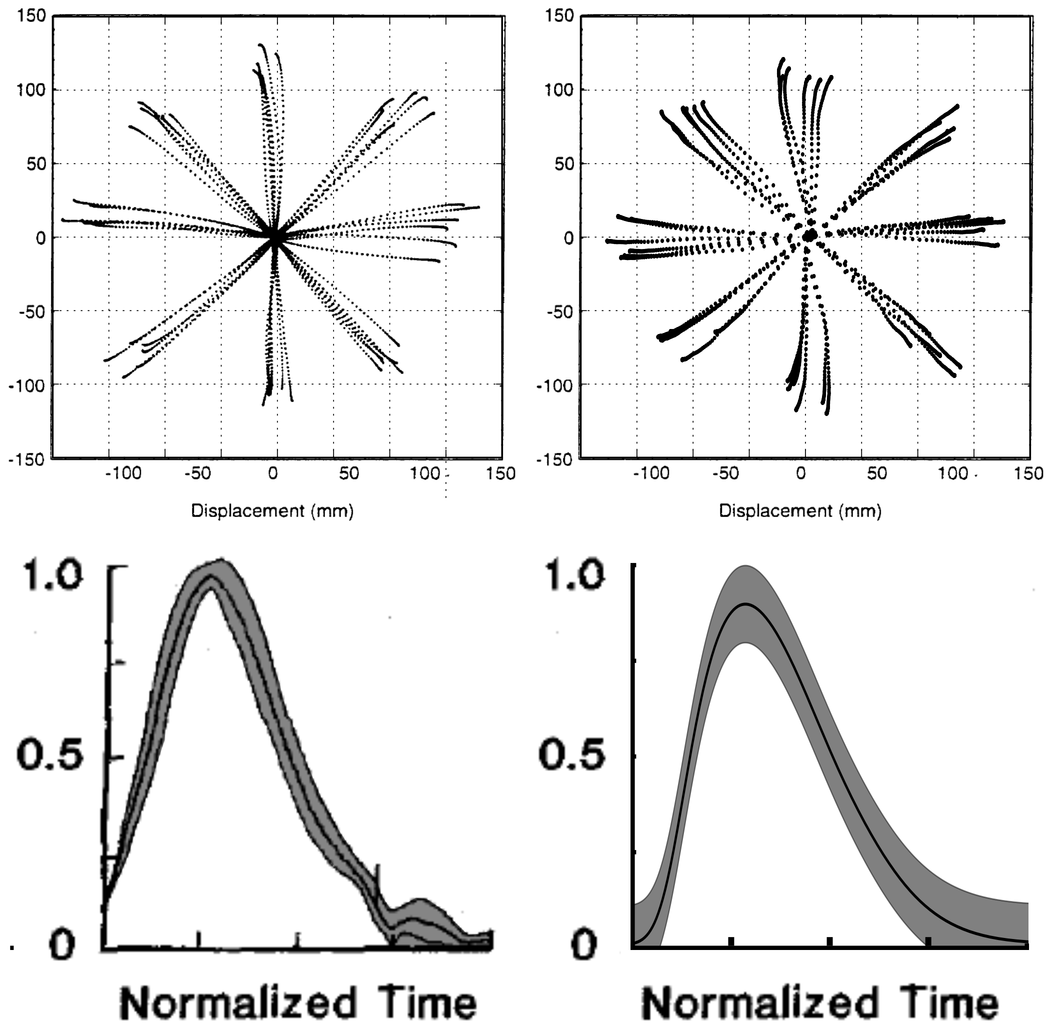


Figure 7.11: Top: Normal reaching from a center point outward to eight equidistant targets, repeated five times. Human data, taken from [168], is on the left and model simulation results are presented on the right. Bottom: The mean (thin center line) and variance (thick grey line) of the velocity profiles for each of the eight reaches in the above figure. Human data, adapted from [173], is displayed on the left, and model simulation results are presented on the right. A simple model of muscle responses has been applied to account for the effects of muscle activation profiles, by convolving the velocity signal with a skew-symmetric Gaussian.

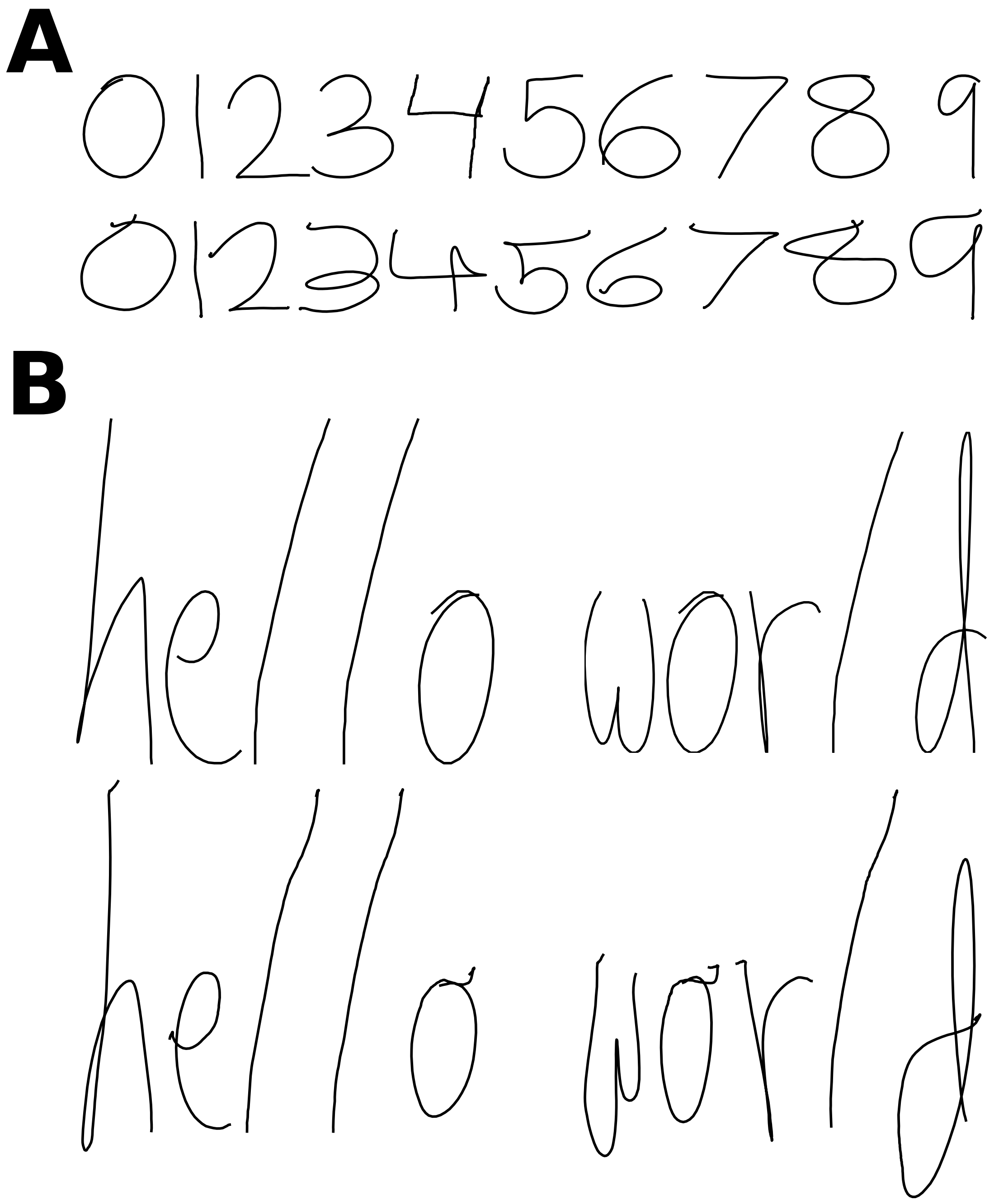


Figure 7.12: A sample of one of the author's handwriting (which was used to train the model) compared with the model' handwriting. These trajectories were learned from single example presentations. A) Writing out the numbers 0-9, the author's handwriting is above, the model's below. B) Writing out the phrase 'hello world'; the author's handwriting above, the model's below.

The same adaptation as implemented in Section 7.2 also works for the fully neural system, as shown in Figure 7.13 and Figure 7.14.

In Figure 7.13 the experimental paradigm used in [168] was simulated. A reaching task was set up where the model performed 8 centre-out reaches to targets approximately 10 centimetres away. While reaching, a force was applied to the hand that was proportional to the velocity of the hand, as in Section 7.2. On the left side of Figure 7.13 the experimental results can be seen, collected from human patients learning to reach with a force-field applied. While the deviations from the normal trajectory are qualitatively similar (i.e. the human and simulated arm are both pushed in the same directions) they are notably larger in the human trajectories. This is likely due to the minimal feedback delay in the REACH model, whereas humans require at least 80-100ms before visual or proprioceptive correction is possible [101]. Extensions to incorporate such feedback delays and compensatory control methods is discussed at the end of chapter.

In the experimental results presented by [168], the subjects were shown the target, the target disappeared, and then the subjects were asked to reach to where the target had been. To simulate the error introduced from reaching without visual feedback, noise was added to the targets presented to the model, providing a similar distribution of movement end-points as seen in human trials.

Figure 7.14 shows the REACH model instantiated with an incorrect Jacobian matrix, calculated as though the arm segment lengths were all 3 feet, rather than 2, 1.2, and .7 feet, just as in Section 7.2. Once again, the system is able to quickly correct the control error such that an ellipse can be traced appropriately. The root squared tracking error averaged over 2 ellipse traces reduces from 272.7m over the first two traces to 64.6m after 30 simulated seconds of learning.

An example showing both the adaptive transform and bias being applied at the same time is shown in Figure 7.15 for both the basic reaching and the force field reaching tasks. Learning both kinematic and dynamic parameters at the same time is only possible for point-to-point movements, such as the target reaching shown in Figure 7.15. Implementing both types of learning for tracking control, such as for the ellipse tracing, requires further development of the nonlinear control methods used, and is discussed at the end of this chapter.

7.4.2 Neural comparisons

The comparisons made to this point, while reflecting underlying neural performance, are purely behavioral. A major benefit of generating a model at the level of individual neurons

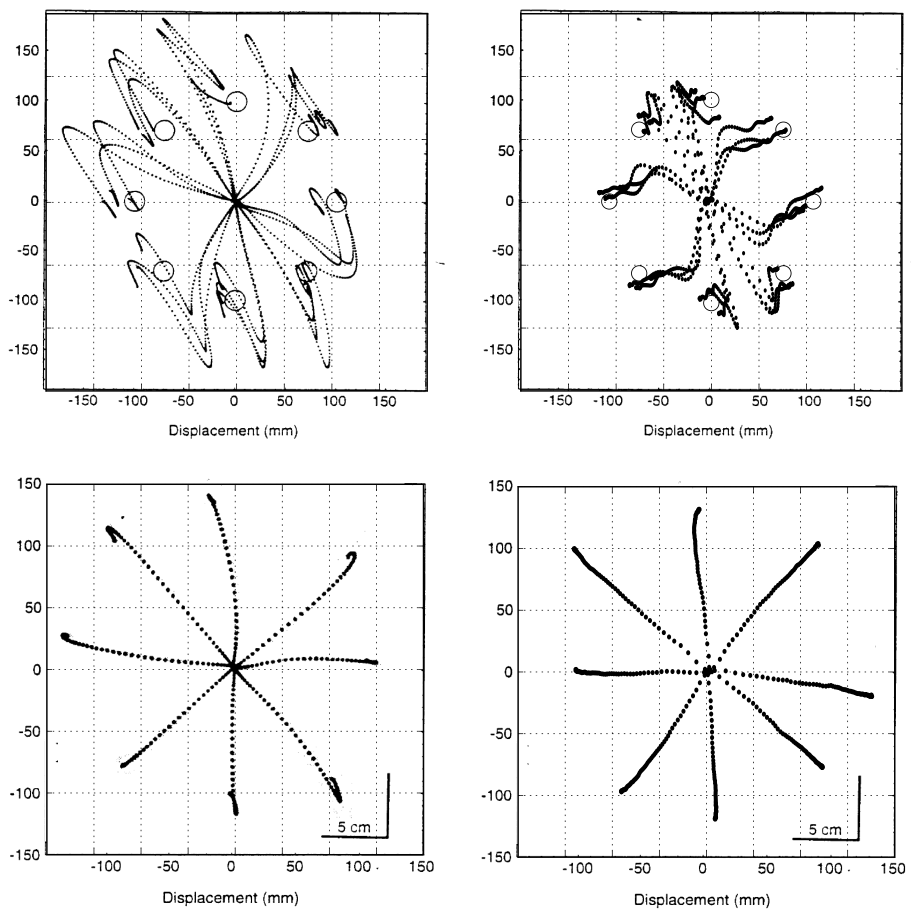


Figure 7.13: Reaching from a center point outward to eight equidistant targets with an end-effector velocity based force field applied. Human subjects required approximately 750 trials to reach an average of 0.9 correlation with the original reach trajectories. The model's adaptation was significantly faster, requiring less than two full trials to reach 0.9 correlation with the original reach trajectories. Top: Movement with the force field applied, before adaptation. Human data, taken from [168], are shown on the left, model simulation results are shown on the right. Bottom: Movement with the force field applied, after adaptation. Human data, taken from [168], are shown on the left, and model simulation results are shown on the right.

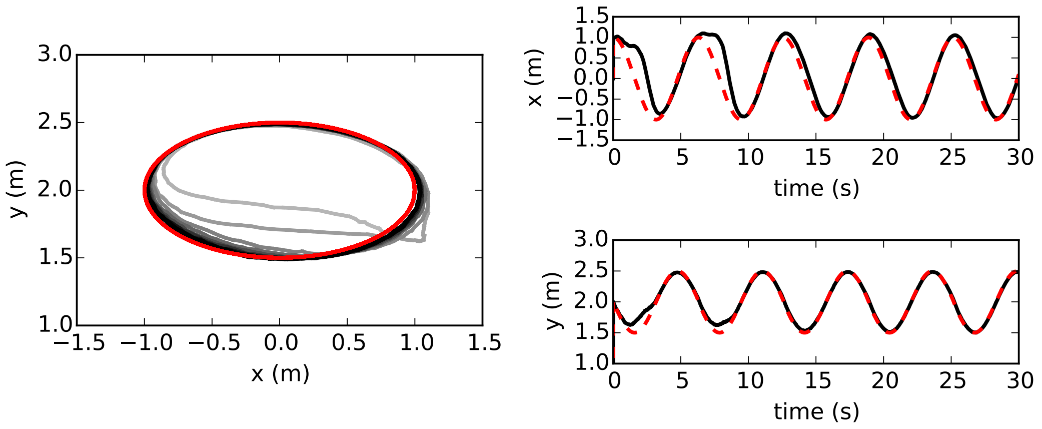
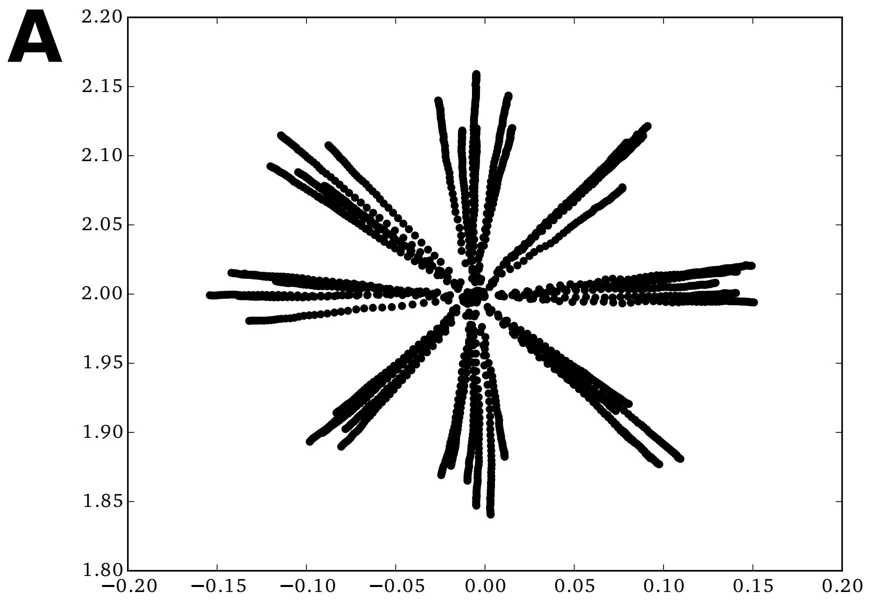


Figure 7.14: The arm tracing a circle starting with an incorrect internal model of the system dynamics. Over time the system learns to correct this model by adapting the Jacobian. Initial average tracking error across two cycles: 272.7 m, final average tracking error across two cycles: 64.6 m.

is that it allows detailed comparison with a large body of experimental evidence from neuroscientific studies. Numerous experimental studies have examined the neural activity of cells in the motor cortices and correlated their activity with various movement parameters (see Table 7.1). Such studies provide insight into the information represented in different brain areas and can help identify likely transformations used by robust natural controllers.

7.4.3 Neural activity correlation with movement parameters

As one classic example of the correlations observed between neurons in M1 and movement parameters, [66] reports that neurons exhibit what is widely known as a "preferred direction" of arm movement in the plane. Figure 7.16 shows a comparison of the spiking activity of a single cell as the arm moves in 8 directions from experimental data and from a neuron with a similar preferred direction in the REACH model. In both cells, a strong preferred direction is visible, shown in the cells' increased activity in reaches to the left. The exhibition of preferred direction is also present predominantly during movement in both cells, where activity either increases or decreases during the movement period and maintains a baseline firing rate both before and after.



