SYNTHETIC NERVOUS SYSTEMS AND DESIGN
TOOLS FOR LEGGED LOCOMOTION

by

NICHOLAS S. SZCZECINSKI

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Department of Mechanical and Aerospace Engineering

CASE WESTERN RESERVE UNIVERSITY

August 2017
CASE WESTERN RESERVE UNIVERSITY
SCHOOL OF GRADUATE STUDIES

We hereby approve the dissertation of

Nicholas S. Szczecinski

candidate for the degree of Doctor of Philosophy*.

Committee Chair
Prof. Roger D. Quinn

Committee Member
Prof. Roy E. Ritzmann

Committee Member
Prof. Kiju Lee

Committee Member
Prof. M. Cenk Cavusoglu

Date of Defense
19 June 2017

*We also certify that written approval has been obtained
for any proprietary material contained therein.
# Contents

List of Tables vii

List of Figures viii

Acknowledgments xiii

Abstract xiv

1 Introduction 1

2 Background 8

2.1 Neurobiology of Insect Locomotion ................................. 8

2.2 Modeling the Nervous System ........................................ 12

3 Design Process and Tools for Dynamic Neuromechanical Models and Robot Controllers 17

3.1 Abstract ................................................................. 17

3.2 Introduction ............................................................ 18

3.3 Modeling Methods ...................................................... 22

3.3.1 Neuron Dynamics ................................................... 22

3.3.2 Single Joint Network Structure ................................... 24

3.4 Control of Static Posture .............................................. 27

3.4.1 Servomotor Control ................................................ 27
CONTENTS

3.4.2 Afferent Feedback and the Resistance Reflex . . . . . . . . . . 30
3.5 Control of Rhythmic Motion . . . . . . . . . . . . . . . . . . . . . . 39
  3.5.1 CPG Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39
  3.5.2 Sensory Feedback to the CPG . . . . . . . . . . . . . . . . . . 45
3.6 Leg Controller Design and Results . . . . . . . . . . . . . . . . . . 49
  3.6.1 Single Leg Network Structure . . . . . . . . . . . . . . . . . . 49
  3.6.2 Single Leg Walking Speed . . . . . . . . . . . . . . . . . . . . 52
3.7 Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56
3.8 Appendix . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61
  3.8.1 Neural Parameters . . . . . . . . . . . . . . . . . . . . . . . . 61
  3.8.2 Synaptic Parameters . . . . . . . . . . . . . . . . . . . . . . . 62
  3.8.3 Mechanical Parameters . . . . . . . . . . . . . . . . . . . . . . 63
  3.8.4 Computer Specifications . . . . . . . . . . . . . . . . . . . . . 63
  3.8.5 Modulatory Pathways . . . . . . . . . . . . . . . . . . . . . . . 63
  3.8.6 feedbackDesign . . . . . . . . . . . . . . . . . . . . . . . . . . 64
  3.8.7 cpgDesign . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 64
  3.8.8 simScan . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 66

4 A Functional Subnetwork Approach to Designing Synthetic Nervous Systems that Control Legged Robot Locomotion 68
  4.1 Abstract . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 68
  4.2 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 69
  4.3 Methods: Models and Approach . . . . . . . . . . . . . . . . . . . 72
    4.3.1 Mapping Between Neural and Mechanical Values . . . . . . . 76
  4.4 Methods: Arithmetic Subnetworks . . . . . . . . . . . . . . . . . . 78
    4.4.1 Signal Transmission Pathways . . . . . . . . . . . . . . . . . . 79
    4.4.2 Signal Modulation Pathways . . . . . . . . . . . . . . . . . . 80
    4.4.3 Addition . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 81
4.4.4 Subtraction ........................................... 82
4.4.5 Division .............................................. 85
4.4.6 Multiplication ....................................... 86
4.5 Methods: Dynamic Networks ........................... 89
  4.5.1 Differentiation ................................. 89
  4.5.2 Integration ........................................ 93
4.6 Application to a Robot Controller ...................... 103
4.7 Discussion ............................................. 106
  4.7.1 Simplifications ................................... 107
  4.7.2 Why put neurons in the way? ................. 108
4.8 Appendix: Derivation of Integrator Eigenvalues and Eigenvectors . 109

5 MantisBot: A Platform for Investigating Mantis Behavior via Real-Time Neural Control 114
  5.1 Abstract .............................................. 114
  5.2 Introduction ......................................... 115
  5.3 Robot Hardware ..................................... 118
    5.3.1 Mechanical and Electrical Design ............ 118
    5.3.2 Mantis Kinematics ............................ 119
    5.3.3 Robot Kinematics ............................... 120
  5.4 Robot Control Architecture ....................... 124
    5.4.1 AnimatLab-MantisBot interface ............. 124
  5.5 Conclusions ........................................ 125

6 Mantisbot is a Robotic Model of Visually Guided Motion in the Praying Mantis 126
  6.1 Abstract ............................................. 126
  6.2 Introduction ....................................... 127
9.6 Results from Mechanisms 1 and 2 ............................................. 225
9.7 Discussion .................................................................................. 228
  9.7.1 Source of commanded leg strain .............................................. 231
  9.7.2 Where does CPG drive come from? ......................................... 231
  9.7.3 Searching with synergies ......................................................... 232
  9.7.4 Walking with all legs ............................................................... 233
9.8 Appendix ...................................................................................... 235

10 Conclusion and Future Work ......................................................... 236
  10.1 Conclusion .................................................................................. 236
  10.2 Future Work ................................................................................ 241
    10.2.1 Neurobiology ...................................................................... 241
    10.2.2 Modeling and Network Tuning ............................................ 244
    10.2.3 Robotics ............................................................................. 245
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Summary of design constraints and free parameters when tuning arithmetic networks.</td>
<td>112</td>
</tr>
<tr>
<td>4.2</td>
<td>Summary of design constraints and free parameters when tuning dynamic networks.</td>
<td>113</td>
</tr>
<tr>
<td>5.1</td>
<td>Summary of relative proportions of a <em>Tenodera sinensis</em> praying mantis and MantisBot.</td>
<td>123</td>
</tr>
</tbody>
</table>
List of Figures

3.1 Diagram of the single-joint controller used in Chapter 3. . . . . . . . 26
3.2 Schematic and verification of servomotor control method. . . . . . . . 31
3.3 Frequency domain analysis for testing servo controller response and
    stability. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
3.4 A visualizable example showing that an interior point optimizer can
    identify optimal parameter sets, subject to constraints on stability mar-
    gins. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
3.5 Examples of the dynamics, bifurcations, and oscillation frequency of a
    half-center CPG with inhibitory interneurons. . . . . . . . . . . . . . 42
3.6 Examination of CPG phase response for different values of δ and stimuli. 46
3.7 Data showing that sensory feedback and motor output strength play
    important roles in determining motor output. . . . . . . . . . . . . . 50
3.8 The full servomotor controller easily entrains with sensory stimuli. . . 51
3.9 Diagram of the leg controller in Chapter 3 . . . . . . . . . . . . . . . 53
3.10 Leg stepping speed is more sensitive to motor output and sensory input
    strengths than to CPG parameters. . . . . . . . . . . . . . . . . . . . 55
3.11 Demonstration of Modulatory network function. . . . . . . . . . . . . 65
4.1 Graphical representation of synaptic dynamics, and mapping between mechanical and neural values. (A) Graphical representation of how synapses couple neural dynamics. Note that $R$ is marked on the plot. (B) Enhanced version of the motor control network from [Szczecinski et al., 2017b], showing how $R$ relates mechanical and neural values. Mechanical values are drawn in red, and neural values are drawn in blue. 75

4.2 Data demonstrating the function of arithmetic networks. 83

4.3 A network can exploit neural dynamics to compute the differential of an incoming signal. 91

4.4 Simulation data from eight trials with the differentiator network are shown. Different values of $\tau_1$ and $\tau_2$ were used in each. 94

4.5 (A) A disinhibitory network can exploit neural dynamics to compute the integral of an incoming signal. (B) When given an applied current in the form of a step, the network response is a ramp whose slope is proportional to the amplitude of the step. (C) A plot of this data in the $(U_1, U_2)$ phase space shows that when stimulated by applied current $u$, the system state, $x(t) = [U_1(t), U_2(t)]^T$ (blue), moves in the $X_1$ direction (green) while maintaining a constant distance from the equilibrium subspace (dashed violet) in the $X_2$ direction (red). This is because the eigenvalue associated with eigenvector $X_1$, $\lambda_1 = 0$, and the eigenvalue associated with eigenvector. 96
4.6 Simulation data from eight trials are shown. Different values of $C_m$ and $g_s$ were used in each. Neural dynamics are plotted as blue lines. The expected final values of the simulations are plotted in dotted red lines. Regions bounded by $k_i,\text{mean} \pm k_i,\text{range}$ are shaded in violet. In every case, the actual outcome is correctly bounded. As demonstrated mathematically in the text, $k_i,\text{mean}$ only depends on $C_m$. In addition, $k_i,\text{range}$ depends on $g_s$, leading to more variation in $k_i$, as indicated by larger shaded areas.  

4.7 A simplified joint-control network from our previous work [Szczecinski et al., 2017a, Szczecinski and Quinn, 2017c], with pathways color-coded based on the functional subnetwork.  

5.1 A photo of MantisBot.  
5.2 Scaled schematic of a male *Tenodera sinensis*, with segments and degrees of freedom labeled.  
5.3 Photos of each leg of MantisBot with joints labeled.  

6.1 A photo of MantisBot.  
6.2 MantisBot’s head sensor detects the azimuth and elevation of the centroid of luminosity in its field of view.  
6.3 Hypothetical network that processes visual input, predicts the future position of the prey, and generates saccades.  
6.4 Illustration and demonstration of memory network.  
6.5 The leg control network, which produces coordinated motion despite its distributed structure.  
6.6 Illustration of the inverse kinematics method used to tune joint control networks.  
6.7 A summary of MantisBot’s ability to track prey with saccades.
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.8</td>
<td>Walking kinematics change as the intended direction is modulated.</td>
<td>149</td>
</tr>
<tr>
<td>6.9</td>
<td>Stance phase foot directions change significantly with altered “prey” location.</td>
<td>150</td>
</tr>
<tr>
<td>6.10</td>
<td>Neural activity illustrating a reflex reversal during typical locomotion.</td>
<td>151</td>
</tr>
<tr>
<td>6.11</td>
<td>The network in Figure 6.5 controls both locomotion and static posture.</td>
<td>152</td>
</tr>
<tr>
<td>7.1</td>
<td>Schematic of one leg’s neural controller.</td>
<td>167</td>
</tr>
<tr>
<td>7.2</td>
<td>Illustration of how intended foot paths are calculated as a function of body rotation, ( \psi ), and body translation, ( r ).</td>
<td>170</td>
</tr>
<tr>
<td>7.3</td>
<td>MantisBot as constrained for this study.</td>
<td>173</td>
</tr>
<tr>
<td>7.4</td>
<td>A contour of each of the front leg’s joints’ PEP is plotted as the body translation (x axis) and rotation (y axis) are varied.</td>
<td>175</td>
</tr>
<tr>
<td>7.5</td>
<td>A contour of each of the middle leg’s joints’ PEP is plotted as the body translation (x axis) and rotation (y axis) are varied.</td>
<td>176</td>
</tr>
<tr>
<td>7.6</td>
<td>A contour of each of the hind leg’s joints’ PEP is plotted as the body translation (x axis) and rotation (y axis) are varied.</td>
<td>177</td>
</tr>
<tr>
<td>7.7</td>
<td>Bar graphs of the front leg’s foot motion are plotted for comparison to the intended, that is, ideal case.</td>
<td>179</td>
</tr>
<tr>
<td>7.8</td>
<td>Bar graphs of the middle leg’s foot motion are plotted for comparison to the intended, that is, ideal case.</td>
<td>180</td>
</tr>
<tr>
<td>7.9</td>
<td>Bar graphs of the hind leg’s foot motion are plotted for comparison to the intended, that is, ideal case.</td>
<td>181</td>
</tr>
<tr>
<td>7.10</td>
<td>Plots of each joint’s motion in stance phase, for the front leg.</td>
<td>183</td>
</tr>
<tr>
<td>7.11</td>
<td>Plots of each joint’s motion in stance phase, for the middle leg.</td>
<td>184</td>
</tr>
<tr>
<td>7.12</td>
<td>Plots of each joint’s motion in stance phase, for the hind leg.</td>
<td>185</td>
</tr>
<tr>
<td>8.1</td>
<td>MantisBot secured to a frame.</td>
<td>197</td>
</tr>
<tr>
<td>8.2</td>
<td>Schematic of a control network for one joint of MantisBot.</td>
<td>199</td>
</tr>
<tr>
<td>FIGURE</td>
<td>DESCRIPTION</td>
<td>PAGE</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>8.3</td>
<td>Demonstration of how feedback may stabilize CPG dynamics.</td>
<td>202</td>
</tr>
<tr>
<td>8.4</td>
<td>Single-segment model informs the strength of positive velocity feedback to the CPGs.</td>
<td>204</td>
</tr>
<tr>
<td>8.5</td>
<td>The results from the single segment model apply directly to MantisBot, enabling it to change speed while the duration of swing phase remains constant.</td>
<td>206</td>
</tr>
<tr>
<td>8.6</td>
<td>All 279 steps taken by the robot were aggregated to analyze gross trends in the stepping speed.</td>
<td>207</td>
</tr>
<tr>
<td>9.1</td>
<td>A photo of MantisBot.</td>
<td>214</td>
</tr>
<tr>
<td>9.2</td>
<td>Diagram of the part of our leg control network responsible for searching and force modulation.</td>
<td>218</td>
</tr>
<tr>
<td>9.3</td>
<td>Two examples of how leg strain evolves as MantisBot steps.</td>
<td>223</td>
</tr>
<tr>
<td>9.4</td>
<td>Our simple model of leg strain learning predicts results from MantisBot.</td>
<td>224</td>
</tr>
<tr>
<td>9.5</td>
<td>MantisBot can adapt its locomotion to obtain the intended leg strain.</td>
<td>226</td>
</tr>
<tr>
<td>9.6</td>
<td>Our searching network replicates neural data from searching stick insects.</td>
<td>227</td>
</tr>
<tr>
<td>9.7</td>
<td>Both searching and force modulation work together to produce adaptive stepping.</td>
<td>229</td>
</tr>
<tr>
<td>10.1</td>
<td>Joint control network from Figure 6.5 with functional subnetworks boxed.</td>
<td>242</td>
</tr>
</tbody>
</table>
Acknowledgments

I have many people to acknowledge for getting this done. I thank both NASA and the U.S. Department of Education for funding my work. I thank Prof. Roger Quinn for being an exceptional mentor. I thank Prof. Roy Ritzmann for drawing me into neurobiology. I thank Prof. Kiju Lee and Prof. M. Cenk Cavusoglu for their feedback in preparing this document, and their participation on my committee.

I thank my family for their support. I thank my fellow graduate students at CWRU for their camaraderie. And above all I thank Meagan, for her endless support and understanding.
Synthetic Nervous Systems and Design Tools for Legged Locomotion

Abstract

by

NICHOLAS S. SZCZECINSKI

This thesis has three aims. The primary aim of this thesis is to develop a model of insect thoracic locomotion control networks, and explore how minimal descending commands could modulate the speed and direction of locomotion. The secondary aim of this thesis is to develop analytical and numerical tools that enable a person to construct a detailed morphological model of neural systems and tune them to produce stable, functional, animal-like motion. The third and final aim of this thesis is to use the model developed in aim 1 and tuned in aim 2 to control specific locomotion tasks in a robot, demonstrating that a model of this type could some day control the locomotion of a real-world robot. The motivation for this work was approximately 40% neurobiology, 40% computational neuroscience, and 20% robotics, and this thesis is relevant to each of these fields.

This thesis presents a design process and associated tools for analyzing dynamic neuromechanical models and robot controllers. This is expanded to produce a functional subnetwork approach to designing synthetic nervous systems that control legged locomotion of artificial agents. Next, this thesis presents Mantisbot, a 29 degree of freedom robot for investigating mantis behavior via real-time neural control. Then, this thesis presents a dynamical neural controller, which we call “synthetic nervous systems” (SNS), that models visually guided motion in the praying mantis with MantisBot. In subsequent chapters, I generalize the structure and tuning of this SNS to control the stepping of all six of MantisBot’s legs; I incorporate a local sensory pathway to change MantisBot’s stepping speed by entraining pattern generating networks to positive velocity feedback; and I incorporate leg-local neural mechanisms that actively adapt MantisBot’s stepping to the environment via force feedback.
Chapter 1

Introduction

Insects exhibit adaptive locomotion, controlled by highly distributed nervous systems. Activity in brain centers correlates with body motion [Guo and Ritzmann, 2013, Martin et al., 2015], and stimulating these regions evokes locomotion in the expected speed and direction [Bender et al., 2010, Bidaye et al., 2014, Martin et al., 2015]. Upon closer examination, the changes in direction are caused by the modulation of local reflexes that coordinate the muscles in one leg into a cohesive motion [Mu and Ritzmann, 2008, Martin et al., 2015]. How might descending signals, which are known to be quite sparse in arthropods [Bowerman and Larimer, 1974a, Bowerman and Larimer, 1974b], cause lower-level (i.e. thoracic) networks to produce directed motion? **The primary aim of this thesis** is to develop a dynamic model of thoracic locomotion control networks, and explore how minimal descending commands could modulate the speed and direction of locomotion.

Other biological models [Dürr et al., 2004, Knops et al., 2012, Toth et al., 2012, Szczecinski et al., 2014] and biologically inspired robots [Manoonpong et al., 2007, Rutter et al., 2011, Schilling et al., 2012] have explored how an animal may modify control to walk in different directions. There are two main approaches to constructing such models: morphological and functional [Buschmann et al., 2015]. Morphological
models begin with detailed models of specific neuromechanical structures from animals, and seek to construct an intact system by adding hypothetical components. Morphological models typically provide further insight into the biological system, but rarely produce motion as coordinated as that found in the animal, because of difficulty of tuning their components. (My M.S. thesis project, which had the same primary goal as this thesis, fits into this category. While it had some similarities with the animal [Szczecinski et al., 2014], the motion was far from animal-like.) In contrast, functional models use machine learning techniques like perceptron networks or reservoir computing to train an abstract model of neural circuits to reproduce animal behavior. This often leads to animal-like behavior, but the resulting control system is a black box that provides little insight into how the animal functions, and may malfunction because the engineer cannot analyze its stability or robustness.

To thoroughly test a hypothesis about how descending commands control locomotion, a modeling approach is needed that bridges the gap between the morphological and functional approach. A morphological model may demonstrate some underlying principles, but a poorly tuned model may lead to incorrect conclusions about whether or not the model is plausible. A functional model may demonstrate the correct behavior, but its black-box composition does not produce any testable hypotheses for neurobiology. Thus, the secondary aim of this thesis is to develop analytical and numerical tools that enable a person to construct a detailed morphological model of neural systems and tune them to produce stable, functional, animal-like motion, capturing the strengths of both approaches. We call such a system a “synthetic nervous system”, or SNS.

In this thesis, the SNSs were used to perform closed-loop control of a 29 degree of freedom robot, MantisBot. Implementing the SNS on a hardware robot further tests its stability when confronted with the nonlinear, discontinuous dynamics of real-world motion, subject to noise from mechanical and electrical vibrations. At the
time of writing, MantisBot has not walked freely; however, it has performed a variety of important prerequisite tasks, such as using sensory feedback to coordinate the stepping of individual legs (Chapter 6), adjusting its direction and speed with very sparse descending commands (Chapters 7, 8) and actively adapting its locomotion over time based on external forces (Chapter 9). Thus, the third and final aim of this thesis is to use the SNS that I developed in aim 1 and tuned in aim 2 to control specific locomotion tasks in a robot, demonstrating that an SNS could some day control the locomotion of a real-world robot.

This thesis focuses on praying mantises to address these aims. Mantises are especially useful for addressing aim 1 because they are predators. This means that the target of their behavior is clear: they hunt prey. This is in contrast to animals whose niche is exclusively prey, such as cockroaches, who may escape a predator by moving in any direction within a 180° span. The behaviors modeled in this thesis are mantis-like: MantisBot uses saccades to visually track simulated prey [Lea and Mueller, 1977, Rossel, 1980], and locomotion is guided by these sensory signals [Mittelstaedt, 1957]. Because the purpose of this work is not to create a model of the brain, the brain networks that control these behaviors were constructed to be as simple as possible to avoid making assumptions about what these networks actually look like. Many of the details of the thoracic networks, however, are based on other, more thoroughly studied insects, specifically locusts, cockroaches and stick insects. This is justified for two reasons. First, mantises are genetic cousins of cockroaches [Svenson and Whiting, 2004], meaning that many fundamental circuits are likely conserved. Second, the mantis species we have the most experience with, Tenodera sinensis, and stick insects live in similar, vegetative habitats, suggesting that they have been subjected to similar evolutionary pressures for locomotion.

Each chapter in this thesis after the Background has been published or submitted for publication in a peer-reviewed journal or conference proceeding. I was the first
author on each. They are not presented in chronological order. Instead, they are ordered to tell a logical story. Chapter 2 reviews literature from insect neurobiology and computational and robotic models of insect behavior.

Chapter 3 presents a serial design process with associated tools for selecting parameter values for a posture and locomotion controller for a simulation of a robot. The controller is constructed from dynamic neuron and synapse models, and simulated with the open-source neuromechanical simulator AnimatLab 2 [Cofer et al., 2010]. Each joint has a central pattern generator (CPG) whose neurons possess persistent sodium channels. The CPG rhythmically inhibits motor neurons that control the servomotor’s velocity. Sensory information coordinates the joints in the leg into a cohesive stepping motion.

The parameter value design process is intended to run on a desktop computer, and has three steps. First, my tool feedbackDesign uses classical control methods to find neural and synaptic parameter values that stably and robustly control servomotor output. This method is fast, testing over 100 parameter value variations per minute. Next, my tool cpgDesign generates bifurcation diagrams and phase response curves for the CPG model. This reveals neural and synaptic parameter values that produce robust oscillation cycles, whose phase can be rapidly entrained to sensory feedback. It also designs the synaptic conductance of inter-joint pathways. Finally, to understand sensitivity to parameters and how descending commands affect a leg’s stepping motion, my tool simScan runs batches of neuromechanical simulations with specified parameter values, which is useful for searching the parameter space of a complicated simulation. These design tools are demonstrated on a simulation of a robot, but may be applied to neuromechanical animal models or physical robots as well.

Chapter 4 presents the neuron and synapse models used, and introduces design tools and methods for tuning their parameters. Due to increasingly detailed data on the connectivity and dynamics of insect nervous systems, controlling a legged body
with an SNS is largely a problem of parameter tuning. My approach to this problem is to design functional subnetworks that perform specific operations, and then assemble them into larger models of the nervous system. In this chapter, I present networks that perform addition, subtraction, multiplication, division, differentiation, and integration of incoming signals. Starting with the operating range of neural activity in the network, $R$, the gain of the operation, $k$, and bounds based on biological values, parameters can be constrained to produce one functional set of values. Assembling large networks from functional subnetworks that are tuned analytically underpins my recent results with MantisBot, such as Chapters 6 through 9.

Chapter 5 introduces MantisBot, a 29 degree of freedom robot controlled in real-time by an SNS. MantisBot was designed as a 13.3:1 model of a male *Tenodera sinensis* with the animal’s predominant degrees of freedom. In order to more directly use data from the animal, the robot mimics its kinematics and range of motion as closely as possible, uses strain gages on its legs to measure femoral strain like insects, and is controlled by a realistic neural simulation of networks in the thoracic ganglia. This chapter summarizes the mechanical, electrical, and software design of the robot.

Chapter 6 presents the basic stepping controller for all of MantisBot’s legs, tuned using methods from Chapters 3 and 4. Using the literature on mantis prey tracking and insect locomotion, I designed a hierarchical, distributed neural controller that establishes the goal, coordinates various joints, and executes prey-tracking motion. In my controller, brain networks perceive the location of prey and predict its future location, store this location in memory, and formulate descending commands for ballistic saccades like those seen in the animal. The descending commands are simple, indicating only 1) whether the robot should walk or stand still, and 2) the intended direction of motion. Each joint’s controller uses the descending commands differently to alter sensory-motor interactions, changing the sensory pathways that coordinate the joints’ CPGs into one cohesive motion. Experiments with one leg of MantisBot
show that visual input produces simple descending commands that alter walking kine-
matics, change the walking direction in a predictable manner, enact reflex reversals
when necessary, and can control both static posture and locomotion with the same
network.

Chapter 7 expands on Chapter 6 by applying the controller to each pair of legs. Each pair of legs is kinematically distinct from the others. The networks were tuned using methods from Chapters 3 and 4. In this chapter, I successfully apply an auto-
mated method to tune the control network for all three pairs of legs of MantisBot in only 90 seconds with a desktop computer (that is, the resulting networks solve the
inverse kinematics problem for all six legs). Each foot’s motion changes appropriately
as the body’s intended direction of travel changes. In addition, several results from
studies of walking insects are captured by this model. This chapter both suggests
neural mechanisms underlying observations from walking insects, and demonstrates
the broad applicability of this control method for robots.

Chapter 8 presents an extension of the controller in Chapters 6 and 7, enabling
MantisBot to step at different speeds. The networks were tuned using methods from
Chapters 3 and 4. This chapter demonstrates and analyzes how CPGs can entrain
joints of MantisBot to positive velocity feedback resulting in a duration change of a
leg’s stance phase. I use a model of a single leg segment, as well as previously presented
design techniques to understand how the gain of positive velocity feedback to the
CPGs should be modulated to successfully implement a stance-phase reinforcement
reflex, termed the active reaction (AR), during walking. The results suggest that
the AR simplifies the descending control of walking speed, naturally producing the
asymmetrical changes in stance and swing phase duration seen in walking animals. I
implement the AR in the SNS developed in Chapter 6, and experiments confirm that
the robot modulates its walking speed as the simple model predicted. Aggregating
the data from hundreds of steps in different walking directions show that the robot
changes speed by altering the duration of stance phase while swing phase remains unaffected, as seen in walking animals.

Chapter 9 presents an extension of the controller in Chapters 6 and 7, enabling MantisBot to search for ground contact and modulate the force exerted with each step. The networks were tuned using methods from Chapters 3 and 4. Adapting motor output based on environmental forces is critical for successful locomotion in the real world. Arthropods use at least two neural mechanisms to adjust muscle activation while walking based on detected forces. Mechanism 1 uses negative feedback of leg depressor force to ensure that each stance leg supports an appropriate amount of the body’s weight. Mechanism 2 encourages searching for ground contact if the leg supports no body weight. In this chapter, I expand the neural controller for MantisBot to include these mechanisms by incorporating leg-local memory and command neurons, as observed in arthropods. I present results from MantisBot transitioning between searching and stepping, mimicking data from animals as reported in the literature.

Chapter 10 contains a discussion of the implications of this work for neurobiology and robotics, as well as future work.
Chapter 2

Background

2.1 Neurobiology of Insect Locomotion

Arthropods (insects, crustaceans, arachnids, and myriapods) evolved from a common ancestor made from unspecialized body segments. Over time, each segment’s ganglion specialized to produce the supraesophageal, subesophageal, and thoracic ganglia (analogous to the brain, brain stem, and spinal cord in vertebrates) [Ruppert et al., 2004]. Because of this evolutionary legacy, each ganglion has a substantial amount of autonomy; communication between centers tends to be sparse and efficient [Bowerman and Larimer, 1974a, Bowerman and Larimer, 1974b], and cutting connectives between centers alters motion, but in many cases does not disable the animal [Ridgel and Ritzmann, 2005, Roeder, 1937].

In insects, the thoracic ganglia control locomotion via a system of oscillators called central pattern generators (CPGs) [Ryckebusch and Laurent, 1993, Büschges et al., 1995]. Each joint in each leg has its own CPG, within that leg’s thoracic ganglion. Sensory signals such as joint motion [Hess and Büschges, 1997, Hess and Büschges, 1999, Bucher et al., 2003] and leg strain [Akay et al., 2004, Zill et al., 2004] provide feedback to the oscillators, adjusting their relative phase and producing coordinated
motion. Drastic changes in coordination, such as those seen while insects turn or walk backward, may be caused by the sign of this feedback reversing, which causes the opposite transition in a CPG [Hess and Büschges, 1999, Akay et al., 2006, Ridgel et al., 2007, Mu and Ritzmann, 2008, Hellekes et al., 2011, Szczecinski et al., 2014]. Stimulating areas of the brain’s central complex (CX) whose activity correlates to walking direction can evoke such a reversal [Martin et al., 2015]. Other behavioral transitions, such as switching from standing still to walking, are the result of a “reflex reversal” between negative and positive velocity feedback in individual joint controllers [Hellekes et al., 2011].

Insects’ legs must communicate to properly time steps and form gaits. Much behavioral data on insect inter-leg coordination exists [Cruse, 1990, Dürr et al., 2004], but less is known about the neural communication that produces it [Borgmann et al., 2009, Tóth et al., 2015]. Inter-leg coordination rarely resembles the clean gaits people visualize, such as a tripod gait [Grabowska et al., 2012], and is better described as probability distributions than hard and fast rules [Dürr and Ebeling, 2005]. The front legs, which experience the world first in a forward-walking insect, appear to be the most volitional [Grabowska et al., 2012]. When the animal changes direction, the front legs change their direction during stance before the other legs, “pulling” the others along [Dürr and Ebeling, 2005]. In addition, the CPGs in the middle and hind legs only activate when a front leg is actively stepping [Borgmann et al., 2009]. Additional behavioral data from arthropods suggest that more rules may be encoded in the nervous system to help coordinate leg speed and positioning [Cruse, 1990, Bender et al., 2011]. However, many force regulating [Foth and Graham, 1983, Libersat et al., 1987, Zill and Seyfarth, 1996] and timing [Borgmann et al., 2009, Zill et al., 2009] mechanisms are controlled by sensory-driven controllers at the leg-level. All of these findings suggest that posture and locomotion are controlled via distributed neural structures that depend on feedback from sensors as well as descending commands.
from higher command centers [Buschmann et al., 2015].

**Descending Commands** The competence of the thoracic networks to control walking suggests that descending commands do not need to be detailed to direct locomotion (for reviews, see [Ritzmann et al., 2012] and [Borgmann and Büschges, 2015]). Neurons in the central complex (CX, see [Pfeiffer and Homberg, 2014]) show activity that correlate with the speed and direction of cockroach walking [Guo and Ritzmann, 2013, Martin et al., 2015]. Stimulating these populations of neurons through the recording electrodes causes the cockroach to alter walking in a manner that is consistent with patterns of activity recorded during free walking [Martin et al., 2015], suggesting that the CX either generates descending commands that control locomotion, or is upstream of the center that does [Ritzmann et al., 2012]. Work in *Drosophila* (fruit flies) has identified a “Moonwalker Descending Neuron” (MDN) whose activation is necessary and sufficient to make the fly reverse its walking direction [Bidaye et al., 2014]. This is reminiscent of earlier studies with crayfish in which tonic [Bowerman and Larimer, 1974a] and phasic [Atwood and Wiersma, 1967, Bowerman and Larimer, 1974b] stimuli to descending nerves to the thoracic ganglia evoked complete behaviors, such as swimmeret beating, claw grasping, and “freezing” in place. While the supraesophageal ganglion (i.e. brain) may not send such simple descending commands *in vivo*, these studies show just how competent the thoracic ganglia are, and suggest that descending commands could be very sparse in the behaving animal.

Numerous studies suggest that descending commands alter locomotion by changing the strength or direction of reflexes. Many studies focus on how stretch of the femoral chordotonal organ (fCO, signals tibial flexion) affects muscle activity elsewhere in the leg. During forward walking, stretch of the fCO plays two roles: it reinforces tibial flexion via positive feedback (called the “active reaction”, or AR) [Bässler, 1976, Akay and Büschges, 2006], and affects femoral depressor muscle activ-
ity, to transition between stance and swing phase [Hess and Büschges, 1999, Bucher et al., 2003]. Walking in different directions, however, requires that these reflexes be modified to change the motion of the leg, and thus the direction of body motion. Studies with stick insects showed that an AR occurs in the front legs when they walk forwards, but rarely as they walk backwards [Hellekes et al., 2011]. Similarly, when the insects walked in a curve, an AR occurs in the front leg on the inside of the turn, but not on the front leg on the outside of the turn. Work with cockroaches has shown that stimulating the CX to cause curve walking changes how fCO stretch affects femoral depressor activity, depending on the walking direction [Martin et al., 2015]. Severing the connection from the sub- and supraesophageal ganglia makes these reflexes less predictable, possibly because muscle activation is less regulated without these ganglia [Mu and Ritzmann, 2008].

Descending commands can modify other sensory-processing pathways. Load feedback from campaniform sensilla (CS), which sense leg strain, are critical for coordination of walking motion [Noah et al., 2004]. Signals from the CS apparently reset the phase of CPGs throughout the leg, coordinating motion [Akay et al., 2001, Akay et al., 2007]. When the animal walks backward or along a curve, some of these reflexes must reverse their sign, a phenomenon that has been observed in stick insects [Akay et al., 2007]. Descending commands appear to control locomotion not by providing detailed instructions to the thoracic ganglia, but by modulating local sensory processing. Any model that seeks to understand these mechanisms must include local sensory-processing networks (Chapter 6) and have means by which to modulate network sensitivity and activity (Chapter 4).

**Adaptive Mechanisms**  Besides walking on flat ground, insects have neural mechanisms that enable them to adapt to unpredictable terrain. Even without their brains, insects are capable of making some long-term adjustments to motor output, for ex-
ample, to avoid painful stimuli [Eisenstein, 1972]. Each leg is individually capable of many adaptive postural and locomotive tasks, such as controlling body height [Cruse et al., 1993], searching for ground contact [Bläsing and Cruse, 2004, Berg et al., 2015], resteping under excessive load [Zill et al., 1992], and elevating the swing phase when impeded [Cruse, 1980]. All of these contribute to insects’ adaptive walking. This thesis does not model all of these behaviors. However, it does model force regulating and searching reflexes by adding a few additional pathways to the baseline walking controller. Because I established an adaptable joint controller based on the literature, the resulting networks can either be based directly on intracellular recordings in animals [Berg et al., 2015], or serve as detailed hypotheses of circuits underlying extracellular recordings [Foth and Graham, 1983, Libersat et al., 1987, Zill and Seyfarth, 1996].

2.2 Modeling the Nervous System

The nervous system can be modeled in varying degrees of detail (See Figure 9 in [Buschmann et al., 2015]). There are two primary approaches. The Morphological Approach is an anatomy-first approach. The modeler attempts to build a model by beginning with what is known about neurons, local networks, and the nervous system. Additional components are added if biological data is insufficient, but these additions serve as testable hypotheses for future neurobiological studies. Morphological models are rarely coordinated enough to fully reproduce animal motion because the current body of neurobiology knowledge is not complete, and modelers seek to keep hypothetical additions tractable (i.e. minimal). The second approach is the Functional Approach, which is a behavior-first approach. The modeler constructs an abstracted model, such as a finite state machine (FSM), artificial neural network (ANN), or reservoir computing (RC) network, trained to reproduce the behavior of the animal. Such models are frequently used to control robots, because they can be
tuned to arbitrary precision. However, the resulting systems are black boxes that do not directly test neurobiological hypotheses.

**Comparison of Neural Models** In this thesis, ANN refers to a neural model in which the neurons have no internal dynamics. The response of the \(i^{th}\) neuron in the network, \(u_i\), is simply a function of the sum of its inputs,

\[
u_i^{\text{out}} = \tanh \left( \sum_{j=1}^{n} w_{ij} u_j^n \right), \tag{2.1}\]

where neuron \(i\) has \(n\) inputs, weighted by \(w_{ij}\) [Trappenberg, 2009]. Several biologically inspired robots use this type of neuron model because of its ease of use [Von Twickel et al., 2011, Schilling et al., 2012]. In addition, a network of this type of neuron models is said to be a universal “function approximator”, meaning that a large enough network is trained properly, it can recreate any non-dynamic relationship between inputs and outputs [Trappenberg, 2009]. Nervous systems, however, are dynamical systems, and many critical structures exploit these dynamics to perform calculus [Reichardt, 1961, Seung et al., 2000b, Loewenstein and Sompolinsky, 2003, Szczecinski et al., 2017a] or generate rhythmic signals for locomotion [Perkel and Mulloney, 1974, Daun-Gruhn et al., 2009, Szczecinski et al., 2017b]. Therefore, we choose not to use this model in our work.

The reasoning behind RC models are similar in principle to ANNs: a network can produce any desired output given a particular input, and a large enough, properly tuned network. RC methods, in contrast to ANNs, use continuous-time recurrent neural networks (CTRNNs), in which the response of the \(i^{th}\) neuron in the network, \(u_i\), is

\[
\tau_i \cdot \dot{u}_i = -u_i + \sum_{j=1}^{n} w_{ij} \cdot g(u_j) + I_{\text{app},i}, \quad (2.2)
\]

where \(\tau_i\) is the time constant of this neuron, and \(g\) is a function that maps between
the presynaptic neuron voltage and the conductance of the synapse [Dasgupta et al., 2015]. This model is the simplest continuous-time dynamical neuron model, and is similar to that used in this thesis (see Equation 4.6). Even though RC methods endow neural models with dynamic properties, their uniform, reservoir structure (or lack thereof) does not reflect the highly-organized networks in the thoracic ganglia or central complex of insects. Therefore, we choose not to model our system with RC methods.

The most detailed neuron models are conductance-based models. These models add additional current terms to Equation 2.2, in which each current refers to the flow of one ion (e.g. calcium, sodium, etc., up to eight different channels. See [Prinz et al., 2003]). The inclusion of different ion channels are necessary to construct limit cycles. Models of thoroughly studied, rhythmic networks, such as the stomatogastric ganglion of crustaceans [Mulloney and Selverston, 1974], often use these models because data exists that aids in tuning of such networks [Marder and Taylor, 2011]. Such details are not available for insect locomotion networks, so using such a detailed model would overcomplicate the tuning process.

Most animal locomotion models model neurons with a compromise between a CTRNN model and a conductance-based model [Daun-Gruhn et al., 2009, Daun-Gruhn and Tóth, 2010, Szczecinski et al., 2014, Hunt et al., 2015a, Markin et al., 2016]. Rather than simulating the dynamics of eight different ions, only one ion channel is included, which gives the neurons relaxation dynamics that can be tuned to produce stable limit cycles [Strogatz, 1994]. We choose to model neurons in this way, as continuous-time neurons with persistent sodium channels (Eqs. 3.1 through 3.7), because it is the simplest model that exhibits dynamical behavior, and enables the generation of relaxation dynamics to build CPGs.
Tuning Neuromechanical Models  Morphological models tend to have fewer components, because they start with what is known, and only add what is necessary for it to function. These tend to model small, functional networks [Daun-Gruhn et al., 2009, Von Twickel et al., 2011, O’Leary et al., 2013], or deliberately reduce networks to smaller, essential systems that are more tractable [Haferlach et al., 2007, Izquierdo and Beer, 2013, Prinz et al., 2004]. Tuning these systems is still complicated, despite their reduced scope, because the individual components are usually more detailed, including conductance-based neuron models like those used in our work [Haferlach et al., 2007, Izquierdo and Beer, 2013, Agmon and Beer, 2013], and possibly nonlinear networks like bursting oscillators [Daun-Gruhn et al., 2009, Daun-Gruhn and Tóth, 2010]. Thus, they are oftentimes tuned by hand [Daun-Gruhn et al., 2009, Daun-Gruhn and Tóth, 2010, Szczecinski et al., 2014] or by complicated, time-consuming evolutionary algorithms [Haferlach et al., 2007, Izquierdo and Beer, 2013, Agmon and Beer, 2013]. Both of these approaches take a lot of time and are generally imprecise.

Because of this imprecision, some models (especially neuro-robots) are controlled by neural networks that are highly abstracted to make them trainable [Beer et al., 1992, Cruse et al., 1998, Manoonpong et al., 2013, Schilling et al., 2013a]. These include using static neuron models with no dynamics [Beer et al., 1992, Cruse et al., 1998, Schilling et al., 2013a] and reservoir computing methods that use huge dynamic networks [Dasgupta et al., 2015]. No matter the model type, this approach treats the nervous system as a “function approximator”, that is, a system which tries to learn the correct relationship between inputs (i.e. sensory information and control inputs) and outputs (i.e. motor output) [Trappenberg, 2009]. While this may be true in some sense, the resulting systems tend to be exceptionally large, and opaque to experimentation. Thus, they are not particularly useful as neurobiological tools. They are useful as robot controllers, but their size and opacity make their stability and robustness difficult to prove.
CHAPTER 2. BACKGROUND

Few models seek to bridge this gap by using analytical or numerical tools to directly select parameters for robustly stable network function [Pasemann et al., 2001, Von Twickel et al., 2011]. This is surprising because several groups have reported success in tuning networks when constraints are used to ensure that segments of the network perform specific tasks [Pasemann et al., 2001, Haferlach et al., 2007] or tune components in a specific order [Zakotnik et al., 2006]. The work in this thesis seeks to bridge the gap between Morphological and Functional Models by applying engineering analysis techniques to neuromechanical models.

AnimatLab is a program that has enabled us to make progress towards automatically tuned morphological models [Cofer et al., 2010]. AnimatLab is an all-in-one neuromechanical simulator. The user can build a 3D rigid body model of robot or animal and actuate it with motors or muscles. Passive elements such as springs and dampers can also be added. Neural systems can be constructed and interfaced with sensors and actuators. In this way, one can construct a closed-loop neural controller: neural activity causes actuator force, actuator force causes rigid body motion, motion causes sensor activity, and sensor activity changes neural activity. In this thesis, I use AnimatLab and the AnimatLab Robotics Toolkit to test SNSs and control MantisBot (Chapter 5, [Szczecinski et al., 2015b]).
Chapter 3

Design Process and Tools for Dynamic Neuromechanical Models and Robot Controllers

This chapter was originally published as


Minor edits have been made, particularly in adding references to other parts of this thesis.

3.1 Abstract

We present a serial design process with associated tools for selecting parameter values for a posture and locomotion controller for a simulation of a robot. The controller is constructed from dynamic neuron and synapse models, and simulated with the open-source neuromechanical simulator AnimatLab 2. Each joint has a central pat-
tern generator (CPG) whose neurons possess persistent sodium channels. The CPG rhythmically inhibits motor neurons that control the servomotor’s velocity. Sensory information coordinates the joints in the leg into a cohesive stepping motion.

The parameter value design process is intended to run on a desktop computer, and has three steps. First, our tool feedbackDesign uses classical control methods to find neural and synaptic parameter values that stably and robustly control servomotor output. This method is fast, testing over 100 parameter value variations per minute. Next, our tool cpgDesign generates bifurcation diagrams and phase response curves for the CPG model. This reveals neural and synaptic parameter values that produce robust oscillation cycles, whose phase can be rapidly entrained to sensory feedback. It also designs the synaptic conductance of inter-joint pathways. Finally, to understand sensitivity to parameters and how descending commands affect a leg’s stepping motion, our tool simScan runs batches of neuromechanical simulations with specified parameter values, which is useful for searching the parameter space of a complicated simulation. These design tools are demonstrated on a simulation of a robot, but may be applied to neuromechanical animal models or physical robots as well.

3.2 Introduction

Legged animals can be remarkably agile in their environments. They use their legs to move in many different ways, including standing still, walking, turning, climbing, and more. Their ability to smoothly and stably transition between these could be exploited for more adaptable, autonomous robotic control. To explore such transitions, scientists often model the animal’s neural system with simplified Hodgkin-Huxley models [Szczecinski et al., 2014, Toth et al., 2013a]. The networks are intended to be reduced while still capturing specific phenomena observed in insects. Such networks have great potential to be used as walking robot controllers, an application that we
have particular interest in.

To use systems like these to control a robot that is powered by electric servomotors, we must make two considerations. The first is that without muscles, a neural controller for an electric servomotor must be devised that will capture some of what is known about neuromuscular control. Second, parameter values within the network must be tuned carefully to enable the hardware to produce the forces necessary for stable posture and locomotion. In this work, we present a design process and accompanying tools to automate the transition from an animal model to a robot controller and the tuning of the network, by using these tools to tune the neural controller for a simulation of a robot. First, we prove the stability of the neuron-servomotor interface, then analyze the response properties of pattern-generating networks, and finally use these results to design inter-joint sensory pathways that coordinate the stepping of a multi-jointed leg in simulation.

Animals control their legs through a combination of feedforward and feedback mechanisms. While standing still, a resistance reflex causes the joint to hold its position via negative position feedback, but while walking, local positive velocity feedback (LPVF) dominates intra-joint reflexes [Hellekes et al., 2011]. Changes in brain activity precede this and other locomotory transitions, suggesting that descending commands modulate low-level feedback pathways to change walking speed or direction [Guo and Ritzmann, 2013, Martin et al., 2015]. Additionally, models of human posture show that feedback gains that stabilize posture in one kinematic configuration may destabilize it in another [Bingham et al., 2011]. These examples highlight the need for a detailed understanding of how neuromechanical systems use sensory feedback, and how a leg’s motion is affected by changing feedback gains.

In insects, each leg segment is capable of independent, feedforward oscillation generated by a neural oscillator, called a central pattern generator (CPG) [Büschges et al., 1995, Ryckebusch and Laurent, 1993]. The CPGs are coordinated by sensory
CHAPTER 3. DESIGN PROCESS AND TOOLS FOR DYNAMIC NEUROMECHANICAL MODELS AND ROBOT CONTROLLERS

information from other leg segments. When the stance phase of the walking cycle begins, force feedback entrains the CPGs in the leg to support and propel the body [Akay and Büschges, 2006, Rosenbaum et al., 2010]. When the leg has completed its rearward push, proprioceptive feedback entrains the CPGs to enter swing phase and protract the leg [Cruse, 1985b]. Therefore, a CPG’s oscillation must be stable to provide a sustained walking rhythm, but must also entrain to sensory signals to coordinate all the segments in the leg. In addition, descending commands should be able to modulate CPG oscillation, preventing oscillation while the robot stands still, and adjusting sensitivity to sensory information while walking.

Many dynamical neuromechanical models have validated portions of what is known about animal locomotion [Hunt et al., 2015b, Knops et al., 2012, Markin et al., 2016, Szczecinski et al., 2014], clarifying the theoretical basis of animal locomotion in the process. However, they often make simplifications that the animal or a robot cannot, such as using virtual sensory feedback [Knops et al., 2012], or constraining the motion of the body [Hunt et al., 2015b, Markin et al., 2016]. Feedback gains are typically tuned by hand such that the model reproduces kinematic [Hunt et al., 2015b, Szczecinski et al., 2014] or EMG [Markin et al., 2016] data from animals. This process may cause a simulation to replicate animal motion, but it does not reveal anything about the stability of the controller or its sensitivity to parameter values. In reality, motor plans are typically habitual rather than optimal [de Rugy et al., 2012], and there may be many sets of parameters that may produce indistinguishable behavior [Marder and Taylor, 2011]. Therefore, we need tools that help us eliminate unstable or sensitive model parameter values, and provide a set of stable systems from which we may pick any combination, perhaps motivated by other performance metrics such as particular kinematics or forces.

How can the goal of stable and reliable motion guide the selection of parameter values? Even though the nonlinear nature of neuromechanical systems make analysis
difficult, one can use local linear approximations of a nonlinear system to compute its stability margins to determine robustness to parameter values or noise [Astrom, 2002]. In addition, the equilibrium points of nonlinear differential equations can be found with Newton’s method [Dennis and Schnabel, 1983], and Lyapunov’s methods can be used to find the stability of the equilibria [Khalil, 2002]. These tools enable the modeler to analyze a system’s response to a variety of disturbances. In this work, our design tool feedbackDesign automates frequency domain analysis to find stability margins of negative feedback loops composed of servomotors and neurons, and cpgDesign uses Newton’s method to find the equilibria of coupled nonlinear differential equations and characterizes them with numerical eigenvalue analysis.

More challenging features to analyze are the models’ discontinuous and oscillatory nature, which requires repeating these analyses in different regimes. Rather than using stochastic optimization methods to search large parameter spaces and pick a successful parameter set, one can run batches of simulations to “sweep” parameter values and extract the system’s sensitivity to them. In related studies, modelers constructed numerical phase response curves for oscillating subsystems [Tóth et al., 2015] and scanned feedback parameter values to examine subsystem and system performance [Bingham et al., 2011]. This not only enables the modeler to understand how networks behave, but also avoids unsatisfactory solutions in which parameters are found that accomplish one task optimally but fail in other scenarios. Our design tool simScan automates such scans, enabling the modeler to assemble a controller of individually characterized subsystems.

In this paper we apply these analysis techniques to develop stable stepping motions for two joints of a simulation of our robot, MantisBot (Chapter 5, [Szczecinski et al., 2015a]). We first introduce the neuron models used (3.3.1) and the structure of the control network (3.3.2). Then we describe our design process: first, we describe how to interface the neural system with a servomotor (3.4.1) and use feedbackDesign to
explore the stability of the resistance reflex as the neural and mechanical parameter values are varied (3.4.2). Next, we use cpgDesign to study the control of rhythmic output by analyzing a CPG’s equilibria, oscillation mechanism, and phase response properties (3.5.1). Then, we use simScan to explore how feedback from the mechanical system affects these properties of the CPG (3.5.2). Using these results, we design the synaptic weights of different intra- and inter-joint reflexes, and present a sensory-coupled controller for two joints in one leg (3.6.1) based on the sideways stepping reduced preparation in insects [Fischer et al., 2001, Schmidt et al., 2001]. We then use simScan again to explore the stepping motion’s sensitivity to descending commands that modulate joint-level reflexes (3.6.2). All of these analyses are automated by our design tools feedbackDesign (3.8.6), cpgDesign (3.8.7), and simScan (3.8.8), which take only seconds or minutes to run on a desktop computer (specifications in the Appendix). We conclude with a discussion of relevant design decisions that resulted from our analysis, as well as how these methods may be applied to neuromechanical models of animals and legged robots.

### 3.3 Modeling Methods

All neuromechanical modeling in this work was conducted in AnimatLab 2 [Cofer et al., 2010]. All neural and synaptic models are native to the program. All design tools were implemented in Matlab (The Mathworks, Natick, MA, USA).

### 3.3.1 Neuron Dynamics

We model all neurons as nonspiking Hodgkin-Huxley compartments with leak current, synaptic currents, a voltage-gated persistent sodium channel, and applied current. The primary dynamical variable is the voltage across the cell membrane, $V$, with
\[ C_m \dot{V} = I_{\text{leak}} + I_{\text{syn}} + I_{NaP} + I_{\text{app}} \]  

(3.1)

where

\[ I_{\text{leak}} = G_m (E_r - V), \]  

(3.2)

\[ I_{\text{syn}} = \sum_{i=1}^{n} G_{s,i} (E_{s,i} - V), \]  

(3.3)

\[ I_{NaP} = G_{NaP} m_{\infty}(V) h (E_{NaP} - V), \]  

(3.4)

and \( I_{\text{app}} \) is an arbitrary external stimulus. Eqs. 3.2, 3.3, and 3.4 define the leak, synaptic and sodium channel currents and follow the same basic form of a conductance \( G \) multiplied by the difference between the postsynaptic neuron’s membrane voltage, \( V \) and a constant reference voltage (i.e. reversal potential) \( E \). \( E_r \) is the resting potential of the neuron, and \( C_m \) and \( G_m \) are the capacitance and conductance of the cell membrane, respectively.

Neurons that make up the central pattern generators (CPGs) will have \( G_{NaP} \neq 0 \) in Equation 3.4. In this case, \( h \), the sodium channel deactivation is the second dynamical variable, and changes according to

\[ \dot{h} = (h_\infty - h)/\tau_h(V). \]  

(3.5)

\( m_\infty \) and \( h_\infty \) from Eqs. 3.4 and 3.5 are sigmoids of the form:

\[ z_\infty(V) = \frac{1}{1 + A_z \cdot \exp(S_z \cdot (V - E_z))}, \]  

(3.6)

where \( z \) represents either \( m \) or \( h \), and \( A, S, \) and \( E \) are constant parameters, specific
to \( m \) and \( h \). The time constant of \( h \) is a function of \( V \),

\[
\tau_h(V) = \tau_{h,\text{max}} \cdot h_\infty(V) \cdot \sqrt{A_h \cdot \exp(S_z \cdot (V - E_z))}. \tag{3.7}
\]

Neurons communicate via synapses. If a neuron has \( n \) incoming synapses, then the conductance of the \( i^{th} \) one is:

\[
G_{s,i} = g_i \cdot \min \left( \max \left( \frac{V_{\text{pre}} - E_{\text{lo},i}}{E_{\text{hi},i} - E_{\text{lo},i}}, 0 \right), 1 \right), \tag{3.8}
\]

where \( g_i \) is the maximum conductance, \( E_{\text{lo},i} \) is the lower threshold, and \( E_{\text{hi},i} \) is the saturation limit of the synapse. \( V_{\text{pre}} \) is the voltage of the sending (i.e., presynaptic) neuron.

The parameter values used in this paper are given in the Appendix.

### 3.3.2 Single Joint Network Structure

Much is now known about the structure of insect locomotion controllers (for review, see [Buschmann et al., 2015]), but a model or controller cannot realistically incorporate all of this knowledge. Because our goal is to understand how sensory feedback affects postural stability and the coordination of walking rhythms, and how descending commands could alter feedback pathways to change locomotion, our model of joint control is simplified to that in Figure 3.1. It is divided into three tiers: A) descending commands, B) the CPG, and C) the neural-actuator interface.

The descending commands are volitional signals that modulate the behavior of the rest of the network. The first is the strength of the CPG to motor neuron (MN) connections. It is known that modulating this connection changes locomotion speed [Rosenbaum et al., 2010], so our network includes a pathway that changes the strength of MN inhibition from the CPGs. The second descending command modulates CPG speed and phase-response characteristics by modulating the strength
of mutual inhibition (see Section 3.5.1). The third descending command changes the strength of afferent feedback, which produces the resistance reflex in standing insects [Bässler, 1983], but can be weakened, disabled, or reversed when the animal walks [Hellekes et al., 2011]. Finally, the fourth descending command sets the controller’s intended angle between two segments (or the “joint rotation”).

The CPG is modeled as a half-center (HC) oscillator whose persistent sodium channels give rise to nonlinear oscillatory dynamics. The neurons labeled “HC” in Figure 3.1 possess persistent sodium channels. Each HC neuron excites an inhibitory interneuron (IN), which inhibits the opposite HC neuron. Section 3.5.1 describes how parameter values determine CPG behavior, including whether or not periodic orbits exist, the frequency of oscillation, and the phase response properties.

The neural-actuator interface is abstracted from the animal because our robot uses electric servomotors for actuation and sensing instead of muscles. Each joint has two slow motor neurons (MNs), whose activity rotate the segment in opposite directions. Fast and inhibitory MNs are not included in our model. In cockroaches, MN activation corresponds to the rotational velocity of the segment [Watson and Ritzmann, 1998]. In our model, the difference in activation between the MNs corresponds to the rotational velocity of the servomotor, shown in Section 3.4.1. Insects use negative feedback to generate a resistance reflex while standing still [Bässler, 1983], but the gain of this feedback varies between species [Bässler et al., 1996], and can destabilize the system if too high. Thus, there is no single correct feedback gain for stable posture and locomotion, and we analyze the effect of feedback gain, as well as physical parameters of the limb, on the stability of the controller in Section 3.4.2.
Single-Joint Controller Based on Insect Neurobiology

A. Descending commands

B. CPG

C. Neural-actuator interface

Figure 3.1: Diagram of the single-joint controller used in this work. A) Descending command neurons, which modulate the behavior of the joint’s controller. B) Central pattern generator, which produces rhythm for locomotion. C) Neural-actuator interface, which includes MNs that control motor velocity, and feedback proportional to the rotation of the servomotor.
3.4 Control of Static Posture

3.4.1 Servomotor Control

Motor neuron activity correlates to joint angular velocity in locomoting insects [Watson and Ritzmann, 1998]. This model produces the same result by using one neuron to interface with a servomotor (Servo Interface neuron in Figure 3.2). This section derives this result and verifies it with simulation.

The membrane voltage of the neuron commands the servomotor’s position, and the servo injects a current into the neuron proportional to its position at that instant. The equation of motion of the servomotor with inertia and internal viscous damping, and controlled by proportional feedback is

\[
\tau = I \ddot{\theta} + c \dot{\theta} = k \cdot (\theta_{ref} - \theta),
\]

in which \(\tau\) is the actuator’s torque on the joint, \(\theta\) is the rotation of the joint, \(I\) is the inertia of the rotor and attached limb, \(c\) is the viscous damping coefficient, and \(k\) is the proportional feedback gain. We define an affine mapping between the neuron’s voltage \(V\) and the intended servomotor rotation,

\[
\theta_{ref} = aV + b.
\]

Substituting Equation 3.10 into Equation 3.9 shows how the voltage of the Servo Interface neuron affects servomotor rotation,

\[
k \cdot (aV + b) = I \ddot{\theta} + c \dot{\theta} + k \theta.
\]

The dynamics of the Servo Interface neuron can be calculated using Equation 3.1 with applied currents for the position feedback \(I_{fb}\) and the control input current \(I_{in}\),
and no synaptic or persistent sodium currents. The membrane voltage is

\[ C_m \dot{V} = G_m \cdot (E_r - V) + I_{fb} + I_{in}. \]  

(3.12)

To put the equation in a linear form, we make the substitution \( V = U + E_r \). Then we \( I_{in} = u \), the control input, and the feedback current,

\[ I_{fb} = \theta/a, \]  

(3.13)

which is inversely related to the mapping in Equation 3.10. We tune the constants \( a \) and \( b \) such that \( \theta_{\text{min}} = a \cdot E_{\text{rest}} + b \), and \( \theta_{\text{max}} = a \cdot (E_{\text{rest}} + 20) + b \). The 20 mV is an arbitrary upper limit on the membrane voltage of the Servo Interface neuron. Solving these equations simultaneously shows that \( a = (\theta_{\text{max}} - \theta_{\text{min}})/20 \), and \( b = 3\theta_{\text{max}} - 2\theta_{\text{min}} \).

Finally, we must tune \( G_m \) and \( E_r \). To simplify scaling, \( G_m = 1 \mu S \). To ensure that the simulation initializes with an initial acceleration \( \ddot{\theta} = 0 \), \( U = 0 \) when \( \theta = 0 \). Otherwise, the controller will produce a corrective torque in the first time step, producing needless motion upon startup. Solving Equation 3.10 with this condition yields \( E_r = -b/a \).

Making all substitutions simplifies Eqs. 3.9 and 3.12 to:

\[ I \ddot{\theta} = kaU - c\dot{\theta} - k\theta, \]  

(3.14)

\[ C_m \dot{U} = -U + \theta/a + u. \]  

(3.15)

These interactions are summarized graphically in Figure 3.2A. Equations 3.14 and
3.15 can be written in the form \( \dot{x} = A \cdot x + B(u) \), in which

\[
x = 
\begin{bmatrix}
U \\
\theta \\
\dot{\theta}
\end{bmatrix},
\tag{3.16}
\]

\[
A = 
\begin{bmatrix}
-1/C_m & 1/(a \cdot C_m) & 0 \\
0 & 0 & 1 \\
k \cdot a/I & -k/I & -c/I
\end{bmatrix},
\tag{3.17}
\]

\[
B(u) = 
\begin{bmatrix}
u/C_m - b/(a \cdot C_m) \\
0 \\
k \cdot b/I
\end{bmatrix}.
\tag{3.18}
\]

This system of equations was simulated in Matlab using Euler’s forward method, that is, \( x_{i+1} = x_i + \Delta t \cdot (A \cdot x_i + B(u_i)) \). Results from these simulations, shown in Figure 3.2B, reveal that in steady-state, a constant \( u \) causes a constant \( \dot{\theta} = v \) and \( \ddot{\theta} = 0 \). Substituting these conditions into Equation 3.14,

\[
\theta/a = U - \frac{c \cdot v}{a \cdot k}.
\tag{3.19}
\]

Plugging Equation 3.19 into Equation 3.15 and solving for \( v \),

\[
v = \frac{a \cdot k}{c} \cdot (u - C_m \dot{U}).
\tag{3.20}
\]

This shows that the steady-state velocity depends on the input \( u \). However, if \( C_m \) is large, the contribution from \( \dot{U} \) will make \( u \) alone a poor estimate of \( v \). We can improve the estimate, however, by approximating \( \dot{U} \) as a function of \( v \). This is accomplished by solving Equation 3.15 and ignoring the transient component. Equation 3.15 can be rewritten in the form \( \tau \dot{U} = U_{ref} - U \), in which \( \tau = C_m \) and \( U_{ref} = \theta/a + u \). This
linear equation has the well-known solution \( U(t) = U_{ref} \cdot (1 - \exp(-t/\tau)) \). Once the transient component (i.e. \(-U_{ref} \cdot \exp(-t/\tau)\)) decays, then \( U(t) = U_{ref} = \theta/a + u \).

Because we are solving for the scenario in which \( u \) is constant and \( \dot{\theta} = v \) is also constant, \( \dot{U} = \dot{\theta}/a \) in steady-state. Substituting \( \dot{U} = v/a \) into Equation 3.20,

\[
v = \frac{a \cdot k}{c} \cdot \left( u - \frac{C_m v}{a} \right).
\]

After expanding and recollecting terms,

\[
v = \frac{a \cdot k}{c + k \cdot C_m} \cdot u.
\]

Data from simulations of this system (Figure 3.2B, bottom row) verify that this constant scaling of \( u \) accurately predicts the steady-state rotational velocity of the servomotor, even as \( C_m \) is varied over several orders of magnitude. As \( C_m \) is increased, the servomotor’s steady-state velocity increasingly lags behind the commanded velocity (dashed red lines, Figure 3.2B, bottom row), demonstrating that \( C_m \) functions as a damping term in the velocity control of the servomotor.

**Full Implementation.** To complete the network shown in Figure 3.1C, two MNs are added to the network. These feed into a comparator network, neurons Flexion Compare and Extension Compare in Figure 3.1C, which inject current into the Servo Interface neuron proportional to the difference in the MNs’ membrane voltages. Exciting the Extensor MN relative to the Flexor MN causes proportional, positive joint velocity, and vice versa.

### 3.4.2 Afferent Feedback and the Resistance Reflex

Insects produce static posture via the resistance reflex (RR), which is the result of negative feedback from sensors that monitor joint rotation [Bässler, 1983]. Our model
Servo Velocity is Controlled by the Neuron’s Input Current, Scaled by the Neuron’s Membrane Capacitance

A. 

\[ C_m \ddot{U} = -U + I_{fb} + u \]

\[ k\dot{\Theta}_{\text{intended}} = I\dot{\Theta} + c\dot{\Theta} + k\Theta \]

\[ I_{fb} = \frac{\Theta}{a} \]

B. 

\[ u, C_m = 5 \text{ nF} \]

\[ u, C_m = 50 \text{ nF} \]

\[ u, C_m = 500 \text{ nF} \]

Figure 3.2: Schematic and verification of servomotor control method. A) Summary of model equations and how the neuron and servomotor exchange information. B) Examples of servomotor motion when subjected to multiple step inputs, as the neuron’s membrane capacitance spans several orders of magnitude. These show that Equation 3.22 accurately predicts the steady-state velocity of the servomotor.
reproduces the RR via an inhibitory connection from the Perceived Rotation neuron in Figure 3.3A to the Extension MN. Thus, extending the joint inhibits the Extension MN, causing the servomotor to actively flex to counteract the imposed movement.

**Equilibrium State.** The equilibrium rotation of the joint, $\theta_\infty$, is set by the Intended Rotation descending neuron’s voltage. This can be verified by solving for the equilibrium voltage of the Extension MN neuron. This neuron has no persistent sodium channels (i.e. $G_{NaP} = 0$), so Equation 3.1 can be solved directly when $\dot{V} = 0$,

$$V_\infty = \frac{G_m \cdot E_r + \sum_{i=1}^{n} G_{s,i} \cdot E_{s,i} + I_{app}}{G_m + \sum_{i=1}^{n} G_{s,i}}.$$

(3.23)

At equilibrium, the servomotor’s velocity is 0, so the voltage of the Extension MN and Flexion MN neurons must be equal. The Flexion MN has no inputs, and so remains at its resting voltage, $E_{r,\text{Flex}} = -60$. Therefore, $V_{\infty,\text{Ext}} = -60$ mV.

$$V_{\infty,\text{Ext}} = -60 = \frac{G_m \cdot E_r + G_{s,\text{desc}} \cdot E_{s,\text{desc}} + G_{s,\text{fb}} \cdot E_{s,\text{fb}}}{G_m + G_{s,\text{desc}} + G_{s,\text{fb}}},$$

(3.24)

where $E_r = -60$ and $G_m = 1$, $G_{s,\text{desc}}$ and $E_{s,\text{desc}}$ are the synaptic conductance and reversal potential associated with the descending command, and $G_{s,\text{fb}}$ and $E_{s,\text{fb}}$ are the synaptic conductance and reversal potential associated with the feedback. If $E_{s,\text{desc}} = -20$ and $E_{s,\text{fb}} = -100$ (excitatory and inhibitory synapses, respectively), then Equation 3.24 becomes

$$-60 = \frac{-60 - 20 \cdot G_{s,\text{desc}} - 100 \cdot G_{s,\text{fb}}}{1 + G_{s,\text{desc}} + G_{s,\text{fb}}}.$$

(3.25)

If

$$G_{s,\text{desc}} = g \cdot \frac{\theta_{\text{desired}} - \theta_{\text{min}}}{\theta_{\text{max}} - \theta_{\text{min}}}$$

(3.26)
and

\[ G_{s,fb} = g \cdot \frac{\theta_{actual} - \theta_{min}}{\theta_{max} - \theta_{min}}, \]  

(3.27)

then solving Equation 3.25 shows that

\[ G_{s,desc} = G_{s,fb} \]  

(3.28)

at equilibrium. Substituting Eqs. 3.26 and 3.27 into Equation 3.28 makes it trivial to show that \( \theta_{desired} = \theta_{actual} \) at steady state.

The numerical values of \( g \), \( E_{s,desc} \) and \( E_{s,fb} \) in this work are given in the Appendix, but any may be used as long as \( g > 0 \), and \( (E_{s,desc} + E_{s,fb})/2 = E_{r,flex} \).

**Frequency Domain Analysis.** Once the equilibrium rotation of the servomotor is known, one can use frequency domain analysis to find the magnitude and phase lag of the servomotor’s rotation as a function of the input signal’s frequency. These quantities define the margins of stability and robustness to parameter values.

The frequency response of a single neuron is the Laplace transform of its dynamical equation, rearranged as the ratio between the input signal and the neuron’s membrane voltage. A simple, illustrative example is a neuron with leak and applied current only. Making the substitution \( V = U + E_r \), Equation 3.1 can be expressed as

\[ C_m \dot{U}(t) = -G_m U(t) + I_{app}(t). \]  

(3.29)

Defining the neuron’s time constant as \( \tau = C_m/G_m \) gives Equation 3.29 the form

\[ \tau \dot{U}(t) = -U(t) + I_{app}(t)/G_m. \]  

(3.30)

Writing this expression in terms of \( U \) instead of \( V \) ensures that \( U(t = 0) = 0 \), because
$V(t = 0) = E_r$. The Laplace transform of Equation 3.30 is

$$\tau s U(s) = -U(s) + \frac{I_{\text{app}}(s)}{G_m}. \quad (3.31)$$

Solving for $U(s)$ shows that a neuron’s response to a sinusoidal applied current of magnitude $I_{\text{app}}$ and phase of 0 is

$$U(s) = \frac{I_{\text{app}}(s)}{G_m \cdot (\tau s + 1)}. \quad (3.32)$$

This expression will serve as an analog the response to synaptic inputs.

The neuron’s response to $n$ synaptic inputs in the frequency domain can be found in a similar fashion. A neuron that has only leak and synaptic currents, after the substitution $V = U + E_r$, has the dynamics

$$C_m \dot{U}(t) = -G_m U(t) + \sum_{i=1}^{n} G_{s,i} \cdot (E_{s,i} - E_r - U(t)) \quad (3.33)$$

Factoring out terms that contain $U(t)$ yields

$$C_m \dot{U}(t) = -\left( G_m + \sum_{i=1}^{n} G_{s,i} \right) U(t) + \sum_{i=1}^{n} G_{s,i} \cdot (E_{s,i} - E_r). \quad (3.34)$$

We define the neuron’s time constant as

$$\tau_{\text{syn}} = \frac{C_m}{G_m + \sum_{i=1}^{n} G_{s,i}}. \quad (3.35)$$

Note that the time constant depends on both the neuron’s membrane conductance, as well as the conductance of the incoming synapses. Substituting this definition for $\tau$, Equation 3.34 becomes

$$\tau_{\text{syn}} \dot{U}(t) = -U(t) + \frac{\sum_{i=1}^{n} G_{s,i} \cdot (E_{s,i} - E_r)}{G_m + \sum_{i=1}^{n} G_{s,i}}. \quad (3.36)$$
Now Eqs. 3.34 and 3.36 are analogs to Eqs. 3.29 and 3.30, respectively, with the last term in each representing an external current. Note that in Equation 3.36, \( E_{s,i} - E_r \) is a synapse-specific constant that scales \( G_{s,i} \). But what do the \( G_{s,i} \) terms represent? In the denominator, \( G_m \) is the constant membrane conductance of the neuron; in a similar way, the sum in the denominator is the sum of the DC conductance of the incoming synapses, that is, their conductance when the system is at equilibrium. Because this section began by finding the equilibrium state, these values are known for the entire network.