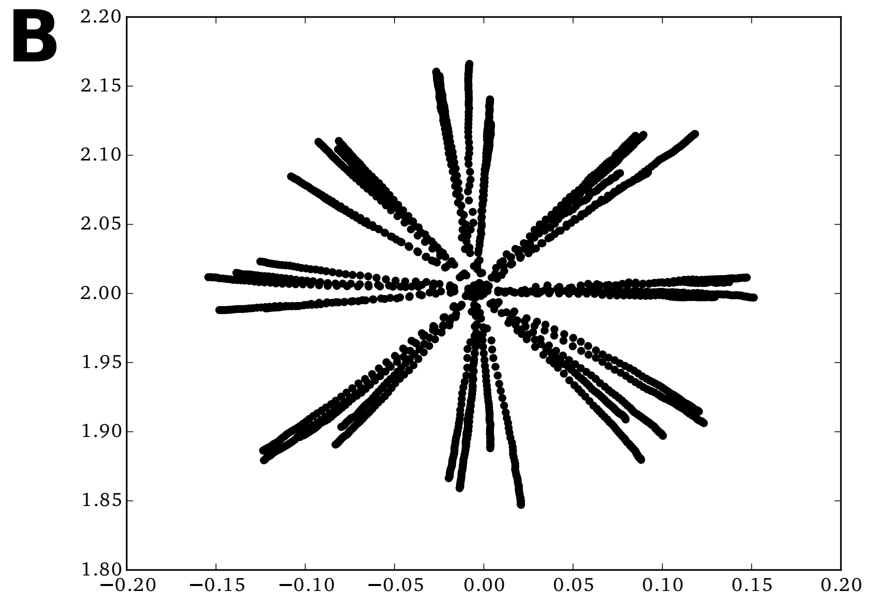
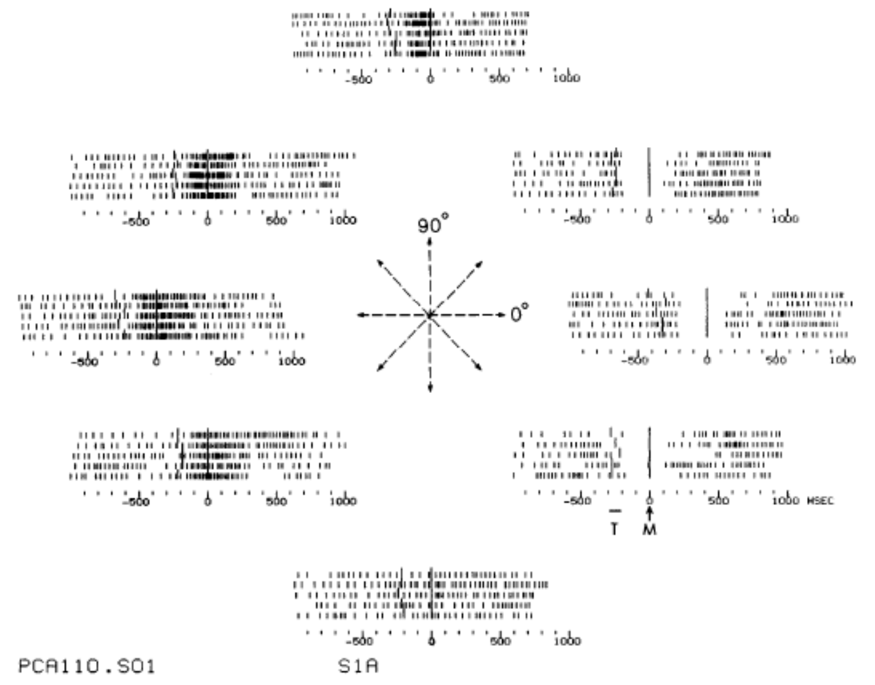


Figure 7.15: The arm performing 8 centre-out reaches with both bias and transformation adaptation applied. A) Normal reaching. B) Reaching under a force-field post adaptation.



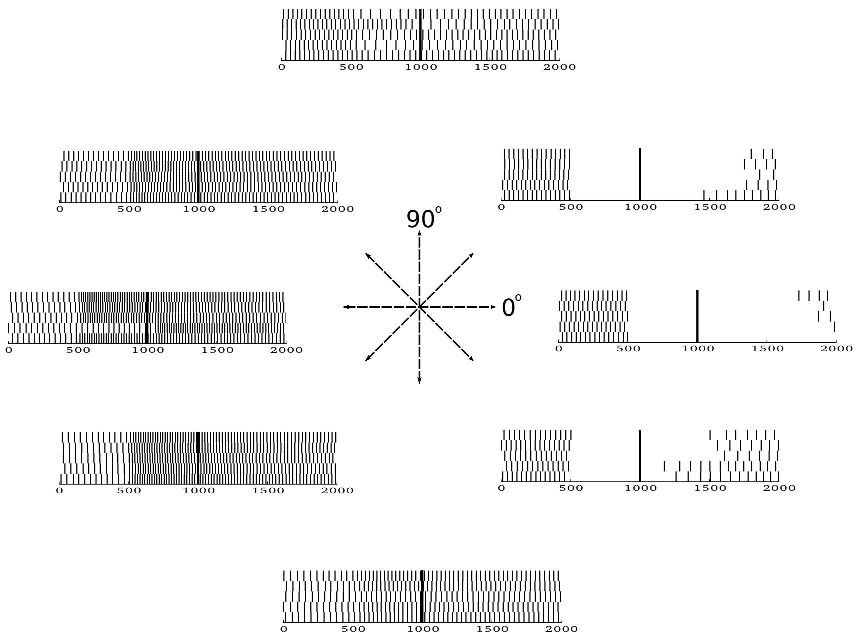


Figure 7.16: Examining a neuron's activity profile in response to the arm reaching out to 8 surround-center targets over 5 trials. Top: original results from [66]. Bottom: model results. In both cases a clear preferred direction for the neuron can be observed.

Given the structure of the REACH model, and its realization in spiking neurons, it can be demonstrated that the REACH model accounts for the same correlations with movement parameters identified in experimental research, listed in Table 7.1. Figures 7.17 and 7.18 show the correlations from the primary motor cortex and cerebellum as examples. To generate those figures, the actual and target hand positions are provided to the primary motor cortex as shown in Figure 7.1, as they are required for the computation of the task-space control signal and Jacobian matrix. Figure 7.17 shows that neurons in the M1 area of the model correlate with acceleration, arm position, distance to target, hand position, movement direction, movement magnitude, and movement velocity. The premotor cortex is given a signal indicating which action to carry out as well as system position information which it requires to generate an error signal and modulate the speed of movement execution. Consequently, neurons in this area of the model correlate with distance to target, hand position, and movement direction. Finally, the cerebellum is provided with position and velocity information, as well as an efferent copy of the motor signal, which is necessary for dynamics compensation and adaptive error correction. The representation of these signals means that neurons in this area in the model will correlate with arm position and movement direction, magnitude, and velocity. These correlations, as well as others are as shown in Figure 7.18.

While these neural correlations with movement parameters and preferred direction similarities are expected as a result of the network structure and implementation with the NEF, they provide support for the functions being performed in each of the modeled brain areas. That is to say that each of the variables giving rise to the correlations observed in the REACH model are a requirement of the algorithms, and finding the same correlations in experimental work provides evidence for this information being available to the modeled brain areas, suggesting that the functions implemented by the REACH model are biologically plausible.

7.4.4 Analysis of neural activity with jPCA

More recent work on the motor system has suggested that a simplistic view of neural activity as correlated with basic movement parameters, like movement direction or velocity, does not adequately capture neural response properties, especially dynamic ones [30]. To demonstrate this claim, a novel variation on principal components analysis called ‘jPCA’ was presented in [30] and applied to single-cell neural recordings from monkeys made during a similar reach task.

This method was devised as a population measure that characterized the changes of neural activity during a reach movement. Briefly, in jPCA the neural data is projected into

Movement parameter	**Model representation**	**References**
Acceleration	system state	M1 [56, 7]
Arm position	system state	CB [156] M1 [111, 166]
Distance to target	control signal	{M1, PM} [60]
Hand position	system state	M1 [7], PM [106]
Movement direction	control signal	CB [58, 156], M1 [66, 7], PM [26]
Movement force	control signal	M1 [106, 167]
Movement magnitude	control signal	CB [58], M1 [135]
Movement velocity	system state	CB [156] M1 [4]

Table 7.1: A table summarizing the correlations with movement parameters found in neural recordings from different areas of the motor system. The model representation identifies the signal in the model that carries information about the given movement parameter. CB: cerebellum; M1: primary motor cortex; PM: pre-motor cortex.

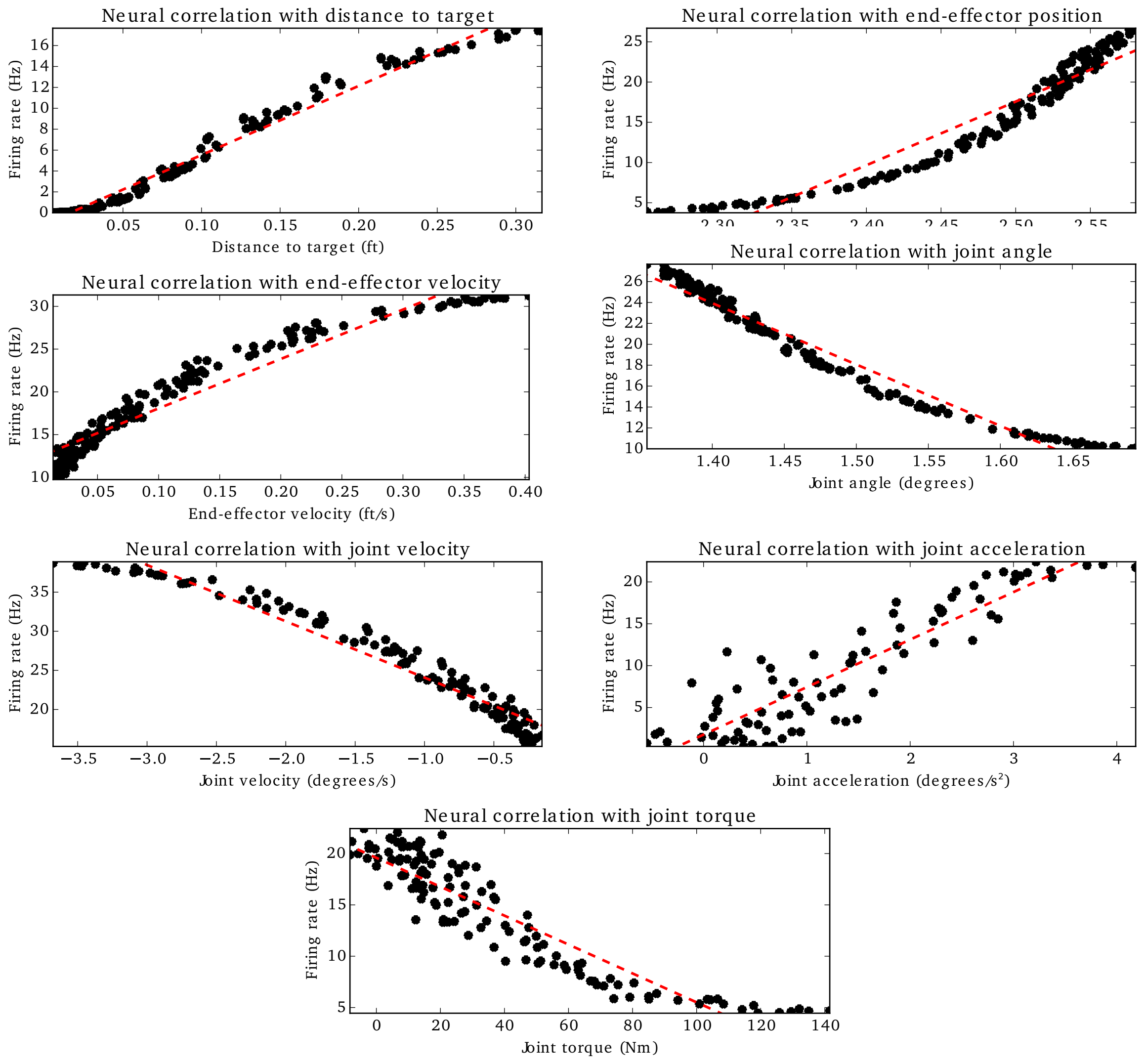


Figure 7.17: Plots of the neurons from the primary motor cortex correlating with various high-level movement parameters, reflecting the same correlations found in experimental research as detailed in Table 7.1.

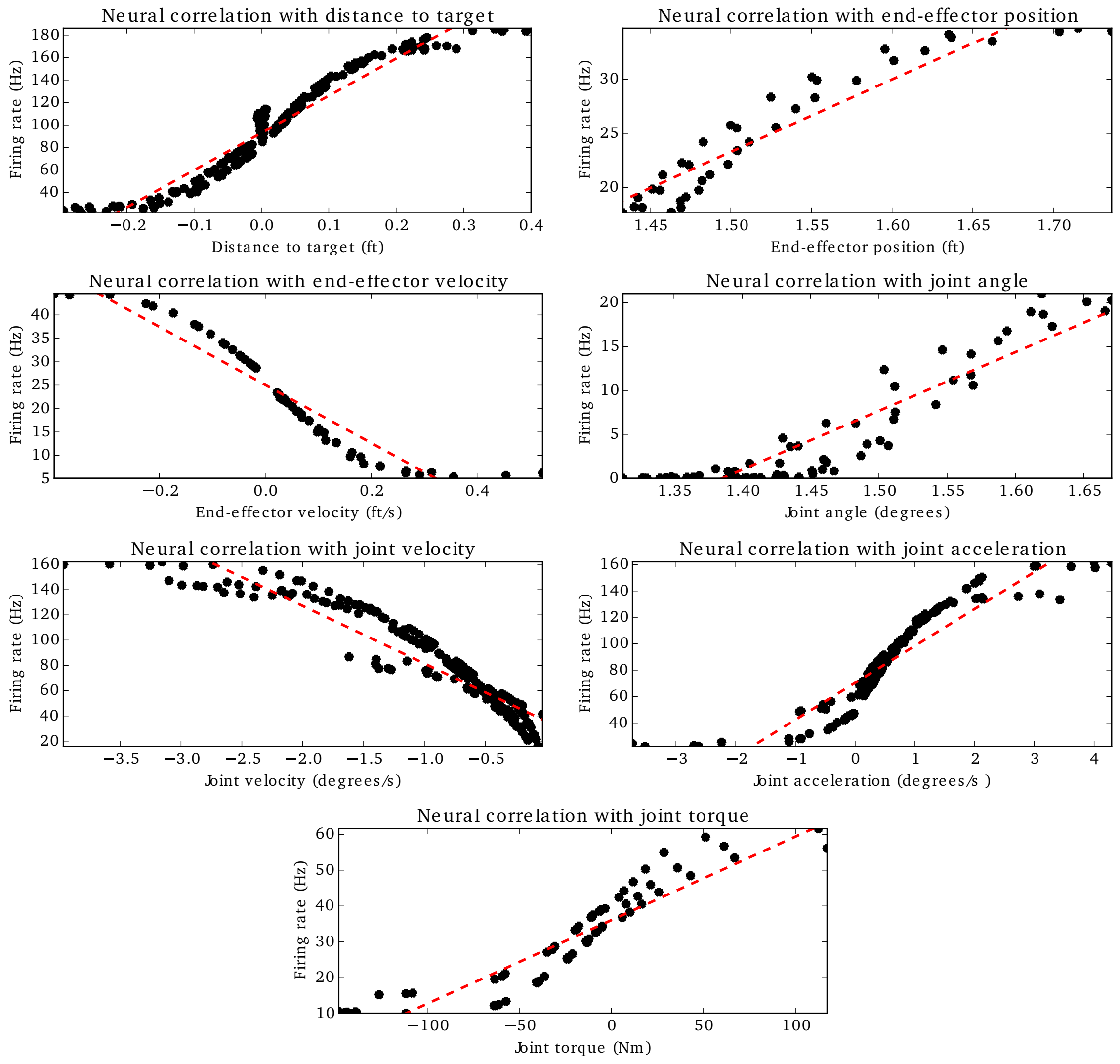
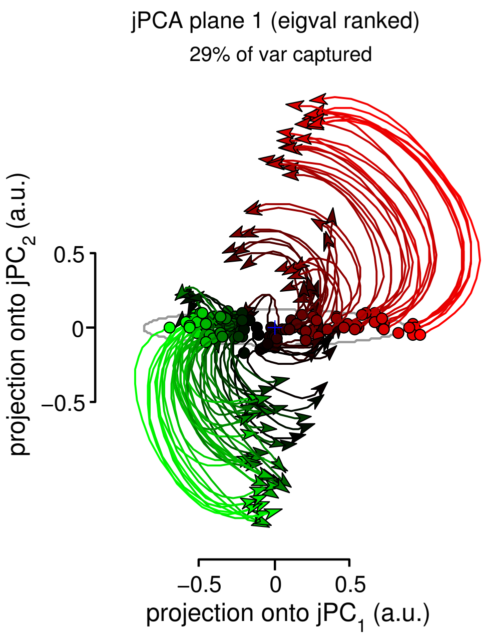


Figure 7.18: Plots of the neurons from the cerebellum correlating with various high-level movement parameters, reflecting the same correlations found in experimental research as detailed in Table 7.1.