What do the \( G_{s,i} \) terms in the numerator represent? Like \( I_{app} \) in Equation 3.30, these terms refer to the AC component of the synaptic conductance. Referring to Equation 3.8, the Laplace transform of \( G_{s,i} \) is simply

\[
G_{s,i}(s) = U_i(s) \cdot \frac{g_i}{E_{hi,i} - E_{lo,i}},
\]

(3.37)
as long as \( E_{lo,i} \leq V_i \leq E_{hi,i} \). If \( V_i \) is outside this range, then the response is scaled by the magnitude of the first harmonic of the clipped signal [Roberge, 1975]. With this understanding, we can take the Laplace transform of Equation 3.36, yielding:

\[
U(s) = \frac{\sum_{i=1}^{n} G^{AC}_{s,i}(s) \cdot (E_{s,i} - E_r)}{(G_m + \sum_{i=1}^{n} G^{DC}_{s,i}) \cdot (\tau_{syn} + 1)}.
\]

(3.38)

One can compute the magnitude and phase response of any open-loop network of neurons and synapses with this formula.

To analyze a closed-loop system of neurons and servomotors, we need to compute the frequency response of our neuron-servomotor complex, inside the dashed box in Figure 3.3A. The Laplace transform of Equation 3.14 is

\[
U(s) = \frac{I_k^2 + cs + k}{k \cdot a} \cdot \theta(s)
\]

(3.39)
and the Laplace transform of Equation 3.15 is

\[ C_m s U(s) = -U(s) + \theta(s)/a + u(s) \]  

(3.40)

Substituting Equation 3.39 into Equation 3.40 yields

\[ \frac{\theta(s)}{u(s)} = \frac{k \cdot a}{(C_m s + G) \cdot (Is^2 + cs + k) - k}. \]  

(3.41)

Equations 3.30, 3.39, and 3.41 enable the calculation of the open-loop response of any feedforward network of neurons, synapses, and servomotors. The closed-loop response is calculated from the open-loop response. For a linear system,

\[ \frac{Y(s)}{F(s)} = \frac{H_{\text{plant}}(s)}{1 + H_{\text{plant}}(s) \cdot H_{\text{feedback}}(s)}, \]  

(3.42)

in which \( Y(s) \) is the system’s response, \( F(s) \) is the system’s input, \( H_{\text{plant}}(s) \) is the transfer function of the plant, and \( H_{\text{feedback}}(s) \) is the transfer function of the feedback pathway.

Our design tool feedbackDesign automates these calculations, and enables us to find the equilibrium state, open- and closed-loop response, and stability margins of a controller in just 10 ms. Figure 3.3B shows the magnitude and phase response of the system as the forcing frequency \( \omega \) is varied over several orders of magnitude. If a system’s open-loop magnitude response crosses below 0 dB before the phase response crosses below \(-180^\circ\), then the closed-loop system is stable. Modeling errors may cause these crossing points to differ in the real system, so controllers are tuned with margins of safety. The phase margin is \(-180^\circ\) minus the phase angle when the gain crosses below 0 dB. The gain margin is 0 minus the gain when the phase angle crosses below \(-180^\circ\). Together, a phase margin of 30° and a gain margin of 5 dB make a system stable with safety [Astrom, 2002]. Figure 3.3B graphically demonstrates how the gain
Figure 3.3: Frequency domain analysis for testing system response and stability. All data in this figure was generated in less than one minute. A) Schematic of the network analyzed (Figure 3.1C). Key parameters are labeled near the objects they pertain to. B) Bode plots of the open-loop response of the system, that is, the voltage of the Perceived Rotation neuron when the Feedback synapse is removed, reveal the gain and phase margins of the system for a particular set of parameters. C) Bode plots of the closed-loop response show that for frequencies less than about 10 Hz, the control loop responds with the same amplitude as the input signal, with little phase lag. Transfer function calculations (red) agree with those from simulation (blue), verifying the method. D) Four key parameters were varied by two orders of magnitude above and below nominal values, drawn as red circles. Shaded regions represent gain margins less than 5 dB and phase margins less than 30°.
and phase margins are extracted from Bode plots. The pink shaded regions in Figure 3.3D correspond to responses that are stable, but outside the stability margins.

Figure 3.3C shows the closed-loop Bode plot for the system. It verifies that this combination of parameter values produces servomotor motion with same magnitude as the input with a phase lag less than 45° until 10 Hz, well beyond intended stepping frequency of about 0.5 Hz.

Figs. 3.3B and 3.3C also verify that the frequency domain models match time domain simulation. Points in blue represent data calculated from time domain simulations subject to a forcing current $I_{app} = \sin \omega t$ nA. The frequency domain models, however, have the advantage of being much faster to calculate. At low forcing frequencies, simulations that represent hundreds or thousands of “neuron” seconds must be run, which takes a lot of time. At high forcing frequencies, the simulation time step must be small to keep the simulation stable, which also increases runtime. FeedbackDesign’s relative speed enables testing stability and robustness for a wider range of parameter values.

Figure 3.3D shows plots of the gain and phase margins of the network in Figure 3.3A as the labeled parameter values are changed over several orders of magnitude. Several trends are apparent. First, increasing the feedback strength more than approximately a factor of five will destabilize the system, placing a hard limit on the synaptic conductances. Second, stability is very sensitive to the damping coefficient, $\xi$. This represents a constraint, rather than a design decision, since $\xi$ is an unchangeable property of the servomotor. Third, the system maintains high stability margins even as the inertia is increased 63-fold from that of the fully extended leg. This suggests that external forces on the leg, or the inertia of the body will not impact the stability of static posture. Finally, $C_m = C_0$ minimizes the stability margins, so the controller could be made even more stable by increasing or decreasing $C_m$. These plots reveal the safety margins of the controller, and quantify its sensitivity to each
parameter. An desktop computer took only 45 seconds to generate all of the plots in Figure 3.3, highlighting the speed of this method.

The speed of this method can be leveraged to constrain the optimization of network parameters, ensuring that the resulting network is both stable and robust. Figure 3.4 shows the parameter values found by an interior point constrained minimizer seeking to maximize the stiffness of the controller \((G_{syn})\), while maintaining a phase margin of 30°. If we initialize \((G_{syn}, C_m) = (1, 500)\) and let the solver modify \(G_{syn}\), then it is constrained to stay along the line \(C_m = 500\), and finds the stiffest controller possible (black dot, Figure 3.4). If the solver is also allowed to modify \(C_m\), then the solver can find stiffer controllers with the same phase margin (red dot, Figure 3.4). For such a low-dimensional problem, hand tuning can be used to find the optimal parameter values. However, considering more parameters requires such an optimization tool, and this example illustrates this point. No matter how the parameters are chosen, feedbackDesign enables them to be quickly tested for stability and robustness, without running simulations of the dynamics.

3.5 Control of Rhythmic Motion

3.5.1 CPG Design

The study of CPG dynamics is a rich field with many open questions. The goal of our analysis is to choose parameters that produce a CPG whose oscillations are stable and responsive to sensory feedback. We seek to understand how mutual inhibition predicts rhythmicity, modulates oscillation frequency, and determines phase response sensitivity. The CPG has many more parameters, but for our purpose of engineering a control system, the knowledge our analysis provides is sufficient.
CHAPTER 3. DESIGN PROCESS AND TOOLS FOR DYNAMIC NEUROMECHANICAL MODELS AND ROBOT CONTROLLERS

CPG Structure. A central pattern generator can be formed by the mutual inhibition of two tonically-active neurons with persistent sodium channels [Daun-Gruhn, 2010, Daun-Gruhn et al., 2009]. These two neurons are called “half-centers” (HCs). This structure can produce a wide range of oscillation frequencies, which can be controlled by the excitatory input to the HC neurons, or as we will show, the strength of the mutual inhibition. To control the strength of coupling, we implement the mutual inhibition via a pair of inhibitory interneurons (INs), as shown in Figure 3.5. In our model, all neural and synaptic parameter values are symmetrical, that is, the flexor and extensor halves have the same values. The full list of parameters is given in the Appendix.

Figure 3.4: A visualizable example showing that an interior point optimizer can identify optimal parameter sets, subject to constraints on stability margins. Minimum gain margins were also specified, but the resulting boundaries lie entirely outside of the phase margin boundaries, and are not shown. The contour lines represent the objective function to minimize, \( f(x) = -G_{syn}^2 \). The scale on the right corresponds to the color of the contour lines.
CPG Rhythmicity. Each HC neuron is tonically excited by its persistent sodium channel. This activation inhibits the other HC neuron. When one of the inhibitory pathways is disabled, one HC neuron is tonically active and inhibits the other HC neuron. If the inhibited neuron’s activation is below the synaptic threshold, it cannot inhibit the active neuron, and the system will be stable in a one-up, one-down configuration. This is only possible because the synapse is piecewise, allowing the conductance to be exactly 0 for some neural activations.

The strength of the synaptic inhibition establishes the difference between the inhibited neuron’s resting potential $V_{\infty, inh}$, and the synaptic threshold $E_{lo}$, which we define as $\delta$:

$$\delta = V_{\infty, inh} - E_{lo}.$$  \hfill (3.43)

If $\delta < 0$, then the inhibited neuron will remain below the resting threshold for all time, establishing a stable equilibrium. The neuron’s activations can be swapped, producing a second stable equilibrium. Two stable points are separated by a separatrix, which includes an unstable equilibrium point when the two neuron’s voltages are equal. Behavior from this regime is shown in Figure 3.5A, and the equilibrium points are located on the bifurcation diagram in Figure 3.5E.

If the mutual inhibition is weakened such that $\delta \in (0, 5)$, we observe that the voltage of each HC neuron oscillates out of phase with the other. This is because increasing $\delta$ causes the system’s equilibria to become unstable and collapse into a single unstable equilibrium point (Figure 3.5E). In this case, a periodic orbit forms about the unstable equilibrium.

Examining the equilibria of just one of the HC neurons gives insight into controlling the system’s oscillation. According to our definition in Equation 3.43, the instantaneous equilibrium voltage of the inhibited neuron (which is not the equilibrium voltage of the fully-connected system) is greater than the synaptic threshold when $\delta > 0$. This can be seen in Figs. 3.5H and 3.5I, in which the red and blue
CHAPTER 3. DESIGN PROCESS AND TOOLS FOR DYNAMIC NEUROMECHANICAL MODELS AND ROBOT CONTROLLERS

CPG Behavior Can Be Predicted by Bifurcation Parameter $\delta$

Inhibitory interneurons (possess no persistent Na$^+$ channel)

Half-Center neurons (possess persistent Na$^+$ channel)

A. $\delta = -0.1$

B. $\delta = 0.1$

C. $\delta = 3$

D. $\delta = 7$

E. Bifurcation Diagram

F. Oscillation Frequency

G. Synaptic Inhibition

H. Flex HC Phase Portrait, $\delta = 0.1$ (B)

I. Flex HC Phase Portrait, $\delta = 3$ (C)

Figure 3.5: Examples of the dynamics, bifurcations, and oscillation frequency of a
nullclines intersect to the right of the dashed synaptic threshold. As the neuron’s state approaches this equilibrium point, it will cross over the synaptic threshold and begin to inhibit the other HC neuron, called escape. One can see from the spacing of the dots on the trajectories in Figs. 3.5H and 3.5I that this transition is quicker than the slow approach to the equilibrium state. The larger δ is, the smaller the mutual inhibition is, and the farther from the equilibrium point the trajectory remains, causing the CPG to oscillate faster (Figure 3.5F). So, although when δ > 0 it no longer corresponds to the system’s equilibrium voltages, it remains a useful bifurcation parameter. δ also directly corresponds to the strength of mutual inhibition (Fig 3.5G), giving it physical meaning, and explaining why δ is so effective at predicting behavior; the inhibited neuron’s escape could be aided either by adding excitation [Daun-Gruhn et al., 2009] or by weakening mutual inhibition, as shown here.

Increasing δ eventually causes the unstable node to become stable. In this case, the only equilibrium is for both neurons to have equal membrane voltage (Figure 3.5D). The primary implication of this regime from a design standpoint is that it should be avoided when designing a locomotory system.
**CPG Phase Response.** To determine how the CPG responds to different types of sensory input, phase response curves (PRCs) were numerically calculated by applying a stimulus current to one inhibitory IN in an oscillating CPG and measuring the advancement or delay of the cycle. In this work, the stimulus is either $-5 \text{ nA}$ or $5 \text{ nA}$ applied to the inhibitory interneuron, in a square pulse with a duration 5% of the oscillation period.

Figure 3.6Ai shows that when the active interneuron is inhibited, the period of activity is cut short and the phase is advanced. For smaller values of $\delta$, the CPG entrains almost immediately to stimuli, and the phase is advanced proportionally to when the stimulus is applied. As $\delta$ increases, however, the CPG is less sensitive to inputs. This is because when $\delta$ is near 0, the state velocities $dV/dt$ and $dh/dt$ are closer to 0, for a longer period of time. Therefore, the internal dynamics of the neurons are proportionally less important than the applied stimulus, and the phase changes immediately [Shaw et al., 2014]. No matter the value of $\delta$, a period completely insensitive to inputs occurs during the state transition near 0 phase. This analysis applies equally to Figure 3.6Bii, in which the inactive neuron is given a positive stimulus.

When the inactive neuron is inhibited (Figure 3.6Aii), no effect on phase is observed for most of the cycle. This is because an inhibitory input suppresses the interneuron farther below its synaptic threshold $E_{lo}$, and thus does not change the strength of inhibition. The exception is when the phase is greater than approximately 0.90, at which point the interneuron’s activation is rising due to the imminent transition. Inhibition at this point catastrophically affects the phase, resetting the escape. Figure 3.6C shows this resetting when $\delta = 0.1$. However, when $\delta = 3$, the stimulus is largely overpowered by the dynamics of the transition, as shown in Figure 3.6E. This example shows that $\delta$ can be tuned to design a particular sensitivity to inputs.

Unlike inhibiting the inactive interneuron, exciting the active interneuron may
affect the phase, depending on $\delta$ (Figure 3.6Bi). As $\delta$ approaches 0, the inhibited HC neuron must get closer to its equilibrium point before crossing the synaptic threshold and escaping. Exciting the active interneuron will further inhibit the inhibited HC neuron during this approach, increasing its $h_\infty$ value, and swinging the trajectory upward and around the equilibrium point (Figure 3.6D). This causes a transition sooner than one would have occurred otherwise, advancing the phase. When $\delta$ is larger, the trajectory stays farther away from the equilibrium point, preventing this same premature escape.

This analysis reveals that $\delta$, a single parameter calculated from the other parameter values of the CPG, can reveal the existence of oscillation and its sensitivity to inputs. Increasing $\delta$ increases the CPG’s frequency, which allows the CPG Speed descending neuron to match the CPG’s oscillation with that of the leg. Increasing $\delta$ also reduces sensitivity to inputs, a strategy that fast-running insects may use [Fuchs et al., 2011]. We avoid making parameter value decisions until the next section, in which we analyze the impact of local positive velocity feedback on oscillation.

### 3.5.2 Sensory Feedback to the CPG

Sensory signals from adjacent leg segments, as well as the segment itself, provide CPGs with continuous input. Therefore, understanding how a network oscillates when deaffarented has only limited utility [Bässler, 1986]. In this section, we expand on the results from the previous section to explore how local positive velocity feedback (LPVF) and local negative position feedback (LNPF) affect the motion of the joint. This corresponds to the Flexion Velocity, Extension Velocity, and Flexion Trigger neurons in Figure 3.1, and their connections to the CPG. In this section, the CPG→MN Strength and CPG Speed neurons’ activations will be modulated to explore their effects on the joint’s motion. In addition, the Afferent Strength neuron disables position control.
CPG’s Phase Response Depends on Sense of Input and δ

A. Phase Response Curve, Inhibitory Input

i. Perturbed neuron “active”

ii. Perturbed neuron “inactive”

B. Phase Response Curve, Excitatory Input

i. Perturbed neuron “active”

ii. Perturbed neuron “inactive”

C. φ = 0.99, δ = 0.1

D. φ = 0.28, δ = 0.1

E. φ = 0.97, δ = 3

F. φ = 0.28, δ = 3

Figure 3.6: Examination of CPG phase response for different values of δ and stimuli.

A) Phase response curves (PRCs) for various values of δ, when perturbed by inhibitory
LPVF is a mechanism used to reinforce ongoing motion of a joint, observed in insects while they walk [Hellekes et al., 2011]. Excitatory LPVF to the CPGs has two effects in our model, as shown in Figure 3.7A and 3.7B. First, without changing the speed of motion, it ensures that the joint achieves its full range of motion while oscillating. As a part of a leg controller, this property ensures that each joint continues rotating until sensory signals from other joints change its phase. Second, the joint will reverse its direction of travel when motion is impeded (Figure 3.7C), due to the second transition shown in Figure 3.6D.

In our model, the strength of sensory and motor pathways is more important than CPG parameters for setting the oscillation frequency. Increasing the strength of the connection between the CPG and MNs causes the joint to oscillate faster because it reaches the end of its range of motion, and thus zero velocity, faster (Figure 3.7D). LNPF, implemented via the Flexion Trigger neuron, discourages extension as the joint extends, shortening the period and causing faster oscillation (Figure 3.7E).

Using our simScan tool, we tested a wide variety of parameter values automatically, and aggregated the results to reveal larger trends. Figure 3.7F-H show how the oscillation frequency changes with the CPG→MN activation, $\delta$ from Section 3.5.1 (via the CPG Speed descending neuron), and tonic stimulus to the Flexion Trigger
neuron (which changes the angle at which extension is inhibited). Together, these plots show that frequency is the least sensitive to changes in $\delta$. Figure 3.7F shows a variance in the frequency over a factor of 3 as the MN→CPG connection is modulated over its full range of activation. Figure 3.7G shows that the LNPF directly controls the frequency by cutting the period short. Finally, Figure 3.7H shows that changing the LNPF and the CPG→MN strength together can produce a wide range of frequencies.

Figs. 3.8A and 3.8B show PRCs of the entire segment control network. These PRCs were generated using six different stimuli: direct $\pm 5$ nA current pulses (purple), large synaptic inputs (dashed lines, $g = 0.50 \mu S$, inhibitory and excitatory), and small synaptic inputs (fine lines, $g = 0.10 \mu S$, inhibitory and excitatory). The synaptic inputs function as phase-gated inputs, because their reversal potentials are the same as the CPG’s equilibrium voltages ($-40$ mV and $-60$ mV, the nullcline intersections in Figs. 3.5H and 3.5I. Therefore, an inhibitory synapse will apply no current to the inactive HC neuron, and an excitatory synapse will apply no current to the active HC neuron. This is a mechanism used in many animals’ locomotor systems to improve the stability of oscillation [DiCaprio, 1999, Wolf, 1995].

Ideally, the PRC in Figure 3.8A would be a line with a slope of $-1$ for $\phi \in [0, 0.5]$, and zero otherwise. This would mean that the CPG immediately entrains with a stimulus applied to the active HC neuron, and ignores stimuli applied to the inactive HC neuron. The PRC in Figure 3.8B would look the same, but with the right and left halves of the plot swapped. Phase-gated inputs make the PRC more closely resemble the ideal case. Exciting the inactive interneuron or inhibiting the active interneuron advances the phase proportionally to when the stimulus is applied, and the system is less responsive in any other case. This is in contrast to direct current pulses, which can delay the phase prior to transitions ($\psi = 1$, Figure 3.7A, $\psi = 0.5$, Figure 3.7B).

LPVF also affects the PRC by filtering out weak inputs, and by extending the
oscillatory regime of the controller. Small sensory stimuli, such as sensor noise, should be ignored lest they cause unintended phase adjustments in the CPG. Figs. 3.7A and B show that weak stimuli, drawn in fine lines, hardly affect the phase at all, because the velocity feedback reinforces the ongoing rhythm. Velocity feedback also enables CPGs with large $\delta$ values to oscillate stably. Figs. 3.7C and D show PRCs for four values of $\delta$, including $\delta = 7$, which cannot produce CPG oscillation without velocity feedback. Therefore, LPVF makes oscillation more robust, reducing its sensitivity to the value of $\delta$.

We have shown that $\delta$ does not control the oscillation speed in a CPG system with LPVF. However, the joint will not oscillate when $\delta < 0$, so the CPG Speed descending neuron plays an important role in the generation and maintenance of rhythmic output. If $\delta > 5$, the CPG alone will not oscillate (Figure 3.5E and F). When LPVF is included, the joint can oscillate above this range, but its PRCs show unpredictable fluctuations in phase response. Therefore, we choose $\delta = 0.1$ for the leg controller in the next section, because this guarantees responsive, robust oscillation.

### 3.6 Leg Controller Design and Results

#### 3.6.1 Single Leg Network Structure

Using the design methods presented, a simulation of two leg segments of MantisBot was made to step at various speeds by modulating descending commands. A schematic of the leg and control system are shown in Figure 3.9. A servomotor at the coxa-trochanter (CTr) joint actuates the femur. The CTr is rigidly attached to the ground reference frame, mimicking the commonly-used “reduced preparation” in which the animal’s thorax is fixed in space, and the thorax-coxa joints are locked in place via cement [Fischer et al., 2001, Schmidt et al., 2001]. The servomotor at the femur-tibia (FTi) joint is rigidly attached to the femur, and actuates the tibia.
Sensory Feedback Parameter Values are More Important to Determining Oscillation Frequency than CPG Parameter Values

Figure 3.7: Data showing that sensory feedback and motor output strength play important roles in determining motor output. A) Under normal conditions, the joint oscillates over its full range of motion. B) When LPVF is removed, the CPG dynamics dictate the oscillation frequency, and reduce the range of motion. C) LPVF causes the joint to reverse direction when motion is impeded. D) Increasing the activation of the CPG→MN descending neuron increases oscillation frequency. E) LNPF increases frequency but reduces range of motion. F) Frequency is more sensitive to CPG→MN than to δ. G) Frequency is more sensitive to Feedback bias than to δ. H) Modulating both CPG→MN and Feedback bias can produce a wide range of frequencies.
Figure 3.8: The full servomotor controller easily entrains with sensory stimuli. PRCs were generated using six different stimuli, illustrated at the top of the figure. A) The PRC for phase-gated synaptic input shows that the CPG rapidly entrains with inputs applied during the first half of the cycle, and ignores those applied during the second.
Figure 3.8: (contd.) half. This improves robustness of transitions, such as smoothing the plot near \( \psi = 1 \). B) The PRC for excitatory phase-gated synaptic input shows that the CPG rapidly entrains with inputs applied during the second half of the cycle, and ignores those applied during the first half. The PRCs in both A. and B. show that the CPG completely ignores weak synaptic inputs, which is useful for filtering noise. C and D) PRCs for various values of \( \delta \) show that LPVF enables stable oscillation even when \( \delta \) predicts no oscillation (Figure 3.5). However, fluctuations in Cii. and Di. when \( \delta \geq 3 \) suggests that the phase response may not be as predictable as when \( \delta \) is small.

Each servomotor has a control system like that in Figure 3.1. The only modification is that the Flexion Trigger neuron has been replaced by inter-joint pathways that coordinate the motion. Specifically, FTi flexion causes CTr levation, and FTi extension causes CTr depression [Hess and Büschges, 1999]; and contact force on the tibia, registered by the Leg Load neuron, excites CTr depression and FTi flexion. Together, these pathways produce inward-pulling walking like that seen in single-leg insect preparations [Fischer et al., 2001, Schmidt et al., 2001]. Synaptic parameter values are the same as those used to generate the PRCs in Figure 3.8 (\( g = 0.50 \mu S \), \( E = -60 \text{ mV} \) or \( -40 \text{ mV} \)).

The single leg produces walking motions when the CTr joint is fixed at a height that allows the foot to reach the ground. To replicate treadmill experiments performed on stick insects, the simulation also includes a bar with a mass of 100 g placed on the ground, such that the foot must drag the bar when it moves in stance phase. The coefficient of friction with the ground is 0.2, and that with the foot is 1. This allows the belt to slide easily when pushed by the leg, but forces the leg to overcome some inertia and friction when stepping.

### 3.6.2 Single Leg Walking Speed

The most fundamental inter-joint coordinating pathway is that information from load sensors excite stance-phase motor activity [Zill et al., 2015], including in the FTi joint [Akay and Büschges, 2006]. In insects, the latency between ground contact and FTi
Two-Joint Controller Based on Insect Neurobiology

Figure 3.9: Diagram of the leg controller used in this work. Each joint has a controller like that in Figure 3.1. A) Two of the descending commands (CPG Speed and Aff. Stren.) are shared between the joints, controlling the CPG speed and stiffness of the leg, respectively. Two of the descending commands (CPG→MN and Int. Rot.) are unique to each joint, controlling the magnitude of joint rotation during stepping, and the static position of the leg, respectively. The final descending command (Kin. Bias) adjusts the extension of the leg during stepping. Pathways are added at the CPG level (B) to coordinate the motion of the joints. Justification for these are provided in the text. At the joint level (C), the Flexion Trigger neuron was removed, because it was a proxy for studying the inter-joint pathways that adjust CPG phase to coordinate stepping.
flexion is very short, on the order of milliseconds [Rosenbaum et al., 2010]. Therefore, the inter-joint synapses transmitting leg loading information match those in Figure 3.8 for rapid transitions: $E_s = -40, g = 0.50$. Stance phase is also prolonged by continual force feedback [Cruse, 1985b], so all incoming connections to the CPGs were made weaker by comparison.

Motion of the FTi joint affects the rhythms of the CTr joint [Hess and Büschges, 1999]. Therefore, FTi extension halts CTr levation to depress the leg, and FTi flexion halts CTr depression to elevate the leg. The synapses provide graded input, causing stronger inhibition as the FTi joint moves past the reference positions, overpowering the load signal only once the joint has rotated 20% past the positions encoded in the FTi Ext. and FTi Flex. neurons. These can be biased upward by the Kin. Bias (Kinematic Bias) descending neuron, which reduces the joint’s range of motion.

The LPVF synapses are weaker than the load and motion synapses, with only half the current output of the FTi joint motion synapses. This establishes an important hierarchy of pathways: lacking any other feedback, a joint should continue to move in the same direction. In swing phase, kinematic information can interrupt this motion to enter stance phase. In stance phase, load feedback dominates, unless a strong signal encourages the transition to swing.

Many parameter values could be changed to effect the walking motion, but here we focus on descending commands and their effect on walking speed and phase asymmetry. The Int. Rot. (Intended Rotation) neurons in Figure 3.9 are set such that the foot just reaches the ground when the CPGs are disabled. The Aff. Stren. (Afferent Strength) neuron is tonically active at $-40$ mV. Activations for CPG $\rightarrow$ MN, CPG Speed, and Kin. Bias neurons were swept for comparison.

The mean speed of the simulated belt for different descending commands is shown in the top row of Figure 3.10. The data show that the walking speed is much more sensitive to motor output strength and sensory feedback than to CPG parameters.
Feedback Parameters are More Important to Walking Speed than CPG Parameters

Figure 3.10: Leg stepping speed is more sensitive to motor output and sensory input strengths than to CPG parameters. As the leg walks on the belt, the speed changes primarily due to changing the CPG→MN and Kinematic Bias neurons’ tonic stimulus. As the speed increases, the duration of swing phase remains fairly constant.

For instance, the leg can walk at speeds up to 40 cm/s by activating the CPG→MN neuron, which increases FTi flexor speed during stance, or activating the Kin. Bias neuron, which facilitates transitions between stance and swing. Varying either of these with $\delta$, however, shows that even as $\delta$ varies over its entire range of values that induce oscillation (Figure 3.5F), the stepping speed is not affected.

As animals increase their speed, they asymmetrically speed up their motion, leaving swing phase at roughly the same duration. The bottom row of Figure 3.10 shows that this also is true in our model. Swing phase lasts 1 s on average, and the standard deviation among trials that produce stable walking is low.
3.7 Discussion

Designing effective controllers for robots or neuromechanical models of animals requires that the intended behaviors can be designed with certainty. This work provides analysis, a design process, and tools to tune a neuromechanical simulation of part of our hexapod robot, MantisBot (Chapter 5, [Szczecinski et al., 2015a]). First, the stability and robustness of each servomotor’s neuromechanical controller is determined by our automated software tool feedbackDesign. Next, the stability and responsiveness of each segment’s CPG is designed by tuning the mutual inhibition to produce a specific value of $\delta$, the bifurcation parameter. Finally, we use our software tool simScan to run batches of AnimatLab 2 simulations and analyze the output to characterize subsystems or examine the system’s sensitivity to different parameters. This process and accompanying tools enable component-wise design of the controller rather than using expensive optimization techniques to tune the entire controller at once.

The goal of this work is not to produce networks that optimally complete a task, but networks that can be modulated by descending commands to perform many different tasks. Modulation means changing the system’s parameter values, so we must understand how they affect behavior. Our analysis helps rule out some parameter values, such as unstable servomotor controllers or CPGs that do not oscillate, but does not necessarily point to one best set. Individual behaviors, such as standing still, walking, or climbing each use sensory feedback and motor output differently, and the controller’s higher command centers, whether autonomous or remote-controlled, will need to modulate parameter values to perform these actions without destabilizing the system. This analysis and the accompanying software tools enable the modeler to identify combinations of parameter values that will cause instability, and avoid descending commands that would destabilize the system.

We demonstrate the utility of these tools by analyzing a reduced simulation of MantisBot’s middle leg. The leg has two joints, and each has its own neuromechanical
servomotor controller with parameters from Sections 3.4 and 3.5. Understanding the response properties of the joint controllers enabled us to quickly and easily add inter-joint pathways based on those in insects to coordinate a two-jointed leg into walking on a treadmill, as performed in animal experiments [Fischer et al., 2001, Schmidt et al., 2001]. We then changed the descending commands and showed that they control the behavior of the leg.

**Joint Controller Design** The servomotor controller in this work was designed to let us interface some of what is known about insect locomotion control with commercial off-the-shelf components. Of course animals do not have servomotors, and the neural networks that control locomotion evolved specifically to interface with muscles. However, we built some of what is known about insect motor control into a platform that is simpler to control than antagonistic actuators. For instance, our servomotor’s velocity is proportional to the MN activation [Watson and Ritzmann, 1998], negative feedback is used to stabilize the movement of the femur and the tibia [Bässler et al., 1996], and local positive velocity feedback may reinforce ongoing motion in walking animals [Hellekes et al., 2011]. In the future, feedbackDesign will be expanded to include muscles for actuators, making this tool useful for tuning neuromechanical models of animals. Muscle parameter values are notoriously difficult to tune [Hooper et al., 2016], so applying fast-to-simulate frequency domain models may be beneficial.

**CPG Parameter Value Exploration** In this paper we analyzed how weakening the mutual inhibition in a CPG increases its oscillation frequency. This is effectively no different from adding tonic current, as done in other studies [Daun-Gruhn, 2010, Daun-Gruhn et al., 2009], but exploring \( \delta \) gave us insight into the escape mechanism that drives oscillation in this model, how mutual inhibition controls timing, and double- transitions caused by excitatory inputs. It also has the benefit of not affecting the neurons’ maximal activations, which ultimately affect motion. In practice, a robot
or model might use one or both techniques to control frequency and sensitivity, and cpgDesign can be used to find equilibria and their stability to predict oscillation.

cpgDesign designs the synaptic connections within the CPGs to obtain the specified value of $\delta$, enabling us to quickly tune parameter values based on output behavior. However, it assumes that the CPG’s parameter values are symmetrical. This, as well as previous studies have shown that the apparent asymmetry (i.e. extensor dominated) in animal locomotion can be produced by afferent feedback, rather than parameter value asymmetry [Markin et al., 2010]. However, the sensory inputs that produce asymmetrical motion are still breaking the symmetry of the CPG, suggesting that such analysis may provide insight into the mechanisms behind asymmetrical timing.

Coordinating multiple joints The single-leg stepping model presented is meant to be the simplest demonstration that thorough characterization of the segment (i.e. single servomotor) controller simplifies more complicated design problems, such as tuning inter-segmental reflexes. Rather than trying to tune an entire system at once, building the leg model was trivial once the segment controllers were stable, rhythmic, and responsive. Not only are most of the parameter values of the system determined before assembly, but the remaining parameter values can be set by only a handful of descending commands. For instance, to make this model step, it was necessary to try a few different combinations of CPG → MN neuron activations. Once the joints produced torques sufficient to move the belt, the leg stepped. The feedback pathways from biology naturally produce a stepping motion, and our analysis of segmental control networks let us design our different feedback pathways to enforce these rules.

That said, the parameter values used here are by no means unique. Studies of animal control networks have shown repeatedly that many different parameter value combinations may produce similar behavior in different organisms (for a review, see
Marder et al. [Marder and Taylor, 2011]). However, to construct a model or robot, the designer needs to settle on one set of parameter values. The non-uniqueness of stable parameter values creates an opportunity to explore other combinations that may have other benefits, such as working stably over a range of walking speeds or directions. Controlling full legs of MantisBot in the future may require more complete analysis of how network parameters affect stability, which may reveal additional constraints on the values of the parameters studied here.

**Descending commands act on low-level controllers** In this work, we assumed that descending commands’ main role in controlling locomotion is to modulate the function of low-level control networks. In cockroaches, it is known that stimulating parts of the brain that respond to locomotion heading cause an inter-segmental reflex to reverse, changing coordination and thus the direction of travel [Martin et al., 2015]. In stick insects, the optomotor response changes the balance between negative and positive feedback in at least one leg joint, showing that descending commands can change the balance, and thus the gain of different feedback loops [Hellekes et al., 2011]. In humans, one set of feedback gains cannot stabilize all possible leg postures, showing that higher command centers can monitor joint-level feedback gains [Bingham et al., 2011]. Our design tools not only identify unstable parameter values, but also show how the system’s performance will change when descending commands modulate controllers. For example, feedbackDesign reveals what Afferent Strength descending neuron activities may destabilize the system at higher frequencies (Figure 3.3).

We also assumed that descending commands alter feedforward output from the joint control networks. In this work, the CPG→MN descending neuron controls the flexion speed of the FTi servomotor. However, getting our model to step required individual manipulation of this connection in each joint. Animals instead use “muscle synergies” [Zill et al., 2015] or “motor modules” [Ting et al., 2015] to move their legs in
a particular direction, which may be the result of modulating the CPG to MN strength of all the joints in a leg in a predetermined ratio. Changing direction would require a different combination, which could be mapped out in higher command centers, or in the thoracic networks themselves. This is a major concern for legs with more degrees of freedom, and we plan to investigate this functional organization as a solution for MantisBot in the future.