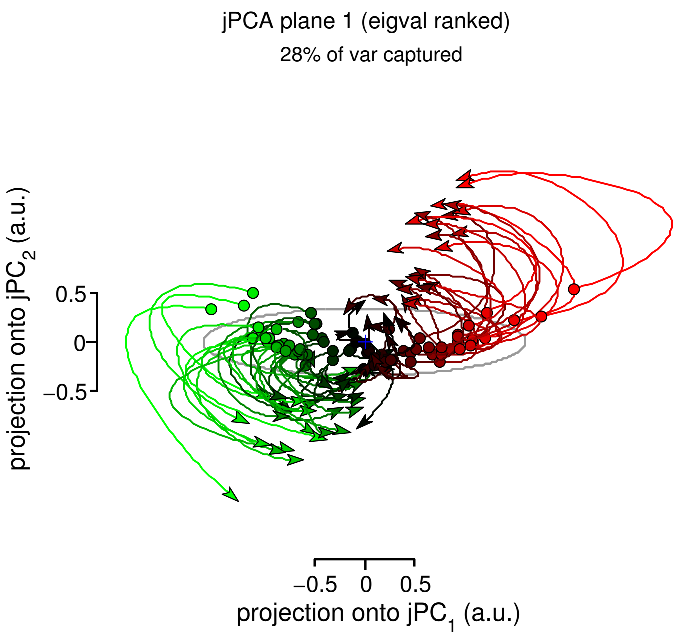


Figure 7.19: jPCA analysis applied to reach data generated from monkeys and the model. Results from the original analysis is on the left, and from model analysis performed on randomly chosen neurons from the primary motor cortex on the right. The color is determined by the starting position of the neural activity. Both analyses exhibit a similar rotating path through the low-dimensional state space over time.

the space of the top two principle components of the population activity. This provides a low-dimensional representation of the neural activity during each trial. A matrix that best captures the dynamics of the low-dimensional trajectory over time is calculated, with the constraint that the matrix must be skew-symmetric. The skew-symmetry forces the dynamics to be captured in terms of a set of oscillators. The principle components of this skew-symmetric matrix are found, and the low-dimensional representation is projected into this rotation-centric space.

The original authors note that even during non-rhythmic, discrete actions, such as the monkey reaches, rotational trajectories were evident. Figure 7.19 shows a comparison of a jPCA analysis performed on the spiking data from our model and the data from [30]. This figure demonstrates that the same kinds of dynamics captured by this analysis are found in the model neural activity as in the empirical data.

7.5 Discussion

In this chapter I have presented the Recurrent Error-driven Adaptive Control Hierarchy (REACH) model; a large-scale spiking neuron model of the motor cortices and cerebellum in the motor control system. The REACH model is capable of accounting for behavioural and single-cell data. The REACH model is the first to provide this broad of a range of empirical match; there are no other models of the motor control system at this scale able to match both behavioural and experimental neuroscience data that we are aware of.

7.5.1 Predictions

As a result of its scope, the REACH model is able to help make a number of testable predictions. One such prediction regards the presence of and location of a secondary controller that keeps the system near a default resting state. In the REACH model this signal comes from the primary motor cortex, but another likely location is the cerebellum. The prediction is that given patients with cerebellar damage who are still able to perform directed reaching movements, their ability to correct for dimensions of the movement not related to completing the task should not be compromised. For example, a patient should be able to start a reach with their elbow in a position away from the normal resting state and move the elbow back towards resting space during the reach without compromising task-space performance.

Another prediction can be generated by examining the division of signals coming from the cerebellum in the REACH model. The nonlinear adaptive bias signal, $\mathbf{u}_{adapt}$, maps to the error correction functionality coming from the intermediate cerebellum (part of the spino-cerebellar system), and the inertia matrix weighted velocity compensation signal, $\mathbf{M}\dot{\mathbf{q}}$, maps to the context sensitive corrective signal received from the lateral cerebellum (part of the cerebro-cerebellar system), where internal models of the body are stored. With these functional assignments made, damage to these signals can be introduced and compared to clinical patients with cerebellar damage during reaching tasks. In the case of lateral cerebellar damage it would be assumed that visual feedback and slow movement will be of central importance for precise reaches, because the internal model of inertia and velocity compensation will have been lost; this would be especially apparent in patients attempting ballistic movements. In the case of intermediate cerebellar damage it would be expected that while errors in movement (such as those introduced by a forcefield) can be observed, the ability to learn to correct them automatically given a context where they consistently appear will be severely impaired. It would be expected that some correction would still be possible, as the trajectory itself could be altered, but it is hypothesized that this change in the trajectory (i.e. the DMP generated signal) would be a much slower change, and result in much longer after-effects when the force-field is removed.

Figure 7.18 presents the correlations of the REACH cerebellum with several high-level movement parameters. Several parameter correlations not found in the literature review but present in the REACH model are the correlations with end-effector position, velocity. As such, another prediction is that these correlations will be found in the cerebellum during reaching tasks.

In the primary motor cortex the approximation of the Jacobian is modified using the high-level control and system velocity signals. This is the only place in the primary motor cortex model that the system velocity signal is used. A prediction that can be made as a result of this structure is that the system velocity signal should only be detectable in the primary motor cortex while the system is learning kinematic parameters. The form of the kinematic adaptation is such that there will be learning whenever $\tilde{\mathbf{x}} = \mathbf{x} - \mathbf{x}_{\text{des}} \neq 0$, i.e. whenever the hand is not at the desired position. When the hand is at the desired position, even when moving, such as in the case of tracing a path, the learning signal will be zero and the REACH model predicts that a system velocity signal will not be present in the primary motor cortex. Being able to successfully detect the presence of a system velocity signal explicitly, and not conflate with neural activity from a high-level control signal, though, will require a very careful experimental setup.

More easily tested, the REACH model also predicts that in the primary motor cortex there will be neurons that respond to high-level control signal, system velocity, and system position. These neurons are necessary for the transformation between high- and low-level control signals. Following upon this prediction, if an experiment were able to successfully identify all of the neurons sensitive to these signals for a given limb (e.g. all neurons sensitive to the high-level control signal and right arm velocity and position) and lesion them, it is predicted that any operational space control of the limb would be destroyed.

In the pre-motor cortex, it is predicted that there will be neurons that fire at a constant rate throughout a practiced movement, reflecting the oscillator signal that is integrated by the forcing function ramp population, as detailed in Section 7.3.1. These neurons should also be insensitive to any parameters of movement aside from possibly the temporal scaling of the movement. This is because these neurons are not a function of any system variable, they serve to hold a constant value that can be steadily integrated to move the learned trajectory forward through time.

7.5.2 Future work

The REACH model is a far from complete model of the motor control system. There are several possible improvements, ranging from including methods described earlier, but not in the neural model, to exploring control methods not yet considered in detail.

In the DMP neural network implementation the generalizability of trajectories is not currently implemented in the neural model, despite being a strength of DMPs. This was left unimplemented due to time constraints. Additionally, learning and adaptation of DMP trajectories was not explored in the neural model. This adaptation is is different than that considered for the controller itself. Instead it is a kind of reinforcement learning, commonly associated with the basal ganglia, and currently outside of the scope of the REACH model. Recent work in hierarchical learning has met success with NEF implementations using a basal model [153], and could possibly serve as a basis for integrating reinforcement learning with the model presented here. Other proposed trajectory learning methods, e.g. the PI^2 algorithm [185] are based on batch learning and a biologically plausible analog has not yet been presented. It is possible that stochastic gradient descent might be an effective and simple method for learning trajectories, but this requires exploration. A further desirable extension to the neural DMP model is an obstacle avoidance system, where an outside signal would provide the obstacle boundaries and the system would adjust any path being taken to avoid the identified objects.

As noted, there are also methods of nonlinear adaptive control, such as the tracking control [29] methods for adapting the Jacobian which have not been thoroughly explored here. Although not shown, the stability proof for non-autonomous systems with bias adaptation to account for unknown system dynamics is essentially the same as that shown in Chapter 4, employing Barbalat's lemma instead of the invariant set theorem. However, a different proof is required for guaranteeing the stability of non-autonomous systems while using both transformation and bias adaptation. This form of adaptive tracking control was omitted due to time constraints, as well as the complexity and hence potentially biologically implausible calculations used in the original formulation. In order to implement the methods neurally, it will likely have to be reworked extensively.

A further significant extension to the model would be the development of the sensory cortices. Currently the feedback signals required in each of the brain areas modeled are directly provided. A more complete model would have the sensory cortices receiving the noisy low-level system feedback and transforming it as required. Additionally this signal should include visual input, as well as a predictive signal developed using the same nonlinear adaptation techniques presented in Chapter 6. In other work, we have shown that the same methods can be used for the development of a nonlinear predictive filter because the prediction problem is a dual of the control problem. The development of such a system will be especially useful in situations such as the control of a real-world system, where feedback and control delays require a predictive model for generating ballistic movements.

An important practical extension to the REACH model is to use it to control a physical robotic arm. Working with Stanford's Brains in Silicon lab, where the NeuroArm [130] discussed in Chapter 5 was developed, I have constructed a prototype of the NeuroArm for our lab. Due to time constraints I was not able to interface the REACH model with the NeuroArm, but it is important future work to be able to apply the neural implementation of nonlinear adaptive control to physical systems with more complex dynamics than simulations, and apply methods for effectively handling real-world constraints such as time-delay and noise and other unmodelled effects. In the pursuit of this extension the delay of feedback signals will also have to be addressed, examining how well the REACH model operates under such conditions. The use of the internal models in the cerebellum, as a means of feed-forward control to compensate for predictable errors, will play a large role in effective control of movements that are too fast to be corrected by feedback signals [164].

Additionally, the brain is capable of learning to control effectively in extreme situations where the effects of activation of different motor neurons or spinal circuits are unknown. An avenue for future work is to look at how well the methods described here can work to control in such situations. Humans have a very large developmental period in early life, and it will be very interesting to see the scale of time required for the REACH model to learn to control from a ground knowledge of zero.

7.5.3 Comparison with the NOCH

As mentioned previously, the REACH model is an extension and update to the NOCH framework. Where there is an agreement of overall function assignment and flow of information between the two, there are also a number of important differences. In both models, the pre-motor cortices are responsible for planning of high-level trajectories and generating desired end-effector forces. The primary motor cortex is used to translate these high-level commands into low-level signals that can be sent out to the body. The cerebellum is used for storing internal models and providing error correction. However, the means by which each of these is implemented in the REACH and NOCH models is different.

DMPs are used in the REACH model, where sets of learned actions from which the basal ganglia generates a weighting over to accomplish novel movements are used in the NOCH model. This compositionality of movement is not in the scope of the REACH model, which is missing a basal ganglia, but including this as an extension is discussed below. In the NOCH the learned actions were developed by analytically solving the Bellman equation in a highly non-biologically plausible way (see [40] for details). While how trajectories could be learned is outside the scope of the REACH model, there are potentially biologically plausible methods, as discussed above.

In the NOCH model the low-level output is decided by an iterative optimization method to best match a low-level control signal with a desired high-level movement. In the REACH model an adaptive Jacobian matrix is used to perform this transformation, implemented fully in spiking neurons, The methods used in the REACH model also allow the primary motor cortex to generate a secondary control signal that keeps the body near its default resting state, and incorporate nonlinear adaptive methods to learn to compensate for errors in the model of the system under control. While the iterative optimization used by the NOCH model might be biologically plausible through a model-predictive system, the explicit use of such a system to transform control signals at every point in time is unrealistically expensive for use in a biological system.

In the REACH model error correction is performed by the nonlinear adaptive bias methods presented in Chapter 6, while in the NOCH model the error correction is simply an additional PD signal generated from system feedback. The use of internal models is also explicit in the REACH model, where the inertia matrix for the arm is stored in the cerebellum and used to cancel out unwanted velocity and interacting forces between the different arm segments to help linearize their control, while there is no explicit use of internal models in the NOCH model. Internal models allow the system to learn to predict upcoming errors in the system signal and compensate in a feed-forward way, rather than only being able to correct the outgoing control signal once an error is present, as in PD or

PID control.

In the NOCH model implemented in [50] the model was able to control a muscle based arm, because quadratic minimization was used to transform the control signal between levels using iterative approximation. In the REACH model the control of a muscle based system is not as straightforward, because the transformation is based on having a Jacobian matrix rather than an iterative optimization process. There are several possible solutions to this. The first is to apply the adaptive transformation learning algorithm from nonlinear adaptive control reformulated for neural networks in Chapter 6. Given sufficient time this process should be able to learn the necessary transformation. The difficulties of this approach are common to all learning systems, the most prevalent being the concern over how many neurons would be required to be able to approximate such a transformation well. There is a significant increase in the complexity of the transformation to muscles; where even the response of a control signal is highly dependent on the state (i.e. length and current activation of the muscle). Another complexity is added in having muscles that span several joints, where the analytic development of Jacobian matrices is unclear. However, there has been success recently in control of complex muscle based systems through a method called muscle based feature control [62]. Further examination of this method will be required to see if a biologically plausible implementation may be possible.

7.5.3.1 Incorporating the basal ganglia

As mentioned, where the NOCH framework has compositionality of movement as a central feature and the basal ganglia as a prominent part of the system. However, it is outside of the scope of the REACH model. Nevertheless, further work on the compositionality of movement and the role of the basal ganglia, was presented in [43]. In that paper I presented a model of the development of expertise in humans, based on using previously learned actions as a starting point to learning a novel action. Initially, the system is unsure how to perform an action as it has not seen that action before. The basal ganglia in this model explores combinations of previously learned actions as an approximation of the desired action. As it converges to a solution, the basal ganglia trains the motor cortex to generate the same action automatically (i.e. without basal ganglia intervention). This transfer of knowledge from the subcortical basal ganglia loop to the faster cortical loop represents the development of expertise, as fewer system resources are required and the system executes the novel action faster and more consistently.

The integration of this automaticity model presented [43] with the REACH model is not straightforward. This is because, unlike the NOCH model, which was based on having a set of learned movements stored in the primary motor cortex that are weighted to generate an action, in the REACH model, the primary motor cortex is providing a transformation between a high-level control signal and a low-level control signal. Consequently, the relationship of the REACH model to a compositionality based model is much less clear. Considering just the pre-motor cortex, however, the basal ganglia and the REACH models have a more natural relationship. In the DMP literature, work has been done on classification of movement based on the set of weights applied to the basis function. Hence the DMPs are weighted compositions of simpler movements. In this work, the weights of similar trajectories have a larger dot-product than weights for dissimilar trajectories. This similarity can serve as a means of generating a set of weights for a desired trajectory given its similarity to previously learned trajectories.

Finally, the basal ganglia was also assigned the role of scaling movements in the NOCH framework, which fits well with the REACH application of DMPs, which can be scaled and rotated through the addition of a gain term into their dynamics and goal signals. Having the basal ganglia responsible for this scaling will allow some direct and potentially interesting comparisons with patients with basal ganglia damage (e.g. Parkinson's patients) at a behavioural level, looking at their ability to generalize straight reaches and more complex trajectories. In general, however, it remains for future work to consider in more detail precisely how the basal ganglia and related compositional approaches can be integrated with the REACH model presented here.

7.6 Conclusions

Validating claims about explicit algorithms used by the brain is very difficult. However, the REACH model presented in this thesis provides a detailed mechanistic explanation of a large number of phenomena seen in the motor system. Because this implementation is in a spiking neural network, biological constraints have been applied allowing the generation of specific, testable predictions. Continued improvement and testing of these kinds of detailed, yet functional, models is a promising avenue to improving our understanding of the biological basis of behaviour.

APPENDICES

7.7 Forward transformation matrices

Given a serial robot with two rotating joints, as shown in Figure 7.20, let the joint angles be denoted q_1 and q_2, and the control signal be a set of torques acting on these joint angles. To be able to transform forces between the Cartesian coordinate space of the end-effector and joint space it is necessary to know the relationship between the position of the end-effector and each of the joint angles.

The 'point of view' from any given point on the robot is referred to as that point is 'frame of reference'. The 'origin frame of reference' is a static frame from which all of the points of the system can be described and related to each other. By characterizing the position of the end-effector in terms of the joints of the robot it is possible to understand how movement of the joints affects the movement of the end-effector.

In this section the relationship of each of the frames of reference of the robot's links to the origin, or base, frame of reference is described. The characterization of these relationships is done using forward transformation matrices.

7.7.1 Forward transformation matrices in 2D

Looking from the reference frame of q_2, in Figure 7.20, the end-effector point $\mathbf{p}$ is length d_2 away along its x-axis. Similarly, the reference frame of q_2 is length d_1 away from the reference frame of q_1 along q_1's x-axis, and q_1 is length d_0 away from the origin along the y-axis of the origin. The problem is how to calculate the position of $\mathbf{p}$ in terms of the origin's frame of reference.

In this configuration pictured on the left hand side of Figure 7.20 it is straightforward to figure out: $(x = d_2, y = d_0 + d_1)$. Things become more complicated, however, when the

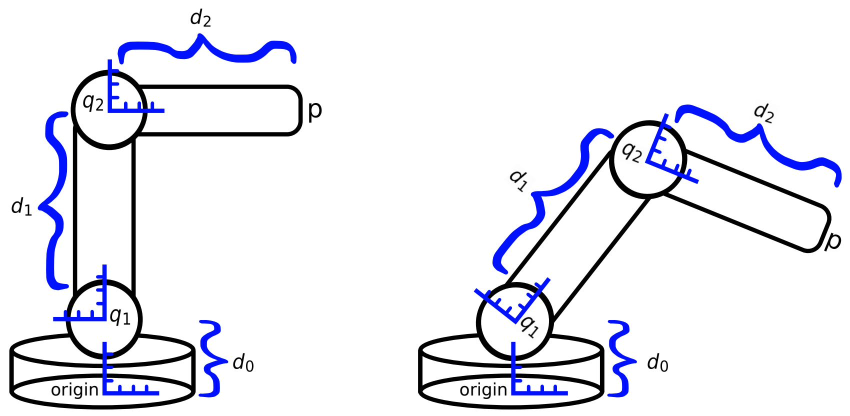


Figure 7.20: A two-link revolute-revolute(RR) robot. Left: A configuration where the position of the end-effector is easily approximated by visual inspection. Right: A configuration of the robot where the exact position of the end-effector is unclear from visual inspection.

robot changes configuration into something like that shown in on the right hand side of Figure 7.20.

A method of being able to explicitly calculate the position of the end-effector in the origin frame of reference that accounts for the rotations and translations of points between different coordinate frames is desired, such that when the current angles of the robot joints and the relative positions of their coordinate frames are known it is possible to quickly calculate the position of the point of interest in terms of the origin coordinate frame.

This task can be broken up into two parts: 1) Accounting for the rotation between different reference frames, and 2) accounting for the translation between different reference frames.