Design tool runtime We want fast design tools that run on a desktop computer. Fast tools enable the testing of many parameter sets, and using a desktop computer increases accessibility and simplifies use. feedbackDesign can calculate the stability margins of the network in Figure 3.3 in 10 ms, enabling us to generate all of the data in that figure in a mere 45 seconds. This made it practical to couple with an interior point optimizer, as shown in Figure 3.4. cpgDesign simulates CPG dynamics six times faster than real time, enabling us to create PRCs in about a minute. simScan automates the execution of AnimatLab 2 simulations, so the runtime depends on the complexity of the simulation and the amount of time simulated. That said, parallelizing their execution and eliminating the graphical rendering enables parameter values to be tested much faster than by hand. The speed of these tools make them practical for engineers and modelers alike, and enable trying many parameter values to find those that produce stable behavior.

Application to neuromechanical animal models or hardware robots The methods presented in this paper were tested by tuning a simulation of part of MantisBot’s leg. However, these tools may be used to tune a neuromechanical model of an animal or a hardware robot. For example, we recently expanded feedbackDesign to include the linear Hill muscle model for actuating limbs, enabling study of the stability and robustness of neuromuscular systems. We have also used simScan to adjust muscular properties to achieve the rapid, overdamped motion of running insects.
CHAPTER 3. DESIGN PROCESS AND TOOLS FOR DYNAMIC
NEUROMECHANICAL MODELS AND ROBOT CONTROLLERS

[Hooper et al., 2016]. These tools enable us to select parameter values for a model
that capture the dynamical features (rise time, overshoot, etc.) of animal neuromus-
cular systems, accelerating the development of the model, and enabling more detailed
hypotheses about animal locomotion to be addressed.

These tools also accelerate the design of controllers for hardware robots. The
robustness analysis provided by feedbackDesign ensures that the controller developed
in simulation is not sensitive to physical parameters that are difficult to measure,
such as motor damping, joint friction, or limb inertia. In addition, the AnimatLab
Robotics Toolkit (Chapter 5, [Szczecinski et al., 2015a]) exports simulation files that
control the robot (see Section 3.8.8), enabling simScan to perform hardware-in-the-
loop optimization. Such functionality is critical to tuning difficult-to-model features
of the robot’s controller, such as feedback gain from strain sensors in the legs. The
tools presented in this paper have broad application to the tuning of neuromechanical
models of animals and robot controllers alike.

3.8 Appendix

3.8.1 Neural Parameters

The parameter values for the network in Figure 3.1 are listed below. All neural
properties are scaled to ms time scale, producing base units of mV, uS, nF, and nA.
Unless otherwise noted, all neurons have the parameters $C_m = 5$, $G_m = 1$, $E_r = -60$,
$G_{NaP} = 0$.

**HC neurons.** $C_m = 5$, $G_m = 1$, $E_r = -60$, $G_{NaP} = 1$, $E_{NaP} = 50$, $A_h = 0.5$,
$S_h = -0.046$, $E_h = -60$, $A_m = 1$, $S_m = 0.046$, $E_m = -40$.

**Servo Interface.** $C_m = 50$, $E_r = -60 - 20 \cdot \theta_{\text{min}}/ (\theta_{\text{max}} - \theta_{\text{min}})$. This ensures that
when the simulation starts with $\theta = 0$, the Servo Interface neuron starts at its resting
potential. Any value will work, but a different value will cause unnecessary motion at startup.

**Perceived Rotation.** Like the Servo Interface neuron, \( E_r = -60 - 20 \cdot \theta_{\text{min}} / (\theta_{\text{max}} - \theta_{\text{min}}) \).

Rotation (slow). \( C_m = 50 \).

Flexion Trigger. \( E_r = -75 \).

Rotation (slow) \( C_m = 10 \).

### 3.8.2 Synaptic Parameters

Unless stated otherwise, all synapses have \( E_{\text{lo}} = -60 \) and \( E_{\text{hi}} = -40 \).

**HC \to IN.** \( g = 0.118, E_s = 300, E_{\text{hi}} = -20 \).

**IN \to HC.** \( g = 1.041, E_s = -100, E_{\text{hi}} = -20 \).

**MN \to Compare.** Excitatory connection: \( g = 1, E_s = -20, E_{\text{lo}} = -70, E_{\text{hi}} = -30 \). Inhibitory connection: \( g = 1, E_s = -100, E_{\text{lo}} = -70, E_{\text{hi}} = -30 \).

**Compare \to Servo Interface.** Excitatory connection: \( g = 1, E_s = 0 \). Inhibitory connection: \( g = 1, E_s = -100 \).

**Perceived Rotation \to Extension MN.** \( g = 0.5, E_s = -100 \).

**Perceived Rotation \to Rotation (fast/slow).** \( g = 0.0588, E_s = 300 \). This parameter combination makes the steady-state voltage of the postsynaptic neuron the current voltage of the presynaptic neuron.
CHAPTER 3. DESIGN PROCESS AND TOOLS FOR DYNAMIC NEUROMECHANICAL MODELS AND ROBOT CONTROLLERS

Rotation (fast/slow) \(\rightarrow\) Velocity. \(g = 0.133, E_s = -20, E_{hi} = -60.\)

Rotation (fast/slow) \(\rightarrow\) Velocity. \(g = 0.133, E_s = -100, E_{hi} = -60.\)

3.8.3 Mechanical Parameters

The servomotors run a proportional feedback loop with an experimentally determined gain of \(k = 15.2\) Nm/rad. Damping has been determined to be \(c = 2.5\) Nms/rad. The inertia was that of one MX-64T servomotor placed 30 cm from the joint, \(I = 13.5 \times 10^{-3}\) kg·m².

3.8.4 Computer Specifications

All data was collected using a desktop computer with an Intel i5-4690K CPU running at 3.50 GHz and 8 GB of RAM. feedbackDesign, cpgDesign, and simScan all parallelize their computations. Four parallel threads were used in this work.

3.8.5 Modulatory Pathways

To simplify figures in this paper, modulatory connections were drawn as synapses onto other synapses, as shown in Figure 3.11A. In practice, this was implemented as a disinhibitory pathway that changed the conductance, and thus sensitivity, of an interneuron (Figure 3.11B). This is similar to GABAergic synapses, which change the effective size of the postsynaptic neuron, reducing its sensitivity to incoming currents [Trappenberg, 2009]. The Modu. (Modulatory) neuron’s voltage controls the effective conductance of the connection between the Input and Output neurons between 0% when at \(-60\) mV and 100% when at \(-40\) mV. The plots in Figure 3.11C., generated with simScan, show that the activation of the Modu. neuron directly scales the Output neuron’s activation for the same Input activations.
All neurons in Figure 3.11B. have the typical set of parameters listed in Section 3.8.1, except the Inter. 1 neuron, whose resting potential is $E_r = -40$. The inhibitory synapses are both identical, with $G_s = 20$, $E_s = -61$, $E_{lo} = -60$, and $E_{hi} = -40$.

### 3.8.6 feedbackDesign

feedbackDesign automates the analysis presented in Secs. 3.4.1 and 3.4.2. The user provides parameter values for a closed- or open-loop network of neurons, synapses, and a servomotor and limb if desired. Neurons that interface as inputs or outputs to the servomotor, and that feedback to the input neuron, are specified by the user. The equilibrium state is found by simulating the system until the energy goes to 0. If this happens, then the eigenvalues of the equilibrium point are found to ensure that the point is stable. If instead the energy diverges, the system is deemed unstable and no further analysis is conducted.

The transfer function is generated for each neuron and neuron-servomotor complex, and the open-loop transfer function of each node (i.e. neuron or servomotor) is calculated by compounding Eqs. 3.37 and 3.38 for each node along the path. The closed loop transfer function is calculated by Equation 3.42. The user can also query the stability margins, which are calculated by using a Newton minimizer to find the crossing points of the gain and phase responses (Section 3.4.2). The system’s parameter values can be varied and these analyses repeated to produce plots like those in Figs. 3.3 and 3.4. feedbackDesign can be downloaded at http://biorobots.case.edu/download/neural_design_tools/feedbackDesign.zip.

### 3.8.7 cpgDesign

cpgDesign automates the analysis presented in Section 3.5.1. It assumes the network structure in Figure 3.5. For a set of parameter values, it will calculate $\delta$, or if $\delta$
Figure 3.11: Modulatory networks, as drawn in this paper (A), were implemented in
the model as disinhibitory pathways (B). C) Changing the Modu. neuron’s activation
adjusts the input-output gain. For a full explanation of this behavior, see Chapter
4.4.6.
is provided, it will compute the corresponding strength of mutual inhibition. The user can simulate the dynamics, with the option to add external stimuli. When the simulation is complete, the equilibrium points will be found at every time step, using a log-bounded interior point Broyden-Fletcher-Goldfarb-Shanno (BFGS) minimizer. The eigenvalues and eigenvectors will also be calculated. The user then has the option to animate the phase portrait for each neuron over time, and calculate the nullclines corresponding to the equilibrium states. In this way the user can gain an intuitive understanding of the CPG’s dynamics. Finally, the program facilitates studying phase response properties by calculating whether or not the output was periodic, the period of oscillation, and the rising edges of one HC’s activity. cpgDesign can be downloaded at

\[ http://biorobots.case.edu/download/neural_design_tools/cpgDesign.zip \]

### 3.8.8 simScan

simScan automates the analysis presented in Secs. 3.5.2 and 3.6.2. The user makes a simulation with AnimatLab 2 and exports a “Standalone” simulation, which can be run from the command line. The user then tells simScan the directory of this file, the parameters to change and their values, and an objective function with which to process the data (in this paper, it simply extracts the speed of the “treadmill”). simScan then runs the simulations with the desired values and saves the output from the objective function. This process can easily be parallelized, and multiple objective functions can be used to measure different quantities. simScan will also produce a function handle that takes the desired parameters as inputs, and runs the simulation and computes the objective as an output. This handle can be used to optimize the parameters of the simulation to perform a specific task. simScan can be downloaded at
http://biorobots.case.edu/download/neural_design_tools/simScan.zip.
Chapter 4

A Functional Subnetwork Approach to Designing Synthetic Nervous Systems that Control Legged Robot Locomotion

This chapter is currently under review in the journal Frontiers in Neurorobotics.

4.1 Abstract

A dynamical model of an animals nervous system, or synthetic nervous system (SNS), is a potentially transformational control method. Due to increasingly detailed data on the connectivity and dynamics of both mammalian and insect nervous systems, controlling a legged robot with an SNS is largely a problem of parameter tuning. Our approach to this problem is to design functional subnetworks that perform specific operations, and then assemble them into larger models of the nervous system. In this paper, we present networks that perform addition, subtraction, multiplication, divi-
sion, differentiation, and integration of incoming signals. Parameters are set within each subnetwork to produce the desired output by utilizing the operating range of neural activity, $R$, the gain of the operation, $k$, and bounds based on biological values. The assembly of large networks from functional subnetworks underpins our recent results with MantisBot.

4.2 Introduction

The development of robotic control that can closely match the dexterity and adaptability found in the animal kingdom has so far remained elusive. This is because the control of locomotion is a complex process controlled by dynamic systems which are not fully understood. However, recent advances in neural imaging and recording have lead to an increase in the abundance and detail of our knowledge of how an animal’s nervous system controls its body within the context of its environment (for a recent review, see [Buschmann et al., 2015]).

These advances have lead to an explosion of bio-inspired robotic systems in recent years (for a review, see [Ijspeert, 2014]). These models can be broadly categorized into a range of template and anchor models. In a template model, biological principles are abstracted, such as using a spring-loaded inverted pendulum (SLIP) model to investigate bipedal locomotion [Blickhan, 1989] or using Whegs to investigate insect locomotion [Allen et al., 2003, Schroer et al., 2004]. These models seek to explain how specific characteristics of animal locomotion lead to desired behaviors, or they exploit certain principles of animal locomotion for more agile robotic systems. Anchor models, such as Pleurobot [Karakasiliotis et al., 2016], Puppy [Hunt et al., 2017], MantisBot [Szczecinski et al., 2017a, Szczecinski and Quinn, 2017c] and others are developed to explore how specific animals are capable of providing adaptable locomotion with their unique morphology and physical constraints.
The template versus anchor model distinction is not limited to physical models; it can also be applied to control systems. The majority of robotic controllers so far have been template models, either mathematical abstractions of neural systems, or black box artificial neural networks. This is because effective tools for setting parameters in more realistic, dynamic neural models to produce reliable behavior in a robotic system do not yet exist. In spite of growing knowledge about the neural connectivity that underlies locomotion control, detailed data for tuning these systems (neural time constants, ion channel conductivities, synaptic conductivities, etc.) remain largely unavailable, requiring the modeler or engineer to tune these parameter values. However, this is an inherently difficult task because there are many parameters to be tuned in a model, and likely many different parameter combinations that lead to indistinguishable performance [Prinz et al., 2004, Marder and Taylor, 2011]. Thus, the emphasis in choosing parameter values should not be on selecting the singular “correct” values, but rather sufficiently “effective” values. In this work, we set parameter values in functional subnetworks for addition, subtraction, multiplication, division, differentiation, and integration of incoming signals, and use analytical techniques to identify constraints on the parameter values that must be met for the intended calculations to occur. Larger networks can then be assembled from these subnetworks with no additional tuning [Szczecinski et al., 2017a, Szczecinski and Quinn, 2017c].

Oftentimes neuromechanical models are tuned by hand [Szczecinski et al., 2014, Daun-Gruhn and Tóth, 2010, Markin et al., 2016], which is a time-consuming and imprecise process. Such imprecision may be acceptable in simulation studies, but provides many difficulties for robots that must interact with real environments. Techniques do exist for tuning controllers based on animal locomotion data [Schilling et al., 2013a, Hunt et al., 2015b, Karakasiliotis et al., 2016, Hunt et al., 2017]. However, collecting kinematic and dynamic data from animals is time-consuming and expensive, and once collected, must be further processed to scale the dynamics of
the animal to the robot [Karakasiliotis et al., 2016, Hunt et al., 2017]. In addition, using cross-individual average values for tuning dynamical neural models may fail in many cases, because the average may not represent any one individual [Golowasch et al., 2002, Marder and Taylor, 2011]. Machine learning methods may be used to tune more abstract control networks to produce competent, animal-like locomotion [Schilling et al., 2013a], but because the control network is abstracted, so are the biological insights gained from the model.

Another approach is to use genetic algorithms (GAs) [Beer and Gallagher, 1992, Haferlach et al., 2007, Izquierdo and Beer, 2013, Agmon and Beer, 2013] or reservoir computing (RC) [Dasgupta et al., 2015] to test many different networks and parameter values, based on a simulated agent’s performance, rather than mimicking animal data. GAs can be effective at finding networks that perform specific operations, such as oscillating [Beer and Gallagher, 1992], navigating [Haferlach et al., 2007], or switching between foraging tasks [Agmon and Beer, 2013]. However, this approach has some drawbacks. Specifically, the evolution process is slow, requiring the simulation of hundreds or thousands of parameter combinations [Agmon and Beer, 2013], which may take days without great computing power. The speed and likelihood of success can be increased by embedding functional subnetworks in the network [Pasemann et al., 2001, Haferlach et al., 2007], which may be identified by brute-force [Prinz et al., 2003], dynamical systems analysis [Hunt et al., 2017], or constraints on network connectivity and parameter values [Haferlach et al., 2007]. In this paper, we analytically derive parameter constraints to eliminate the need for GAs altogether, and guarantee network performance.

RC methods simulate large “reservoirs” of randomly connected dynamical neuron models, and then use machine learning to map reservoir activity to learned useful values. While this method can produce capable robotic controllers [Dasgupta et al., 2015], the final system is likely more complicated than is ultimately necessary, increas-
ing its computational cost to implement. In addition, the final system is a black box, which does not provide any insights about nervous system function. The methods in this paper enable the direct assembly and tuning of dynamical networks without the need of large reservoirs of neurons.

This work analytically derives constraints that govern the behavior of synthetic nervous systems (SNSs) built from dynamical neural networks. These constraints were derived as a result of our previous network design work (Chapter 3, [Szczecinski et al., 2017b]), and have enabled the rapid assembly and testing of my recent robot control networks (Chapters 6 through 9, [Szczecinski et al., 2017a, Szczecinski and Quinn, 2017c]). An SNS designer can apply these constraints to find parameter values needed for a functional network. Section 4.3 presents the neural and synaptic models, and explains how the neural system encodes mechanical inputs and outputs. Section 4.4 derives two basic synapse types, “signal transmission” and “signal modulation”, and uses them to derive constraints on synaptic parameters in networks performing addition, subtraction, multiplication, and division of two incoming signals. Section 4.5 derives constraints on neural and synaptic parameters in networks that differentiate and integrate incoming signals as a function of time. Results showing that the networks perform as intended are provided throughout the manuscript, and Tables 4.1 and 4.2 summarize the design constraints. Finally, Sec. 4.7 explores how these techniques may be used to tune robot controllers and neuromechanical models of animals, and how they may be improved in the future.

4.3 Methods: Models and Approach

We model neurons as nonspiking Hodgkin-Huxley compartments [Cofer et al., 2010], the same basic model as used in continuous-time recurrent neural networks [Haferlach et al., 2007, Agmon and Beer, 2013]. The leaky integrator dynamics capture the
most basic behavior of neurons and allows more complex behaviors to be added with additional ion channels, if desired. This work is not concerned with the specifics on how action potentials are generated and have left out Hodgkin-Huxley sodium and potassium currents. The membrane voltage, $V$, may be seen as a proxy for the spiking frequency of a spiking neuron. $V$ varies according to the differential equation

$$C_m \frac{dV}{dt} = I_{\text{leak}} + I_{\text{syn}} + I_{\text{app}} \quad (4.1)$$

where

$$I_{\text{leak}} = G_m \cdot (E_r - V), \quad (4.2)$$

$$I_{\text{syn}} = \sum_{i=1}^{n} G_{s,i} \cdot (E_{s,i} - V), \quad (4.3)$$

and $I_{\text{app}}$ is an optional external stimulus. Eqs. 4.2 and 4.3 define the leak and synaptic currents, respectively. Both follow the same basic form of a conductance $G$ multiplied by the difference between the current membrane voltage, $V$, and a constant reference voltage (i.e. reversal potential), $E$. $E_r$ is the resting potential of the neuron, and $C_m$ and $G_m$ are the capacitance and conductance of the cell membrane, respectively. Unless otherwise noted, all units in this paper are scaled to nA for current, mV for potentials, nF for capacitances, and $\mu$S for conductances.

Neurons communicate via synapses. The conductance, $G_{s,i}$ in Eq. 4.3, is a threshold linear function of the $i^{th}$ incoming (i.e. presynaptic) neuron’s voltage. Synapses communicate via piecewise-linear functions described as

$$G_{s,i} = \begin{cases} 
0, & \text{if } V_{\text{pre}} < E_{lo}, \\
g_{s,i} \cdot \frac{V_{\text{pre}} - E_{lo}}{E_{hi} - E_{lo}}, & \text{if } E_{lo} < V_{\text{pre}} < E_{hi}, \\
g_{s,i}, & \text{if } V_{\text{pre}} > E_{hi}. 
\end{cases} \quad (4.4)$$
The parameters $g_{s,i}$, $E_{lo}$, and $E_{hi}$ are constants representing the synapse’s maximum conductance, its lower threshold, and its upper threshold, respectively. The relationship between the presynaptic neuron voltage, synaptic conductance, and postsynaptic neuron voltage is illustrated in Fig. 4.1A. We prefer this piecewise-linear representation better than a sigmoidal function for several reasons. First, the thresholds ensure that for low activations, synapses conduct exactly 0 current. This could represent a reduced model of a spiking neuron, which transmits no information while it is not spiking. Second, Eq. 4.4 contains no transcendental terms, facilitating analytical manipulation of the equations. A discontinuous system does complicate traditional gradient-based optimization methods, but this structure can be exploited to make these methods unnecessary. In the following sections, we show how networks of three or four neurons with synapses between them can be constructed and analytically tuned to perform mathematical operations on the input signals, such as addition or differentiating with respect to time.

Instead of analyzing $V$ when designing these networks, we shift the neural activity to simplify analysis. For each neuron, we substitute $U = V - E_r$, the activation level above the resting voltage. A typical value is $E_r = -60$ mV, but using $U$ for analysis rather than $V$ lets us apply the same analysis no matter what $E_r$ is. We also set $G_m = 1 \, \mu S$, which is a typical value [Daun-Gruhn et al., 2009, Daun-Gruhn, 2010].

For the synapses, we set $E_{lo} = E_r$ of the presynaptic neuron, and introduce a new parameter $R = E_{hi} - E_{lo}$. Thus, a synapse transmits more information as the presynaptic neuron’s voltage rises above its resting potential, and exhibits an “operating range” of $R$ mV. The constraints we apply ensure that $U_{pre} \in [0, R]$, meaning that the synapse is always active, but never saturates. Thus, we can replace $G_s$ with the second line of Eq. 4.4. Applying the substitutions described so far,

$$G_s = g_s \cdot \frac{V_{pre} - E_{lo}}{E_{hi} - E_{lo}} = g_s \cdot \frac{U_{pre}}{R} = \frac{g_s}{R} \cdot U_{pre}. \quad (4.5)$$
CHAPTER 4. A FUNCTIONAL SUBNETWORK APPROACH TO DESIGNING SYNTHETIC NERVOUS SYSTEMS THAT CONTROL LEGGED ROBOT LOCOMOTION

Figure 4.1: Graphical representation of synaptic dynamics, and mapping between mechanical and neural values. (A) Graphical representation of how synapses couple neural dynamics. Note that $R$ is marked on the plot. (B) Enhanced version of the motor control network from [Szczecinski et al., 2017b], showing how $R$ relates mechanical and neural values. Mechanical values are drawn in red, and neural values are drawn in blue.
For each synapse we also introduce the parameter $\Delta E_{s,i} = E_{s,i} - E_{r,post}$, where $E_{r,post}$ is the resting potential of the postsynaptic, or receiving neuron.

Making all of these substitutions in Eqs. 4.1, 4.2, and 4.3 gives the response

$$C_m \frac{dU}{dt} = -U + \sum_{i=1}^{n} \frac{g_{s,i}}{R} \cdot U_{pre,i} \cdot (\Delta E_{s,i} - U) + I_{app}.$$  (4.6)

When $U = R$, the neuron is fully active, and when $U = 0$, the neuron is inactive. We can use this knowledge to categorize synapses as excitatory or inhibitory, depending on the sign of $\Delta E_{s,i}$. If $\Delta E_{s,i} \geq R$, then the $i^{th}$ synapse will always transmit positive current, no matter the instantaneous value of $U$. Thus, this synapse will cause $U$ to increase, and is therefore excitatory. Similarly, if $\Delta E_{s,i} \leq 0$, then the $i^{th}$ synapse will always transmit negative current, no matter the instantaneous value of $U$. Thus, this synapse will cause $U$ to increase, and is therefore inhibitory.

### 4.3.1 Mapping Between Neural and Mechanical Values

The nervous system encodes physical quantities as neural activity. In insects, the firing rate of sensory neurons encode the stretch of chordotonal organs [Field and Matheson, 1998] and the strain of campaniform sensilla [Zill et al., 2004], among other physical quantities. Typical robot controllers perform operations on these signals to provide meaningful information for control actions. These operations may include subtractions of measured and reference values, differentiation or integration of error values, or gain adjustments. Neural systems perform these same operations, but in a transformed space. The exact transformation that nervous systems use is not known, but for reliable behavior, it is necessary that sensory information is mapped to neural activity in a predictable way. Thus, we map any sensory input, $\theta$, to an applied current,

$$I_{app} = R \cdot \frac{\theta - \theta_{\min}}{\theta_{\max} - \theta_{\min}}.$$  (4.7)
where $R$ is the “operating range” specified in the previous section. Figure 4.1Bi and Biii graphically illustrate this relationship. If a sensory neuron has only this applied current and leak current, Eq. 4.6 shows that

$$C_m \frac{dU}{dt} + U = R \cdot \frac{\theta - \theta_{\text{min}}}{\theta_{\text{max}} - \theta_{\text{min}}}.$$  

This means that the sensory neuron acts as a low-pass filter with time constant $\tau = C_m$. It is trivial to show that when the neuron is at equilibrium (i.e. $dU/dt = 0$),

$$U^* = R \cdot \frac{\theta - \theta_{\text{min}}}{\theta_{\text{max}} - \theta_{\text{min}}},$$  

where the superscript “$*$” specifies the equilibrium value. (Throughout this manuscript, the equilibrium activation of neuron $U$ will be referred to as $U^*$, and the neuron itself will be referred to as $U$.) Equation 4.9 means that the neuron’s activation above its rest potential encodes the sensory signal. In addition to perceiving sensory information, commands must be issued in the same transformation. Thus, we map the commanded sensory quantity, $\theta_{\text{comm}}$, to the commanded neural activation, $U_{\text{comm}}$, with the inverse function of Eq. 4.9,

$$\theta_{\text{comm}} = \theta_{\text{min}} + \frac{U_{\text{comm}}}{R} \cdot (\theta_{\text{max}} - \theta_{\text{min}}).$$  

Figure 4.1B demonstrates such a transformation within a diagram of a neural feedback loop, controlling the position of a motor. The purpose of this paper is not to analyze how this particular network functions; for a detailed analysis of this network and its function, see [Szczecinski et al., 2017b].

In this way the nervous system may specify an intended motion, such as the rotation of a joint, encoded in neural activity. In our synthetic nervous systems, $R$ specifies how mechanical quantities and neural activation are related. Thus, the
tuning of every functional subnetwork described in this work relies on \( R \), which the designer specifies before tuning the rest of the network. Two other parameters are critical for tuning these subnetworks: the amplification of synaptic transmission, \( k_{syn} \) (discussed in Sec. 4.4.1), and the synaptic reversal potential, \( \Delta E_s \). From these values, biological parameters such as synaptic conductance and neural tonic drive can be directly calculated. This makes network design intuitive, enabling the designer to select biological parameter values based on functional ones.

4.4 Methods: Arithmetic Subnetworks

This section describes how to use typical engineering quantities to design neural and synaptic pathways. We can understand how these pathways work by manipulating their equilibria, something that a machine learning technique does not leverage. The steady-state activation \( U^* \) is calculated by solving for \( U \) when \( dU/dt = 0 \),

\[
0 = -U^* + \sum_{i=1}^{n} \frac{g_{s,i}}{R} \cdot U_{pre,i} \cdot (\Delta E_{s,i} - U^*) + I_{app}. \tag{4.11}
\]

Moving all \( U^* \) terms to the left hand side,

\[
U^* \cdot (1 + \sum_{i=1}^{n} \frac{g_{s,i}}{R} \cdot U_{pre,i}) = \sum_{i=1}^{n} \frac{g_{s,i}}{R} \cdot U_{pre,i} \cdot \Delta E_{s,i} + I_{app}. \tag{4.12}
\]

Solving for \( U^* \),

\[
U^* = \frac{\sum_{i=1}^{n} \frac{g_{s,i}}{R} \cdot U_{pre,i} \cdot \Delta E_{s,i} + I_{app}}{1 + \sum_{i=1}^{n} \frac{g_{s,i}}{R} \cdot U_{pre,i}}. \tag{4.13}
\]

This solution is the basis for the remainder of Sec. 4.4.
4.4.1 Signal Transmission Pathways

The goal of a signal transmission pathway is to cause the postsynaptic neuron’s voltage to be some ratio of the presynaptic neuron’s voltage. We call this ratio $k_{syn}$. The $U_{pre,i}$ terms in the denominator of the right hand side of Eq. 4.13 mean that $k_{syn}$ changes as $U_{pre}$ changes, so we approximate $k_{syn}$ as $U_{post}/U_{pre}$ when the presynaptic neuron is fully activated (i.e. $U_{pre} = R$). The steady-state response of a neuron with a single synaptic input and no applied current can be written based on Eq. 4.13,

$$U_{post}^* = \frac{g_s \cdot R \cdot U_{pre} \cdot \Delta E_s}{1 + \frac{g_s \cdot R}{R} \cdot U_{pre}}.$$  

To find $k_{syn}$ for this synapse, we first divide both sides of Eq. 4.14 by $U_{pre}$,

$$\frac{U_{post}^*}{U_{pre}} = \frac{g_s \cdot R \cdot U_{pre} \cdot \Delta E_s}{U_{pre} \cdot (1 + \frac{g_s \cdot R}{R} \cdot U_{pre})}.$$  

Next, we want to find $k_{syn}$, which can be calculated for any value of $U_{pre}$. To simplify analysis and improve the clarity of this derivation, we set find $k_{syn}$ when $U_{pre} = R$. Then, we show how to set parameter values to keep $k_{syn}$ nearly constant, even as $U_{pre}$ changes. Making this substitution,

$$\frac{U_{post}^*}{R} = k_{syn} = \frac{g_s \cdot R \cdot \Delta E_s}{R \cdot (1 + \frac{g_s \cdot R}{R} \cdot R)}.$$  

Finally, reducing $R/R$ terms reveals

$$k_{syn} = \frac{g_s \cdot \Delta E_s}{R \cdot (1 + g_s)}.$$  

Rearranging to solve for $g_s$,

$$g_s = \frac{k_{syn} \cdot R}{\Delta E_s - k_{syn} \cdot R}.$$  

79
Because $g_s$ must be positive, and the numerator of Eq. 4.18 is always positive, Eq. 4.18 is also subject to the constraint

$$\Delta E_s > k_{syn} \cdot R.$$ (4.19)

Equation 4.18 will be used to tune addition, subtraction, multiplication, and division networks (Secs. 4.4.3 through 4.4.6).

### 4.4.2 Signal Modulation Pathways

We may also use synapses to modulate a neuron’s sensitivity to other inputs. Based on Eq. 4.13, the steady-state response of a neuron with only an applied current $I_{app}$ is simply

$$U_{post}^* = I_{app},$$ (4.20)

if we set $G_m = 1$. For example, this is the case for a sensory neuron that receives applied current proportional to a sensor’s state, such as a joint angle (Fig. 4.1B), muscle stretch, or touch sensor. However, the nervous system may need to actively increase or reduce the sensitivity of the sensory neuron depending on context. Hyperpolarizing or depolarizing the neuron, however, would cause sensory information to be truncated (i.e. $V_{pre} < E_{lo}$). We can change the sensitivity of this neuron without losing sensory information by adding a synaptic input to the response from Eq. 4.20:

$$U_{post}^* = \frac{g_s \cdot U_{pre} \cdot \Delta E_s + I_{app}}{1 + g_s \cdot U_{pre}}.$$ (4.21)

To quantify how $U_{pre}$ modulates $U_{post}^*$ for a given $I_{app}$, we introduce the parameter $c_{syn}$, which quantifies this degree of modulation. We define $c_{syn}$ as $U_{post}^*/U_{pre}$, the same as $k_{syn}$, but with the understanding that $U_{pre}$ will decrease $U_{post}^*$ in this case.
Dividing both sides of Eq. 4.21 by $U_{\text{pre}}$ and using the definition of $c_{\text{syn}}$,

$$
\frac{U^*_{\text{post}}}{U_{\text{pre}}} = c_{\text{syn}} = \frac{\frac{g_s}{R} \cdot U_{\text{pre}} \cdot \Delta E_s + I_{\text{app}}}{U_{\text{pre}} \cdot (1 + \frac{g_s}{R} \cdot U_{\text{pre}})}. 
$$

(4.22)

As in the previous section, we will solve for $c_{\text{syn}}$ when $U_{\text{pre}} = R$ to simplify analysis. Making this substitution and reducing $R/R$ terms,

$$
c_{\text{syn}} \cdot R = \frac{g_s \cdot \Delta E_s + R}{1 + g_s}. 
$$

(4.23)

Multiplying both sides by the denominator of the right hand side and expanding,

$$
c_{\text{syn}} \cdot R + c_{\text{syn}} \cdot R \cdot g_s = g_s \cdot \Delta E_s + R. 
$$

(4.24)

Collecting $g_s$ terms on the left hand side,

$$
c_{\text{syn}} \cdot R \cdot g_s - g_s \cdot \Delta E_s = R - c_{\text{syn}} \cdot R. 
$$

(4.25)

Solving Eq. 4.25 for $g_s$,

$$
g_s = \frac{c_{\text{syn}} \cdot R - R}{\Delta E_s - c_{\text{syn}} \cdot R}. 
$$

(4.26)

Just as in Sec. 4.4.1, $g_s > 0$ depends only on $R$, which the designer specifies beforehand, $\Delta E_s$, which is limited by biological constraints, and $c_{\text{syn}}$, which the designer picks based on network function. $\Delta E_s$ should be negative, and as close to 0 as possible to minimize hyperpolarization of the postsynaptic neuron. Equation 4.26 will be used to tune division and multiplication networks (Secs. 4.4.5 and 4.4.6).

### 4.4.3 Addition

A subnetwork that approximates linear addition of the form $U^*_{\text{post}} = k_{\text{syn}} \cdot (U_{\text{pre},1} + U_{\text{pre},2})$ may underlie positive feedback mechanisms, which increase motor neuron ac-
CHAPTER 4. A FUNCTIONAL SUBNETWORK APPROACH TO DESIGNING SYNTHETIC NERVOUS SYSTEMS THAT CONTROL LEGGED ROBOT LOCOMOTION

tivation proportional to sensory inputs such as force sensing organs [Zill et al., 2004], or used to sum sensory signals from different body segments [Mittelstaedt, 1957]. We construct such a network by using two Signal Transmission pathways as presented in Sec. 4.4.1.

Let us rewrite Eq. 4.13 here, for clarity:

\[ U^* = \frac{\sum_{i=1}^{n} \frac{g_{s,i}}{R} \cdot U_{pre,i} \cdot \Delta E_{s,i} + I_{app}}{1 + \sum_{i=1}^{n} \frac{g_{s,i}}{R} \cdot U_{pre,i}}. \]  

(4.27)

This equation shows \( U_{pre,i} \) in both the numerator and denominator. To capture addition, we wish to minimize the impact of \( U_{pre,i} \) on the denominator. This is accomplished by minimizing \( g_s \). However, if \( g_s = 0 \), then the network will not function at all. Therefore, we instead maximize \( \Delta E_s \), which yields a small \( g_s \) (Eq. 4.18). Mathematically, there is no limit on \( \Delta E_s \), but synaptic potentials are limited in biological systems. In our work, we choose the reversal potential of calcium (\( E_s = 134 \) mV), which yields \( \Delta E_s = E_s - E_r = 134 - (-60) = 194 \) mV, and specify \( R = 20 \) mV. To design a pathway where \( k_{syn} = 1 \), for example, we plug these values into Eq. 4.18, which gives \( g_s = 115 \) nS. The contour plots in Fig. 4.2A show that the network matches the ideal behavior very closely over the operating range \( U_{sum} \in [0, R] \). These design constraints are summarized in Table 4.1, and Fig. 4.2B graphically shows the accuracy of the subtraction network.