7.7.1.1 Accounting for rotation

Figure 7.21 displays two frames of reference with the same origin rotated from each other by q degrees. Imagine a point $\mathbf{p} = (p_{x_1}, p_{y_1})$ specified in reference frame 1, to find its coordinates in terms of of the origin reference frame, or (x_0, y_0) coordinates, it is necessary to find out the contributions of the x_1 and y_1 axes to the x_0 and y_0 axes. The contributions to the x_0 axis from p_{x_1} are calculated

$$p_{x_0} = cos(q)p_{x_1}, \tag{7.5}$$

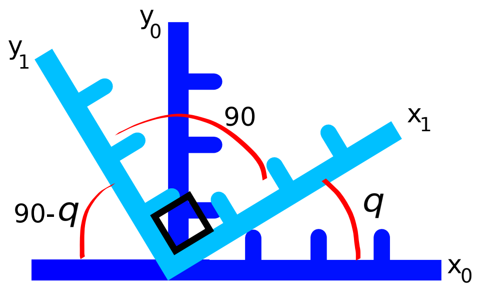


Figure 7.21: Two reference frames at the same origin, rotated q degrees.

which takes the x_1 position of p_{x_1} and maps it to x_0. To calculate how p_{y_1} affects the position in x_0 we calculate

$$p_{y_0} = cos(90 + q)p_{y_1}, \tag{7.6}$$

which is equivalent to $-cos(90 - q)$, as shown in Figure 7.21. This term can be rewritten as $-sin(q)$ because *sin* and *cos* are phase shifted 90 degrees from one another. This leads to the total contributions of a point defined in the (x_1, y_1) axes to the x_0 axis being

$$p_{0_x} = cos(q)p_{x_1} - sin(q)p_{y_1}. \tag{7.7}$$

Similarly for the y_0 axis contributions we have

$$p_{0_y} = sin(q)p_{x_1} + sin(90 - q)p_{y_1}, \tag{7.8}$$

$$p_{0_y} = sin(q)p_{x_1} + cos(q)p_{y_1}. \tag{7.9}$$

Rewriting Eq. 7.7 and Eq. 7.9 in matrix form gives:

$$\mathbf{R}_0^1\,\mathbf{p}_1 = \begin{bmatrix} cos(q_0) & -sin(q_0) \\ sin(q_0) & cos(q_0) \end{bmatrix} \begin{bmatrix} p_{x_1} \\ p_{y_1} \end{bmatrix}, \tag{7.10}$$

where $\mathbf{R}_0^1$ is called a rotation matrix. The notation used here for these matrices is that the reference frame number being rotated *from* is denoted in the superscript, and the reference frame being rotated *into* is in the subscript. $\mathbf{R}_0^1$ denotes a rotation from reference frame 1 into reference frame 0.

To transform any point in reference frame 1 into reference frame 0 is a multiplication by the rotation matrix $\mathbf{R}_0^1$.

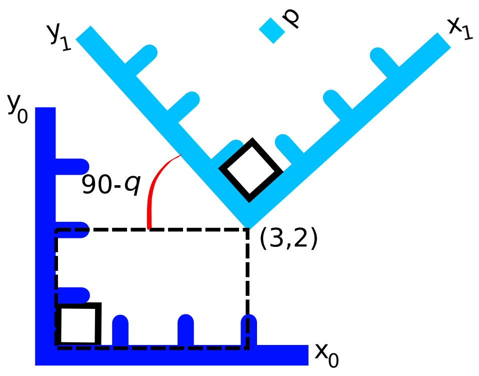


Figure 7.22: Two reference frames with rotated q degrees and translated $(3, 2)$ from each other, and a point $p = (2, 2)$ in reference frame 1.

7.7.1.2 Accounting for translation

The second part of transformation is translation, and so it is also necessary to account for distances between reference frame origins. Given the reference frames 1 and 0 shown in Figure 7.22, where point $\mathbf{p} = (2, 2)$ in reference frame 1. Reference frame 1 is rotated 45 degrees from and located at $(3, 2)$ in reference frame 0. To account for this translation and rotation a new matrix will be created that includes both rotation and translation. It is generated by appending distances, denoted $\mathbf{D}$, to the rotation matrix $\mathbf{R}_0^1$ along with a row of zeros ending in a 1 to get a transformation matrix:

$$\mathbf{T}_0^1 = \begin{bmatrix} \mathbf{R}_0^1 & \mathbf{D}_0^1 \\ \mathbf{0} & \mathbf{1} \end{bmatrix} \tag{7.11}$$

$$\mathbf{T}_0^1 = \begin{bmatrix} cos(q_0) & -sin(q_0) & d_{x0} \\ sin(q_0) & cos(q_0) & d_{y0} \\ 0 & 0 & 1 \end{bmatrix}. \tag{7.12}$$

To make the matrix-vector multiplications work out, a homogeneous representation must be used, which adds an extra row with a 1 to the end of the vector $\mathbf{p}$ to give

$$\mathbf{p} = \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix}. \tag{7.13}$$

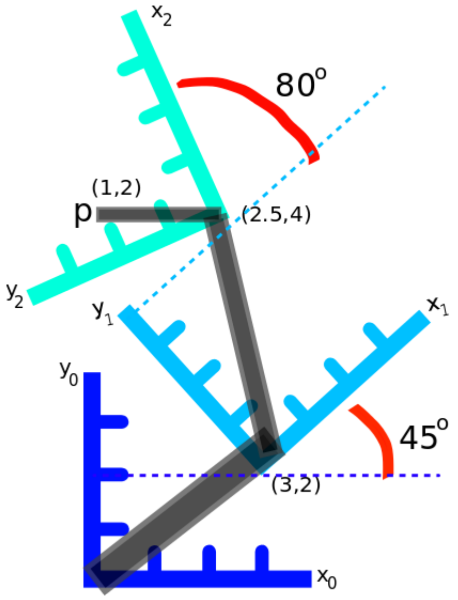


Figure 7.23: A three link, planar revolute-revolute-revolute arm. The point end-effector p is defined in the reference frame of the third link, and must be transformed to the origin reference frame.

When position vector $\mathbf{p}$ is multiplied by the transformation matrix $\mathbf{T}_0^1$ the answer should be somewhere around $(3, 5)$ from visual inspection, and indeed:

$$\mathbf{T}_0^1\,\mathbf{p} = \begin{bmatrix} cos(45) & -sin(45) & 3 \\ sin(45) & cos(45) & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 4.8285 \\ 1 \end{bmatrix}. \tag{7.14}$$

To get the coordinates of $\mathbf{p}$ in reference frame 0 now simply take the first two elements of the resulting vector $\mathbf{p} = (3, 4.8285)$.

7.7.1.3 Applying multiple transformations

Multiple transformation matrices can also be strung together. Given the robot arm shown in Figure 7.23, the end-effector is at point $(1, 2)$ in reference frame 2, which is at an 80 degree angle from reference frame 1 and located at $(x_1 = 2.5, y_1 = 4)$. The transformation matrix for moving from reference frame 2 to reference frame 1 is:

$$\mathbf{T}_1^2 = \begin{bmatrix} cos(80) & -sin(80) & 2.5 \\ sin(80) & cos(80) & 4 \\ 0 & 0 & 1 \end{bmatrix}. \tag{7.15}$$

To get the position of the end-effector in terms of reference frame 0 it is necessary to transform from reference frame 2 into reference frame 1 with $\mathbf{T}_1$ and then from reference frame 1 into reference frame 0 with the previously defined transformation matrix $\mathbf{T}_0^1$:

$$\mathbf{T}_0^1 = \begin{bmatrix} cos(45) & -sin(45) & 3 \\ sin(45) & cos(45) & 2 \\ 0 & 0 & 1 \end{bmatrix}. \tag{7.16}$$

By applying the transformation matrices one after the other it is possible to move from reference frame 2 down to reference frame 0:

$$\mathbf{T}_0^1\,\mathbf{T}_1^2 \begin{bmatrix} p_{x_2} \\ p_{y_2} \\ 1 \end{bmatrix} = \begin{bmatrix} p_{x_0} \\ p_{y_0} \\ 1 \end{bmatrix}, \tag{7.17}$$

$$\begin{bmatrix} cos(45) & -sin(45) & 3 \\ sin(45) & cos(45) & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos(80) & -sin(80) & 2.5 \\ sin(80) & cos(80) & 4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.273 \\ 6.268 \\ 1 \end{bmatrix}, \tag{7.18}$$

which gives the result the point $\mathbf{p}$ is at $(-0.273, 6.268)$ in reference frame 0.

It is also often worthwhile to define a single matrix to perform the entire translation in one step:

$$\mathbf{T}_0^2 = \mathbf{T}_0^1\,\mathbf{T}_1^2, \tag{7.19}$$

such that multiplication of the point in reference frame 2 by this new transformation matrix $\mathbf{T}_0^2$ transforms it into the coordinates of reference frame 0.

7.7.2 Forward transform matrices in 3D

The previous examples of transformation matrices were all for 2D systems, but it is straightforward to generalize to 3 dimensions. The example here is based on Samir Menon's RPP control tutorial [3].

[3] `http://web.stanford.edu/~smenon/code/rppbot/MathTutorial_01_RPPBot.htm`

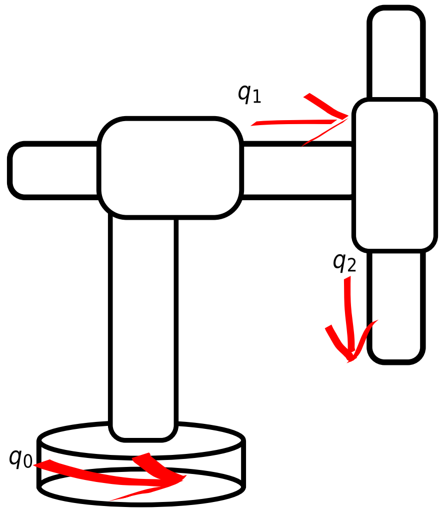


Figure 7.24: A three-link revolute-prismatic-prismatic (RPP) robot.

Given a standard revolute-prismatic-prismatic robot, a sketch of which is shown in Figure 7.24, where the base rotates around the z axis, and the distance from reference frame 0 to reference frame 1 is 1 unit, also along the z axis. The rotation matrix from reference frame 1 to reference frame 0 is:

$$\mathbf{R}_0^1 = \begin{bmatrix} cos(q_0) & -sin(q_0) & 0 \\ sin(q_0) & cos(q_0) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7.20}$$

and the translation vector is

$$\mathbf{D}_0^1 = [0, 0, 1]^T. \tag{7.21}$$

The transformation matrix from reference frame 1 to reference frame 0 is then:

$$\mathbf{T}_0^1 = \begin{bmatrix} cos(q_0) & -sin(q_0) & 0 & 0 \\ sin(q_0) & cos(q_0) & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{7.22}$$

where the third column indicates that there was no rotation around the z axis in moving between reference frames, and the forth (translation) column shows that we move 1 unit along the z axis. The fourth row is again then only present to make the multiplications work out and provides no information.

For transformation from the reference frame 2 to reference frame 1, there is no rotation (because it is a prismatic joint), and there is translation along the y axis of reference frame 1 equal to $.5 + q_1$. This gives a transformation matrix:

$$\mathbf{T}_1^2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0.5 + q_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{7.23}$$

The final transformation, from the origin of reference frame 2 to the end-effector position is similarly another transformation with no rotation (because this joint is also prismatic), that translates along the z axis:

$$\mathbf{T}_2^{ee} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -0.2 - q_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{7.24}$$

The full transformation from reference frame 0 to the end-effector is found by combining all of the above transformation matrices:

$$\mathbf{T}_0^{ee} = \mathbf{T}_0^1 \, \mathbf{T}_1^2 \, \mathbf{T}_2^{ee} = \begin{bmatrix} cos(q_0) & -sin(q_0) & 0 & -sin(q_0)(0.5 + q_1) \\ sin(q_0) & cos(q_0) & 0 & cos(q_0)(0.5 + q_1) \\ 0 & 0 & 1 & 0.8 - q_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{7.25}$$

To transform a point from the end-effector reference frame into terms of the origin reference frame, simply multiply the transformation matrix by the point of interest relative to the end-effector. If it is the end-effector position that is of interest to us, $p = [0, 0, 0, 1]^T$. For example, let $q_0 = \frac{\pi}{3}$, $q_1 = .3$, and $q_2 = .4$, then the end-effector location is:

$$\mathbf{T}_0^{ee} \, \mathbf{p} = \begin{bmatrix} cos(q_0) & -sin(q_0) & 0 & -sin(q_0)(0.5 + q_1) \\ sin(q_0) & cos(q_0) & 0 & cos(q_0)(0.5 + q_1) \\ 0 & 0 & 1 & .8 + q_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.693 \\ 0.4 \\ 0.8 \\ 1 \end{bmatrix}. \tag{7.26}$$

7.7.2.1 Inverting the transformation matrices

If a point is defined in terms of reference frame 0 and needs to be translated into a reference frame relative to the end-effector, an inverse transform matrix can be used. Because of the way the transformation matrices are constructed, calculating the inverse transform ends up being straight-forward. Continuing the same robot example and configuration as above, and denoting the rotation part of the transform matrix $\mathbf{R}$ and the translation part $\mathbf{D}$, the inverse transform is defined:

$$(\mathbf{T}_0^{ee})^{-1} = \begin{bmatrix} (\mathbf{R}_0^{ee})^T & -(\mathbf{R}_0^{ee})^T \, \mathbf{D}_0^{ee} \\ 0 & 1 \end{bmatrix}. \tag{7.27}$$

Given a point $\mathbf{p} = [1, 1, .5, 1]^T$ in reference frame 0, then relative to the end-effector it is at:

$$\begin{bmatrix} p_{x_{ee}} \\ p_{y_{ee}} \\ p_{z_{ee}} \\ 1 \end{bmatrix} = (\mathbf{T}_0^{ee})^{-1} \begin{bmatrix} p_{x_0} \\ p_{y_0} \\ p_{z_0} \\ 1 \end{bmatrix} = \begin{bmatrix} 1.37 \\ -1.17 \\ -0.3 \\ 1 \end{bmatrix}. \tag{7.28}$$

7.8 Jacobians, velocity, and force

A Jacobian defines the dynamic relationship between two different representations of a system. Given a robot arm with joint angles and velocity, $\mathbf{q}$ and $\dot{\mathbf{q}}$, respectively, and end-effector position and velocity, $\mathbf{x}$ and $\dot{\mathbf{x}}$, respectively, the Jacobian relates how movement of the description of $\mathbf{q}$ causes movement in the description of $\mathbf{x}$. The Jacobian is a linear approximation of the system kinematics at a given point in time, and so must be constantly updated to ensure that any calculations using it are accurate.

Formally, a Jacobian is defined through as a set of partial differential equations:

$$\mathbf{J}(\mathbf{q}) = \frac{\partial \mathbf{x}}{\partial \mathbf{q}}. \tag{7.29}$$

Rearranging this equation gives

$$\mathbf{J}(\mathbf{q}) = \frac{\partial \mathbf{x}}{\partial t} \frac{\partial t}{\partial \mathbf{q}} \rightarrow \frac{\partial \mathbf{x}}{\partial t} = \mathbf{J}(\mathbf{q}) \frac{\partial \mathbf{q}}{\partial t}, \tag{7.30}$$

or

$$\dot{\mathbf{x}} = \mathbf{J}(\mathbf{q}) \, \dot{\mathbf{q}}. \tag{7.31}$$

This says that the end-effector velocity is equal to the Jacobian, $\mathbf{J}$, multiplied by the joint angle velocity, $\dot{\mathbf{q}}$.

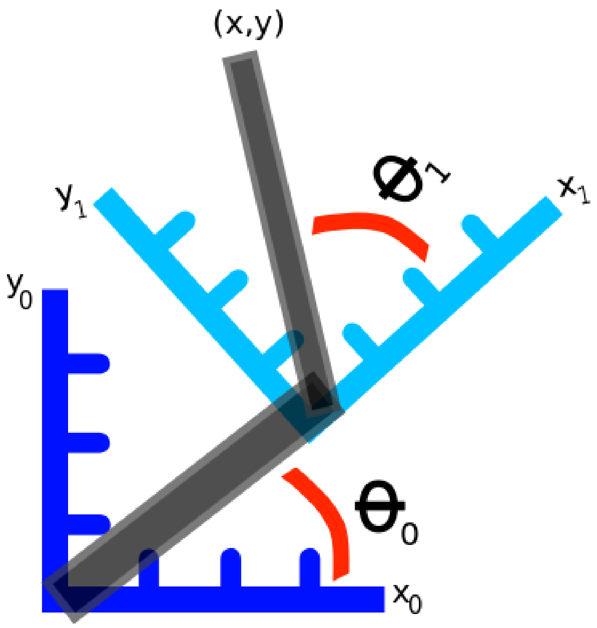


Figure 7.25: A serial 2-link robot arm, with joint angles θ_0 and θ_1. The reference frame from the first joint is along the axes (x_0, y_0), and the reference frame of the second joint is along the axes (x_1, y_1).

The importance of the Jacobian comes from the desire to control in operational (or task) space, i.e. to plan a trajectory in a different space than can be controlled directly. Jacobians provide a way to calculate what the control signal is in the space that we control (torques) given a control signal in one we do not (end-effector forces). The above equation, however, relates the velocities between two different spaces, but what is needed is a relationship between forces in these spaces.

7.8.1 Energy equivalence and Jacobians

Conservation of energy is a property of all physical systems where the amount of energy expended is the same no matter how the system in question is being represented. The planar two-link robot arm shown in Figure 7.25 will be used for illustration. Let the joint angle positions be denoted $\mathbf{q} = [q_0, q_1]^T$, and end-effector position be denoted $\mathbf{x} = [x, y, 0]^T$.

Work is the application of force over a distance

$$\mathbf{W} = \int \mathbf{F}^T \mathbf{v} dt \tag{7.32}$$

, where $\mathbf{W}$ is work, $\mathbf{F}$ is force, and $\mathbf{v}$ is velocity.