4.4.4 Subtraction

A subnetwork that approximates linear subtraction of the form \( U_{post}^* = k_{syn} \cdot (U_{pre,1} - U_{pre,2}) \) may underlie negative feedback mechanisms, which are important for controlling many parameters in locomotion [Pearson, 1993, Peterka, 2003, Buschmann et al., 2015]. Just as in the previous section, Eq. 4.18 is used to find \( g_s \) for each pathway.

Designing a subtraction network requires that we pay attention to how the two
Figure 4.2: Data demonstrating the function of arithmetic networks. Each contour plot represents cross sections of the response surface, as depicted at the top. The network diagram, relevant parameters, and data are shown for addition (A), subtraction (B), division (C), and multiplication (D). Triangular synaptic terminations stand for excitatory inputs, and filled round terminations stand for inhibitory inputs. For each operation, the contour on the right is the ideal output, and the contour on the left is the actual operation for the parameter values listed. Free parameters from Table 4.1 are highlighted in gray.
synapses affect one another. Since the reversal potentials of hyperpolarizing ion channels are not much more negative than typical resting potentials, larger \( g_{s,2} \) values are required to transmit information than for depolarizing ion channels. This makes it harder to minimize \( g_s \) like we did in the previous section. Equation 4.13 enables us to constrain \( g_{s,2} \) such that when \( U_{pre,1} = R \) and \( U_{pre,2} = R \), \( U^*_{post} = 0 \). Starting with the neuron response in Eq. 4.13 for two synaptic currents and no applied current,

\[
U^*_{post} = \frac{g_{s,1}/R \cdot U_{pre,1} \cdot \Delta E_{s,1} + g_{s,2}/R \cdot U_{pre,2} \cdot \Delta E_{s,2}}{1 + g_{s,1}/R \cdot U_{pre,1} + g_{s,2}/R \cdot U_{pre,2}}.
\] (4.28)

Substituting in \( U_{pre,1} = R \), \( U_{pre,2} = R \), and \( U^*_{post} = 0 \),

\[
0 = \frac{g_{s,1}/R \cdot R \cdot \Delta E_{s,1} + g_{s,2}/R \cdot R \cdot \Delta E_{s,2}}{1 + g_{s,1}/R \cdot R + g_{s,2}/R \cdot R} \quad (4.29)
\]

\[
0 = \frac{g_{s,1} \cdot \Delta E_{s,1} + g_{s,2} \cdot \Delta E_{s,2}}{1 + g_{s,1} + g_{s,2}} \quad (4.30)
\]

\[
0 = g_{s,1} \cdot \Delta E_{s,1} + g_{s,2} \cdot \Delta E_{s,2} \quad (4.31)
\]

\[
g_{s,2} = \frac{\Delta E_{s,1}}{\Delta E_{s,2}} \cdot -g_{s,1} \quad (4.32)
\]

Substituting Eq. 4.18 for \( g_{s,1} \),

\[
g_{s,2} = \frac{\Delta E_{s,1}}{\Delta E_{s,2}} \cdot -\frac{k_{syn} \cdot R}{\Delta E_{s,1} - k_{syn} \cdot R} \quad (4.33)
\]

To be physically realizable, \( g_{s,2} > 0 \). Because \( g_{s,1} > 0 \) and \( \Delta E_{s,1} > 0 \), \( g_{s,2} > 0 \) if and only if \( \Delta E_{s,2} < 0 \). Thus, it is critical that \( \Delta E_{s,2} < 0 \).

Just as for the addition network, we minimize \( g_{s,1} \) by maximizing \( \Delta E_{s,1} \). If \( R = 20 \) mV and \( k_{syn} = 1 \), then \( g_{s,1} = 115 \) nS and \( \Delta E_{s,1} = 194 \) mV. To tune \( g_{s,2} \), we first select \( \Delta E_{s,2} = -40 \) mV, then we solve Eq. 4.33 to find \( g_{s,2} = 558 \) nS. These design constraints are summarized in Table 4.1.
4.4.5 Division

A subnetwork that approximates division of the form

\[
U_{\text{post}} = \frac{U_{\text{pre},1}}{1 + \frac{1-c_{\text{syn}}}{c_{\text{syn}} \cdot R} \cdot U_{\text{pre},2}} \tag{4.34}
\]

replicates the function of GABA synapses that regulate activity in the brain. A key reason for this behavior is that the reversal potential of GABA-ergic synapses is about equal to the resting potential of the postsynaptic neuron [Trappenberg, 2009]. Equation 4.26 is used to find \(g_s\) for the division pathway.

The synapse from \(U_{\text{pre},1}\) to \(U_{\text{post}}\) is tuned as an excitatory Signal Transmission pathway with \(k = 1\), as in Sec. 4.4.1. In our work, \(R = 20\) mV, \(\Delta E_{s,1} = 194\) mV, and Eq. 4.18 tells us that \(g_{s,1} = 115\) nS. Such a small \(g_s\) ensures that the signal from \(U_{\text{pre},1}\) to \(U_{\text{post}}\) is transmitted without greatly affecting the sensitivity of \(U_{\text{post}}\) to inputs. That is, the effect of \(U_{\text{pre},1}\) on the denominator of \(U_{\text{post}}^*\) is very nearly zero.

The synapse from \(U_{\text{pre},2}\) to \(U_{\text{post}}\) is tuned as a Signal Modulation pathway, as analyzed in Sec. 4.4.2. Setting \(\Delta E_{s,2} = 0\) will eliminate \(U_{\text{pre},2}\)'s influence on the numerator of \(U_{\text{post}}^*\). Substituting this case into Eq. 4.26 and reducing,

\[
g_{s,2} = \frac{1 - c_{\text{syn}}}{c_{\text{syn}}}, \tag{4.35}
\]

where \(U_{\text{post}}^* = c_{\text{syn}} \cdot R\) when \(U_{\text{pre},1} = U_{\text{pre},2} = R\), their maximal value. Equation 4.35 also reveals that since \(g_{s,2} > 0\), \(0 < c_{\text{syn}} < 1\).

The steady-state response of the network is the result of these two synaptic inputs, as written in Eq. 4.28. Substituting Eq. 4.35, and specifying that \(k_{\text{syn},1} = 1\), \(U_{\text{post}}^*\) simplifies to

\[
U_{\text{post}}^* \approx \frac{g_{s,1} / R \cdot \Delta E_{s,1} \cdot U_{\text{pre},1} + g_{s,2} / R \cdot \Delta E_{s,2} \cdot U_{\text{pre},2}}{1 + g_{s,1} / R \cdot U_{\text{pre},1} + \frac{1-c_{\text{syn}}}{c_{\text{syn}} \cdot R} \cdot U_{\text{pre},2}} \approx \frac{U_{\text{pre},1}}{1 + \frac{1-c_{\text{syn}}}{c_{\text{syn}} \cdot R} \cdot U_{\text{pre},2}} \tag{4.36}
\]
In our network, we wished $U^*_{\text{post}} = 1$ when $U_{\text{pre},2} = R$, so we set $c_{\text{syn}} = 1/R = 0.05$, which makes $g_{s,2} = 19 \mu S$. When $c_{\text{syn}}$ is close to 0, $U_{\text{pre},2}$ can strongly reduce $U_{\text{post}}$'s sensitivity to inputs. When $c_{\text{syn}}$ is close to 1, $U_{\text{pre},2}$ can only weakly reduce $U_{\text{post}}$’s sensitivity to inputs. Figure 4.2C shows that this network performs the intended division of the signals. Table 4.1 summarizes these design constraints.

### 4.4.6 Multiplication

A subnetwork that approximates multiplication of the form $U^*_{\text{post}} = U_{\text{pre},1} \cdot U_{\text{pre},2}/R$ can be used to control the gain of a sensory feedback loop, a frequently observed characteristic of neural systems that control locomotion [Cruse, 1981, Gabriel and Büschges, 2007] and posture [Peterka and Loughlin, 2004].

A multiplication network can be assembled by replacing the Modulatory Pathway in the division network with two identical Modulatory Pathways in series, connected into a disinhibitory network (see Fig. 4.2D). This works because the product of two numbers, $a \cdot b = a/(1/b)$. However, tuning the Modulatory Pathway for the multiplication network differs from tuning the division network. This is because the right-side pathway of the network in Figure 4.2D must make $U^*_{\text{post}} = 0$, no matter how active $U_{\text{pre},1}$ becomes (because $a \cdot 0 = 0$, no matter the value of $a$). Thus, according to Eq. 4.22, $c_{\text{syn}} = 0$, unlike the division network, for which $0 < c_{\text{syn}} < 1$. Solving Eq. 4.26 when $c_{\text{syn}} = 0$ reveals that

$$g_{s,2} = -R/\Delta E_{s,2}. \quad (4.37)$$

To solve for $g_{s,2}$, we must first select $\Delta E_{s,2}$. If $\Delta E_{s,2} = 0$ like for the division network, then Eq. 4.37 divides by 0. If $\Delta E_{s,2} > 0$, then $g_{s,2} < 0$, which is physically not realizable. Therefore, we must choose a value $\Delta E_{s,2} < 0$. The more negative $\Delta E_{s,2}$ is, the more small-amplitude signals are clipped; however, the less negative it is, the
larger $g_{s,2}$ must be. Therefore, $g_{s,2}$ is the limiting factor to maintain biological realism. We have chosen $g_{s,2} = 20 \, \mu S$ and $R = 20 \, mV$, making $\Delta E_{s,2} = -1$.

Now that we have designed one of the Modulatory synapses, we can calculate the response of the complete multiplication network seen in Fig. 4.2D, which includes two identical Modulatory Pathways in series. When $U_{\text{pre,2}}$ is inactive, then it does not inhibit $U_{\text{inter}}$, which is tonically active. In this case, $U_{\text{inter}}$’s activity completely desensitizes $U_{\text{post}}$ to inputs. When $U_{\text{pre,2}}$ is active, then it inhibits $U_{\text{inter}}$. In this case, $U_{\text{inter}}$ is hyperpolarized, and cannot desensitize $U_{\text{post}}$ to inputs. To show that this is the case, let us find the full response of the system. We first calculate $U^*_{\text{inter}}$, which has one Modulatory Pathway input and a tonic applied current $I_{\text{app}} = R$. Its response is the same as in Eq. 4.21, with the constraint from Eq. 4.37, which causes terms to cancel:

$$U^*_{\text{inter}} = \frac{g_{s,2} R \cdot U_{\text{pre,2}} \cdot \frac{R}{g_{s,2}} + R}{1 - \frac{U_{\text{pre,2}}}{\Delta E_{s,2}}} = \frac{R - U_{\text{pre,2}}}{1 - \frac{U_{\text{pre,2}}}{\Delta E_{s,2}}}.$$  \hspace{1cm} (4.38)

$U_{\text{post}}$ has two presynaptic neurons, $U_{\text{pre,1}}$ and $U_{\text{inter}}$. The synapse from $U_{\text{pre,1}}$ is a Signal Transmission synapse, and the synapse from $U_{\text{inter}}$ is a Signal Modulation synapse. Its response is found via Eq. 4.13,

$$U^*_{\text{post}} = \frac{g_{s,3} R \cdot U_{\text{inter}} \cdot \Delta E_{s,3} + g_{s,1} R \cdot U_{\text{pre,1}} \cdot \Delta E_{s,1}}{1 + \frac{g_{s,3} R}{U_{\text{inter}}} + \frac{g_{s,1} R}{U_{\text{pre,1}}}}.$$  \hspace{1cm} (4.39)

We showed in Sec. 4.4.3 that Eq. 4.18 can be used to design a synapse that transmits the presynaptic neuron’s activity to the postsynaptic neuron, while minimizing its impact on the denominator of the postsynaptic neuron’s steady-state response, $U^*_{\text{post}}$. This enables us to approximate $U_{\text{pre,1}}$’s effect on $U^*_{\text{post}}$ as an applied current $I_{\text{app}} \approx U_{\text{pre,1}}$. Making this substitution in Eq. 4.39,

$$U^*_{\text{post}} \approx \frac{g_{s,3} R \cdot U_{\text{inter}} \cdot \Delta E_{s,3} + U_{\text{pre,1}}}{1 + \frac{g_{s,3} R}{U_{\text{inter}}}}.$$  \hspace{1cm} (4.40)
Because we previously specified that the Modulatory Pathways are identical, we can apply the constraint from Eq. 4.37,

\[ U_{\text{post}}^* = \frac{U_{\text{pre},1} - U_{\text{inter}}}{1 - \frac{U_{\text{inter}}}{\Delta E_{s,3}}}. \]  

(4.41)

We can now substitute Eq. 4.38 for \( U_{\text{inter}} \),

\[ U_{\text{post}}^* = \frac{U_{\text{pre},1} - \frac{R - U_{\text{pre},2}}{1 - \frac{U_{\text{pre},2}}{\Delta E_{s,2}}}}{1 - \frac{1}{\Delta E_{s,3}} \cdot \frac{R - U_{\text{pre},2}}{1 - \frac{U_{\text{pre},2}}{\Delta E_{s,3}}}}. \]  

(4.42)

This expression can be simplified. First, as noted previously, synapses 2 and 3 are identical, so \( \Delta E_{s,2} = \Delta E_{s,3} = \Delta E_s \). Second, we can multiply the first term in both the numerator and denominator by the factor \( (1 - U_{\text{pre},2}/\Delta E_s) \), which enables us to combine terms. Performing these simplifications,

\[ U_{\text{post}}^* = \frac{U_{\text{pre},1} \cdot U_{\text{pre},2}/\Delta E_s - R + U_{\text{pre},2}}{1 - U_{\text{pre},2}/\Delta E_s - R/\Delta E_s + U_{\text{pre},2}/\Delta E_s}; \]  

(4.43)

\[ U_{\text{post}}^* = \frac{-U_{\text{pre},1} \cdot U_{\text{pre},2}/\Delta E_s + U_{\text{pre},1} + U_{\text{pre},2} - R}{1 - R/\Delta E_s}. \]  

(4.44)

Equation 4.44 contains a lot of information about how the multiplication network functions. First, \( U_{\text{post}}^* \)'s response indeed contains a term that multiplies \( U_{\text{pre},1} \) and \( U_{\text{pre},2} \). When \( \Delta E_s = -1 \), then \( U_{\text{post}}^* \) scales with \( U_{\text{pre},1} \cdot U_{\text{pre},2} \) in a 1:1 fashion. Second, the numerator will be \( \leq 0 \) if either \( U_{\text{pre},1} = 0 \) or \( U_{\text{pre},2} = 0 \), \( U_{\text{post}}^* \leq 0 \). This is because \( U_{\text{pre},1} \) and \( U_{\text{pre},2} \) must each be less than or equal to \( R \). If either input is greater than \( R \), then their synaptic inputs to \( U_{\text{post}} \) will saturate (see Eq. 4.4), preventing this condition from being violated. Third, the denominator does not depend on the input values. Technically, because of the approximation made in Eq. 4.40, the denominator does change slightly with \( U_{\text{pre},1} \). However, with our chosen values of \( R \) (20), \( \Delta E_s \) (-1), and \( g_{s,1} \) (0.115), this change is less than 1%, justifying this approximation. Fig.
CHAPTER 4. A FUNCTIONAL SUBNETWORK APPROACH TO DESIGNING SYNTHETIC NERVOUS SYSTEMS THAT CONTROL LEGGED ROBOT LOCOMOTION

4.2D demonstrates that this network multiplies the two inputs.

Table 4.1 summarizes the function, component pathways, constraint equations, and free parameters of each network from this section. This analysis enables direct construction and parameter selection for functional subnetworks that can be assembled into more complex networks capable of performing real-time robotic control (e.g. [Szczecinski et al., 2017a, Szczecinski and Quinn, 2017c]). Additionally, one of the key advantages to using dynamic neural systems for motor control is the handling of time varying signals. The next section examines how the dynamics of these neurons can be exploited to perform calculus on signals.

4.5 Methods: Dynamic Networks

The differential equation for a single neuron’s response (Eq. 4.1) can be solved analytically. Solving an equation $dx/dt = f(x)$ is simplified if the equilibrium state is $x^* = 0$, so as in Sec. 4.4, the substitution $U = V - E_r$ is made. Additionally, the membrane conductance $G_m$ and capacitance $C_m$ can be combined into a new parameter $\tau = C_m/G_m$, which is a more intuitive parameter when discussing dynamic networks. This section uses analysis from the previous section, plus additional analysis, to derive design constraints for networks that differentiate or integrate input signals over time.

4.5.1 Differentiation

One dynamic response neural systems are known to utilize is differentiation of signals. Early examination of neural networks led to the discovery of the Reichardt detector network [Reichardt, 1961], an autocorrelation network with delays that approximates the differential of an incoming signal. Other examples include human balance, which relies on feedback proportional to the position, velocity, and acceleration of the center
CHAPTER 4. A FUNCTIONAL SUBNETWORK APPROACH TO DESIGNING SYNTHETIC NERVOUS SYSTEMS THAT CONTROL LEGGED ROBOT LOCOMOTION

of mass [Peterka, 2003, Safavynia and Ting, 2012]. Also, positive velocity feedback plays an important role in insect muscle control [Cruse, 1981].

We have developed differentiation networks based on the Reichardt detector network, shown in Fig. 4.3A. We can understand its function by examining a neuron’s response to a ramp input, \( I_{app} = A \cdot t \), where \( A \) is an arbitrary slope of the ramp. The response of the network should be a step with a magnitude proportional to \( A \), as shown in Fig. 4.3B. Inserting this applied current into Eq. 4.6, a single neuron’s response is

\[
C_m \cdot \frac{dU}{dt} = -U + A \cdot t \\
C_m \cdot \frac{dU}{dt} + U = A \cdot t.
\] (4.45)

The response of the neuron, \( U(t) \), is the sum of the particular and homogeneous solutions to Eq. 4.46, \( U_p(t) \) and \( U_h(t) \), respectively. Simulating the dynamics of Eq. 4.46 suggests that the particular solution is a ramp of slope \( A \), which lags behind the input with a time constant \( C_m \). To confirm this, we can substitute a candidate solution and its derivative into Eq. 4.46, and check for equality. The result is the particular (i.e. steady-state) response,

\[
U_p(t) = A \cdot (t - C_m)
\] (4.47)

This means that if the same \( I_{app} \) were injected into neurons with different \( \tau \) values, and then their outputs were subtracted from one another with a network from Sec. 4.4.4, the network would perform a finite-difference approximation of the derivative of \( I_{app} \), once the transient response decays (illustrated in Figure 4.3A-B). Calculating the homogeneous solution, \( U_h(t) \), informs us how quickly the transient response decays.

The homogeneous solution to first-order linear equation like Eq. 4.46 is well-known,

\[
U_h(t) = b \cdot \exp(-t/C_m).
\] The constant \( b \) is found by plugging the initial condition
Figure 4.3: (A) A network can exploit neural dynamics to compute the differential of an incoming signal. (B) When given an applied current in the form of ramps, the network returns steps whose heights are proportional to the slopes of the ramps. (C) The amplification of the differential, $k_d$, and the time constant of the network, $\tau_d$, depend on the time constant of the neurons, $\tau_1$ and $\tau_2$. (D) Frequency domain analysis enables the identification of the cutoff frequency $\omega_c$, enabling the network to naturally filter out high-frequency noise.
into the full response, $U(t) = U_p(t) + U_h(t)$.

$$b = A \cdot C_m$$ \hfill (4.48)

To tune this network, the response of $U_{post}$ is written as the difference between neuron $U_{pre,1}$ with $C_{m,1}$ and neuron $U_{pre,2}$ with $C_{m,2} > C_{m,1}$.

$$U_{post}(t) = U_{pre,1}(t) - U_{pre,2}(t)$$

$$= A \cdot t - A \cdot C_{m,1} \cdot (1 - \exp(-t/C_{m,1})) - \left( A \cdot t - A \cdot C_{m,2} \cdot (1 - \exp(-t/C_{m,2})) \right).$$ \hfill (4.49)

Canceling the terms that are linear in $t$ and expanding,

$$U_{post}(t) = A \cdot (C_{m,2} - C_{m,1}) + A \cdot (C_{m,1} \cdot \exp(-t/C_{m,1}) - C_{m,2} \cdot \exp(-t/C_{m,2})).$$ \hfill (4.50)

Properly tuning a differentiator network requires tuning $C_{m,1}$ and $C_{m,2}$ to obtain the intended gain of the network, $k_d$, and an appropriately high cutoff frequency, $\omega_c$. Equation 4.50 reveals how these may be tuned. First, the steady-state response of this network to a ramp input defines $k_d = (C_{m,2} - C_{m,1})$. Second, the cutoff frequency $\omega_c = 1/\tau_d$ quantifies the frequency of incoming signals (i.e. $I_{app} = A \cdot \sin(\omega \cdot t)$) above which the network’s response has less than half the energy of a lower-frequency signal. This is especially useful because although differential calculations amplify high-frequency noise, this network filters out noise with a frequency $\omega > \omega_c$. Because $C_{m,2} > C_{m,1}$, the time constant $\tau_d = C_{m,2}$.

Figure 4.3C shows contours of $k_d$ and $\tau_d$ as $C_{m,1}$ and $C_{m,2}$ change. The plots show that increasing $C_{m,2}$ relative to $C_{m,1}$ increases $k_d$, which may be valuable for amplifying signals. However, this also increases $\tau_d$, making $\omega_c$ impractically low, which will cause the network’s output to lag behind the input substantially. The contour
for $k_d = 1$ is drawn on the contour of $\tau_d$, showing that the smallest $\tau_d$ achievable for this gain value is 1000 ms, which would filter out all incoming signals for which $\omega > \omega_c = 1/(1s) = 1 \text{ rad/s (0.159 Hz)}$.

We can gain further insight into tuning $\tau_d$ using our FeedbackDesign tool [Szczecinski et al., 2017b]. Figure 4.3D shows Bode plots for this network’s response, given two different values for $C_{m,2}$. When $C_{m,2} = 1000 \text{ nF}$, like in Fig. 4.3B, the network functions properly for inputs with $\omega < 1 \text{ rad/s}$, as predicted in the previous paragraph. Lowering $C_{m,2}$ to 50 nF increases $\omega_c$ to 20 rad/s (3.18 Hz). Lowering $C_{m,2}$ also lowers the magnitude response as a function of $\omega$, that is, it decreases $k_d$. To regain this lost gain, we may increase $k_{syn}$ in the subtraction network. Figure 4.4 shows simulation data that explores this tradeoff. Table 4.2 lists how to use $\tau_d$ and $k_d$ to tune the entire differentiation network.

### 4.5.2 Integration

Our neuron model is a leaky integrator, which means that the membrane voltage will integrate an applied current, but “leak” current to return to its resting potential. As a result, data cannot be stored in individual neurons, because neurons only have one stable equilibrium point. A network that is constructed to have a marginally stable equilibrium subspace will not leak. A network will have this property if the determinant of the Jacobian matrix is 0, or in other words, if it is not full rank [Khalil, 2002]. Instead of leaking, it will maintain its activation when no external currents are applied; when currents are applied, the state of the system will change continuously. This is analogous to the position of a box on a table with friction; it will remain wherever it is placed indefinitely, unless an external force is applied. In this section, we show how to construct a network that is marginally stable by ensuring that its Jacobian matrix has insufficient rank when the network is at equilibrium, and demonstrate that such a network can be used to integrate signals over time.
Figure 4.4: Simulation data from eight trials with the differentiator network are shown. Different values of $\tau_1$ and $\tau_2$ were used in each. $U^*_\text{post}$ is plotted in blue, $U_2 - U_1$ is plotted in dotted red, and the actual rate of change of the input, $d/dt(I_{\text{app}})$, is plotted in gold.
CHAPTER 4. A FUNCTIONAL SUBNETWORK APPROACH TO DESIGNING SYNTHETIC NERVOUS SYSTEMS THAT CONTROL LEGGED ROBOT LOCOMOTION

Marginally stable networks are hypothesized to play an important role in navigation [Haferlach et al., 2007] and the regulation of muscle forces in posture [Lévy and Cruse, 2008]. Some memory models use carefully tuned self-excitation to cancel the leak current with excitatory synaptic current [Seung et al., 2000b]. In a similar vein, our network uses self-disinhibition (Fig. 4.5A) to produce a line-attractor network in which a continuum of marginally stable equilibrium states exist. In previous work, we analytically proved that this network always has one eigenvalue equal to zero [Szczecinski et al., 2017a], meaning that it will integrate inputs without leaking [Khalil, 2002]. In this section, we summarize this analysis and provide additional calculations to tune the network’s rate of integration, $k_i$, where $\dot{U}_1 = k_i \cdot I_{\text{app}}$. This requires performing eigenvalue decomposition, finding how $I_{\text{app}}$ affects the steady-state velocity along the equilibrium manifold, and then transforming that result back into $\dot{U}_1$.

Each neuron has leak current, synaptic current, and an applied current. Let all parameter values be symmetrical between the two neurons. We make the same substitutions as before; $U = V - E_r$, $E_r = E_{lo}$, $\Delta E = E_s - E_r$, and $R = E_{hi} - E_{lo}$.

If $I_{\text{app}} = R$,

$$C_m \cdot \frac{dU_1}{dt} = R - U_1 + g_s \cdot \frac{U_2}{R} \cdot (\Delta E - U_1)$$

(4.51)

$$C_m \cdot \frac{dU_2}{dt} = R - U_2 + g_s \cdot \frac{U_1}{R} \cdot (\Delta E - U_2).$$

(4.52)

Moving dynamical terms to the left hand side, and applied current to the right hand side,

$$\frac{dU_1}{dt} + \frac{1}{C_m} \left( U_1 - g_s \cdot \frac{U_2}{R} \cdot (\Delta E - U_1) \right) = \frac{R}{C_m}$$

(4.53)

$$\frac{dU_2}{dt} + \frac{1}{C_m} \left( U_2 - g_s \cdot \frac{U_1}{R} \cdot (\Delta E - U_2) \right) = \frac{R}{C_m}$$

(4.54)
A disinhibitory network can exploit neural dynamics to compute the integral of an incoming signal. When given an applied current in the form of a step, the network response is a ramp whose slope is proportional to the amplitude of the step. A plot of this data in the \((U_1, U_2)\) phase space shows that when stimulated by applied current \(u\), the system state, \(x(t) = [U_1(t), U_2(t)]^T\) (blue), moves in the \(X_1\) direction (green) while maintaining a constant distance from the equilibrium subspace (dashed violet) in the \(X_2\) direction (red). This is because the eigenvalue associated with eigenvector \(X_1\), \(\lambda_1 = 0\), and the eigenvalue associated with eigenvector...
Figure 4.5: (contd.) \( X_2, \lambda_2 < 0 \). \( X_1 \) and \( X_2 \) are drawn in multiple places because they depend on \( x(t) \), as shown in Appendix 4.8. (D) The mean rate of integration, \( k_{i,\text{mean}} \) (left), and the range of the rate of integration, \( k_{i,\text{range}} \) (right), depend on the synaptic conductance of mutual inhibition, \( g_s \), and the membrane capacitance of the neurons, \( C_m \). Note that the \( x \)-axis of these plots are \( 1/C_m \), to better space the contour lines.

Solving Eq. 4.53 when \( dU_1/dt = 0 \) reveals the equilibrium manifold

\[
U_2 = \frac{R \cdot (U_1 - R)}{g_s \cdot (\Delta E - U_1)}. \tag{4.55}
\]

Solving Eq. 4.54 when \( dU_2/dt = 0 \) reveals the equilibrium manifold

\[
U_1 = \frac{R \cdot (U_2 - R)}{g_s \cdot (\Delta E - U_2)}, \tag{4.56}
\]

which can be algebraically rearranged to be the same as Eq. 4.55. Thus, they are the same manifold.

To aid analysis, let us write Eqs. 4.53 and 4.54 together in matrix form,

\[
\begin{bmatrix}
\dot{U}_1 \\
\dot{U}_2
\end{bmatrix} + \frac{1}{C_m} \cdot \begin{bmatrix}
1 + U_2 \cdot \frac{g_s}{R} & -\frac{g_s}{R} \cdot (\Delta E - U_1) \\
-\frac{g_s}{R} \cdot (\Delta E - U_2) & 1 + U_1 \cdot \frac{g_s}{R}
\end{bmatrix} \cdot \begin{bmatrix}
U_1 \\
U_2
\end{bmatrix} = \frac{1}{C_m} \cdot \begin{bmatrix}
R \\
R
\end{bmatrix}, \tag{4.57}
\]

in which the square matrix is \( J \), the system Jacobian. Because \( J \) contains \( U_1 \) and \( U_2 \) terms, it is not constant, but still describes the stability of the system for specific values of \( U_1 \) and \( U_2 \). To construct a marginally stable equilibrium subspace for the network, we must show that \( J \) has insufficient rank (i.e. the rows are identical) no matter the values of \( U_1 \) and \( U_2 \). The rows are identical if we apply the constraint

\[
-\frac{g_s}{R} \cdot \Delta E = 1 \tag{4.58}
\]
to Eq. 4.57. Applying Eq. 4.58 and collecting terms,

$$\begin{bmatrix} \dot{U}_1 \\ \dot{U}_2 \end{bmatrix} + \frac{1}{C_m} \cdot \begin{bmatrix} 1 + \frac{g_s}{R} \cdot U_2 & 1 + \frac{g_s}{R} \cdot U_1 \\ 1 + \frac{g_s}{R} \cdot U_2 & 1 + \frac{g_s}{R} \cdot U_1 \end{bmatrix} \cdot \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} = \frac{1}{C_m} \cdot \begin{bmatrix} R \\ R \end{bmatrix}. \quad (4.59)$$

Let us define

$$a = 1 + g_s/R \cdot U_1 \quad (4.60)$$

$$b = 1 + g_s/R \cdot U_2. \quad (4.61)$$

These expressions let us write Eq. 4.59 as simply

$$\begin{bmatrix} \dot{U}_1 \\ \dot{U}_2 \end{bmatrix} + \begin{bmatrix} b/C_m & a/C_m \\ b/C_m & a/C_m \end{bmatrix} \cdot \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} = \begin{bmatrix} R/C_m \\ R/C_m \end{bmatrix}. \quad (4.62)$$

Simulation data in Fig. 4.5B shows that stimulating $U_1$ with an additional applied constant current $u$ causes $U_1$ to increase at an apparently constant rate. But how fast does $U_1$ change? Plotting the simulation data on a phase-space diagram (Fig. 4.5C) suggests that $u$ causes $U_1$ and $U_2$ to change in such a way that the state of the system $(x(t), \text{blue})$ moves along the equilibrium manifold (dashed violet), with some constant distance away from it. This suggests that the state can be generalized into two decoupled degrees of freedom in the phase-space: unresisted, marginally stable motion along the equilibrium manifold ($X_1$, green in Fig. 4.5C); and resisted, stable motion away from the equilibrium manifold ($X_2$, red). The natural coordinates, $\vec{x} = [U_1, U_2]^T$, are transformed into generalized coordinates, $\vec{q} = [q_1, q_2]^T$, by a matrix $X$ comprised of the eigenvectors of $J$. This same transformation matrix is used to transform $J$ into the generalized coordinate system, yielding $J_q$. $J_q$ is diagonal, decoupling the dynamics of the generalized coordinates and enabling us to quantify how quickly $\vec{x}$ moves along the equilibrium manifold.
Appendix 4.8 shows the calculation of $X$, with $q_1$ representing the marginally stable mode and $q_2$ representing the stable mode. Using $X$, we can transform the system into generalized coordinates. First, we write the dynamics from Eq. 4.62 in a compact format.

$$\dot{x} + J\ddot{x} = F,$$  \hspace{1cm} (4.63)

where $J$ is the square matrix in Eq. 4.62, and

$$F = \begin{bmatrix} \frac{R}{C_m} + \frac{u}{C_m} \\ \frac{R}{C_m} \end{bmatrix}. \hspace{1cm} (4.64)$$

The generalized coordinates, $\ddot{q}$, are defined as

$$\ddot{x} = X\ddot{q}. \hspace{1cm} (4.65)$$

To transform Eq. 4.63 into generalized coordinates, premultiply both sides of Eq. 4.63 by $X^{-1}$,

$$\ddot{q} + J_q\ddot{q} = \ddot{Q}, \hspace{1cm} (4.66)$$

where $J_q = X^{-1}JX$ and $\ddot{Q} = X^{-1}\ddot{F}$. The top and bottom rows of Eq. 4.66 are decoupled because $J_q$ is a diagonal matrix. Furthermore, $J_q^{ii} = \lambda_i$, meaning that the $J_q^{1,1} = 0$, meaning that the system simplifies even further.

To find the particular solution of this system, we can guess the form of $\ddot{q}$ and substitute those in to Eq. 4.66. We observe that $\ddot{q}_1(t) = B \cdot u$ in steady state, where $B$ is a constant that relates $\dot{q}_1(t)$ and $u$. $q_1(t)$ would be the integral of $\dot{q}_1(t)$, but because the top row of $J_q$ is zeros, it will not appear in the particular solution, and thus need not be explicitly included. We also observe that $\ddot{q}_2(t) = 0$ in steady state, so $q_2(t) = D$, a constant. We can calculate $\ddot{Q} = X^{-1}F$ using $X^{-1}$, which is calculated
in Appendix 4.8 (Eq. 4.100). Solving for the particular solution of this system, \( \dot{q}_p(t) \),

\[
\dot{q}_p(t) + J_q \ddot{q}_p = \begin{bmatrix} B \cdot u \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & \frac{a+b}{c_m} \end{bmatrix} \cdot \begin{bmatrix} q_{p,1} \\ D \end{bmatrix} = \begin{bmatrix} \frac{a \cdot d}{c_m \cdot (a+b)} \cdot u \\ \frac{a \cdot d}{c_m \cdot (a+b)} \cdot u \end{bmatrix},
\]

(4.67)

\[
B = \frac{a \cdot d}{c_m \cdot (a+b)},
\]

(4.68)

where \( d \) is defined in Eq. 4.90. \( B \) describes how quickly \( q_{p,1} \) varies with \( u \), but we want to know how quickly \( U_1 \) varies with \( u \). Therefore, we use Eq. 4.65 to transform \( \ddot{q}_p = [B \cdot u, 0]^T \) into natural coordinates to find \( \ddot{x} \),

\[
\begin{align*}
\ddot{x}_p &= X \ddot{q}_p \\
\dot{U}_{1,p}(t) &= \begin{bmatrix} 1/d & 1/\sqrt{2} \\ -b/(ad) & 1/\sqrt{2} \end{bmatrix} \cdot \begin{bmatrix} \frac{a \cdot d}{c_m \cdot (a+b)} \cdot u \\ 0 \end{bmatrix} \\
\dot{U}_{2,p}(t) &= \begin{bmatrix} 1/d & 1/\sqrt{2} \\ -b/(ad) & 1/\sqrt{2} \end{bmatrix} \cdot \begin{bmatrix} \frac{a \cdot d}{c_m \cdot (a+b)} \cdot u \\ 0 \end{bmatrix} \\
\dot{k}_i &= \frac{a}{C_m \cdot (a+b)}.
\end{align*}
\]

(4.69, 4.70, 4.71, 4.72)

Recall that \( a \) and \( b \) are functions of \( U_1 \) and \( U_2 \), respectively. This means that \( k_i \) is not a constant. To place bounds on \( k_i \), let us substitute Eqs. 4.60 and 4.61 into Eq. 4.72,

\[
k_i = \frac{1 + g_s/R \cdot U_1}{C_m \cdot (2 + g_s/R \cdot (U_1 + U_2))}.
\]

(4.73)

We can now plug in different values of \( U_1 \) and \( U_2 \) to see how \( k_i \) varies. Using Eqs. 4.55 and 4.56, we find that the most extreme cases are when \([U_1, U_2] = [0, R]\) and \([U_1, U_2] = [R, 0]\). We can plug these cases into Eq. 4.73 to find the minimum and maximum values for \( k_i \),

\[
k_{i, \text{min}} = \frac{1 + g_s/R \cdot 0}{C_m \cdot (2 + g_s/R \cdot (0 + R))} = \frac{1}{C_m \cdot (2 + g_s)}.
\]

(4.74)
and

\[
k_{i,\text{max}} = \frac{1 + g_s/R \cdot R}{C_m \cdot (2 + g_s/R \cdot (R + 0))} = \frac{1 + g_s}{C_m \cdot (2 + g_s)}.
\]  

(4.75)

The difference between \(k_{i,\text{min}}\) and \(k_{i,\text{max}}\)

\[
k_{i,\text{range}} = \frac{1 + g_s}{C_m \cdot (2 + g_s)} - \frac{1}{C_m \cdot (2 + g_s)} = \frac{g_s}{C_m \cdot (2 + g_s)}.
\]  

(4.76)

To find the mean rate of integration, we can calculate \(k_{i,\text{mean}} = (k_{i,\text{min}} + k_{i,\text{max}})/2\),

\[
k_{i,\text{mean}} = \frac{1}{2 \cdot C_m}.
\]  

(4.77)

This is the same value of \(k_i\) obtained from computing \(k_i\) when \(U_1 = U_2\). This simple expression is a useful relationship for tuning the integrator network. One may select \(C_m\) to obtain the intended mean integration rate, and then minimize the variation of the integration rate by minimizing \(g_s\), as long as Eq. 4.58 is satisfied.

Figure 4.5D graphically demonstrates how \(k_{i,\text{mean}}\) and \(k_{i,\text{range}}\) determine \(C_m\) and \(g_s\). Just as in Eq. 4.77, \(k_{i,\text{mean}}\) is a function only of \(C_m\). Therefore, the contour only shows vertical lines. The value of \(k_{i,\text{range}}\) is minimized by decreasing either \(g_s\) or \(C_m^{-1}\) (i.e. increasing \(C_m\)). Figure 4.6 shows simulation data of the integrator’s response to a step input with eight different parameter value combinations. In every case, the change in \(U_1\) is bounded by the values of \(k_{i,\text{mean}}\) and \(k_{i,\text{range}}\). As shown in Fig. 4.5D, increasing \(C_m\) decreases the integration rate, and increasing \(g_s\) increases the variation in the integration rate. Table 4.2 summarizes the design approach for this integrator network. The mean and range of the integration rate are free parameters that are determined by the intended network performance. Using these values and the constraint in Eq. 4.58, the neurons’ \(C_m\) value and the synapses’ \(g_s\) and \(\Delta E_s\) values can be fully specified.
CHAPTER 4. A FUNCTIONAL SUBNETWORK APPROACH TO DESIGNING SYNTHETIC NERVOUS SYSTEMS THAT CONTROL LEGGED ROBOT LOCOMOTION

Figure 4.6: Simulation data from eight trials are shown. Different values of $C_m$ and $g_s$ were used in each. Neural dynamics are plotted as blue lines. The expected final values of the simulations are plotted in dotted red lines. Regions bounded by $k_{i,\text{mean}} \pm k_{i,\text{range}}$ are shaded in violet. In every case, the actual outcome is correctly bounded. As demonstrated mathematically in the text, $k_{i,\text{mean}}$ only depends on $C_m$. In addition, $k_{i,\text{range}}$ depends on $g_s$, leading to more variation in $k_i$, as indicated by larger shaded areas.
4.6 Application to a Robot Controller

We have used the methods in this paper to tune several different networks that control robotic stepping [Szczecinski et al., 2017a, Szczecinski and Quinn, 2017c] and visual tracking [Szczecinski et al., 2017a]. Figure 4.7 shows a simplified joint control network in which different functional pathways are color-coded. This illustrates how these functional subnetworks enable the direct assembly of control networks based on neurobiology. The neurobiological inspiration for these networks and the results of robotic experiments are presented in [Szczecinski et al., 2017a, Szczecinski and Quinn, 2017c], and so are omitted here. The joint network in Fig. 4.7 uses three simple descending commands (body heading, stride length, and reference leg load) to control the walking motion of one joint of a leg. The descending commands modulate the output of a central pattern generator (CPG) to control the speed of the motion, and sensory feedback is used to adjust both the timing and amplitude of motor output. Addition pathways are drawn in red. These include the mapping between body heading and stride length (i.e. descending commands, drawn in gray) to the PEP and AEP [Szczecinski and Quinn, 2017c]. The PEP can also be modulated by force feedback, which compares the load on the leg to a reference value (Szczecinski and Quinn, in review). This requires a subtraction network, drawn in orange, to compute if there is too much or too little load on the leg. The difference is used to adjust the PEP Memory network, which is an integration network, drawn in blue. This network adjusts the PEP over time, and remembers the motor command that produces the intended force.

The output of the CPG, drawn in purple, excites the motor neurons. Tuning CPG dynamics is discussed in our previous work [Szczecinski et al., 2017b]. The PEP and AEP neurons adjust the motor output via multiplication pathways, drawn in green, which scale CPG output to the motor neurons based on the intended range of motion. Motor neuron activity controls the motor velocity, and the $\theta$ neuron receives position
Figure 4.7: A simplified joint-control network from our previous work [Szczecinski et al., 2017a, Szczecinski and Quinn, 2017c], with pathways color-coded based on the functional subnetwork.
feedback from the motor via the mappings in Fig. 4.1B. The motor velocity, computed by the cyan differentiation pathway, reinforces ongoing CPG behavior through the $\dot{\theta}$ neuron [Szczecinski et al., 2017b]. The $\dot{\theta}$ neuron also receives some input from the Load neuron, ensuring that stance phase is stable (Szczecinski and Quinn, accepted). By using the functional subnetworks and the design constraints presented in this paper, we can rapidly and directly assemble models of neural systems that perform as intended without hand-tuning or optimization methods.

How are the “Free Parameters” in Tables 4.1 and 4.2 chosen? The free parameters fall into two classes: reversal potentials (i.e. $\Delta E_s$) and dynamical constants (e.g. $k$, $\tau$, etc.). The reversal potentials are informed by biology. In this paper, we kept $-40 < \Delta E_s < 194$ mV ($-100 < E_s < 134$ mV). The modeler could use reversal potentials from specific synapses if that data were available. The dynamical constants are informed by the function of the robot. For example, the $k_{syn}$ of the subtraction network in Fig. 4.1B controls the stiffness of the controller, and may destabilize the system if not tuned to match the mechanical properties of the robot [Szczecinski et al., 2017b].

As another example, $\tau_d$ and $k_d$ of the differentiator network in Fig. 4.7 determines the robustness of CPG rhythms, and how well it entrains to sensory feedback [Szczecinski et al., 2017b]. A slow, adaptively-walking robot may want a high $k_d$ to regularize CPG oscillations, whereas a fast running robot may want a low $k_d$ to be less sensitive to sensory feedback. Picking specific values for these free parameters ultimately depends on the intended behavior of the robot. The constraints in this paper enable the designer to think in terms of more traditional robotics quantities, and use these to set neural and synaptic parameter values, which may be less intuitive.
4.7 Discussion

In this paper we presented analytical methods for setting parameters in dynamical neural networks that can add, subtract, multiply, divide, differentiate, and integrate incoming signals. Such operations are at the core of control, and these techniques enable control networks to be assembled rapidly and tuned directly. This work primarily identifies constraint equations, not unique values, that govern how parameters should be tuned. Thus, many different networks may perform the same function with different parameter values, as observed in real neural circuits [Prinz et al., 2004]. Since these results are analytical, not based on machine learning or optimization, there is no concern about these networks over- or under-fitting training data, and their behavior is provable. These techniques build on our previous analysis of synthetic nervous systems [Szczecinski et al., 2017b], and have been validated through several studies with our robot, MantisBot [Szczecinski et al., 2017a, Szczecinski and Quinn, 2017c].

All of the results from this paper make it easier to tune neuromechanical models of animals, as well. Many such models have been created to study the principles underlying insect [Daun-Gruhn and Tóth, 2010, Szczecinski et al., 2014] and mammalian [Hunt et al., 2015a, Markin et al., 2016] locomotion alike. Oftentimes, parameters of these models are tuned by hand to obtain the intended motion, which is a painstaking, slow, and imprecise process. The analysis in this paper can make neuromechanical models come together more quickly, and have more predictable behavior, leading to more thorough scientific investigations. More precise tuning methods enable more thorough validation or invalidation of hypotheses. Faster tuning methods enable more rapid validation or invalidation of hypotheses. For example, these methods could be used to improve the coordination our previous cockroach model [Szczecinski et al., 2014]. In the model, curve walking of varying radii was achieved by modulating muscle activations with broad descending commands. However, the coordination,
reliability, and repeatability of such motion could be improved with the methods of this paper, enabling us to improve or reject the model.

4.7.1 Simplifications

Some of the calculations in this paper are based on approximations, which lead to inaccuracies in the calculations of the subnetworks. One example is that the subtraction network does not produce linear output. This nonlinearity occurs because the reversal potentials of synapses are rarely much lower than the resting potentials of neurons, requiring large values of $g_{s,2}$ to build a subtractor where $k_{syn} = 1$. A large $g_{s,2}$ value increases $U_{pre,2}$’s effect on the denominator of $U_{post}$’s response, causing the synaptic input to reduce $U_{post}$’s sensitivity to inputs. This is particularly noticeable in the differentiator’s response (Fig. 4.4), especially as $k_{syn}$ increases.

Another example of a simplification we made is that our calculation of $k_i$ only used the particular solution of the system. This means that a transient response also exists, which we did not compute. In addition, $k_i$ is a function of $U_1$ and $U_2$. This means that $k_i$ is not a constant for this network. However, the impact of $U_1$ and $U_2$ on $k_i$ can be minimized by minimizing $g_s$ and maximizing $R$, as we showed in Sec. 4.5.2.

However, the developed networks are not intended to act as perfect analogs to their mathematical counterparts. These networks are intended to act as representations of real neural circuits, which likely do not act as perfect adders, multipliers, differentiators, etc. Dynamic and transient effects are a real part of biological control systems, and effective neural controllers have developed around these idiosyncrasies and has likely evolved to even exploit many of these aspects. In spite of these issues, the methods in this paper are valuable. Our recent robotics work [Szczecinski et al., 2017a, Szczecinski and Quinn, 2017c], as well as related work in progress, is proof of the effectiveness of this approach.
4.7.2 Why put neurons in the way?

The methods in this paper enable the direct construction of networks that perform arithmetic and dynamic calculations. Why bother building neural networks just to recreate mathematical operators? We believe there are several reasons to take this approach. From a neurobiology perspective, the constraints that we have identified may help explain why certain structures are common in the nervous system (David Friel, personal correspondence). For instance, mutually-inhibitory parallel pathways are common in the thoracic control of insect locomotion [Büschges and Wolf, 1995], which may function as subtraction networks in negative feedback loops. As another example, networks in the retina of the rabbit are selectively sensitive to motion in one direction or the other [Barlow and Levick, 1965]. Such a network could be constructed by using adjacent cells in the retina as inputs to differentiator networks. This would be consistent with both the function of direction-sensitivity, as well as the laterally inhibitive structure. The design methods in this paper may aid in understanding the function of neural networks found in animals, beyond tuning models of them.

Additionally, the constraints that we identified may be used to constrain parameter values in large brain models. Rather than using global search techniques to understand the dynamics of a large pool of neurons, we believe it may be faster to begin with a number of functional subnetworks, and then use local search techniques to tune the connections between them. In this way, the designer is certain that parts of the network perform specific, useful computations, rather than naively optimizing a large network [Haferlach et al., 2007, Agmon and Beer, 2013, Izquierdo and Beer, 2013]. The end result is something like a genetic program, but in a neuroscience context.
4.8 Appendix: Derivation of Integrator Eigenvalues and Eigenvectors

We find the eigenvalues $\lambda_1$ and $\lambda_2$ and the associated eigenvectors $X_1$ and $X_2$ of the Jacobian matrix by the eigenvalue problem,

$$\det(J - \lambda_i \cdot I) = 0$$

$$J \cdot X_i = \lambda_i \cdot X_i,$$

where $i$ is the index of the eigenvalue (1 or 2), $I \in \mathbb{R}^{2 \times 2}$ is an identity matrix, and $J$ is the square matrix from Eq. 4.62. Solving for $\lambda$,

$$\det \begin{bmatrix} b/C_m - \lambda & a/C_m \\ b/C_m & a/C_m - \lambda \end{bmatrix} = 0$$

$$(b/C_m - \lambda) \cdot (a/C_m - \lambda) - (a/C_m \cdot b/C_m) = 0$$

$$\frac{ab}{C_m^2} + \frac{a + b}{C_m} \cdot \lambda + \lambda^2 - \frac{ab}{C_m^2} = 0$$

$$\lambda \cdot (\lambda - \frac{a + b}{C_m}) = 0$$

$$\lambda_1 = 0, \lambda_2 = \frac{a + b}{C_m}.$$

We use the eigenvalues to find their associated eigenvectors. First, $X_1$:

$$J \cdot X_1 = \lambda_1 \cdot X_1$$

$$\begin{bmatrix} b/C_m & a/C_m \\ b/C_m & a/C_m \end{bmatrix} \cdot \begin{bmatrix} 1 \\ x_1 \end{bmatrix} = 0 \cdot \begin{bmatrix} 1 \\ x_1 \end{bmatrix}.$$
CHAPTER 4. A FUNCTIONAL SUBNETWORK APPROACH TO DESIGNING SYNTHETIC NERVOUS SYSTEMS THAT CONTROL LEGGED ROBOT LOCOMOTION

The equations in Eq. 4.86 are redundant, so we may choose either row to solve to find \( x_1 \). Solving the first row,

\[
\frac{b}{C_m} + \frac{a}{C_m} \cdot x_1 = 0
\]

Solving the first row,

\[
b/C_m + a/C_m \cdot x_1 = 0
\]  \hspace{1cm} (4.87)

\[
x_1 = -\frac{b}{a}
\]  \hspace{1cm} (4.88)

\[
X_1 = \begin{bmatrix}
1 \\
-\frac{b}{a}
\end{bmatrix}
\]  \hspace{1cm} (4.89)

Normalizing \( X_1 \) to 1,

\[
X_1 = \begin{bmatrix}
\frac{1}{d} \\
-\frac{b}{(ad)}
\end{bmatrix} \hspace{1cm} \text{where} \hspace{1cm} d = \sqrt{1^2 + (-\frac{b}{a})^2}.
\]  \hspace{1cm} (4.90)

Next, we calculate \( X_2 \):

\[
J \cdot X_2 = \lambda_2 \cdot X_2
\]  \hspace{1cm} (4.91)

\[
\begin{bmatrix}
\frac{b}{C_m} & \frac{a}{C_m} \\
\frac{b}{C_m} & \frac{a}{C_m}
\end{bmatrix} \begin{bmatrix}
1 \\
x_2
\end{bmatrix} = \frac{a + b}{C_m} \begin{bmatrix}
1 \\
x_2
\end{bmatrix}.
\]  \hspace{1cm} (4.92)

Solving the first row for \( x_2 \),

\[
b/C_m + a/C_m \cdot x_2 = (a + b)/C_m
\]  \hspace{1cm} (4.93)

\[
b + a \cdot x_2 = a + b
\]  \hspace{1cm} (4.94)

\[
x_2 = 1.
\]  \hspace{1cm} (4.95)

Normalizing \( X_2 \) to 1,

\[
X_2 = \begin{bmatrix}
1/\sqrt{2} \\
1/\sqrt{2}
\end{bmatrix}.
\]  \hspace{1cm} (4.96)
CHAPTER 4. A FUNCTIONAL SUBNETWORK APPROACH TO DESIGNING SYNTHETIC NERVOUS SYSTEMS THAT CONTROL LEGGED ROBOT LOCOMOTION

We now know the transformation matrix between the natural coordinates, \( \vec{x} = [U_1, U_2]^T \), and the generalized coordinates, \( \vec{q} = [q_1, q_2]^T \):

\[
\vec{x} = X \cdot \vec{q}, \quad X = \begin{bmatrix} X_{1,1} & X_{1,2} \\ X_{2,1} & X_{2,2} \end{bmatrix} = \begin{bmatrix} 1/d & 1/\sqrt{2} \\ -b/(ad) & 1/\sqrt{2} \end{bmatrix}.
\] (4.97)

We will also make use of \( X^{-1} \) when transforming between natural and generalized coordinates. The inverse of a 2 x 2 matrix can be computed directly,

\[
X^{-1} = \begin{bmatrix} X_{1,1} & X_{1,2} \\ X_{2,1} & X_{2,2} \end{bmatrix}^{-1} = \frac{1}{\det X} \begin{bmatrix} X_{2,2} & -X_{1,2} \\ -X_{2,1} & X_{1,1} \end{bmatrix}.
\] (4.98)

Substituting values from Eq. 4.97,

\[
\det X = \frac{1}{d} \cdot \frac{1}{\sqrt{2}} - \frac{-b}{a \cdot d} \cdot \frac{1}{\sqrt{2}} = \frac{a}{a \cdot d \cdot \sqrt{2}} + \frac{b}{a \cdot d \cdot \sqrt{2}} = \frac{a + b}{a \cdot d \cdot \sqrt{2}}
\] (4.99)

\[
X^{-1} = \frac{a \cdot d \cdot \sqrt{2}}{a + b} \cdot \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} \\ b/(ad) & 1/d \end{bmatrix} = \begin{bmatrix} \frac{a \cdot d}{a + b} & -\frac{a \cdot d}{a + b} \\ \frac{b \cdot \sqrt{2}}{a + b} & \frac{a \cdot \sqrt{2}}{a + b} \end{bmatrix}
\] (4.100)
Table 4.1: Summary of design constraints and free parameters when tuning arithmetic networks. This table assumed that the designer has already selected a value of $R$ for the subnetwork. In this table, “minimize” refers to making a value as negative as possible, and “maximize” refers to making a value as positive as possible.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Component Pathways</th>
<th>Constraint Equations</th>
<th>Free Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>Syn. 1, Transmission</td>
<td>$g_{s,1} = \frac{k_{syn,1} \cdot R}{\Delta E_{s,1} - k_{syn,1} \cdot R}$</td>
<td>$k_{syn,1}$</td>
</tr>
<tr>
<td></td>
<td>Syn. 2, Transmission</td>
<td>$g_{s,2} = \frac{k_{syn,2} \cdot R}{\Delta E_{s,2} - k_{syn,2} \cdot R}$</td>
<td>$k_{syn,2}$</td>
</tr>
<tr>
<td>Subtraction</td>
<td>Syn. 1, Transmission</td>
<td>$g_{s,1} = \frac{k_{syn} \cdot R}{\Delta E_{s,1} - k_{syn} \cdot R}$</td>
<td>$k_{syn}$</td>
</tr>
<tr>
<td></td>
<td>Syn. 2, Transmission</td>
<td>$g_{s,2} = \frac{\Delta E_{s,1} - k_{syn} \cdot R}{\Delta E_{s,2} - k_{syn} \cdot R} \cdot \frac{\Delta E_{s,1} - k_{syn} \cdot R}{\Delta E_{s,2} - k_{syn} \cdot R}$</td>
<td>$\Delta E_{s,2}, \text{minimize}$</td>
</tr>
<tr>
<td>Division</td>
<td>Syn. 1, Transmission</td>
<td>$g_{s,1} = \frac{k_{syn} \cdot R}{\Delta E_{s,1} - k_{syn} \cdot R}$</td>
<td>$c_{syn}$</td>
</tr>
<tr>
<td></td>
<td>Syn. 2, Modulation</td>
<td>$g_{s,2} = \frac{c_{syn} \cdot R - R}{\Delta E_{s,2} - c_{syn} \cdot R}$</td>
<td>$\Delta E_{s,2}, = 0$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$0 &lt; c_{syn} &lt; 1$</td>
<td></td>
</tr>
<tr>
<td>Multiplication</td>
<td>Syn. 1, Transmission</td>
<td>$g_{s,1} = \frac{k_{syn} \cdot R}{\Delta E_{s,1} - k_{syn} \cdot R}$</td>
<td>$k_{syn} = 1$</td>
</tr>
<tr>
<td></td>
<td>Syn. 2, Modulation</td>
<td>$g_{s,2} = \frac{-R}{\Delta E_{s,2}}$</td>
<td>$\Delta E_{s,2} &lt; 0$</td>
</tr>
<tr>
<td></td>
<td>Syn. 3, Modulation</td>
<td>$g_{s,3} = g_{s,2}$</td>
<td>$\Delta E_{s,3} = \Delta E_{s,2}$</td>
</tr>
</tbody>
</table>
Table 4.2: Summary of design constraints and free parameters when tuning dynamic networks. This table assumed that the designer has already selected a value of $R$ for the subnetwork. In this table, “minimize” refers to making a value as negative as possible, and “maximize” refers to making a value as positive as possible.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Component</th>
<th>Constraints and Useful Relations</th>
<th>Free Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Differentiation</td>
<td>Neuron 1</td>
<td>$\tau_1 &lt; \tau_2$</td>
<td>$\tau_d$</td>
</tr>
<tr>
<td></td>
<td>Neuron 2</td>
<td>$\tau_2 = \tau_d$</td>
<td>$k_d$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\tau_1 = \tau_2 - k_d$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Syn. 1, Transmission</td>
<td>$k_{syn} = 1/k_d$</td>
<td>$\Delta E_{s,1}$, maximize</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$g_{s,1} = \frac{k_{syn} \cdot R}{\Delta E_{s,1} - k_{syn} \cdot R}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\Delta E_{s,1} - k_{syn} \cdot R &gt; 0$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Syn. 2, Transmission</td>
<td>$g_{s,2} = \frac{\Delta E_{s,1}}{\Delta E_{s,2}} \cdot \frac{-k_{syn} \cdot R}{\Delta E_{s,1} - k_{syn} \cdot R}$</td>
<td>$\Delta E_{s,2}$, minimize</td>
</tr>
<tr>
<td>Integration</td>
<td>Neuron 1</td>
<td>$I_{app,1} = R$</td>
<td>$k_{i,mean}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$C_{m,1} = \frac{1}{2 \cdot k_{i,mean}}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Neuron 2</td>
<td>$I_{app,2} = R$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$C_{m,1} = C_{m,2}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Syn. 1, Transmission</td>
<td>$\Delta E_{s,1} = \frac{-R}{g_{s,1}}$</td>
<td>$k_{i,range}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$g_{s,1} = \frac{2 \cdot C_{m,1}}{1/k_{i,range} - C_{m,1}}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Syn. 2, Transmission</td>
<td>$g_{s,2} = g_{s,1}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\Delta E_{s,2} = \Delta E_{s,1}$</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 5

MantisBot: A Platform for Investigating Mantis Behavior via Real-Time Neural Control

This chapter was originally published after peer-review as


Minor edits have been made, specifically, removing “preliminary results” and adding references to other parts of this thesis.

5.1 Abstract

We present Mantisbot, a 28 degree of freedom robot controlled in real-time by a neural simulation. MantisBot was designed as a 13.3:1 model of a male Tenodera sinensis
with the animal’s predominant degrees of freedom. The purpose of this robot is to investigate two main topics: 1. the control of targeted motion, such as prey-directed pivots and striking, and 2. the role of descending commands in transitioning between behaviors, such as standing, prey stalking, and walking. In order to more directly use data from the animal, the robot mimics its kinematics and range of motion as closely as possible, uses strain gages on its legs to measure femoral strain like insects, and is controlled by a realistic neural simulation of networks in the thoracic ganglia. This paper summarizes the mechanical, electrical, and software design of the robot, and how its neural control system generates reflexes observed in insects. It also presents preliminary results; the robot is capable of supporting its weight on four or six legs, and using sensory information for adaptive and corrective reflexes.

5.2 Introduction

Praying mantises make visually-guided posture adjustments to align themselves with prey [Yamawaki et al., 2011], [Cleal and Prete, 1996]. These adjustments require the animal to process visual information in the brain [Yamawaki and Toh, 2003] to produce descending commands to low-level systems that control its body and legs, which execute these translations and rotations. We are especially interested in the central complex (CX), a midline neuropil in the arthropod brain, and its role in controlling behavior. The CX receives multimodal sensory information and communicates with premotor centers, suggesting that it plays a key role in controlling orientation and locomotion. Work with cockroaches suggests that activity in the CX precedes directed motion, specifically linear and angular velocity in the frontal and sagittal planes [Guo and Ritzmann, 2013]. Mantises’ directed motion facilitates studying the CX; therefore we have begun to investigate these questions in the praying mantis, one of the cockroach’s closest living relatives [Svenson and Whiting, 2004],
and an animal that exhibits deliberate, targeted motion as a predator.

In our previous work, we have constructed detailed neuromechanical models of insects to investigate questions about motor control [Szczecinski et al., 2014, Szczecinski et al., 2015c]. Building realistic models of animal locomotion requires the scientist to confront the details of network parameters and dynamics, an approach that has led to improved understanding of how animals generate rhythms for walking [Spardy et al., 2011, Daun-Gruhn, 2010] and modify them to change direction [Knops et al., 2012]. While neuromechanical software simulations are useful investigative tools, robots offer a more physically realistic way to explore animal control strategies [Ritzmann et al., 2000]. Phenomena like ground contact and body strain are important details of controlling motion, yet they are difficult to model. In addition, the real world is noisy, and while noise can be modeled in simulation, a real-world environment is an
excellent test for an experimental controller. Therefore, MantisBot was developed to be controlled by the AnimatLab Robotics Toolkit. This makes the transition from simulation to robot straightforward.

Many other robots have served as models by which to explore animal control systems, and only a few are mentioned here for brevity. ROBOT II, modeled after the stick insect and controlled by a finite state machine, was one of the first to implement leg reflexes based on insect behavior [Espenschied et al., 1996]. One of the most complete robotic models of insect behavior is WALKNET, which is a hierarchical artificial neural network that replicates behavioral data from stick insects [Schilling et al., 2013a]. WALKNET is used to control HECTOR, perhaps the most sophisticated biologically-inspired hexapod robot [Schneider et al., 2012]. The robot SCORPION explores rhythm generation and reflexes through a neural system [Dirk and Frank, 2007], including abstracted CPGs. Our goals for MantisBot are parallel to those for these other robots, except that we seek to explore how neural dynamics themselves affect the control of posture, reflexes, and rhythm. As such, we model the animal’s control system as one hierarchical network of nonspiking neurons and synapses.

MantisBot is a robotic test platform for neural controllers that model insects’ nervous systems, like those in our previous work [Szczecinski et al., 2014, Szczecinski et al., 2015c]. It mimics the anatomical proportions and kinematics of a male praying mantis Tenodera sinensis, and is controlled by a real-time neural simulation implemented with the AnimatLab Robotics Toolkit. This paper describes the robot, explores design decisions that make it like the animal, and explains the control system. Results are also presented from preliminary experiments, in which MantisBot uses sensory signals to coordinate its joints and exploits CPG dynamics to produce reflexive correction steps.
CHAPTER 5. MANTISBOT: A PLATFORM FOR INVESTIGATING MANTIS BEHAVIOR VIA REAL-TIME NEURAL CONTROL

5.3 Robot Hardware

5.3.1 Mechanical and Electrical Design

MantisBot is actuated by Robotis Dynamixel MX-64T and smaller AX-12 smart servo motors (Robotis, South Korea). Each unit can measure position, mechanical load and temperature, and possesses its own microcontroller for communication. Our experiments revealed that MX-64Ts can output sufficient torque at stall, while only weighing 1.24 N.

Motors are controlled by an Arbotix-M (Vanadium Labs LLC, New York), an Arduino-compatible microcontroller based on the ATMEGA644p. Conveniently, chains of Dynamixels can be plugged into the TTL connectors built into the board, providing power and communication. In addition, this board is supported by the AnimatLab Robotics Toolkit, which provides low-level controls for MantisBot. We power both the servos and the Arbotix-M with a 12 VDC 83 A power supply. To avoid running all motors’ current through the Arbotix-M, some motors plug directly into the microcontroller while others plug into a power hub, which when interfaced with the Arbotix-M via a TTL cable functions as a signal repeater.

The Arbotix-M also has eight analog inputs, allowing us to use strain gages for continuous load detection on each leg. Each strain gage is mounted on the proximal dorsal surface of the femur, providing the same kind of information as the femoral campaniform sensilla (fCS), which are crucial to timing stance and swing motions [Zill et al., 2004, Akay et al., 2001]. A 5V rail is used to power an LM324 op-amp and Wheatstone quarter-bridge for each strain gage.

MantisBot’s structural components are all made from polycarbonate (PC). We chose PC over aluminum because PC is sufficiently strong for a robot of this size and offers a better strength to weight ratio. Using PC also allowed most of the components to be 3D printed, allowing for complex geometries. For example, MantisBot’s body
segments are each a 3D truss, a very strong and light shape that would be difficult to produce with subtractive manufacturing. PC also is flexible enough that the amplifier gain for the strain gages can be set low (200), producing a very clean signal.

The microcontroller communicates with a desktop computer (i7 2770K 3.5 GHz, 32 GB RAM) at 256 kbps over a virtual serial connection (USB) using a modified version of the Firmata protocol. The Arbotix-M collects inputs from the robot and writes them to a buffer that AnimatLab uses to update the neural control system. MantisBot’s inputs are the position of all 28 servos, as well as femoral strain, one gage for each of the six legs. The strain is used as a 10-bit analog signal to provide the network with continuous (i.e. not discrete) load signals. The network then writes new motor position commands to the buffer for the Arbotix-M to read.

5.3.2 Mantis Kinematics

Mantises are highly flexible insects with many degrees of freedom (DOF). Figure 5.2 shows a to-scale schematic of the animal with segments and joints labeled.

The prothorax and mesothorax are connected by a multi-DOF joint which allows the mantis to rear and pivot the prothorax and the attached forelegs and head. Each thoracic segment (prothorax or T1, mesothorax or T2, and metathorax or T3) has a pair of multi-jointed legs.

Each leg has four main segments, moving distally: the coxa, trochanter, femur, and tibia, terminating in a tarsus (foot) for gripping the substrate. The T1 legs are highly mobile, possessing three thorax-coxa (ThC) joints, which together function like a ball-and-socket joint. The trochanter and femur are fused, keeping the coxa-trochanter (CTr) and femur-tibia (FTi) joints parallel. CTr extension lowers the tarsus toward the ground, and FTi flexion pulls the tarsus backward toward the body. The T2 and T3 legs each has the same DOF as those in T1, with the addition of a mobile trochanter-femur (TrF) joint.
Because the cockroach has been studied more thoroughly and is closely related [Svenson and Whiting, 2004], it may be helpful to contrast mantis leg anatomy to that of the cockroach. Unlike the cockroach, the mantis’s T2 legs possess inwardly-mobile ThC3 joints, which rotate the leg ventrally about the coxae’s long axis. The ThC3 joint is used to maintain the mantis’s upright hunting posture. The T3 legs are nearly identical to the T2, except that the ThC3 moves the leg dorsally, and the TrF joint is less mobile. Figure 5.2 shows a schematic of each leg and the DOF it possesses.

The T2 and T3 legs also differ from the T1 in that the segments are proportioned differently. The raptorial T1 legs are specialized for grasping, while T2 and T3 provide a wide, stable base for four-legged posture. When walking, however, the T1 legs are used like the front legs of other insects. Table 5.1 shows the measurements of leg segments from a male *Tenodera* on which MantisBot is based. On the front leg, the femur is roughly 150% the length of the coxa, and 250% the length of the tibia. On the middle leg, the coxa is much shorter and the tibia is much longer, making the femur about 350% the length of the coxa, and 130% the length of the tibia.

### 5.3.3 Robot Kinematics

MantisBot has two body segments, a prothorax and mesothorax/metathorax, which are connected by a two-DOF joint. This enables the prothorax to rear and yaw with respect to the main body segment. The yawing motion is directly driven by an MX-64T. The rearing is driven by a four-bar mechanism underneath the thorax, with an MX-64T on the rear of the robot. The four-bar mechanism both provides additional mechanical advantage required to lift the prothorax and moves the center of mass of the robot rearward, which is beneficial for quadrupedal posture.

MantisBot’s T1 legs include all of the degrees of freedom of the animal. This is important because the front legs are the most mobile and volitional [Cleal and Prete,
Figure 5.2: Scaled schematic of a male *Tenodera sinensis*, with segments and degrees of freedom labeled. Joints with green arrows are not included on MantisBot. To see how the robot captures these proportions and DOF, see Figure 5.3.
MantisBot Possesses Most Important Joints of the Animal

Figure 5.3: Photos of each leg of MantisBot with joints labeled. 13.3:1 leg schematics from Figure 5.2 are overlaid to compare the robot’s proportions to the animal’s. This comparison is quantified in Table 5.1.