Power is the rate at which work is performed

$$\mathbf{P} = \frac{d}{dt}\mathbf{W}, \tag{7.33}$$

where $\mathbf{P}$ is power. Substituting in Eq. 7.32 into Eq. 7.33 gives:

$$\mathbf{P} = \frac{d}{dt}\mathbf{F}^T \mathbf{v}. \tag{7.34}$$

Because of energy equivalence, work is performed at the same rate regardless of the characterization of the system. Rewriting Eq. 7.34 in terms of end-effector space gives:

$$\mathbf{P} = \mathbf{F}_{\mathbf{x}}^T \dot{\mathbf{x}}, \tag{7.35}$$

where $\mathbf{F}_{\mathbf{x}}$ is the force applied to the hand, and $\dot{\mathbf{x}}$ is the velocity of the hand. Rewriting Eq. 7.34 in terms of joint-space gives:

$$\mathbf{P} = \mathbf{F}_{\mathbf{q}}^T \dot{\mathbf{q}}, \tag{7.36}$$

where $\mathbf{F}_{\mathbf{q}}$ is the torque applied to the joints, and $\dot{\mathbf{q}}$ is the angular velocity of the joints. Setting Eq. 7.35 and Eq. 7.36 equal to each other and substituting with Eq. 7.31 gives

$$\begin{aligned}
\mathbf{F}_{q_{hand}}^T \dot{\mathbf{q}} &= \mathbf{F}_{\mathbf{x}}^T \dot{\mathbf{x}}, && (7.37)\\
\mathbf{F}_{q_{hand}}^T \dot{\mathbf{q}} &= \mathbf{F}_{\mathbf{x}}^T \mathbf{J}_{ee}(\mathbf{q})\, \dot{\mathbf{q}}, && (7.38)\\
\mathbf{F}_{q_{hand}}^T & \quad \mathbf{F}_{\mathbf{x}}^T \mathbf{J}_{ee}(\mathbf{q}), && (7.39)\\
\mathbf{F}_{q_{hand}} &= \mathbf{J}_{ee}^T(\mathbf{q}) \mathbf{F}_{\mathbf{x}}, && (7.40)
\end{aligned}$$

where $\mathbf{J}_{ee}(\mathbf{q})$ is the Jacobian for the end-effector of the robot, and $\mathbf{F}_{q_{hand}}$ represents the forces in joint-space that affect movement of the hand. This says that not only does the Jacobian relate velocity from one state-space representation to another, it can also be used to calculate what the forces in joint space should be to effect a desired set of forces in end-effector space.

7.8.2 Building the Jacobian

In deriving the Jacobian, the relationship between the end-effector position and the robot is joint angles needs to be defined. To do this, forward transformation matrices, described in Section 7.7, can be employed.

7.8.2.1 The forward transformation matrix

Recall that transformation matrices allow a given point to be transformed between different reference frames. In this case, the position of the end-effector relative to the second joint of the robot arm is known, but where it is relative to the base reference frame (the first joint reference frame in this case) is of interest. This means that only one transformation matrix is needed, transforming from the reference frame attached to the second joint back to the base.

The rotation part of this matrix is straight-forward to define, as in the previous section:

$$\mathbf{R}_0^1 = \begin{bmatrix} cos(q_0) & -sin(q_0) & 0 \\ sin(q_0) & cos(q_0) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{7.41}$$

The translation part of the transformation matrices is a little different than before because reference frame 1 changes as a function of the angle of the previous joint's angles. From trigonometry, given a vector of length r and an angle q the x position of the end point is defined $r\ cos(q)$, and the y position is $r\ sin(q)$. The arm is operating in the (x, y) plane, so the z position will always be 0.

Using this knowledge, the translation part of the transformation matrix is defined:

$$\mathbf{D}_0^1 = \begin{bmatrix} L_0 cos(q_0) \\ L_0 sin(q_0) \\ 0 \end{bmatrix}. \tag{7.42}$$

Giving the forward transformation matrix:

$$\mathbf{T}_0^1 = \begin{bmatrix} \mathbf{R}_0^1 & \mathbf{D}_0^1 \\ \mathbf{0} & \mathbf{1} \end{bmatrix} = \begin{bmatrix} cos(q_0) & -sin(q_0) & 0 & L_0 cos(q_0) \\ sin(q_0) & cos(q_0) & 0 & L_0 sin(q_0) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{7.43}$$

which transforms a point from reference frame 1 (elbow joint) to reference frame 0 (shoulder joint / base).

The point of interest is the end-effector which is defined in reference frame 1 as a function of joint angle, q_1 and the length of second arm segment, L_1:

$$\mathbf{x} = \begin{bmatrix} L_1 cos(q_1) \\ L_1 sin(q_1) \\ 0 \\ 1 \end{bmatrix}. \tag{7.44}$$

To find the position of our end-effector in terms of the origin reference frame multiply the point $\mathbf{x}$ by the transformation $\mathbf{T}_0^1$:

$$\mathbf{T}_0^1\,\mathbf{x} = \begin{bmatrix} cos(q_0) & -sin(q_0) & 0 & L_0 cos(q_0) \\ sin(q_0) & cos(q_0) & 0 & L_0 sin(q_0) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} L_1 cos(q_1) \\ L_1 sin(q_1) \\ 0 \\ 1 \end{bmatrix}, \tag{7.45}$$

$$\mathbf{T}_0^1\mathbf{x} = \begin{bmatrix} L_1 cos(q_0)cos(q_1) - L_1 sin(q_0)sin(q_1) + L_0 cos(q_0) \\ L_1 sin(q_0)cos(q_1) + L_1 cos(q_0)sin(q_1) + L_0 sin(q_0) \\ 0 \\ 1 \end{bmatrix} \tag{7.46}$$

where, by pulling out the L_1 term and using the trig identities

$$cos(\alpha)cos(\beta) - sin(\alpha)sin(\beta) = cos(\alpha + \beta) \tag{7.47}$$

and

$$sin(\alpha)cos(\beta) + cos(\alpha)sin(\beta) = sin(\alpha + \beta) \tag{7.48}$$

the position of our end-effector can be rewritten:

$$\begin{bmatrix} L_0 cos(q_0) + L_1 cos(q_0 + q_1) \\ L_0 sin(q_0) + L_1 sin(q_0 + q_1) \\ 0 \end{bmatrix}, \tag{7.49}$$

which is the position of the end-effector in terms of joint angles. As mentioned above, however, both the position of the end-effector and its orientation are needed; the rotation of the end-effector relative to the base frame must also be defined.

7.8.2.2 Accounting for orientation

Fortunately, defining orientation is simple, especially for systems with only revolute and prismatic joints (spherical joints will not be considered here). With prismatic joints, which are linear and move in a single plane, the rotation introduced is 0. With revolute joints, the rotation of the end-effector in each axis is simply a sum of rotations of each joint in their respective axes of rotation.

In the example case, the joints are rotating around the z axis, so the rotation part of our end-effector state is

$$\begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ q_0 + q_1 \end{bmatrix}, \tag{7.50}$$

where ω denotes angular velocity. If the first joint had been rotating in a different plane, e.g. in the (x, z) plane around the y axis instead, then the orientation would be

$$\begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} 0 \\ q_0 \\ q_1 \end{bmatrix}. \tag{7.51}$$

7.8.2.3 Partial differentiation

Once the position and orientation of the end-effector have been calculated, the partial derivative of these equations need to be calculated with respect to the elements of $\mathbf{q}$. For simplicity, the Jacobian will be broken up into two parts, J_v and J_ω, representing the linear and angular velocity, respectively, of the end-effector.

The linear velocity part of our Jacobian is:

$$\mathbf{J}_v(\mathbf{q}) = \begin{bmatrix} \frac{\partial x}{\partial q_0} & \frac{\partial x}{\partial q_1} \\ \frac{\partial y}{\partial q_0} & \frac{\partial y}{\partial q_1} \\ \frac{\partial z}{\partial q_0} & \frac{\partial z}{\partial q_1} \end{bmatrix} = \begin{bmatrix} -L_0 sin(q_0) - L_1 sin(q_0 + q_1) & -L_1 sin(q_0 + q_1) \\ L_0 cos(q_0) + L_1 cos(q_0 + q_1) & L_1 cos(q_0 + q_1) \\ 0 & 0 \end{bmatrix}. \tag{7.52}$$

The angular velocity part of our Jacobian is:

$$\mathbf{J}_\omega(\mathbf{q}) = \begin{bmatrix} \frac{\partial \omega_x}{\partial q_0} & \frac{\partial \omega_x}{\partial q_1} \\ \frac{\partial \omega_y}{\partial q_0} & \frac{\partial \omega_y}{\partial q_1} \\ \frac{\partial \omega_z}{\partial q_0} & \frac{\partial \omega_z}{\partial q_1} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}. \tag{7.53}$$

The full Jacobian for the end-effector is then:

$$\mathbf{J}_{ee}(\mathbf{q}) = \begin{bmatrix} \mathbf{J}_v(\mathbf{q}) \\ \mathbf{J}_\omega(\mathbf{q}) \end{bmatrix} = \begin{bmatrix} -L_0 sin(q_0) - L_1 sin(q_0+q_1) & -L_1 sin(q_0+q_1) \\ L_0 cos(q_0) + L_1 cos(q_0+q_1) & L_1 cos(q_0+q_1) \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}. \tag{7.54}$$

7.8.3 Analyzing the Jacobian

Once the Jacobian is built, it can be analyzed for insight about the relationship between $\dot{\mathbf{x}}$ and $\dot{\mathbf{q}}$.

For example, there is a large block of zeros in the middle of the Jacobian defined above, along the row corresponding to linear velocity along the z axis, and the rows corresponding to the angular velocity around the x and y axes. This means that the z position, and rotations ω_x and ω_y are not controllable. This can be seen by going back to the first Jacobian equation:

$$\dot{\mathbf{x}} = \mathbf{J}_{ee}(\mathbf{q})\ \dot{\mathbf{q}}.$$

No matter what the values of $\dot{\mathbf{q}}$, it is impossible to affect ω_x, ω_y, or z, because the corresponding rows during the above multiplication with the Jacobian are $\mathbf{0}$. These rows of zeros in the Jacobian are referred to as its 'null space'. Because these variables can't be controlled, they will be dropped from both $\mathbf{F_x}$ and $\mathbf{J}(\mathbf{q})$.

Looking at the variables that can be affected it can be seen that given any two of x, y, ω_z the third can be calculated because the robot only has 2 degrees of freedom (the shoulder and elbow). This means that only two of the end-effector variables can actually be controlled. In the situation of controlling a robot arm, it is most useful to control the (x, y) coordinates, so ω_z will be dropped from the force vector and Jacobian.

After removing the redundant term, the force vector representing the controllable end-effector forces is

$$\mathbf{F_x} = \begin{bmatrix} f_x \\ f_y \end{bmatrix}, \tag{7.55}$$

where f_x is force along the x axis, f_y is force along the y axis, and the Jacobian is written

$$\mathbf{J}_{ee}(\mathbf{q}) = \begin{bmatrix} -L_0 sin(q_0) - L_1 sin(q_0+q_1) & -L_1 sin(q_0+q_1) \\ L_0 cos(q_0) + L_1 cos(q_0+q_1) & L_1 cos(q_0+q_1) \end{bmatrix}. \tag{7.56}$$

If instead f_{ω_z}, i.e. torque around the z axis, were chosen as a controlled force then the force vector and Jacobian would be (assuming force along the x axis was also chosen):

$$\mathbf{F_x} = \begin{bmatrix} f_x \\ f_{\omega_z} \end{bmatrix}, \tag{7.57}$$

$$\mathbf{J}_{ee}(\mathbf{q}) = \begin{bmatrix} -L_0 sin(q_0) - L_1 sin(q_0 + q_1) & -L_1 sin(q_0 + q_1) \\ 1 & 1 \end{bmatrix}. \tag{7.58}$$

7.8.4 Using the Jacobian

With the Jacobian defined the changes in linear and angular end-effector velocities caused by different joint angle velocities can be calculated, and desired end-effector space forces, $\mathbf{F_x}$, can be transformed into joint-space torques, $\mathbf{F_q}$. Two example cases will now be provided, using the full Jacobian in the first example and the simplified Jacobian in the second example.

7.8.4.1 Example 1

Given known joint angle velocities with arm configuration

$$\mathbf{q} = \begin{bmatrix} \frac{\pi}{4} \\ \frac{3\pi}{8} \end{bmatrix} \quad \dot{\mathbf{q}} = \begin{bmatrix} \frac{\pi}{10} \\ \frac{\pi}{10} \end{bmatrix},$$

and arm segment lengths $L_i = 1$, the (x, y) velocities of the end-effector can be calculated by substituting in the system state at the current time into Eq. 7.31

$$\begin{aligned}
\dot{\mathbf{x}} &= \mathbf{J}_{ee}(\mathbf{q})\,\dot{\mathbf{q}}, \\
\dot{\mathbf{x}} &= \begin{bmatrix} -sin(\frac{\pi}{4}) - sin(\frac{\pi}{4} + \frac{3\pi}{8}) & -sin(\frac{\pi}{4} + \frac{3\pi}{8}) \\ cos(\frac{\pi}{4}) + cos(\frac{\pi}{4} + \frac{3\pi}{8}) & cos(\frac{\pi}{4} + \frac{3\pi}{8}) \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} \frac{\pi}{10} \\ \frac{\pi}{10} \end{bmatrix}, \\
\dot{\mathbf{x}} &= \left[-0.8026, -0.01830, 0, 0, 0, \frac{\pi}{5}\right]^T.
\end{aligned}$$

And so the end-effector is linear velocity is $(-0.8026, -0.01830, 0)^T$, with angular velocity is $(0, 0, \frac{\pi}{5})^T$.

7.8.4.2 Example 2

Given the same system and configuration as in the previous example as well as a trajectory planned in (x, y) space, the goal is to calculate the torques required to get the end-effector to move as desired. The controlled variables will be forces along the x and y axes, and so the reduced Jacobian from the previous section will be used. Let the desired (x, y) forces be

$$\mathbf{F_x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

to calculate the corresponding joint torques the desired end-effector forces and current system state parameters are substituted into Eq. 7.40

$$\begin{aligned} \mathbf{F_q} &= \mathbf{J}_{ee}^T(\mathbf{q})\mathbf{F_x}, \\ \mathbf{F_q} &= \begin{bmatrix} -sin(\frac{\pi}{4}) - sin(\frac{\pi}{4} + \frac{3\pi}{8}) & cos(\frac{\pi}{4}) + cos(\frac{\pi}{4} + \frac{3\pi}{8}) \\ -sin(\frac{\pi}{4} + \frac{3\pi}{8}) & cos(\frac{\pi}{4} + \frac{3\pi}{8}) \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \\ \mathbf{F_q} &= \begin{bmatrix} -1.3066 \\ -1.3066 \end{bmatrix}. \end{aligned}$$

So given the current configuration to get the end-effector to move as desired, without accounting for the effects of inertia and gravity, the torques to apply to the system are $\mathbf{F_q} = [-1.3066, -1.3066]^T$.

7.9 Accounting for mass and gravity

In very simple systems where highly precise control isn't required, the effects of gravity and inertia can be ignored, but in general they must be accounted for and cancelled out. The full dynamics of a robot arm are

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} = (\mathbf{u} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) - \mathbf{g}(\mathbf{q})), \tag{7.59}$$

where $\ddot{\mathbf{q}}$ is joint angle acceleration, $\mathbf{u}$ is the control signal (specifying torque), $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ is a function describing the Coriolis and centrifugal effects, $\mathbf{g}(\mathbf{q})$ is the effect of gravity in joint space, and $\mathbf{M}$ is the mass matrix of the system in joint space.

There are a lot of terms involved in the system acceleration, so while the Jacobian can be used to transform forces between coordinate systems it is clear that just setting the control signal $\mathbf{u} = \mathbf{J}_{ee}^T(\mathbf{q})\mathbf{F_x}$ is not sufficient, because a lot of the dynamics affecting

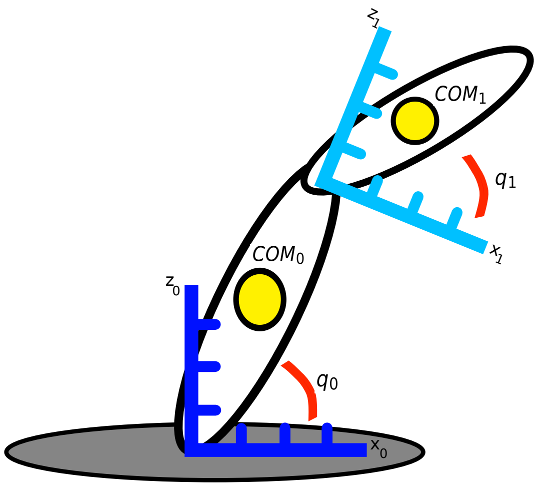


Figure 7.26: A two link arm extending along the (x, z) dimensions with y extending into the diagram. The yellow circles represent the centres-of-mass (COM) of each link.

acceleration aren't accounted for. In this section an effective PD controller operating in joint space will be developed that will allow for more precise control by cancelling out unwanted acceleration terms. To do this the effects of inertia and gravity need to be calculated.

7.9.1 Accounting for inertia

The fact that systems have mass introduces inertia into the system dynamics, making control of how the system will move at any given point in time more difficult. Mass can be thought of as an object's unwillingness to respond to applied forces. The heavier something is, the more resistant it is to acceleration, and the force required to move a system along a desired trajectory depends on both the object's mass and its current acceleration. To effectively control a system, the system inertia needs to be calculated so that it can be included in the control signal and cancelled out.

Assume a robot arm operating in the (x, z) plane, shown in Figure 7.26 with the y axis extending into the picture where the yellow circles represent each links centre-of-mass (COM). The position of each link is COM is defined relative to that link's reference frame, and the goal is to figure out how much each link's mass will affect the system dynamics.

The first step is to transform the representation of each of the COM from Cartesian coordinates in the reference frame of their respective arm segments into terms of joint angles, such that the Jacobian for each COM can be calculated.