1996], and will be necessary for studying directed motions such as striking at prey, or mobile locomotive tasks such as climbing. MantisBot’s T2 and T3 legs possess ThC1, ThC3, CTr and FTi joints, which our previous works suggests are the most crucial for postural tasks [Szczecinski et al., 2015c]. Figure 5.3 shows photos of each of the robot’s legs overlaid with a scale schematic of the corresponding leg from *Tenodera*. The most noticeable discrepancy is that MantisBot’s T2 and T3 coxae are longer than the animal’s. This is necessary because placing the motors as close together as possible would establish a 23.1:1 scale, which would make the legs very long and reduce the mechanical advantage of the proximal leg motors so much that the robot would be unable to support itself. However, a 13.3:1 scale for the femora and tibia would mitigate this problem, establishing the scale used for most of the robot. These proportions are within the range of variation of other mantid species (G. Svenson, personal communication). In total, MantisBot weighs 63.2 N, and when all motors are zeroed, has an envelope of 90 cm wide, 60 cm long, and 50 cm tall.
Table 5.1: Segment lengths for the animal (left) and robot (center) in mm. The ratios between the segments is also shown for comparison (right). Weight is also included in the top row. Compare the lengths to the overlays in Figure 5.3.

<table>
<thead>
<tr>
<th>Animal Segment</th>
<th>Animal T1</th>
<th>Animal T2</th>
<th>Animal T3</th>
<th>Robot Segment</th>
<th>Robot T1</th>
<th>Robot T2</th>
<th>Robot T3</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coxa</td>
<td>13.3</td>
<td>5.50</td>
<td>5.60</td>
<td>Coxa</td>
<td>135</td>
<td>127</td>
<td>127</td>
<td>585:1</td>
</tr>
<tr>
<td>Femur</td>
<td>18.2</td>
<td>18.6</td>
<td>24.0</td>
<td>Femur</td>
<td>175</td>
<td>249</td>
<td>320</td>
<td>9.62:1</td>
</tr>
<tr>
<td>Tibia</td>
<td>7.3</td>
<td>14.2</td>
<td>18.0</td>
<td>Tibia</td>
<td>100</td>
<td>190</td>
<td>240</td>
<td>13.7:1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Weight</th>
<th>Animal</th>
<th>Robot</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>0.108</td>
<td>63.2</td>
</tr>
</tbody>
</table>

123
5.4 Robot Control Architecture

5.4.1 AnimatLab-MantisBot interface

MantisBot is the first mobile robot designed to be controlled with the AnimatLab Robotics Toolkit (ART). MantisBot was designed with AnimatLab 2 by first constructing a virtual model of its body, servos, sensors, and nervous system. The ART lets us assemble a model of the robot in a graphical user interface (GUI), and fully simulate the rigid body model and servos via the open source Bullet physics engine (bulletphysics.org). In simulation, the nervous system interfaces with the virtual body, producing a neuromechanical simulation used for initial testing of the control system. The ART is designed to allow the same nervous system to control both the simulation and the real robot by swapping out the underlying physics engine for a “robotic engine”, which handles communication with the robot.

A link to the robot control hardware is added within the GUI by defining a hardware interface, which contains one or more I/O control modules that interact with a specific type of microcontroller. Part interfaces can be added to an I/O controller to link specific sensors or motors to their counterparts within the simulation.

MantisBot uses the Firmata I/O protocol to communicate with an Arbotix-M. Firmata allows the robotics engine to interface with most servos and sensors without requiring any new programming. The robotics engine runs on the master computer, and configures the I/O of the Arbotix-M slave. For a Dynamixel servo, the membrane voltage from a motor neuron is converted to a position command to control torque output. Motor commands that have changed are sent to the Arbotix-M, which updates the servos simultaneously. Servos are read round-robin with the data from one servo being read and sent back during each update. Digital and analog signals are sent back each time they change. The engine converts the incoming sensor values into currents that are injected into sensory neurons, completing the sensory/motor
feedback loop. The robotics engine ensures that neural processing is kept in synchrony with the real-time I/O of the hardware, but on a per time step basis it easily simulates MantisBot’s control system of 775 neurons and 1258 synapses 150 times faster than real time.

5.5 Conclusions

MantisBot is a research robot designed after the praying mantis *Tenodera sinensis* and controlled with the AnimatLab Robotics Toolkit to explore the neural control of motion by descending commands from the central complex. It is capable of emulating reflexes seen in the animal via biological neural networks, including central pattern generators. These basic capabilities provide a basis for future behavioral development and show the feasibility of controlling a robot in this manner.
Chapter 6

Mantisbot is a Robotic Model of Visually Guided Motion in the Praying Mantis

This chapter was originally published as


Minor edits have been made, particularly in adding references to other parts of this thesis.

6.1 Abstract

Insects use highly distributed nervous systems to process exteroception from head sensors, compare that information with state-based goals, and direct posture or locomotion toward those goals. To study how descending commands from brain centers produce coordinated, goal-directed motion in distributed nervous systems, we
have constructed a conductance-based neural system for our robot MantisBot, a 29 degree-of-freedom, 13.3:1 scale praying mantis robot. Using the literature on mantis prey tracking and insect locomotion, we designed a hierarchical, distributed neural controller that establishes the goal, coordinates different joints, and executes prey-tracking motion. In our controller, brain networks perceive the location of prey and predict its future location, store this location in memory, and formulate descending commands for ballistic saccades like those seen in the animal. The descending commands are simple, indicating only 1) whether the robot should walk or stand still, and 2) the intended direction of motion. Each joint’s controller uses the descending commands differently to alter sensory-motor interactions, changing the sensory pathways that coordinate the joints’ CPGs into one cohesive motion. Experiments with one leg of MantisBot show that visual input produces simple descending commands that alter walking kinematics, change the walking direction in a predictable manner, enact reflex reversals when necessary, and can control both static posture and locomotion with the same network.

6.2 Introduction

Praying mantises make visually-guided saccades toward prey while hunting [Lea and Mueller, 1977, Mittelstaedt, 1957, Rossel, 1980, Yamawaki et al., 2011], making them ideal model organisms for the study of descending commands and goal-directed motion. Their saccades consist of rotating the head, prothorax, and thorax in near-unison in ballistic, predictive pivots, during which they are apparently insensitive to changing visual input [Lea and Mueller, 1977]. This system requires that the mantis locate prey in its visual field, predict its position sometime in the future, remember this position, and execute a planned, coordinated motion with its neck, jointed prothorax, and legs. If starved, mantises will also pursue prey, requiring them to redirect their
locomotion toward their visual goal. All of these tasks require that the animal receive visual input, process it, plan corrective motions, and then communicate these motions to the thoracic ganglia that control motion via descending commands. We present a conductance-based neural controller, based strongly in the literature on mantis sac- cades and insect locomotion, for our robot MantisBot (Chapter 5, [Szczecinski et al., 2015a]). This controller serves as a hypothesis of how the thoracic ganglia are organized to produce static posture and locomotion, and how descending commands from the brain may alter their function to accomplish a goal.

Much is known about how insects control locomotion (for a review, see [Buschmann et al., 2015]). Each joint has its own central pattern generator (CPG), which can produce rhythms, even when deafferented [Ryckebusch and Laurent, 1993, Büschges et al., 1995]. Sensory signals such as joint motion [Hess and Büschges, 1997, Hess and Büschges, 1999, Bucher et al., 2003] and leg strain [Akay et al., 2004, Ridgel et al., 1999] provide feedback to the oscillators, adjusting their relative phase and coordinating motion. Drastic changes in coordination, such as those seen while insects turn or walk backward, are thought to be the result of reflex reversals, wherein a sensory input causes the opposite transition in a CPG [Akay et al., 2006, Mu and Ritzmann, 2008, Hellekes et al., 2011, Szczecinski et al., 2014].

Many studies with insects have sought to explain what type of descending commands are sent from navigational centers in the brain to the thoracic networks to direct locomotion (for reviews see [Ritzmann et al., 2012] and [Borgmann and Büschges, 2015]). Neurons in the central complex are known to predict the direction and speed of cockroach locomotion [Guo and Ritzmann, 2013, Martin et al., 2015], and stimulating these neurons can elicit the same reflex reversals observed when the animal changes its walking direction (Martin et al., 2015). These studies, however, do not elucidate what information the thoracic ganglia are receiving from the brain. In the fruit fly, an individual neuron (called “Moonwalker descending neuron”, or MDN) has
been identified in the brain whose stimulation is necessary and sufficient to reverse the direction of walking, suggesting that in some cases, descending information may be as straightforward as tonic signals from a single neuron [Bidaye et al., 2014]. In our controller, the thoracic networks are provided only with two pieces of information: whether to walk or stand still, and in what direction to orient the body. This simplicity is reminiscent of the MDN, and is consistent with the highly distributed nature of insect locomotion control systems [Cruse, 1990].

The effect of descending commands has been the focus of an increasing number of modeling and robotics studies [Schilling et al., 2013b], especially reflex reversals observed when insects walk backwards or along a curved path [Rutter et al., 2011, Knops et al., 2012, Toth et al., 2012, Szczecinski et al., 2014]. These fall into two main categories: centralized models used to produce detailed descending commands, and simple descending commands used to modify the function of sensory-motor networks. Walknet and the associated robot Hector use a navigational network (Navinet) to direct the stance-phase motion of the legs toward a goal. This is accomplished through a marionette-like internal body model, implemented as a recurrent neural network, which enables Navinet to “pull” the body in the desired direction, and then command the resulting motion to the motor controllers. Such a centralized approach is in contrast with the distributed approach of models from Daun-Gruhn et al., which focus on reversing the phase of some joints’ motion by changing the sign of the connections between the CPGs and motor neurons (MNs) [Knops et al., 2012, Toth et al., 2012].

Previous work from our group has built computational [Szczecinski et al., 2014] and robotic [Rutter et al., 2011, Klein et al., 2014] models that accomplish this same phase shift by changing the sign of sensory information that coordinates the CPGs in one leg. It is possible that the nervous system uses one or all of these methods to direct locomotion [Buschmann et al., 2015]. Our controller for MantisBot supports the hypothesis that simple descending commands may change low-level network function
to direct locomotion, but tuning this network relies on a kinematic model of the whole robot, representing a compromise between these two approaches.

In this paper, we present a conductance-based neural controller (see Chapters 3 and 4) for the robot MantisBot, enabling it to saccade its head toward visual targets as seen in the animal. This includes the processing of simplified visual information, predicting the position of prey in the future, generating ballistic saccades, storing the anticipated position of prey in memory, and using this information to generate descending commands about the desired direction of travel to the thoracic ganglia that enact these corrective saccades. In addition to the desired direction, the high level controller can specify whether the robot is in an active (i.e. mobile) or inactive state. When active, the robot uses a sensory-coupled network of CPGs to walk in the direction specified by the descending commands, mimicking a starved mantis pursuing prey. When inactive, the intended direction causes the leg to move as observed when mantises change their posture to track prey with their feet planted. We hypothesize that descending commands to the thoracic ganglia are comparatively simple (e.g. “turn 30 degrees to the right” or “climb over a 10 cm block”), relying on the distributed computational power of the thoracic ganglia to use these commands to modify reflexes and adjust MN output as appropriate for each leg and leading to the desired change in direction.

### 6.3 Robot Hardware

Figure 6.1 shows MantisBot, a 29 degree of freedom robot modeled after the praying mantis *Tenodera sinensis* (Chapter 5, [Szczecinski et al., 2015a]). MantisBot is controlled by a real-time conductance-based neural simulation running in AnimatLab 2 on a desktop computer (Intel i7-4770K @ 3.50 GHz, 32 GB RAM). On board the robot, an Arbotix-M microcontroller interfaces with the hardware. This gives the
neural system access to the servos’ positions, as well as strain sensors on the proximal end of each femur to simulate the trochanteral campaniform sensilla (trCS). The neural system commands new positions for the servos at each time step. All actuators are Dynamixel Smart Servos (Robotis, CA, USA). The middle and hind legs each have four degrees of freedom: two thorax-coxa (ThC) joints, a coxa-trochanter (CTr) joint, and a femur-tibia (FTi) joint. The front legs each have five degrees of freedom, including an additional ThC joint. The body has three joints: a neck that can yaw the head, and yawing and pitching joints at the mesothorax-prothorax interface.

To enable the study of descending commands’ effects on thoracic networks, a group of simple head sensors was developed, shown in Figure 6.2. Five Sundance (NH, USA) calibrated solar cells are arranged in a convex orientation. An analog circuit buffers the cells’ inputs, then adds the two upper, two lower, two left, and
CHAPTER 6. MANTISBOT IS A ROBOTIC MODEL OF VISUALLY GUIDED MOTION IN THE PRAYING MANTIS

Figure 6.2: MantisBot’s head sensor detects the azimuth and elevation of the centroid of luminosity in its field of view. A. The four primary sensors are each angled 40 degrees from the transverse plane, away from the dorsal and sagittal planes. Each pair’s activation is added and biased to encode the position of the “prey” in the egocentric frame. B. Experimental data shows that this arrangement accurately captures the prey’s position with a 150 degree field of view in both directions.

two right sensors’ voltages. The difference between the top and bottom sensors is returned as one output of the circuit, and the difference between the left and right is returned as another. To facilitate the analog to digital conversion, each channel’s output is kept positive by biasing them upwards by 2.5 V. In this case, a centered light source returns 2.5 V along each channel, which the analog to digital converter interprets as 512 out of 1023. To collect data from the robot, the thorax was rigidly fixed in space. The head and prothorax were free to move, and orient toward “prey” (a 1600 lumen LED lightbulb) in a closed-loop manner. To simulate walking, a
Teflon coated surface was placed under MantisBot’s left middle leg, which provided the force feedback necessary to coordinate its CPGs. This is comparable to the oiled plate preparation used in several insect preparations (Gruhn et al., 2006; Mu and Ritzmann, 2005). Like the head and prothorax, the leg’s motion was also directed toward the prey (i.e. it would have pushed the robot toward the prey if the thorax were not fixed), but in an open-loop manner, because the orientation of the body did not change when the foot moved.

6.4 Robot Controller

6.4.1 Neuron Models

MantisBot is controlled by a real-time conductance-based neural simulation, implemented with the AnimatLab Robotics Toolkit (Chapter 5, Szczecinski et al., 2015a). Neurons are modeled as nonspiking leaky integrators with nonlinear persistent sodium channels. This model is used because it is one of the simplest dynamical neuron models. Its time response enables the construction of useful networks that would not function otherwise, such as the bursting action required for robust locomotory rhythms (Daun-Gruhn et al., 2009), differentiator networks (Chapter 4), and the memory network described in this paper (Section 6.4.2 and Chapter 4). The primary dynamical variable is the voltage across the cell membrane, \( V \), with dynamics:

\[
C_m \dot{V} = I_{\text{leak}} + I_{\text{syn}} + I_{\text{NaP}} + I_{\text{app}}
\]  

(6.1)

where

\[
I_{\text{leak}} = G_m \cdot (E_r - V),
\]  

(6.2)
\[ I_{\text{syn}} = \sum_{i=1}^{n} G_{s,i} \cdot (E_{s,i} - V), \quad (6.3) \]

\[ I_{NaP} = G_{NaP} \cdot m_\infty(V) \cdot h \cdot (E_{NaP} - V), \quad (6.4) \]

and \( I_{\text{app}} \) is an arbitrary external stimulus. These equations define the leak, synaptic, and sodium channel currents and follow the same basic form of a conductance, \( G \), multiplied by the difference between the current membrane voltage, \( V \), and a constant reference voltage (i.e. reversal potential) \( E \). \( E_r \) is the resting potential of the neuron, and \( C_m \) and \( G_m \) capacitance and conductance of the cell membrane, respectively.

Neurons that make up the CPGs have \( G_{NaP} \neq 0 \), making \( h \) the second dynamical variable for the neuron, with the dynamics:

\[ \dot{h} = \frac{(h_\infty - h)}{\tau_h(V)}. \quad (6.5) \]

\( m_\infty(V) \) and \( h_\infty(V) \) are sigmoids that monotonically increase and decrease, respectively.

Neurons communicate via synapses. The synaptic conductance \( G_{s,i} \) is a threshold linear function of the \( i^{th} \) incoming (i.e. presynaptic) neuron’s voltage:

\[ G_{s,i} = g_i \cdot \min \left( \max \left( \frac{V_{\text{pre}} - E_{lo,i}}{E_{hi,i} - E_{lo,i}}, 0 \right), 1 \right), \quad (6.6) \]

where \( g_i \) is the constant maximum conductance of the synapse, \( V_{\text{pre}} \) is the presynaptic neuron’s voltage, and \( E_{lo,i} \) and \( E_{hi,i} \) are the synaptic threshold and saturation, respectively. The full details of the model can be found in Section 3.3.1.
6.4.2 Network Architecture

Head Networks

Mantises have sophisticated visual systems that enable them to extract many features of prey-like stimuli [Prete et al., 2013], as well as foveated vision [Rossel, 1980]. To focus on the core principles of how descending commands affect thoracic network activity, we have omitted these details, opting instead for the simple, faux-visual sensor described in the Robot Hardware section. In spite of this simplification, our network is designed to capture features of mantis saccadic tracking.

Mantises track prey almost exclusively with ballistic saccades of the head, prothorax, and legs. This means that once initiated, the motion cannot be stopped [Lea and Mueller, 1977]. They only saccade if the prey leaves a cone of about 20 degrees from the center of their field of view [Rossel, 1980]. Changing the position of the target after the saccade has been initiated will cause the mantis to orient toward the initial position, strongly suggesting that the saccade does not incorporate visual feedback [Lea and Mueller, 1977]. Ablating the hair plates on the neck, however, will disrupt the saccades’ accuracy, suggesting that proprioceptive feedback is critical [Mittelstaedt, 1957]. Faster moving targets elicit larger saccades, suggesting that the mantis predicts where the prey will be at the end of a saccade, and orients in that direction [Rossel, 1980]. For saccades larger than 30 degrees, the prothorax and legs are also recruited to rotate the head in a planned and coordinated fashion [Yamawaki et al., 2011].

The network in Figure 6.3 is designed to capture these phenomena in spite of simplified visual input. The network receives visual input that encodes the azimuth and elevation as continuous signals (for example, see Figure 6.2 and Figure 6.7). The network uses this input to produce saccades of stereotypical duration by detecting when the prey leaves a 20 degree span in front of the robot. When this is detected,
the network plans the ballistic saccade by storing the predicted position of the prey and the current neck angle in memory. It then disables visual input, and uses the remembered position of the prey and neck to calculate the new orientation of the neck required to accurately saccade. It then uses proprioceptive feedback from the neck to drive the head to the desired position. The predicted position can also be scaled and sent to both the head and prothorax joint, producing one planned, ballistic motion. This ballistic motion is a result of the network’s structure and function. The network, shown in Figure 6.3, works as follows. Continuous light sensor readings are converted to neural activity by injecting the Left neuron with current proportional to the sensor’s reading. When the voltage of the Left and Right neurons are equal, then the prey is directly ahead of the robot. If this is not the case, they excite either the Error Left or Error Right neuron. These neurons are tonically hyperpolarized such that errors less than 20 degrees in either direction elicit no supra-threshold activity, and thus no saccade. If the error is beyond 20 degrees, then this threshold crossing is detected by a Reichardt detector [Reichardt, 1961] made up by the Error Fast and Error Slow neurons. The Error Slow neuron’s response lags behind the Error Fast neuron’s because its membrane capacitance, and thus its time constant, is greater. If either the Error Left or Error Right neuron suddenly becomes active, the Error Incr. (Increasing) neuron is temporarily depolarized, signaling for a saccade.

The Saccade and No Saccade neurons make up opposing halves of a persistent sodium bursting network like those used in locomotion studies [Daun-Gruhn et al., 2009, Szczecinski et al., 2014]. The No Saccade neuron has a tonic stimulus that causes it to remain in a bursting state, inhibiting the Saccade neuron. The Error Incr. neuron temporarily hyperpolarizes the inhibitory interneuron between the Saccade and No Saccade neurons, releasing the Saccade neuron and causing a stereotypical burst of activity to drive the saccade.

The mechanisms described so far control a saccade’s timing, but what about its
Figure 6.3: Hypothetical network that processes visual input, predicts the future position of the prey, and generates saccades. In the top shaded area, visual processing centers convert sensor activity to neural activity, and calculates the position and velocity of the prey. In the center area, brain networks, possibly in the central complex, predict the future position of the prey, encode this position in memory, use the visual information to decide to saccade, and generate descending commands to the thoracic networks. In the bottom shaded area, the thoracic networks fuse their current
Figure 6.3: (contd.) position with the descending commands to produce the intended
motion. Memory networks are outlined in red, dashed boxes, for identification.

magnitude? In the animal, faster prey elicit larger saccades, so our network estimates
the velocity of the prey with another Reichardt detector. The Future Position re-
ceives inputs from the Left, Left Velocity, and Right Velocity neurons to estimate the
location of the prey one saccade’s duration in the future. This value is then remem-
ered once the Saccade neuron is active by inhibiting the incoming Prey to Left and
Prey to Right neurons. This has the added effect of ignoring visual input during the
saccade, and preventing movement-induced changes to the visual neurons.

Studies of saccadic eye movements in goldfish [Koulakov et al., 2002] and humans
[Seung et al., 2000a] suggest that stable working memory is a requirement for such
tracking. These and other studies [Seung et al., 2000b] assume that positive synaptic
feedback is necessary for a model of working memory. In the next section, we present
an alternative composed of two mutually inhibitory neurons, which can be tuned to
be a line attractor capable of stably storing memory for an indefinite amount of time.
The result is the inclusion of Prey Memory and Neck Memory neurons that, upon
the initiation of a saccade, store the predicted position of the prey and the current
angle of the neck, respectively. The location of the prey is then added or subtracted
from the position of the neck by the Sacc. (Saccade) Position neuron, which is used
to drive a negative feedback loop of the neck position.

Starved mantises recruit their neck, prothorax joint, and legs in predetermined
proportions during saccades [Yamawaki et al., 2011], meaning that the prothorax
joint also has a network like that at the bottom of Figure 6.3. Each joint has its own
Sacc. Position neuron, which is the combination of the prey angle and that joint’s
angle. The strength of the synapses between the Move Left or Move Right neuron
and the Sacc. Position neuron can be scaled to adjust the contribution of each joint
to the final motion.
CHAPTER 6. MantisBot is a robotic model of visually guided motion in the praying mantis

Memory Network

Figure 6.4A shows the memory network removed from the network in Figure 6.3. Chapter 4.5.2 mathematically shows that the mutually inhibitory neurons inside the red dashed box form a memory unit. Figure 6.4B shows a phase portrait of the network dynamics with the equilibrium manifold plotted when $G_m = 1$, $E_r = -60$, $g_s = 0.5$, $E_{lo} = -60$, and $E_{hi} = -40$. The flow field suggests that the manifold is attractive and marginally stable, which we verify in Chapter 4.5.2. This marginal stability also implies that this network is an integrator, meaning that writing a specific value to the Mem 1 neuron is not trivial. Therefore, a comparator is used to inject current into the Mem 1 neuron proportional to the difference between Input and Mem 1’s voltage (Chapter 4.4.4). Figure 6.4D shows that Mem 1’s voltage closely follows Input’s, unless the Remember neuron inhibits the Interneurons, storing the memory.

Leg Networks

An insect’s distributed control system is critical to its adaptive behavior [Cruse, 1990]. They must be able to use sensory information to coordinate different joints’ rhythmic output, allowing them to assemble new motions from the same pieces. Therefore we have constructed a leg control network building on our previous modeling work [Szczecinski et al., 2014] and that of others [Ekeberg et al., 2004, Knops et al., 2012] capable of smoothly transitioning its direction of travel by modifying how each joint in the leg uses sensory information.

Figure 6.5A shows a schematic of the hierarchical control network. The high level controller (e.g., networks in the central complex) has two purposes: it sets the intended thorax translation and rotation during stance phase, and whether the robot is pursuing or (statically) tracking prey. If the robot is pursuing, then the desired thorax translation is positive, but if it is tracking, it is zero. The desired rotation is a proportion of the sum of the neck and prothorax angles. This is based on the fact
Figure 6.4: Illustration and demonstration of memory network. A. The memory network proposed in this work. The neurons in the red dashed box together form the marginally stable memory unit. Memory units in 6.3 are also boxed. B. Solving for the network’s equilibrium state reveals an attractive manifold of solutions. C. Analytically computing the eigenvalues for the system reveal that there is always one zero and one negative eigenvalue, guaranteeing that the memory network functions as a line attractor. D. Demonstration that the Memory neuron’s activity follows that of the Input neuron, but remembers its state when the Remembering neuron is active.
that a mantis will rotate its thorax about the vertical axis in a specific proportion to its neck and prothorax, and leg motion may lag behind the motion of these joints [Yamawaki et al., 2011]. These inputs (whether or not the robot is moving, and in what direction) interface with the leg level controller, which has three purposes. First, it maps the intended thorax motion to the leg’s posterior extreme position (PEP) and anterior extreme position (AEP). Second, it biases the PEP position that causes swing phase. The high level controller ceases walking by changing this bias to make the PEP unreachable. In this way, the leg will move to pivot the robot, but will not lift off. Finally, the leg level controller also takes sensory feedback from the joint controllers to determine when the PEP and AEP have been reached to transition between swing and stance phase, respectively.

Figure 6.5B shows a detailed view of the low level controller for one joint, as well as the high- and leg-level controllers. Each joint’s desired PEP and AEP are linear combinations of the body’s intended direction and stance length, formed by the excitatory connections to the Desired PEP and Desired AEP neurons in row i. These are compared to the resting angle of the joint, and the activity of the Ext. PEP and Flx. PEP neurons in row ii indicate how far the joint extends or flexes in stance, respectively. Only one of these will show supra-threshold activity at a time. A parallel structure controls the AEP. These neurons play two roles. First, via the inhibitory interneurons in row iii, they indicate whether strain information from the leg should cause flexion or extension. For instance, when the PEP is more extended than the resting angle, then the blue pathway Figure 6.5B is active, enabling load information to excite the CPG’s extension (Ext.) half. In contrast, when the PEP is more flexed than the resting angle, the red pathway is active, enabling load information to cause flexion (Flx.). An identical structure directs information about leg unloading. Second, via the excitatory synapses to row vi, they control the disinhibition of the Ext. Relay and Flx. Relay interneurons between the CPG and MNs. This disinhibition changes
CHAPTER 6. MANTISBOT IS A ROBOTIC MODEL OF VISUALLY GUIDED
MOTION IN THE PRAYING MANTIS

Figure 6.5: The leg control network, which produces coordinated motion despite its
distributed structure. A. A zoomed out view of the leg controller shows the hierarchy
CHAPTER 6. MANTISBOT IS A ROBOTIC MODEL OF VISUALLY GUIDED MOTION IN THE PRAYING MANTIS

Figure 6.5: (contd.) of control: high level control sets whether the robot is walking or standing still, and how the body should rotate and translate. Leg level control maps the desired body direction to the AEP and PEP, and decides when to enter stance or swing. When the Desired PEP neuron is more active than the Resting Angle neuron, then the pathway from the load sensor (orange) to the Ext. CPG HC. is disinhibited, causing joint extension in stance phase (blue pathway). If instead the Desired PEP neuron is less active than the Resting Angle neuron, then the pathway from the load sensor to the Flx. CPG HC. is disinhibited, causing joint flexion in stance phase (red pathway). These pathways also disinhibit Ext. Relay and Flx. Relax interneurons to control the amplitude of the motion.

The MNs control the motor velocity, which causes motion. This motion is registered by the Joint Rotation neuron in row viii. This information is compared to the desired PEP and AEP in rows i and ii. Whether swing is caused by extending or flexing past the PEP is again modulated by the Ext. PEP and Flx. PEP neurons, which hyperpolarize the irrelevant comparison. For example, if this joint flexes in stance phase, then it should trigger swing when the joint angle is less than the PEP. Therefore the Flx. PEP neuron, which will be active, inhibits the Angle > PEP neuron, because this comparison is not useful for the specified walking motion. This structure is mirrored to compare the joint rotation to the AEP to transition to stance phase.

The ThC1, ThC3, and FTi joint controllers have identical structure. The CTr joint controller varies only in that it takes input from the Enter Stance and Enter Swing neurons where Figure 6.5B shows the Leg Load and Leg Unload neurons. It also includes a connection from the Leg Load neuron, enacting the positive reinforcement of stance phase observed in insects [Zill et al., 2014, Zill et al., 2015].

How is this network tuned to produce the intended motion? Robust CPG rhythms
and stable joint controllers can be designed by a combination of numerical and ana-
ytical methods (Chapter 3, [Szczecinski et al., 2017b]). Here we focus on how simple
descending commands encoding only the body’s intended translation and rotation are
used to modulate the joint control networks.

Figure 6.6 illustrates the process to map from the translation, $p$, of a point on
the body, $Q$, and the rotation of the body, $\psi$, to joint angles, $\theta$. As shown in Figure
6.6, rotating and translating the body reveals a vector $x$ that the foot must move
along to return to its original position, in the reference frame of the body. We can
use inverse kinematics to find the joint angles that move the foot to the new position.
From robotic manipulation, we know that

$$V = J(\theta) \cdot \dot{\theta}, \quad (6.7)$$

in which $V$ is the velocity vector of the foot, $\theta$ is a vector of the joint angles in the
leg, and $J$ is the manipulator Jacobian matrix, which maps between them (Murray
et al., 1994). If we instead look at small displacements of the foot and joint angles
over a small time $\Delta t$, we can write Equation 10 as

$$\Delta x = J(\theta) \cdot \Delta \theta. \quad (6.8)$$

If we have a known starting vector (i.e. resting posture) $\theta_0$, then we can iteratively
update $\theta$ and $J$ to move the foot along the $x$ vector. Because $J$ is not square, we use
the Moore-Penrose pseudo-inverse to update $\theta$ until we reach the end of $x$:

$$\theta_{i+1} = \theta_i + J(\theta_i)^T \cdot (J(\theta_i) \cdot J(\theta_i)^T)^{-1} \cdot \Delta x. \quad (6.9)$$

Figure 6.6B shows that the body transformations map to joint angles in a very plan-
ar way for the leg middle leg. Rather than relying on machine learning algorithms,
CHAPTER 6. MANTISBOT IS A ROBOTIC MODEL OF VISUALLY GUIDED MOTION IN THE PRAYING MANTIS

Figure 6.6: Illustration of the inverse kinematics method used to tune joint control networks. A. For a grid of values, a point on the robot, Q, is translated p, and the body is rotated ψ. This reveals the translation x and rotation φ of the foot required to return to its original position, in the body’s reference frame. The method in Equation 6.9 finds the joint angles required to execute this return. B. This process produces maps between body movement and joint angles for the left middle leg. Lines of no joint movement are drawn in black on white. These divide each map into regions of flexion and extension in stance phase, predicting what body motion elicits reflex reversals. The body motion tested in Figure 6.8 and Figure 6.9 are marked with red “+s”.

[Diagram of inverse kinematics method and joint control networks]
these relationships can be encoded into the network in Figure 6.5 by designing the four
synapses between the leg and joint controller, in a deterministic way. The leg, how-
ever, does not simply play back joint trajectories calculated with inverse kinematics;
the joints each oscillate independently, and are only coordinated by sensory informa-
tion. This method enforces stance- and swing-phase synergies like those insects are
known to use to control posture and locomotion [Zill et al., 2015].

Some pairs of body translation and rotation require joint angles larger than the
rest position (at (0,0)), and others require smaller angles. By drawing a contour line
over the joint angles equal to the rest position, we can divide each joint’s motion
into regimes where it extends or flexes in stance. These contour lines predict what
body motion will produce the reflex reversals observed in insects [Mu and Ritzmann,
2008, Hellekes et al., 2011, Szczecinski et al., 2014, Martin et al., 2015]. The activity
of the Ext. PEP, Flx. PEP, Ext. AEP, and Flx. AEP neurons in Figure 6.5 encode
this information, re-routing information about stance phase to the opposite half of
the CPG, and enacting a reflex reversal. We will see in the following section that this
structure works precisely as intended.

6.5 Robot Performance

6.5.1 Saccade Accuracy

MantisBot uses the network shown in Figure 6.3 to predict the future position of prey,
and accurately saccade toward that position. Figure 6.7A shows that the activity of
the Future Position neuron leads the activity of the Left neuron, which encodes the
egocentric azimuthal position of prey. The mean and standard deviation lead time
is 254 ± 117 ms, which includes our saccade duration of 300 ms. Shifting the Left
neuron’s activity forward in time by this amount shows that they overlap, and that
the system indeed predicts the future position of prey. Figure 6.7B shows that the
CHAPTER 6. MANTISBOT IS A ROBOTIC MODEL OF VISUALLY GUIDED MOTION IN THE PRAYING MANTIS

Figure 6.7: A summary of the robot’s ability to track prey with saccades. A. Our network predicts the future position of prey, just like the animal [Rossel, 1980]. Examining the current position of the prey and the network’s prediction shows an apparent lag of the current position behind the prediction. Shifting the data by the mean lag time shows that the prediction is indeed accurate. B. The robot keeps the prey within ±30 degrees of center for the entire trial (N = 46 saccades). When the prey exits a ±20 degree window, the Left Bias and Right Bias neurons become active, and trigger a stereotypical saccade. The network remembers the predicted position of the prey throughout saccades (drawn in red), because the animal does not use visual feedback during saccades [Lea and Mueller, 1977].
robot accurately saccades with its neck and prothorax joint. The visual target is kept within 30 degrees of center, shown by the red shading, for the entire trial. Saccades are triggered when the target strays beyond 20 degrees of center, indicated by Left Bias and Right Bias activations above \(-60\) mV. The Saccade neuron fires stereotypical bursts when this occurs, and the Prey Memory neuron remembers the predicted location of the prey one saccade later. Prey Memory activations during saccades are highlighted in red, showing that the value does not change, and memory is stored stably throughout the motion.

6.5.2 Visually Guided Posture and Locomotion

The network in Figure 6.5 produces robust stepping by coordinating CPG oscillations with sensory feedback about leg loading and proprioception. Recall that the neck and prothorax joint angles are used to produce an intended direction descending command for the leg. When the robot saccades toward prey in different locations, do the walking kinematics change? Figure 6.8 shows kinematic data during stance phase as the intended walking direction is modulated. As the intended angle decreases, both of the ThC joints increase their amplitude. The FTi, on the other hand, decreases its amplitude and eventually reverses sign, due to a reflex reversal. This reflex reversal is predicted by the FTi plot in Figure 6.6B. Note that the red pluses (+), which signify the intended directions in Figure 6.8, straddle the zero line for the FTi joint, requiring that the joint changes its direction of travel in stance phase. The structure and tuning of the network enact this reflex reversal with only two simple descending commands: whether the leg should walk, and intended direction. Do the different joint kinematics in Figure 6.8 result in different propulsive motions? Figure 6.9A shows that the foot indeed sweeps different paths in stance phase as the intended angle is modulated. If we fit a line to the first 10 cm of stance phase for each step, we can measure the direction that the foot moves. Figure 6.9B shows that for
Figure 6.8: Walking kinematics change as the intended direction is modulated. The top row shows a stick figure of the leg taking a representative step when the prey is placed at the position specified. Each joint’s kinematics are plotted below, which change noticeably as the prey moves. Note how the FTi joint’s excursion steadily decreases from left to right, eventually exhibiting no motion, and then changing direction. This reflex reversal is enacted because of the network structure, not because of an explicit command to do so.
Figure 6.9: Stance phase foot directions change significantly with altered “prey” location. Scatterplots of the calculated foot positions when the leg is in stance phase qualitatively show that the foot sweeps clear and consistent paths when commanded to walk in different directions relative to color coded “prey” location. Examining the best fit line of the first 10 cm of foot motion reveals that in 5 out of 7 cases, the actual foot direction is within one standard deviation of the intended position. All foot directions are significantly different (p < 0.001, one sided ANOVA). The number of steps used in each calculation is shown in Figure 6.8.