Let the COM positions relative to each segment's coordinate frame be

$$\text{com}_0 = \begin{bmatrix} \frac{1}{2}cos(q_0) \\ 0 \\ \frac{1}{2}sin(q_0) \end{bmatrix}, \quad \text{com}_1 = \begin{bmatrix} \frac{1}{4}cos(q_1) \\ 0 \\ \frac{1}{4}sin(q_1) \end{bmatrix}.$$

The first segment's COM is already in base coordinates (since the first link and the base share the same coordinate frame), so all that is required is the position of the second link's COM in the base reference frame, which can be done with the transformation matrix

$$\mathbf{T}_0^1 = \begin{bmatrix} cos(q_1) & 0 & -sin(q_1) & L_0cos(q_0) \\ 0 & 1 & 0 & 0 \\ sin(q_1) & 0 & cos(q_1) & L_0sin(q_0) \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{7.60}$$

Using $\mathbf{T}_0^1$ to transform the com_1 gives

$$\mathbf{T}_0^1\,\text{com}_1 = \begin{bmatrix} cos(q_1) & 0 & -sin(q_1) & L_0cos(q_0) \\ 0 & 1 & 0 & 0 \\ sin(q_1) & 0 & cos(q_1) & L_0sin(q_0) \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{4}cos(q_1) \\ 0 \\ \frac{1}{4}sin(q_1) \\ 1 \end{bmatrix} \tag{7.61}$$

$$\mathbf{T}_0^1\,\text{com}_1 = \begin{bmatrix} L_0cos(q_0) + \frac{1}{4}cos(q_0 + q_1) \\ 0 \\ L_0sin(q_0) + \frac{1}{4}cos(q_0 + q_1) \\ 1 \end{bmatrix}. \tag{7.62}$$

With the COM positions in the origin reference frame and in terms of joint angles, the

Jacobians for each point can be found using Eq. 7.31, which gives

$$\mathbf{J}_0(\mathbf{q}) = \begin{bmatrix} -\frac{1}{2}sin(q_0) & 0 \\ 0 & 0 \\ \frac{1}{2}cos(q_0) & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \end{bmatrix},$$

$$\mathbf{J}_1(\mathbf{q}) = \begin{bmatrix} -L_0 sin(q_0) - \frac{1}{4}sin(q_0+q_1) & -\frac{1}{4}sin(q_0+q_1) \\ 0 & 0 \\ -L_0 cos(q_0) - \frac{1}{4}cos(q_0+q_1) & -\frac{1}{4}cos(q_0+q_1) \\ 0 & 0 \\ 1 & 1 \\ 0 & 0 \end{bmatrix}.$$

7.9.1.1 Kinetic energy

The total energy of a system can be calculated as a sum of the energy introduced from each source. The Jacobians just derived will be used to calculate the kinetic energy each link generates during motion. Each link's kinetic energy will be calculated and summed to get the total energy introduced into the system by the mass and configuration of each link.

Kinetic energy (KE) is one half of mass times velocity squared:

$$\text{KE} = \frac{1}{2}\,\dot{\mathbf{x}}^T \mathbf{M}_{\mathbf{x}}(\mathbf{q})\,\dot{\mathbf{x}}, \tag{7.63}$$

where $\mathbf{M}_{\mathbf{x}}$ is the mass matrix of the system, with the subscript $\mathbf{x}$ denoting that it is defined in Cartesian space, and $\dot{\mathbf{x}}$ is a velocity vector, where $\dot{\mathbf{x}}$ is of the form

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\omega_x} \\ \dot{\omega_y} \\ \dot{\omega_z} \end{bmatrix}, \tag{7.64}$$

and the mass matrix is structured

$$\mathbf{M}_{\mathbf{x}_i}(\mathbf{q}) = \begin{bmatrix} m_i & 0 & 0 & 0 & 0 & 0 \\ 0 & m_i & 0 & 0 & 0 & 0 \\ 0 & 0 & m_i & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{xx} & I_{xy} & I_{xz} \\ 0 & 0 & 0 & I_{yx} & I_{yy} & I_{yz} \\ 0 & 0 & 0 & I_{zx} & I_{zy} & I_{zz} \end{bmatrix}, \tag{7.65}$$

where m_i is the mass of COM i, and the I_{ij} terms are the moments of inertia, which define the object's resistance to change in angular velocity about the axes, the same way that the mass element defines the object's resistance to changes in linear velocity.

As mentioned above, the mass matrix for the COM of each link is defined in Cartesian coordinates in its respective arm segment's reference frame. The effects of mass need to be found in joint angle space, however, because that is where the controller operates. Looking at the summation of the KE introduced by each COM:

$$\text{KE} = \frac{1}{2}\,\Sigma_{i=0}^{n}(\dot{\mathbf{x}}_i^T \mathbf{M}_{\mathbf{x}_i}(\mathbf{q})\,\dot{\mathbf{x}}_i), \tag{7.66}$$

and substituting in $\dot{\mathbf{x}} = \mathbf{J}\,\dot{\mathbf{q}}$,

$$\text{KE}_i = \frac{1}{2}\,\Sigma_{i=0}^{n}(\dot{\mathbf{q}}^T\,\mathbf{J}_i^T \mathbf{M}_{\mathbf{x}_i}(\mathbf{q})\mathbf{J}_i\,\dot{\mathbf{q}}), \tag{7.67}$$

and moving the $\dot{\mathbf{q}}$ terms outside the summation,

$$\text{KE}_i = \frac{1}{2}\,\dot{\mathbf{q}}^T\,\Sigma_{i=0}^{n}(\mathbf{J}_i^T \mathbf{M}_{\mathbf{x}_i}(\mathbf{q})\mathbf{J}_i)\,\dot{\mathbf{q}}. \tag{7.68}$$

Defining

$$\mathbf{M}(\mathbf{q}) = \Sigma_{i=0}^{n}\,\mathbf{J}_i^T(\mathbf{q})\mathbf{M}_{\mathbf{x}_i}(\mathbf{q})\,\mathbf{J}_i(\mathbf{q}), \tag{7.69}$$

gives

$$\text{KE} = \frac{1}{2}\,\dot{\mathbf{q}}\,\mathbf{M}(\mathbf{q})\,\dot{\mathbf{q}}, \tag{7.70}$$

which is the equation for calculating kinetic energy in joint space. Thus, $\mathbf{M}(\mathbf{q})$ denotes the inertia matrix in joint space.

7.9.1.2 Inertia in operational space

Being able to calculate $\mathbf{M}(\mathbf{q})$ allows inertia to be cancelled out in joint-space by incorporating it into the control signal, but to cancel out the inertia of the system in operational space more work is still required. The first step will be calculating the acceleration in operational space. This can be found by taking the time derivative of Eq. 7.31:

$$\frac{d}{dt}\dot{\mathbf{x}} = \frac{d}{dt}(\mathbf{J}_{ee}(\mathbf{q})\,\dot{\mathbf{q}}), \tag{7.71}$$

$$\ddot{\mathbf{x}} = \dot{\mathbf{J}}_{ee}(\mathbf{q})\,\dot{\mathbf{q}} + \mathbf{J}_{ee}(\mathbf{q})\,\ddot{\mathbf{q}}. \tag{7.72}$$

Substituting in the dynamics of the system, Eq. 7.59, but ignoring the effects of gravity for now, gives:

$$\ddot{\mathbf{x}} = \dot{\mathbf{J}}_{ee}(\mathbf{q})\,\dot{\mathbf{q}} + \mathbf{J}_{ee}(\mathbf{q})\,\mathbf{M}^{-1}(\mathbf{q})[\mathbf{u} - \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})]. \tag{7.73}$$

Define the control signal

$$\mathbf{u} = \mathbf{J}_{ee}^T(\mathbf{q})\mathbf{F}_{\mathbf{x}}, \tag{7.74}$$

where substituting in for $\mathbf{F}_{\mathbf{x}}$, the desired end-effector force, gives

$$\mathbf{u} = \mathbf{J}_{ee}^T(\mathbf{q})\,\mathbf{M}_{\mathbf{x}_{ee}}(\mathbf{q})\,\ddot{\mathbf{x}}_{\text{des}}, \tag{7.75}$$

where $\ddot{\mathbf{x}}_{\text{des}}$ denotes the desired end-effector acceleration. Substituting the above equation into Eq. 7.73 gives

$$\ddot{\mathbf{x}} = \dot{\mathbf{J}}_{ee}(\mathbf{q})\,\dot{\mathbf{q}} + \mathbf{J}_{ee}(\mathbf{q})\,\mathbf{M}^{-1}(\mathbf{q})[\mathbf{J}_{ee}^T(\mathbf{q})\,\mathbf{M}_{\mathbf{x}_{ee}}(\mathbf{q})\,\ddot{\mathbf{x}}_{\text{des}} - \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})]. \tag{7.76}$$

Rearranging terms leads to

$$\ddot{\mathbf{x}} = \mathbf{J}_{ee}(\mathbf{q})\,\mathbf{M}^{-1}(\mathbf{q})\,\mathbf{J}_{ee}^T(\mathbf{q})\,\mathbf{M}_{\mathbf{x}_{ee}}(\mathbf{q})\,\ddot{\mathbf{x}}_{\text{des}} + [\dot{\mathbf{J}}_{ee}(\mathbf{q})\,\dot{\mathbf{q}} - \mathbf{J}_{ee}(\mathbf{q})\mathbf{M}^{-1}(\mathbf{q})\,\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})], \tag{7.77}$$

the last term is ignored due to the complexity of modeling it (this will be corrected in Chapter 4), resulting in

$$\ddot{\mathbf{x}} = \mathbf{J}_{ee}(\mathbf{q})\,\mathbf{M}^{-1}(\mathbf{q})\mathbf{J}_{ee}^T(\mathbf{q})\,\mathbf{M}_{\mathbf{x}_{ee}}(\mathbf{q})\,\ddot{\mathbf{x}}_{\text{des}}. \tag{7.78}$$

At this point, to get the dynamics $\ddot{\mathbf{x}}$ to be equal to the desired acceleration $\ddot{\mathbf{x}}_{\text{des}}$, the end-effector inertia matrix $\mathbf{M}_{\mathbf{x}_{ee}}$ needs to be chosen carefully. By setting

$$\mathbf{M}_{\mathbf{x}_{ee}}(\mathbf{q}) = [\mathbf{J}_{ee}(\mathbf{q})\,\mathbf{M}^{-1}(\mathbf{q})\,\mathbf{J}_{ee}^T(\mathbf{q})]^{-1}, \tag{7.79}$$

Eq. 7.78 becomes

$$\ddot{\mathbf{x}} = \mathbf{J}_{ee}(\mathbf{q})\,\mathbf{M}^{-1}(\mathbf{q})\mathbf{J}_{ee}^T(\mathbf{q})\,[\mathbf{J}_{ee}(\mathbf{q})\,\mathbf{M}^{-1}(\mathbf{q})\,\mathbf{J}_{ee}^T(\mathbf{q})]^{-1}\,\ddot{\mathbf{x}}_{\text{des}}, \tag{7.80}$$

$$\ddot{\mathbf{x}} = \ddot{\mathbf{x}}_{\text{des}}. \tag{7.81}$$

7.9.2 Accounting for gravity

With the forces of inertia accounted for, gravity will now be addressed. To compensate for gravity the concept of conservation of energy (i.e. the work done by gravity is the same in all coordinate systems) will once again be used. The controller operates by applying torque on joints, so it is necessary to be able to calculate the effect of gravity in joint space to cancel it out.

While the effect of gravity in joint space isn't obvious, it is quite easily defined in Cartesian coordinates in the base frame of reference. Here, the work done by gravity is simply the summation of the distance each link's center of mass has moved multiplied by the force of gravity. Where the force of gravity in Cartesian space is the mass of the object multiplied by -9.8m/s^2 along the z axis, the equation for the work done by gravity is written

$$\mathbf{W}_g = \Sigma_{i=0}^{n}(\mathbf{F}_{gi}^T \dot{\mathbf{x}}_i), \tag{7.82}$$

where $\mathbf{F}_{g_i}$ is the force of gravity on the ith arm segment. Because of the conservation of energy, the equation for work is equivalent when calculated in joint space, substituting into the above equation with Eq. 7.32:

$$\mathbf{F}_{\mathbf{q}}^T \dot{\mathbf{q}} = \Sigma_{i=0}^{n}(\mathbf{F}_{gi}^T \dot{\mathbf{x}}_i), \tag{7.83}$$

and then substitute in using $\dot{\mathbf{x}}_i = \mathbf{J}_i(\mathbf{q})\ \dot{\mathbf{q}}$,

$$\mathbf{F}_{\mathbf{q}}^T \dot{\mathbf{q}} = \Sigma_{i=0}^{n}(\mathbf{F}_{gi}^T \mathbf{J}_i(\mathbf{q})\ \dot{\mathbf{q}}), \tag{7.84}$$

and cancelling out the $\dot{\mathbf{q}}$ terms on both sides,

$$\begin{aligned} \mathbf{F}_{\mathbf{q}}^T &= \Sigma_{i=0}^{n}(\mathbf{F}_{gi}^T \mathbf{J}_i(\mathbf{q})), && (7.85) \\ \mathbf{F}_{\mathbf{q}} &= \Sigma_{i=0}^{n}(\mathbf{J}_i^T(\mathbf{q})\mathbf{F}_{g_i}) = \mathbf{g}(\mathbf{q}), && (7.86) \end{aligned}$$

which says that to find the effect of gravity in joint space simply multiply the mass of each link by its Jacobian, multiplied by the force of gravity in (x, y, z) space, and sum over each link. This summation gives the total effect of the gravity on the system.

References

[1] A Lorinez. Common control principles of basal ganglia - thalamocortical loops and the hippocampus. *Neural Network World*, pages 1–30, 1997.

[2] Mohamed Nabeel Abdelghani. *The Role of Sensitivity Derivatives in Sensorimotor Learning*. PhD thesis, University of Toronto, 2011.

[3] Afsheen Afshar, Gopal Santhanam, Byron M Yu, Stephen I Ryu, Maneesh Sahani, and Krishna V Shenoy. Single-trial neural correlates of arm movement preparation. *Neuron*, 71(3):555–564, 2011.

[4] Garrett E Alexander and Michael D Crutcher. Neural representations of the target (goal) of visually guided arm movements in three motor areas of the monkey. *J Neurophysiol*, 64(1):164–178, 1990.

[5] Mehran Emadi Andani, Fariba Bahrami, Parviz Jabehdar Maralani, and Auke Jan Ijspeert. MODEM: a multi-agent hierarchical structure to model the human motor control system. *Biological cybernetics*, 101(5-6):361–77, December 2009.

[6] Mehran Emadi Andani, Fariba Bahrami, Parviz Jabehdar Maralani, and Auke Jan Ijspeert. Modem: a multi-agent hierarchical structure to model the human motor control system. *Biological cybernetics*, 101(5-6):361–377, 2009.

[7] James Ashe and Apostolos P Georgopoulos. Movement parameters and neural activity in motor cortex and area 5. *Cerebral Cortex*, 4(6):590–600, 1994.

[8] M Athans. The role and use of the stochastic Linear-Quadratic-Gaussian problem in control system design. *Transaction on automatic control*, 16(6):529–552, 1971.

[9] Frederico A.C. Azevedo, Ludmila R.B. Carvalho, Lea T. Grinberg, Jos Marcelo Farfel, Renata E.L. Ferretti, Renata E.P. Leite, Wilson Jacob Filho, Roberto Lent, and

Suzana Herculano-Houzel. Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *The Journal of Comparative Neurology*, 513(5):532–541, 2009.

[10] Izhar Bar-Gad, Genela Morris, and Hagai Bergman. Information processing, dimensionality reduction and reinforcement learning in the basal ganglia. *Progress in neurobiology*, 71(6):439–73, December 2003.

[11] Amy Bastian. Learning to predict the future: the cerebellum adapts feedforward movement control. *Current Opinions in Neurobiology*, 16(6):645–649, 2006.

[12] Amy Bastian. Understanding sensorimotor adaptation and learning for rehabilitation. *Current opinion in neurology*, 21(6):628, 2008.

[13] Trevor Bekolay. Learning in large-scale spiking neural networks. Master's thesis, University of Waterloo, Waterloo, ON, 09/2011 2011.

[14] Ben Varkey Benjamin, Peiran Gao, Emmett McQuinn, Swadesh Choudhary, Anand R Chandrasekaran, J Bussat, Rodrigo Alvarez-Icaza, John V Arthur, PA Merolla, and Kwabena Boahen. Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE*, 102(5):699–716, 2014.

[15] Max Berniker, Anthony Jarc, Emilio Bizzi, and Matthew C Tresch. Simplified and effective motor control based on muscle synergies to exploit musculoskeletal dynamics. *Proceedings of the National Academy of Sciences of the United States of America*, 106(18):7601–6, May 2009.

[16] Bennett Bertenthal and Claes Von Hofsten. Eye, head and trunk control: the foundation for manual development. *Neuroscience & Biobehavioral Reviews*, 22(4):515–520, 1998.

[17] Aude Billard, Yann Epars, Sylvain Calinon, Stefan Schaal, and Gordon Cheng. Discovering optimal imitation strategies. *Robotics and Autonomous Systems*, 47(2-3):69–77, June 2004.

[18] C. A. Bloxham, T. A. Mindel, and C. D. Frith. Initiation and execution of predictable and unpredictable movements in parkinson's disease. *Brain*, 107(2):371–384, 1984.

[19] Hal Blumenfield. *Neuroanatomy (Blumenfeld,Neuroanatomy through Clinical Cases)*. Sinauer Associates, Inc, 2002.

[20] G Bortoff and PL Strick. Corticospinal terminations in two new-world primates: further evidence that corticomotoneuronal connections provide part of the neural substrate for manual dexterity. *The Journal of neuroscience : the official journal of the Society for Neuroscience*, 13(12):5105–5118, December 1993.

[21] Andreea C Bostan, Richard P Dum, and Peter L Strick. The basal ganglia communicate with the cerebellum. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 107, pages 8452–8456, 2010.

[22] Jonas Buchli, Freek Stulp, Evangelos Theodorou, and Stefan Schaal. Learning variable impedance control. *The International Journal of Robotics Research*, 30(7):820–833, 2011.

[23] Daniel Bullock and Stephen Grossberg. Vite and flete: Neural modules for trajectory formation and postural control. *Advances in Psychology*, 62:253–297, 1989.

[24] Daniele Caligiore, Tomassino Ferrauto, Domenico Parisi, Neri Accornero, Marco Capozza, and Gianluca Baldassarre. Using motor babbling and hebb rules for modeling the development of reaching with obstacles and grasping. *International Conference on Cognitive Systems*, page E18, 2008.

[25] Daniele Caligiore, Giovanni Pezzulo, R Chris Miall, and Gianluca Baldassarre. The contribution of brain sub-cortical loops in the expression and acquisition of action understanding abilities. *Neuroscience & Biobehavioral Reviews*, 37(10):2504–2515, 2013.

[26] R Caminiti, PB Johnson, C Galli, S Ferraina, Y Burnod, and A Urbano. Making arm movements within different parts of space: the premotor and motor cortical representation of a coordinate system for reaching to visual targets. *J Neurosci*, 11(5):1182–1197, 1991.

[27] Adam F Carpenter, Apostolos P Georgopoulos, and Giuseppe Pellizzer. Motor cortical encoding of serial order in a context-recall task. *Science*, 283(5408):1752–1757, 1999.

[28] Sherwin S Chan and Daniel W Moran. Computational model of a primate arm: from hand position to joint angles, joint torques and muscle forces. *Journal of neural engineering*, 3(4):327–37, December 2006.

[29] Chien-Chern Cheah, Chao Liu, and Jean-Jacques E Slotine. Adaptive tracking control for robots with unknown kinematic and dynamic properties. *The International Journal of Robotics Research*, 25(3):283–296, 2006.

[30] Mark M Churchland, John P Cunningham, Matthew T Kaufman, Justin D Foster, Paul Nuyujukian, Stephen I Ryu, and Krishna V Shenoy. Neural population dynamics during reaching. *Nature*, 2012.

[31] Mark M. Churchland, John P. Cunningham, Matthew T. Kaufman, Stephen I. Ryu, and Krishna V. Shenoy. Cortical preparatory activity: Representation of movement or first cog in a dynamical machine? *Neuron*, 68(3):387 – 400, 2010.

[32] Mark M Churchland, John P Cunningham, Matthew T Kaufman, Stephen I Ryu, and Krishna V Shenoy. Cortical preparatory activity: representation of movement or first cog in a dynamical machine? *Neuron*, 68(3):387–400, 2010.

[33] Paul Cisek, Donald J Crammond, and John F Kalaska. Neural activity in primary motor and dorsal premotor cortex in reaching tasks with the contralateral versus ipsilateral arm. *Journal of neurophysiology*, 89(2):922–942, 2003.

[34] Alessandro Crespi and Auke Jan Ijspeert. Salamandra robotica : a biologically inspired amphibious robot that swims and walks. *Robotics*, pages 3–36, 2009.

[35] M. da Silva, F. Durand, and J. Popovic. Linear bellman combination for control of character animation. In *ACM Trans on Graphics (Proc. SIGGRAPH)*, volume 28, pages 82:1–82:10, 2009.

[36] Christian Darlot, L Zupan, O Etard, P Denise, and A Maruani. Computation of inverse dynamics for the control of movements. *Biological cybernetics*, 75(2):173–186, 1996.

[37] Andrea d'Avella, Philippe Saltiel, and Emilio Bizzi. Combinations of muscle synergies in the construction of a natural motor behavior. *Nature neuroscience*, 6(3):300–308, 2003.

[38] Yiannis Demiris and Andrew Meltzoff. The robot in the crib : A developmental analysis of imitation skills in infants and robots. *Infant and Child Development*, 53:43–53, 2008.

[39] Sophie Deneve, Alexandre Pouget, and New York. Basis Functions for Object-Centered Representations. *Neuron*, 37:347–359, 2003.

[40] T. DeWolf. NOCH : A framework for biologically plausible models of neural motor control. Master's thesis, University of Waterloo, 2010.

[41] T. DeWolf and C. Eliasmith. The neural optimal control hierarchy for motor control. *Journal of neural engineering*, 8(6):065009, 2011.

[42] T. DeWolf, T. Stewart, J.J. Slotine, and C. Eliasmith. Methods and systems for nonlinear adaptive control and filtering. *Patent pending*, (US 23208-P46821 US00), 11 2014.

[43] Travis DeWolf and Chris Eliasmith. A neural model of the development of expertise. In *The 12th International Conference on Cognitive Modelling*, Ottawa, Ontario, 2013.

[44] K Doya. What are the computations of the cerebellum, the basal ganglia and the cerebral cortex. *Neural Networks*, 12:961–974, 1999.

[45] Kenji Doya. Complementary roles of basal ganglia and cerebellum in learning and motor control. *Current opinion in neurobiology*, 10(6):732–739, 2000.

[46] Kenji Doya. How can we learn efficiently to act optimally and flexibly? *Proceedings of the National Academy of Sciences of the United States of America*, 106(28):11429–30, July 2009.

[47] Julien Doyon, Pierre Bellec, Rhonda Amsel, Virginia Penhune, Oury Monchi, Julie Carrier, Stéphane Lehéricy, and Habib Benali. Contributions of the basal ganglia and functionally related brain structures to motor learning. *Behavioural brain research*, 199(1):61–75, 2009.

[48] Aaron D'Souza, Sethu Vijayakumar, and Stefan Schaal. Learning inverse kinematics. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 1, pages 298–303. IEEE, 2001.

[49] Chris Eliasmith and Charles H Anderson. *Neural engineering: Computation, representation, and dynamics in neurobiological systems.* MIT Press, 2004.

[50] Chris Eliasmith, Terrence C Stewart, Xuan Choo, Trevor Bekolay, Travis DeWolf, Yichuan Tang, and Daniel Rasmussen. A large-scale model of the functioning brain. *science*, 338(6111):1202–1205, 2012.

[51] Tom Erez, Kendall Lowrey, Yuval Tassa, Vikash Kumar, Svetoslav Kolev, and Emanuel Todorov. An integrated system for real-time model predictive control of humanoid robots. In *International Conference on Humanoid Robots*, 2013.

[52] Edward V Evarts. Relation of pyramidal tract activity to force exerted during voluntary movement. *J Neurophysiol*, 31(1):14–27, 1968.

[53] D J Felleman and D C Van Essen. Distributed hierarchical processing in the primate cerebral cortex. *Cerebral cortex (New York, N.Y. : 1991)*, 1(1):1–47, 1991.

[54] Alon Fishbach, Stephane a Roy, Christina Bastianen, Lee E Miller, and James C Houk. Deciding when and how to correct a movement: discrete submovements as a decision making process. *Experimental brain research. Experimentelle Hirnforschung. Expérimentation cérébrale*, 177(1):45–63, February 2007.

[55] James Fix. *Atlas of the human brain and spinal cord*. Jones & Bartlett Learning, 2007.

[56] D Flament and J Hore. Relations of motor cortex neural discharge to kinematics of passive and active elbow movements in the monkey. *Journal of neurophysiology*, 60(4):1268–1284, 1988.

[57] J Randall Flanagan, Philipp Vetter, Roland S Johansson, and Daniel M Wolpert. Prediction precedes control in motor learning. *Current Biology*, 13(2):146–150, 2003.

[58] PIERRE A Fortier, JOHN F Kalaska, and ALLAN M Smith. Cerebellar neuronal activity related to whole-arm reaching movements in the monkey. *J Neurophysiol*, 62(1):198–211, 1989.

[59] I Fried, A Katz, G McCarthy, K J Sass, P Williamson, S S Spencer, and D D Spencer. Functional organization of human supplementary motor cortex studied by electrical stimulation. *Journal of Neuroscience*, 11(11):3656–3666, 1991.

[60] QG Fu, D Flament, JD COLT&, and TJ Ebner. Temporal encoding of movement kinematics in the discharge. *J Neurophysiol*, 1995.

[61] Alexander Gail and Richard a Andersen. Neural dynamics in monkey parietal reach region reflect context-specific sensorimotor transformations. *The Journal of neuroscience : the official journal of the Society for Neuroscience*, 26(37):9376–84, September 2006.

[62] Thomas Geijtenbeek, Michiel van de Panne, and A Frank van der Stappen. Flexible muscle-based locomotion for bipedal creatures. *ACM Transactions on Graphics (TOG)*, 32(6):206, 2013.

[63] Maurizio Gentilucci and Anna Negrotti. The control of an action in Parkinson s disease. *Experimental Brain Research*, pages 269–277, 1999.

[64] A P Georgopoulos, J T Lurito, M Petrides, A B Schwartz, and J T Massey. Mental rotation of the neuronal population vector. *Science*, 243(4888):234–6, 1989.

[65] A P Georgopoulos, A B Schwartz, and R E Kettner. Neuronal population coding of movement direction. *Science*, 233(4771):1416–1419, 1986.

[66] Apostolos P Georgopoulos, John F Kalaska, Roberto Caminiti, and Joe T Massey. On the relations between the direction of two-dimensional arm movements and cell discharge in primate motor cortex. *The Journal of Neuroscience*, 2(11):1527–1537, 1982.

[67] S. B. Gershwin. A Hierarchical Framework for Discrete Event Scheduling in Manufacturing Systems. In P. Varaiya and H. Kurzhanski, editors, *Discrete Event Systems: Models and Application*, volume 103 of *LNCIS*, pages 197–216, Berlin, Germany, August 1987. Springer-Verlag.

[68] Claude Ghez and Thomas Thach. *The Cerebellum*, chapter 42. McGraw-Hill Professional Publishing, 4th edition, 2000.

[69] Claude Ghez and Thomas Thach. *The Organization of Movement*, chapter 33. McGraw-Hill Professional Publishing, 4th edition, 2000.

[70] Simon F Giszter, Ferdinando A Mussa-Ivaldi, and Emilio Bizzi. Convergent force fields organized in the frog's spinal cord. *The journal of neuroscience*, 13(2):467–491, 1993.

[71] M A Gluck, M T Allen, C E Myers, and R F Thompson. Cerebellar substrates for error correction in motor conditioning. *Neurobiology of learning and memory*, 76(3):314–41, November 2001.

[72] A. Gopnik, A.N. Meltzoff, and P.K. Kuhl. *The scientist in the crib: What early learning tells us about the mind*. Perennial, 2000.

[73] Michael Graziano. The organization of behavioral repertoire in motor cortex. *Annu. Rev. Neurosci.*, 29:105–134, 2006.

[74] Stephen Grossberg and Michael Kuperstein. *Neural dynamics of adaptive sensory-motor control: Ballistic eye movements*, volume 30. Elsevier, 2011.

[75] Leonidas Guibas, Micha Sharir, and Shmuel Sifrony. On the general motion planning problem with two degrees of freedom. *Proceedings of the 4th Annual Symposium on Computational Geometry*, pages 289–298, 1988.

[76] K Gurney, T J Prescott, and P Redgrave. a. A computational model of action selection in the basal ganglia. I. A new functional anatomy. *Biol. Cybern.*, 84:401–410, 2001.

[77] Kevin Gurney, Tony J Prescott, and Peter Redgrave. A computational model of action selection in the basal ganglia. i. a new functional anatomy. *Biological cybernetics*, 84(6):401–410, 2001.

[78] S Hanneton, Alain Berthoz, Jacques Droulez, and Jean-Jacques E Slotine. Does the brain use sliding variables for the control of movements? *Biological cybernetics*, 77(6):381–393, 1997.

[79] Peter S Harper. Review article The epidemiology of Huntington ' s disease. *Human Genetics*, pages 365–376, 1992.

[80] M Haruno, D M Wolpert, and M Kawato. Mosaic model for sensorimotor learning and control. *Neural computation*, 13(10):2201–20, October 2001.

[81] Masahiko Haruno, D Wolpert, and Mitsuo Kawato. Mosaic model for sensorimotor learning and control. *Neural computation*, 13(10):2201–2220, 2001.

[82] Masahiko Haruno, Daniel M Wolpert, and Mitsuo Kawato. Hierarchical mosaic for movement generation. In *International congress series*, volume 1250, pages 575–590. Elsevier, 2003.

[83] G E Hinton and R R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science (New York, N.Y.)*, 313(5786):504–7, July 2006.

[84] S Hocherman and SP Wise. Effects of hand movement path on motor cortical activity in awake, behaving rhesus monkeys. *Experimental brain research*, 83(2):285–302, 1991.

[85] D S Hoffman and P L Strick. Effects of a primary motor cortex lesion on step-tracking movements of the wrist. *Journal of Neurophysiology*, 73(2):891–895, 1995.

[86] R.N. Holdefer and L.E. Miller. Primary motor cortical neurons encode functional muscle synergies. *Experimental Brain Research*, 146(2):233–243, 2002.

[87] Scott L Hooper and Ralph a DiCaprio. Crustacean motor pattern generator networks. *Neuro-Signals*, 13(1-2):50–69, 2004.

[88] Scott L Hooper and Ralph a DiCaprio. Crustacean motor pattern generator networks. *Neuro-Signals*, 13(1-2):50–69, 2004.

[89] Theo Hopman. Introduction to indicial notation. *Dep. of Physics, University of Guelph*, 2002.

[90] Eiji Hoshi, Léon Tremblay, Jean Féger, Peter L Carras, and Peter L Strick. The cerebellum communicates with the basal ganglia. *Nature neuroscience*, 8(11):1491–3, November 2005.

[91] J C Houk, C Bastianen, D Fansler, a Fishbach, D Fraser, P J Reber, S a Roy, and L S Simo. Action selection and refinement in subcortical loops through basal ganglia and cerebellum. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 362(1485):1573–83, September 2007.

[92] Eun Jung Hwang and Reza Shadmehr. Internal models of limb dynamics and the encoding of limb state. *Journal of neural engineering*, 2(3):S266–78, September 2005.

[93] Eun Jung Hwang and Reza Shadmehr. Internal models of limb dynamics and the encoding of limb state. *Journal of neural engineering*, 2(3):S266–78, September 2005.

[94] Aapo Hyvrinen and Patrik Hoyer. Emergence of topography and complex cell properties from natural images using extensions of ica. In *In Advances in Neural Information Processing Systems*, pages 827–833. MIT Press, 1999.

[95] A. Ijspeert, J. Nakanishi, P Pastor, H. Hoffmann, and S. Schaal. Dynamical movement primitives: Learning attractor models formotor behaviors. *Neural Computation*, 25:328–373, 2013.

[96] Auke Jan Ijspeert, Alessandro Crespi, Dimitri Ryczko, and Jean-Marie Cabelguen. From swimming to walking with a salamander robot driven by a spinal cord model. *Science (New York, N.Y.)*, 315(5817):1416–20, March 2007.

[97] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373, 2013.

[98] D.G. Ivashko, B.I. Prilutsky, S.N. Markin, J.K. Chapin, and I.a. Rybak. Modeling the spinal cord neural circuitry controlling cat hindlimb movement during locomotion. *Neurocomputing*, 52-54:621–629, June 2003.

[99] Eugene M Izhikevich et al. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572, 2003.

[100] Lena Ting J. Lucas Mckay. Muscle synergies produce expensive behavioral biases during postural control. In *Neural Control of Movement 2010*, pages 14–15, 2010.

[101] Marc Jeannerod. *The neural and behavioural organization of goal-directed movements.* Clarendon Press/Oxford University Press, 1988.

[102] Daphna Joel, Yael Niv, and Eytan Ruppin. Actor–critic models of the basal ganglia: New anatomical and computational perspectives. *Neural networks*, 15(4):535–547, 2002.

[103] R Jung, T Kiemel, and A H Cohen. Dynamic behavior of a neural network model of locomotor control in the lamprey. *Journal of Neurophysiology*, 75(3):1074–1086, 1996.

[104] M. Kalakrishnan, L. Righetti, P. Pastor, and S. Schaal. Learning force control policies for compliant robotic manipulation. In *International Conference on Machine Learning (ICML)*, 2012.

[105] J F Kalaska, D A Cohen, M L Hyde, and M Prudhomme. A comparison of movement direction-related versus load direction-related activity in primate motor cortex, using a two-dimensional reaching task. *Journal of Neuroscience*, 9(6):2080–2102, 1989.

[106] JF Kalaska, DAD Cohen, M Prud'Homme, and ML Hyde. Parietal area 5 neuronal activity encodes movement kinematics, not movement dynamics. *Experimental Brain Research*, 80(2):351–364, 1990.

[107] Mitsuo Kawato. Internal models for motor control and trajectory planning. *Current Opinion in Neurobiology*, 9(6):718–727, December 1999.

[108] Mitsuo Kawato. Internal models for motor control and trajectory planning. *Current opinion in neurobiology*, 9(6):718–727, 1999.

[109] Mitsuo Kawato and Hiroaki Gomi. A computational model of four regions of the cerebellum based on feedback-error learning. *Biological cybernetics*, 68(2):95–103, 1992.

[110] Mitsuo Kawato, Tomoe Kuroda, Hiroshi Imamizu, Eri Nakano, Satoru Miyauchi, and Toshinori Yoshioka. Internal forward models in the cerebellum: fmri study on grip force and load force coupling. *Progress in brain research*, 142:171–188, 2003.

[111] Ronald E Kettner, Andrew B Schwartz, and Apostolos P Georgopoulos. Primate motor cortex and free arm movements to visual targets in three-dimensional space. iii. positional gradients and population coding of movement direction from various movement origins. *The journal of Neuroscience*, 8(8):2938–2947, 1988.

[112] Oussama Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *Robotics and Automation, IEEE Journal of*, 3(1):43–53, 1987.

[113] Pradeep K Khosla and Takeo Kanade. Parameter identification of robot dynamics. In *Decision and Control, 1985 24th IEEE Conference on*, volume 24, pages 1754–1760. IEEE, 1985.

[114] DL Kleinman. An optimal control model of human response part i: Theory and validation. *Automatica*, 3:357–369, 1970.

[115] Shinya Kuroda, Kenji Yamamoto, Hiroyuki Miyamoto, Kenji Doya, and Mitsuo Kawato. Statistical characteristics of climbing ber spikes necessary for e cient cerebellar learning. *Science And Technology*, 192:183–192, 2001.

[116] Rodrigo Laje and Dean V Buonomano. Robust timing and motor patterns by taming chaos in recurrent neural networks. *Nature neuroscience*, 16(7):925–933, 2013.

[117] Honglak Lee, C Ekanadham, and A Ng. Sparse deep belief net model for visual area v2. *Energy*, 19:1–8, 2007.

[118] Martin Lemay, Sylvain Chouinard, François Richer, and Paul Lesperance. Huntington's disease affects movement termination. *Behavioural brain research*, 187(1):153–8, February 2008.

[119] Weiwei Li, Emanuel Todorov, and Xiuchuan Pan. Hierarchical feedback and learning for multi-joint arm movement control. *Conference Proceedings of the International Conference of IEEE Engineering in Medicine and Biology Society*, 4:4400–4403, 2005.

[120] D Liu and E Todorov. Hierarchical optimal control of a 7-dof arm model. *2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pages(2):50–57, 2009.

[121] David MacNeil and Chris Eliasmith. Fine-tuning and the stability of recurrent neural networks. *PLoS ONE*, 6, 2011.

[122] M A Maier, E Olivier, S N Baker, P A Kirkwood, T Morris, and R N Lemon. Direct and indirect corticospinal control of arm and hand motoneurons in the squirrel monkey (saimiri sciureus). *Journal of Neurophysiology*, 78(2):721–733, 1997.

[123] NK Mani, EJ Haug, and KE Atkinson. Application of singular value decomposition for analysis of mechanical system dynamics. *Journal of Mechanical Design*, 107(1):82–87, 1985.

[124] Yoshiya Matsuzaka, HIROSHI Aizawa, and Jun Tanji. A motor area rostral to the supplementary motor area (presupplementary motor area) in the monkey: neuronal activity during a learned motor task. *Journal of Neurophysiology*, 68:653–653, 1992.

[125] Maurizio Mattia, Pierpaolo Pani, Giovanni Mirabella, Stefania Costa, Paolo Del Giudice, and Stefano Ferraina. Heterogeneous attractor cell assemblies for motor planning in premotor cortex. *The Journal of Neuroscience*, 33(27):11155–11168, 2013.

[126] Neil McKay. *Minimum cost-control of robotic manipulators with geometric path constraints*. PhD thesis, University of Michigan, 1985.

[127] Thomas A. McMahon. *Muscles, Reflexes and Locomotion.* Princeton University Press, 1984.

[128] F. Meier, P. Hennig, and S. Schaal. Efficient bayesian local model learning for control. In *Proceedings of the IEEE International Conference on Intelligent Robotics Systems*, 2014.

[129] Andrew Meltzoff and M. Keith Moore. Explaining facial imitation: A theoretical model. *Exarly Development and Parenting*, 6:179–192, 1997.

[130] Samir Menon, Sam Fok, Alex Neckar, Oussama Khatib, and Kwabena Boahen. Controlling articulated robots in task-space with spiking silicon neurons. In *IEEE International Conference on Biomedical Robitics and Biomechatronics*. IEEE Press, 2014.

[131] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[132] RC Miall and Daniel M Wolpert. Forward models for physiological motor control. *Neural networks*, 9(8):1265–1279, 1996.

[133] John G. Milton, Steven S. Small, and Ana Solodkin. On the road to automatic: Dynamic aspects in the development of expertise. *Journal of Clinical Neurophysiology*, 21:134–143, 2004.

[134] Jonathan W Mink. The basal ganglia: focused selection and inhibition of competing motor programs. *Progress in neurobiology*, 50(4):381–425, 1996.

[135] Daniel W Moran and Andrew B Schwartz. Motor cortical representation of speed and direction during reaching. *Journal of Neurophysiology*, 82(5):2676–2692, 1999.

[136] M. Morari, C.E. Garcia, and D. M. Prett. Model predictive control: Theory and practice–A survey. *Automatica*, 25(3):335–348, 1989.

[137] Igor Mordatch, Zoran Popović, and Emanuel Todorov. Contact-invariant optimization for hand manipulation. In *Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation*, pages 137–144. Eurographics Association, 2012.

[138] Igor Mordatch, Emanuel Todorov, and Zoran Popović. Discovery of complex behaviors through contact-invariant optimization. *ACM Transactions on Graphics (TOG)*, 31(4):43, 2012.

[139] Susanne M Morton and Amy J Bastian. Cerebellar control of balance and locomotion. *The Neuroscientist*, 10(3):247–259, 2004.

[140] Susanne M Morton and Amy J Bastian. Cerebellar control of balance and locomotion. *The Neuroscientist : a review journal bringing neurobiology, neurology and psychiatry*, 10(3):247–59, June 2004.

[141] Grant H. Mulliken, Sam Musallam, and Richard A. Andersen. Decoding trajectories from posterior parietal cortex ensembles. *The Journal of neuroscience : the official journal of the Society for Neuroscience*, 28(48):12913–12926, November 2008.

[142] Z Nenadic, Charles H Anderson, and B Ghosh. Control of arm movement using population of neurons. *Proceedings of the 39th IEEE Conference on Decision and Control Cat No00CH37187*, 35(11-12):1776–1781, 2000.

[143] Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15(3):267–273, 1982.

[144] B A Olshausen and D J Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607–609, 1996.

[145] B.A. Olshausen. *Sparse codes and spikes*, pages 257–272. MIT Press, 2002.

[146] Eustace Painkras, Luis A Plana, Jim Garside, Steve Temple, Francesco Galluppi, Cameron Patterson, David R Lester, Andrew D Brown, and Steve B Furber. Spinnaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation. *Solid-State Circuits, IEEE Journal of*, 48(8):1943–1953, 2013.

[147] Christopher Parisien, Charles H. Anderson, and Chris Eliasmith. Solving the problem of negative synaptic weights in cortical models. *Neural Computation*, 20:1473–1494, 2008.

[148] Wilder Penfield and T Rasmussen. *The Cerebral Cortex of Man*, volume 40. Macmillan, 1950.

[149] Jan Peters and Stefan Schaal. Reinforcement learning by reward-weighted regression for operational space control. *Proceedings of the 24th International Conference on Machine Learning (2007)*, pages 745–750, 2007.

[150] J. Piaget. *The origins of intelligence in children.* New York: International University Press, 1952.

[151] Alexandre Pouget and Terrence J. Sejnowski. Spatial transformations in the parietal cortex using basis functons. *Journal of Cognitive Neuroscience*, 9(2):222–237, 1997.

[152] Daniel Rasmussen and Chris Eliasmith. A neural reinforcement learning model for tasks with unknown time delays. In *35th Annual Conference of the Cognitive Science Society*, pages 3257–3262, 2013.

[153] Daniel Rasmussen and Chris Eliasmith. A neural model of hierarchical reinforcement learning. In Paul Bello, Marcello Guarini, Marjorie McShane, and Brian Scassellati, editors, *Proceedings of the 36th Annual Conference of the Cognitive Science Society*, pages 1252–1257, Austin, 2014. Cognitive Science Society.

[154] Alexa Riehle and Eilon Vaadia, editors. *Motor Cortex in Voluntary Movements: A distributed system for distributed functions.* CRC Press, 2005.

[155] Michel Rijntjes, Christian Dettmers, Christian Büchel, Stefan Kiebel, Richard SJ Frackowiak, and Cornelius Weiller. A blueprint for movement: functional and

anatomical representations in the human motor system. *The Journal of neuroscience*, 19(18):8043–8048, 1999.

[156] Alexander V Roitman, Siavash Pasalar, Michael TV Johnson, and Timothy J Ebner. Position, direction of movement, and speed tuning of cerebellar purkinje cells during circular manual tracking in monkey. *The Journal of neuroscience*, 25(40):9244–9257, 2005.

[157] Jerome N Sanes and John P Donoghue. Plasticity and primary motor cortex. *Annual review of neuroscience*, 23:393–415, 2000.

[158] Robert M Sanner and J-JE Slotine. Gaussian networks for direct adaptive control. *Neural Networks, IEEE Transactions on*, 3(6):837–863, 1992.

[159] Stefan Schaal. Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In *Adaptive Motion of Animals and Machines*, pages 261–280. Springer, 2006.

[160] Stefan Schaal, Auke Ijspeert, and Aude Billard. Computational approaches to motor learning by imitation. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 358(1431):537–547, 2003.

[161] AB Schwartz. Direct cortical representation of drawing. *Science*, 265:540–542, 1994.

[162] Andrew B Schwartz. Direct cortical representation of drawing. *Science*, 265(5171):540–542, 1994.

[163] Andrew B. Schwartz, X. Tracy Cui, DouglasJ. Weber, and Daniel W. Moran. Brain-controlled interfaces: Movement restoration with neural prosthetics. *Neuron*, 52(1):205 – 220, 2006.

[164] Nicolas Schweighofer, Jacob Spoelstra, Michael A Arbib, and Mitsuo Kawato. Role of the cerebellum in reaching movements in humans. ii. a neural model of the intermediate cerebellum. *European Journal of Neuroscience*, 10(1):95–105, 1998.

[165] Stephen H Scott. Optimal feedback control and the neural basis of volitional motor control. *Nature Reviews Neuroscience*, 5(7):532–546, 2004.

[166] Stephen H Scott and John F Kalaska. Changes in motor cortex activity during reaching movements with similar hand paths but different arm postures. *Changes*, 73(6), 1995.

[167] Lauren E Sergio, Catherine Hamel-Pâquet, and John F Kalaska. Motor cortex neural correlates of output kinematics and kinetics during isometric-force and arm-reaching tasks. *Journal of neurophysiology*, 94(4):2353–2378, 2005.

[168] Reza Shadmehr and Ferdinando A Mussa-Ivaldi. Adaptive representation of dynamics during learning of a motor task. *The Journal of Neuroscience*, 14(5):3208–3224, 1994.

[169] KV Shenoy, D Meeker, and S Cao. Neural prosthetic control signal from plan activity. *Neuroreport*, 14:591–596, 2003.

[170] Jean-Jacques E Slotine and Weiping Li. On the adaptive control of robot manipulators. *The International Journal of Robotics Research*, 6(3):49–59, 1987.

[171] M a Smith, J Brandt, and R Shadmehr. Motor disorder in Huntington's disease begins as a dysfunction in error feedback control. *Nature*, 403(6769):544–9, February 2000.

[172] Maurice A Smith and Reza Shadmehr. Intact ability to learn internal models of arm dynamics in huntington's disease but not cerebellar degeneration. *Journal of Neurophysiology*, 93(5):2809–2821, 2005.

[173] JF Soechting. Effect of target size on spatial and temporal characteristics of a pointing movement in man. *Experimental Brain Research*, 54(1):121–132, 1984.

[174] Aaron D Souza, Sethu Vijayakumar, and Stefan Schaal. Learning inverse kinematics. In *Proceedings of IROS*, pages 298–303, 2001.

[175] Cesare Stefanini, Stefano Orofino, Luigi Manfredi, Stefano Mintchev, Stefano Marrazza, Tareq Assaf, L Capantini, Edoardo Sinibaldi, Sten Grillner, Peter Wallen, et al. A compliant bioinspired swimming robot with neuro-inspired control and autonomous behavior. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 5094–5098. IEEE, 2012.

[176] T.C. Stewart and C. Eliasmith. Large-scale synthesis of functional spiking neural circuits. *Proceedings of the IEEE*, 102(5):881–898, May 2014.

[177] Terrence Stewart and Robert West. Testing for equivalence: a methodology for computational cognitive modelling. *Journal of Artificial General Intelligence*, 2(2):69–87, 2010.

[178] Terrence C Stewart, Xuan Choo, and Chris Eliasmith. Dynamic Behaviour of a Spiking Model of Action Selection in the Basal Ganglia. In *International Conference on Cognitive Modeling*, pages 235–240, 2010.

[179] Norikazu Sugimoto, Masahiko Haruno, Kenji Doya, and Mitsuo Kawato. Mosaic for multiple-reward environments. *Neural computation*, 24(3):577–606, 2012.

[180] R S Sutton and A G Barto. Reinforcement Learning - an Introduction. *MIT Press, Cambridge, MA.*, 1998.

[181] Kenji Tahara, Suguru Arimoto, Masahiro Sekimoto, and Zhi-Wei Luo. On control of reaching movements for musculo-skeletal redundant arm model. *Applied Bionics and Biomechanics*, 6(1):11–26, 2009.

[182] Kenji Tahara and Hitoshi Kino. Reaching movements of a redundant musculoskeletal arm: Acquisition of an adequate internal force by iterative learning and its evaluation through a dynamic damping ellipsoid. *Advanced Robotics*, 24(5-6):783–818, 2010.

[183] M Taira, J Boline, N Smyrnis, A Georgopoulos, and J Ashe. On the relations between single cell activity in the motor cortex and the direction and magnitude of three-dimensional static isometric force. *Exp Brain Res*, 109:367–376, 1996.

[184] WT Thach. Correlation of neural discharge with pattern and force of muscular activity, joint position, and direction of intended next movement in motor cortex and cerebellum. *J Neurophysiol*, 41(3):654–676, 1978.

[185] E. A. Theodorou, J. Buchli, and S. Schaal. Learning policy improvements with path integrals. In *International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, 2010.

[186] E Todorov. Parallels between sensory and motor information processing. In *The Cognitive Neurosciences*. MIT Press, 4 edition, 2008.

[187] E Todorov. Compositionality of optimal control laws. *Advances in Neural Information Processing Systems*, 22:1856–1864, 2009.

[188] E Todorov, W Li, and X Pan. From task parameters to motor synergies: A hierarchical framework for approximately-optimal feedback control of redundant manipulators. *Journal of Robotic Systems*, vol:22pp691–710, 2005.

[189] Emanuel Todorov. Direct cortical control of muscle activation in voluntary arm movements: a model. *Nature neuroscience*, 3(4):391–398, 2000.

[190] Emanuel Todorov. On the role of primary motor cortex in arm movement control. *Cognitive Science*, 3:125–166, 2003.

[191] Emanuel Todorov. Efficient computation of optimal actions. *Proceedings of the national academy of sciences*, 106(28):11478–11483, 2009.

[192] Emanuel Todorov. Efficient computation of optimal actions. *Proceedings of the National Academy of Sciences of the United States of America*, 106(28):11478–83, July 2009.

[193] Emanuel Todorov and Michael I Jordan. Optimal feedback control as a theory of motor coordination. *Nature neuroscience*, 5(11):1226–1235, 2002.

[194] Emanuel Todorov and Weiwei Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *American Control Conference, 2005. Proceedings of the 2005*, pages 300–306. IEEE, 2005.

[195] Emmanuel Todorov. *Optimal Control Theory*, chapter 12. MIT Press, 2006.

[196] Mette Helene Toft, Ole Gredal, and Bente Pakkenberg. The size distribution of neurons in the motor cortex in amyotrophic lateral sclerosis. *Journal of anatomy*, 207(4):399–407, 2005.

[197] J C Trinkle, J M Abel, and R P Paul. An investigation of frictionless enveloping grasping in the plane. *International Journal Of Robotics Research*, 7(3):33–51, 1988.

[198] Bryan Tripp. *A search for principles of basal ganglia function.* PhD thesis, University of Waterloo, 2009.

[199] Ya-weng Tseng, Jörn Diedrichsen, John W Krakauer, Reza Shadmehr, and Amy J Bastian. Sensory prediction errors drive cerebellum-dependent adaptation of reaching. *Journal of Neurophysiology*, 98(1):54–62, 2007.

[200] Luca Turella, Andrea C Pierno, Federico Tubaldi, and Umberto Castiello. Mirror neurons in humans: consisting or confounding evidence? *Brain and language*, 108(1):10–21, January 2009.

[201] Robert S Turner, Michel Desmurget, Corresponding Turner, and Robert S. Basal ganglia contributions to motor control : a vigorous tutor. *Current Opinion in Neurobiology*, 20(6):704–716, 2010.

[202] Francisco J Valero-Cuevas, Madhusudhan Venkadesan, and Emanuel Todorov. Structured variability of muscle activations supports the minimal intervention principle of motor control. *Journal of neurophysiology*, 102(1):59–68, July 2009.

[203] M Velliste, S Perel, M C Spalding, A S Whitford, and A B Schwartz. Cortical control of a prosthetic arm for self-feeding. *Nature*, 453(7198):1098–1101, 2008.

[204] Sethu Vijayakumar, Aaron D'souza, and Stefan Schaal. Incremental online learning in high dimensions. *Neural computation*, 17(12):2602–2634, 2005.

[205] Tim Waegeman, Michiel Hermans, and Benjamin Schrauwen. Macop modular architecture with control primitives. *Frontiers in computational neuroscience*, 7, 2013.

[206] Z. Wang, C Lampert, K Muelling, B. Schoelkopf, and J. Peters. Learning anticipation policies for robot table tennis. In *IEEE/RSJ International Conference on Intelligent Robot Systems (IROS)*, 2011.

[207] a M Wing. Motor control: Mechanisms of motor equivalence in handwriting. *Current biology : CB*, 10(6):R245–8, March 2000.

[208] Alan M Wing. Motor control: Mechanisms of motor equivalence in handwriting. *Current Biology*, 10(6):R245–R248, 2000.

[209] Steven P Wise. The primate premotor cortex: past, present, and preparatory. *Annual review of neuroscience*, 8(1):1–19, 1985.

[210] Daniel M Wolpert and Mitsuo Kawato. Multiple paired forward and inverse models for motor control. *Neural Networks*, 11(7):1317–1329, 1998.

[211] Daniel M Wolpert, R C Miall, and M Kawato. Internal models in the cerebellum. *Trends in Cognitive Sciences*, 2:338–347, 1998.

[212] Daniel M Wolpert, R Chris Miall, and Mitsuo Kawato. Internal models in the cerebellum. *Trends in cognitive sciences*, 2(9):338–347, 1998.

[213] Tadashi Yamazaki and Soichi Nagao. A computational mechanism for unified gain and timing control in the cerebellum. *PloS one*, 7(3):e33319, 2012.