Insects and robots must be able to change their direction midstride to compensate for changing environmental conditions. Can MantisBot stably change its walking direction? Figure 6.10 shows that it can, even while enacting a reflex reversal. As the prey moves in the robot’s field of view, the head and prothorax saccade to track its position. During the first half of the trial, leg strain causes the FTi CPG to flex. As the prey moves in the negative direction (contralateral to the leg), the FTi PEP and AEP approach one another, eventually intersecting halfway through the trial. The joint’s amplitude of motion likewise decreases until it reaches zero. As the prey continues to move contralaterally to the leg, the intended direction of travel likewise
Figure 6.10: Neural activity illustrating a reflex reversal during typical locomotion. As the prey moves, the head and prothorax saccade to follow. In the first half of the trial, the FTi CPG is in the flexing state during stance phase (indicated by gray bars). As the heading changes, the PEP and AEP for the FTi joint approach one another, eventually crossing around 30 s. At this point, the network redirects leg load information to the extension half of the FTi, causing the opposite stance phase motion. The network in Figure 6.5 converts a simple descending command of heading direction into both continuous changes in joint motion amplitude and discrete changes in load utilization.
CHAPTER 6. MANTISBOT IS A ROBOTIC MODEL OF VISUALLY GUIDED MOTION IN THE PRAYING MANTIS

Figure 6.11: The network in Figure 6.5 controls both locomotion and static posture. During the first half of the trial, the robot walks, and each joint oscillates within its PEP and AEP. When the signal is given to cease pursuing prey, the legs still participate in tracking saccades. By inhibiting the interneuron that signals that the PEP has been reached, the leg remains in stance phase, and tracks the PEP.
changes, again increasing the FTi’s amplitude, but now causing extension in stance phase. This change is evident both in the CPG activity and the joint motion. The joint controller’s only input is the body’s intended direction of travel; the structure of the network automatically causes the FTi’s motion to change its sign and amplitude when necessary.

Mantises also use their legs to rotate their bodies while statically tracking prey. Figure 6.11 shows that our leg control network can also be used to control this motion. While walking, the intended body direction modulates the PEP and AEP of each joint, establishing limits for each joint’s motion. When one or all of the joints have surpassed its PEP, swing phase is triggered by either the Angle $>\text{PEP}$ or Angle $<\text{PEP}$ neuron. However, inhibiting the PEP Bias neuron decreases excitation to the Angle $>\text{PEP}$ and Angle $<\text{PEP}$ neurons, preventing supra-threshold activity and thus the signal to enter swing phase. Therefore, when the network switches from pursuit to tracking at 40 s, the joint angles no longer oscillate, but instead follow the PEP. In this way the same pathways can be used to control both walking motion and static posture.

6.6 Discussion

We present a conductance-based neural robot controller that mimics visually guided motions seen in praying mantises. A head sensor was constructed that enables the robot to detect the position of a bright light, which it perceives as prey. The control network produces ballistic saccades of the head and prothorax to accurately orient the head toward the prey. To accomplish this, it must predict the position of the prey one saccade’s time in the future, and remember this position while moving without visual feedback. Our data show that this network completes these tasks, and that the system as a whole replicates the function of mantis saccades [Lea and Mueller,

We also present a hypothetical leg control network in which only very simple descending commands, i.e. whether to walk or not, and in what direction, are interpreted by the thoracic ganglion to control both the timing and amplitude of joint motion. Walking motion is the result of sensory coupled CPGs; there are no direct coordinating pathways between the joints. As in insects, straining the leg quickly transitions the joints to stance phase activity [Akay et al., 2001, Akay et al., 2007] and reinforces supportive muscle synergies [Zill et al., 2015], reaching the PEP transitions the leg to lift [Hess and Büschges, 1999], and a reduction in leg strain may be used to transition to swing phase [Ridgel et al., 1999]. In spite of its distributed nature, our network can coordinate walking motion in many different directions by modulating the amplitude of joint motion.

6.6.1 Why Build a Robotic Model?

Many software models have aided in explaining neurobiological results or proposed hypothetical neurobiological structures [Ekeberg et al., 2004, Knops et al., 2012, Toth et al., 2012, Toth et al., 2013b, Szczecinski et al., 2014]. Why, then, go through the trouble of building a robotic model? Controlling a robot is a more thorough test of the robustness of a neural model than a kinematic or dynamic simulation. Sensors introduce randomness to the control network, which prevents the network from sitting at unstable equilibrium points. For example, the CPGs in our network initialize at an unstable equilibrium point, and will remain there unless noisy feedback dislodges it, or an asymmetrical stimulus is deliberately applied by the modeler. For a system this large, it is impractical to identify every possible scenario like this, and apply stimuli to disrupt them. Therefore, using a robotic model more thoroughly tests the neural model than a kinematic or dynamic simulation of the body would.

Robots also have necessarily more accurate dynamics than a simulation, particu-
larly when interacting with their environment. This confronts the neural model with situations that it could not have encountered in simulation, and more thoroughly tests its effectiveness. For example, MantisBot’s strain sensors would be very difficult to model accurately in software. Dynamic simulations most frequently use rigid-body models, which means that the body and leg segments do not strain under force. When we first tested our controller in hardware, vibrations of the leg in swing phase were detected by the strain sensors, prematurely causing stance phase, or causing a resonance between the leg and the control network. To correct this, we changed the network to only read from the strain gage when the CTr servomotor pulled a substantial amount of current. This means the strain gage would only show activity when the leg’s motion is resisted, a phenomenon seen in insects [Zill et al., 2014]. In this case, implementing the neural model on a robot urged us to include more biological data, both helping to explain an insect reflex and improving the robot’s performance.

We also build robotic models because we are interested in developing legged robots with greater mobility and autonomy. Robots as agile as insects could be used to search disaster sites for survivors, work in dangerous locations like mines, or explore extreme terrains such as on other planets. As more neurobiological data becomes available, we can continue to expand this control network to achieve more animal-like locomotion. Insect-like robots from the past and present have been largely constrained to laboratory settings [Lewinger and Quinn, 2010, Rutter et al., 2011, Schilling et al., 2013a, Klein et al., 2014], but continued advancements in computing, actuators, and energy storage technology mean that it may soon be practical to take neurorobots into the field. Neural controllers, specifically, have the advantage that they are inherently parallelizable, meaning that small multi-core computers may enable legged robots to roam while carrying both their own brains as well as enough energy to run for a practical amount of time. The work presented in this paper represents a thorough test of components of a larger control network, which in the future could enable the
CHAPTER 6. MANTISBOT IS A ROBOTIC MODEL OF VISUALLY GUIDED MOTION IN THE PRAYING MANTIS

robot to autonomously identify goals in the real world, and pursue or avoid them to do useful work.

6.6.2 Robot Controller as a Hypothesis of Nervous System Organization

Recently it was found that the reflex reversal in which cockroaches change the direction of FTr motion in stance phase could be elicited by stimulating parts of the central complex whose activity correspond to walking in a curved direction [Martin et al., 2015]. The model we present here is consistent with this observation. It is not clear from these data, however, what kinds of descending commands are used to change this reflex. In our model, we assumed that the simplest possible signal, the intended direction of travel, was sent to each thoracic ganglion, which uses the information to adjust local reflexes. It is possible, however, that to execute sophisticated adaptive motions, the insect may need an internal model of itself to plan trajectories [Schilling et al., 2012, Schilling et al., 2013a], in which case such simplistic descending commands would be insufficient. This might require that postural information be sent upward toward the brain, where one central model is manipulated, and the resulting motion is then commanded back down to the joints. Until more biological data is available, we prefer our approach because it agrees with observations showing that descending commands can alter local network function, rather than simply issuing new commands to each joint [Akay et al., 2007, Hellekes et al., 2011, Martin et al., 2015].

We also prefer our current approach because its animal-like network structure allows the study of network reuse. Related distributed walking controllers are capable of a variety of different behaviors [Schilling et al., 2013b]. Their structure, however, is described as “heterarchical,” in which different subnetworks are swapped in and out to perform different tasks. Indeed, insects use selective hyperpolarization in
their nervous systems to weaken or disable various pathways when relevant to do so, effectively disabling parts of their control systems [Laurent, 1993]. However, it is now known that many of the same networks are used to control both static posture and active locomotion [Buschmann et al., 2015]. Our model also does this, using the same descending commands, sensory signals, and processing networks to control both walking and standing. We believe that this approach can give us further insight into how insects' leg controllers are structured, and how they transition between locomotory states.

### 6.6.3 Use of Memory Network

Besides studies of saccadic eye movements [Seung et al., 2000b, Seung et al., 2000a, Koulakov et al., 2002], memory networks have many applications. Memory is especially important to high-level controllers, enabling navigation based on past motion [Haferlach et al., 2007, Varga and Ritzmann, 2016] and procedural memory for controlling intended locomotion [Schilling et al., 2013b, Toth et al., 2013a]. Due to its marginal stability, the memory network also acts as an ideal, non-leaky integrator. Searching, which is mediated by local interneurons [Berg et al., 2015], (Chapter 9), is an inherently incremental motion, requiring a change of joint kinematics over time [Dürre, 2001]. It is possible that local memory networks like that shown in Figure 6.4 are used to slowly increase the leg’s reach over time. We will experiment with these applications for MantisBot in the future (See Chapter 9).

### 6.6.4 Application of Leg Network to Other Legs

All of the results in this paper come from a single middle leg on MantisBot. But the goal, of course, is to have the entire robot mobile, and supporting its own weight. Prior modeling studies have also started this way, first modeling the motion of one leg, and then modifying the controller to control an entire “organism” [Ekeberg et al.,
2004, Szczecinski et al., 2014]. Our network structure in Figure 6.5 and tuning method in Figure 6.6 are both general. Preliminary results suggest that they can be quickly and easily applied to the hind and front legs, potentially enabling MantisBot to direct free walking toward visual goals.

Two additional considerations must be made before this step. First, how do the legs communicate with one another to coordinate stepping? Behavioral studies of stick insects and other arthropods (Cruse, 1990) reveal a number of coordination rules, which have been successfully applied to produce different gaits in walking robots [Espenschied et al., 1996, Lewinger and Quinn, 2010, Schilling et al., 2013a]. More recently, the neural basis of inter-leg coordination has begun to be investigated [Borgmann and Büschges, 2015], which points to a large role for sensory feedback in coordinating the legs [Zill et al., 2009]. However, not enough data are available to paint a complete picture, necessitating the use of behavioral rules.

6.6.5 CX Model in the Future

We did not model the central complex in any meaningful detail in this study. Our model was as simple as possible to predict the future position of prey, hold memory of its past location, and generate descending commands for the neck, prothorax joint, and legs. But the central complex [Pfeiffer and Homberg, 2014] also plays a role in directing locomotion [Ritzmann et al., 2012, Guo and Ritzmann, 2013, Martin et al., 2015], integrating sensory information [Wessnitzer and Webb, 2006], and processing visual information [Homberg, 2008]. The neural organization of our model is particularly amenable to integrating a CX model and its descending commands to future iterations leading to more mobile robotic solutions. As animal data becomes available, the robot’s satiety will be used to modulate the reliability of transitions between different behavioral states. For example, a sated mantis may completely ignore prey, or choose only to track it with its head. A hungrier mantis, however, may decide to
actively pursue the prey. In this paper, such transitions were artificially administered, but will be autonomous in the future.
Chapter 7

Template for the Neural Control of Directed Stepping Generalized to All Legs of MantisBot

This chapter was originally published as


Minor edits have been made, particularly in adding references to other parts of this thesis.

7.1 Abstract

We previously developed a neural controller for one leg of our six-legged robot, MantisBot, that could direct locomotion toward a goal by modulating leg-local reflexes with simple descending commands from a head sensor. In this work, we successfully apply an automated method to tune the control network for all three pairs of
legs of our hexapod robot MantisBot in only 90 seconds with a desktop computer. Each foot’s motion changes appropriately as the body’s intended direction of travel changes. In addition, several results from studies of walking insects are captured by this model. This paper both demonstrates the broad applicability of this control method for robots, and suggests neural mechanisms underlying observations from walking insects.

7.2 Introduction

Insects use their highly distributed nervous systems to produce coordinated, directed locomotion (for a review, see [Buschmann et al., 2015]). A walking robot with a control system like an insect’s nervous system could potentially benefit from both the relatively low computational power required to run a distributed controller and the insect-like agility it may afford.

Early attempts at insect-like walking controllers required that networks be highly abstracted [Espenschied et al., 1996, Cruse et al., 1998], but decades of biological research and increasing computational power have made biologically-based controllers more capable and practical [von Twickel et al., 2011, Dasgupta et al., 2015, Schilling et al., 2013a, Schilling et al., 2013b]. Building detailed neural controllers for legged robots may simultaneously produce more capable robots while testing and refining hypotheses about the neural control of locomotion.

Insects have distributed nervous systems, a legacy of their evolutionary history as segmented animals [Ruppert et al., 2004]. Each leg and joint has significant autonomy. How, then, do the legs and muscles of insects work together to produce coordinated, directed locomotion? Recent findings in the cockroach show that regions of the central complex (CX) in the insect brain show activity that strongly correlates to the animal’s speed and heading direction [Martin et al., 2015]. Additionally, stimulating the CX
modulates reflexes that control the motion of specific leg joints (i.e. the effect of femoral chordotonal organ stretch on the slow depressor motor neurons), meaning that body level control (i.e. the brain) can influence joint level control [Martin et al., 2015].

At the individual-leg level, each joint has a central pattern generator (CPG) that produces rhythms which, in the deaffarented animal, resemble patterns that would produce walking, searching, and other rhythmic motions [Ryckebusch and Laurent, 1993]. Experiments with deaffarented legs suggest that pattern generating networks that influence different leg segments are not centrally coupled, because different, uncoordinated patterns can be observed [Büsches et al., 1995]. Further experiments with reduced preparations have identified interjoint reflex pathways that act in an all-or-nothing fashion, suggesting that sensory feedback is affecting the phase of CPGs to coordinate the leg’s motion (joint rotation: [Hess and Büschges, 1997], [Hess and Büschges, 1999], [Bucher et al., 2003]; leg strain: [Akay et al., 2004], [Zill et al., 2004]). Together, these pattern generating networks and interjoint reflexes comprise a leg level controller. Walking is a cyclic motion, and the combination of these influences cause transitions between stance phase, when the legs support and propel the body, and swing phase, when the foot is moved through the air to the start of stance. When the leg joints rotate to a particular pre-stance-phase position, called the anterior extreme position (AEP), the foot is lowered to the ground. When ground contact is detected, the remaining joints begin rotating to support and propel the body. When the joints rotate to a particular pre-swing-phase position, called the posterior extreme position (PEP), the foot is raised off the ground. The decrease in force on the leg causes the joints to again rotate toward the AEP. Many prior models have verified this basic progression of sensory information driving the next stage of the step [Cruse et al., 1998, Ekeberg et al., 2004, Knops et al., 2012, Szczecinski et al., 2014].
When an insect walks along a curved path, these reflexes [Hellekes et al., 2011, Martin et al., 2015] and the resulting motor activity [Mu and Ritzmann, 2005, Gruhn et al., 2016] and motion [Dürr and Ebeling, 2005, Gruhn et al., 2006, Szczecinski et al., 2014] change with respect to forward walking. Each leg undergoes specific alterations, due to its location on the body. The hind legs hardly change direction, although they may shorten or lengthen their steps on the inside or outside of a turn, respectively [Dürr and Ebeling, 2005, Mu and Ritzmann, 2005]. The inside middle leg pulls sideways; in cockroaches, the femur-tibia (FTi) joint switches from extension to flexion in stance phase [Mu and Ritzmann, 2005, Szczecinski et al., 2014], a reversal of a well-studied interjoint reflex. The outside middle leg, by contrast, shows little change in its motion, except lengthening its stride [Dürr and Ebeling, 2005, Mu and Ritzmann, 2005]. The front legs in stick insects and cockroaches are very mobile, and both species exploit this mobility to make large, sweeping motions while the animal turns [Dürr and Ebeling, 2005, Szczecinski et al., 2014]. Some mechanisms, such as reversing the influence of interjoint reflexes [Akay et al., 2004, Martin et al., 2015] and asymmetric modulations of CPG activity [Gruhn et al., 2016] have been suggested as possible mechanisms driving these changes. However, it is likely that the structure of the underlying neural circuits does not drastically change. Can we develop a network model that can replicate these changes without major structural adjustments, and applies equally to every leg, despite their different roles and location on the body?

In previous work, we developed such a controller for one leg of our 29 degree of freedom hexapod robot, MantisBot (Chapter 6, [Szczecinski et al., 2017a]). Before MantisBot can walk freely, it must be able to find the AEP and PEP angles for all of the legs (i.e. 8-10 angles per leg), based on the direction of walking. This paper presents an extension of this controller, in which we apply a network with the same connectivity, but different tuning, to each joint of each leg. Each leg’s joints are coordinated into directed stepping by the same set of sensory signals and inter-joint
pathways. The body control network directs each leg to move in the proper direction as if the body were moving at a particular speed and path curvature. Section 7.3 describes the control network implementation, structure, and tuning. Section 7.4 presents data from MantisBot taking steps in different directions with different legs, and compares these data to the intended motion. Section 7.5 discusses this work as a model of insect locomotion, its use as a robot controller, and future directions for this research.

7.3 Methods

7.3.1 Control Network Structure

MantisBot’s neural controller was assembled in AnimatLab [Cofer et al., 2010]. The AnimatLab Robotics Toolkit (ART) enables real-time neural control of the robot (Chapter 5, [Szczecinski et al., 2015a]). The control network is made up of Hodgkin-Huxley compartments with no spiking ion channels (see 7.6 for a full description of the model). Nodes are added to the network that convert neural activity to servomotor commands, or convert sensor values to neural activity. The ART computes neural activity on a laptop computer (Intel Core-i5-5300U 2.3 GHz), and updates these nodes with signals from the robot over a serial connection.

MantisBot is actuated by 29 Dynamixel smart servomotors (Robotis, Seoul, South Korea), arranged to give the robot most of the degrees of freedom seen in mantises. Our dissections show that the thorax-coxa (ThC) joint is highly mobile, rotating along all three axes. MantisBot’s front coxae have all three degrees of freedom, but the remaining legs have only two, rotating about the lateral axis of the thorax (ThC1), and the long axis of the coxa (ThC3) (for graphical representation see Chapter 5 [Szczecinski et al., 2015a]). The ThC joints rotate the plane of the leg, which also contains coxa-trochanter (CTr) and femur-tibia (FTi) joints. Each servomotor receives posi-
tion and velocity commands from the neural controller, and reports its position and current draw (i.e. torque). Each leg has a strain gage on the trochanterofemur to measure ground reaction force like an insect’s campaniform sensilla [Zill et al., 2004].

Network structure, shown in figure 7.1A, is based on findings from insect neurobiology. The primary challenge is producing coordinated, directed leg motion from a highly distributed control system. To regulate the amplitude and direction of joint rotation based on the body’s heading, connections from the body level to the joint level are shown in figure 7.1A as dashed red pathways [Martin et al., 2015]. These are critical to producing coordinated locomotion, and designing them is the focus of section 7.3.2.

Sensory influences from throughout the leg, such as joint rotation and leg strain, coordinate all of the joints in the leg into a cohesive stepping motion. Leg level influences known to exist include FTi motion resetting the coxa-trochanter (CTr) rhythm [Hess and Büschges, 1999, Bucher et al., 2003, blue dashed lines in figure 7.1A], leg load resetting thorax-coxa (ThC) rhythm [Akay et al., 2004, violet solid lines], load resetting FTi rhythm [Akay et al., 2001, green dotted lines], and leg load reinforcing CTr stance phase motion [Borgmann et al., 2011, orange fine lines]. Figure 7.1A also shows hypothetical connections from the ThC joints to the leg level. Such an influence is not known to exist in insects, but without them the leg could not coordinate “outside” turning, in which the FTi joint motion is very small. Such an influence is similar to using all joint angles to determine if the PEP has been reached, similar to “PEP net” in Walknet [Cruse et al., 1998]. As the animal walks in different directions, the leg’s motion [Szczecinski et al., 2014], and the strength and sign of these leg level influences change [Martin et al., 2015, Hellekes et al., 2011]. So, to walk in a specific direction, each joint must not only change its input, as discussed in the previous paragraph, but must also expect different sensory input, enabling it to signal when it has reached the AEP or PEP angle.
Figure 7.1B shows the joint level network in detail to explain how each joint’s output and expected sensory feedback is modulated by descending commands. The body translation and body rotation are encoded in signals from the CX. These are mapped to the joint’s PEP and AEP angles (figure 7.1B, synapses shaded in gray). The PEP and AEP angles are compared to determine which way the joint must move in stance phase. If the PEP angle is less than the AEP angle, then the joint must flex in stance phase. Neurons that are active in this case are shaded gray in figure 7.1B. Incoming and outgoing connections are color coded as in figure 7.1A, with two additions: the rate of strain in the leg is critical for proper timing during locomotion [Ridgel et al., 1999, orange neurons], and each joint possesses its own CPG [Ryckebusch and Laurent, 1993, Büschges et al., 1995, yellow neurons].

Three features of the network must be modulated to adjust motion. First, the active ‘Flx. PEP’ and ‘Ext. AEP’ neurons disinhibit the ‘Flx. Load’ and ‘Ext. Unload’ neurons, which route information about leg strain to the appropriate half of the CPG (in this case, to flex in stance phase). Second, the ‘Flx. PEP’ and ‘Ext. PEP’ neurons establish the amplitude of flexion in stance phase and extension in swing phase, respectively, by synapsing onto the MNs. Finally, the expected sensory feedback must change, which is accomplished by inhibiting irrelevant comparisons between sensory information and the PEP and AEP. If the joint is flexing (i.e. joint angle decreases) in stance phase, then the joint has reached the PEP when the joint angle is less than the PEP, so the ‘Angle > PEP’ neuron is inhibited. If the joint’s PEP and AEP are too similar, then all four comparison neurons are inhibited, because otherwise two would always be active. This is why it was necessary to include information from the ThC joints in the leg level controller; if moving in a particular direction required no FTi motion, then the leg could not use sensory information to transition between stance and swing.
Figure 7.1: Schematic of one leg’s neural controller. A. Each network has three levels: the body’s heading, the leg’s coordinating pathways, and the joints’ motion. Pathways in long-dashed red lines are based on [Martin et al., 2015]. Those in dashed blue lines are based on [Hess and Büschges, 1999]. Those in dotted green lines are based on [Akay et al., 2001]. Those in solid violet lines are based on [Akay et al., 2004]. Those in fine orange lines are based on [Borgmann et al., 2011]. Those in solid gray lines are hypothesized, and parallel the other influences that are known to exist. B. Each
CHAPTER 7. TEMPLATE FOR THE NEURAL CONTROL OF DIRECTED STEPPING GENERALIZED TO ALL LEGS OF MANTISBOT

Figure 7.1: (contd.) joint’s controller decides whether to flex or extend in stance phase, and directs information about strain to the appropriate half of the CPG. The pathways active when the joint must flex in stance phase are shaded in gray, while those active when the joint must extend have dotted boundaries. Incoming and outgoing pathways are coded as in (A). Additionally, the joint’s CPG is shaded yellow [Büschges et al., 1995], and pathways that signal increasing and decreasing load, which coordinate the CPGs, are shaded orange [Ridgel et al., 1999].

7.3.2 Body-level Controller Tuning

How is the neural system tuned to produce coordinated locomotion? When each half-center of the CPG is active, it drives its joint toward the PEP or AEP. The PEP and AEP, however, change as the robot walks in different directions, so the four highlighted synapses at the top of figure 7.1B must be tuned to map the body’s motion to each joint’s PEP and AEP. This requires first finding the motion of the feet that moves the body as commanded, and then using each foot’s motion to compute the PEP and AEP of each joint.

Figure 7.2 illustrates how foot paths are found. Figure 7.2A shows a top view of a simplified schematic of the robot, with a global reference frame with origin \( \mathbf{0} \) and unit vectors \( \mathbf{i}_0 \) pointing forward and \( \mathbf{k}_0 \) pointing to the right. Unit vector \( \mathbf{j}_0 \) is positive out of the page, and represents the robot’s height above the ground. The position of the left middle foot relative to the origin is defined as \( \mathbf{p} = p_x \cdot \mathbf{i}_0 - p_z \cdot \mathbf{k}_0 \).

Figure 7.2B shows the body rotated relative to the initial position, for instance, if the robot were to pivot in place about the origin of the \( \mathbf{0} \) frame. We call this angle \( \psi \). It represents the change in the robot’s heading caused by taking one step (i.e. the curvature of the presumed walking path). We define a second coordinate system \( \mathbf{1} \), rotated angle \( \psi \) from the \( \mathbf{0} \) frame about \( \mathbf{j}_0 \).

Figure 7.2C shows the body translated distance \( r \) along the \( \mathbf{i}_1 \) axis. The new location of the foot is defined as \( \mathbf{q} = p_x \cdot \mathbf{i}_1 - p_z \cdot \mathbf{k}_1 + r \cdot \mathbf{i}_1 \). The vector that describes the foot motion in stance phase is \( \mathbf{v} = \mathbf{p} - \mathbf{q} \), computed in the \( \mathbf{1} \) frame. This requires that we transform \( \mathbf{p} \) into the \( \mathbf{1} \) frame by premultiplying by the transformation matrix:
CHAPTER 7. TEMPLATE FOR THE NEURAL CONTROL OF DIRECTED STEPPING GENERALIZED TO ALL LEGS OF MANTISBOT

\[ R_y(\psi) = \begin{bmatrix} \cos(\psi) & 0 & \sin(\psi) \\ 0 & 1 & 0 \\ -\sin(\psi) & 0 & \cos(\psi) \end{bmatrix}, \quad (7.1) \]

which yields

\[ \mathbf{v} = (R_y(\psi) - I) \cdot \mathbf{p} + r \cdot \mathbf{i}_1. \quad (7.2) \]

One can repeat this calculation with multiple ordered pairs \((r, \psi)\) to generate a family of \(\mathbf{v}\)s. Such \(\mathbf{v}\)s corresponding to \(r = 15\) cm and \(\psi \in [-.35, .35]\) radian \([-20, 20]\) degrees) are shown in figure 7.2D. The arrows point in the direction of motion in stance phase. Each leg shows distinct motion. The hind leg predominantly pushes rearward no matter the value of \(\psi\), as observed in cockroaches [Mu and Ritzmann, 2005]. The middle leg makes longer strides on the outside of the turn, but shorter, inward-directed strides on the inside of the turn. The front leg makes long, sideways strides whether on the inside or outside of the turn. Mantises also use the prothorax joint to increase the front legs’ range of motion.

How are these \(\mathbf{v}\) values used to design the neural system to control MantisBot’s joints? An inverse kinematics problem can be solved to find the joint angles that produce the intended foot locations (that is, those points along \(\mathbf{v}\)). For an open-chain manipulator, such as a leg, the velocity of the endpoint can be expressed as a function of the joint angles and their velocity:

\[ \mathbf{V} = J(\theta) \cdot \dot{\theta}, \quad (7.3) \]

where \(\mathbf{V} \in \mathbb{R}^{3 \times 1}\) is the velocity of the foot in the reference frame of the base (in our case, the body), \(\theta \in \mathbb{R}^{n \times 1}\) is a vector of joint angles for an \(n\) degree of freedom manipulator, and \(J \in \mathbb{R}^{3 \times n}\) is the manipulator Jacobian, which uses kinematic information about the manipulator to map from the joint velocities to the foot velocity [Murray...
Figure 7.2: Illustration of how intended foot paths are calculated as a function of body rotation, \( \psi \), and body translation, \( r \). A. A top view of a simple diagram of the robot. Forward is to the right. B. The body rotates about point \( O \). A new reference frame is defined such that \( \mathbf{i}_1 \) points forward in the body’s frame of reference. C. The robot translates distance \( r \) in the \( \mathbf{i}_1 \) direction. The difference between the new foot placement and original foot placement is \( \mathbf{v} \). D. An isometric view of a schematic of the robot. Each leg is color coded. The axes from A-C are labeled. E. Foot paths for each leg of the robot. The heading angle \( \psi \) varies in increments of 10°. The translation distance \( r \) is 15 cm.
et al., 1994]. Equation (7.3) can be discretized in time and rewritten purely in terms of joint angles \( \theta \), as long as a starting vector is given:

\[
\theta_i = \theta_{i-1} + J^+ (\theta_{i-1}) \cdot \Delta v,
\]

where the pseudo-inverse of \( J \) is \( J^+ = J^T \cdot (J \cdot J^T)^{-1} \). This amounts to solving a least squares problem, which moves the foot in the intended direction while minimizing \( \dot{\theta} \). More information on this method can be found in Chapter 6 [Szczecinski et al., 2017a].

Once the joint angles for the PEP and AEP are known for every joint, for every \([r, \psi]\) pair, then a network is tuned for each joint that maps between these values. In this way, the descending command from higher control centers (i.e. the central complex [Martin et al., 2015]) may be simple, encoding only the body’s motion; each leg, and each joint, decode this information to control locomotion.

Most of the resulting \([r, \psi] \rightarrow \theta\) maps are nearly planar (for an example, see figure 7.4). Rather than using naive optimization to design a network that encodes this relationship, we design such a network directly using functional subnetworks and identified constraints between parameter values (Chapter 4, [Szczecinski et al., 2017b]). This is accomplished in two steps: find the contribution of \( r \) and \( \psi \) to \( \theta_{PEP} \), and then tune a network to match. First, we assume that \( \theta_{PEP} = k_r \cdot r + k_\psi \cdot \psi \). Using the data generated by (7.2) and (7.4), we find \( k_r \) and \( k_\psi \) by solving the least squares problem:

\[
\begin{bmatrix}
  r_1 & \psi_1 \\
  \vdots & \vdots \\
  r_m & \psi_m
\end{bmatrix}
\begin{bmatrix}
  k_r \\
  k_\psi
\end{bmatrix} =
\begin{bmatrix}
  \theta_{PEP,1} \\
  \vdots \\
  \theta_{PEP,m}
\end{bmatrix},
\]

in which \( m \) is the number of \([r, \psi]\) ordered pairs tested with (7.2). In this work, \( m = 45 \), although values of \( m \) as low as 15 gave negligibly different results.
Once \( k_r \) and \( k_\psi \) are known, then we use the technique presented in Chapter 4 [Szczecinski et al., 2017b] to directly assemble a network that adds \( r \) and \( \psi \) with the proper proportions. The simplicity of this approach enables us to tune every joint network in the robot’s controller in only 90 seconds total. This process is automated in our Matlab toolbox, kinematicOrganism. Once this process is complete, the controller can direct locomotion in any direction as soon as it is commanded.

7.3.3 Experimental Setup

These networks were tested by running experiments with MantisBot. For these tests, the robot’s thorax was fixed in place, as shown in figure 7.3. Each leg performed five, 60 second long bouts of stepping on the platform shown in the picture (the number of steps taken is shown in figures 7.7, 7.8, and 7.9). The platform’s height was set to that used in the analysis in the previous section. For each stepping bout, the intended body translation was 15 cm, and the intended body rotation was either -20, -10, 0, 10, or 20 degrees. The ideal footpaths for these five scenarios are illustrated in figure 7.2E.

Please note that these trials did not simply “play back” predetermined stepping kinematics. The method described above was used to tune the connections from the CPGs to their respective MNs, but stepping only occurred because of the sensory coupling produced by each leg’s strain gage and the motion of all the joints in the leg. Thus, as the leg steps on the platform, it supports some of the robot’s weight to strain the leg. This force also causes some friction between the foot and the platform, although the platform has a Teflon top to enable smooth sliding.

Rather than using cameras and a visual tracking system, the resulting robot servomotor rotations were recorded, run through a kinematic model of each leg (using the kinematicOrganism toolbox), and used to determine actual foot trajectories for comparison to the intended motion. Joint rotation and strain gage data were saved.
Figure 7.3: MantisBot as constrained for this study. The thorax was bolted to an aluminum frame. Only one leg was tested at a time. As each leg was tested, an adjustable-height platform was placed under the leg to enable ground contact. The green stripe on the platform is 15 cm long, for scale.

in Matlab (Mathworks, Natick, MA), and individual steps were determined by when the leg strain surpassed a threshold that implied ground contact. The joint angles, stance direction of the foot, and stride length of the leg were recorded and plotted as shown in the figures in the next section.

7.4 Results

The presented method was evaluated in three ways. First, the accuracy of the body to joint maps was quantified. Second, the stance angle and stride length from the robot trials were compared to the intended values. Finally, the joint kinematics in the stance phase were examined to confirm that the direction of interjoint reflexes changed where predicted.
7.4.1 Posture Network Tuning

Figures 7.4, 7.5, and 7.6 show the PEP for each joint as a function of the body’s translation and rotation. Each point on the \((r, \psi)\) plane represents the stride length and resulting body rotation of each step. The first row of each shows contour plots of the PEP as calculated by the inverse kinematics method in Section 7.3.2. One can see that especially for the proximal ThC joints, the maps are nearly planar. The second row shows the output of the network that encodes the data from the first row (highlighted synapses at the top of figure 7.1B). The third row shows the difference between the first and second rows. Even the networks for the distal CTr and FTi joints, whose PEPs vary nonlinearly with the body’s transformation, never produce an error more than 10\(^\circ\).

Figures 7.4, 7.5, and 7.6 also predict what stride lengths and body heading directions will require interjoint reflexes to change sign. In the second row of each figure, regions in which joint angles are less than the static posture are shaded in gray. Therefore, if the robot transitions from one heading pair \((r, \psi)\) to another, it may need to switch a particular joint from flexing in stance phase to extending, or vice versa. In this paper, each leg walked with \(p = 15\) cm as \(\psi\) was varied between -0.35 radian (-20 degrees) and 0.35 radian (20 degrees). Inspecting the right-most boundary of the second row in figures 7.4, 7.5, and 7.6, we may predict that as the robot transitions from turning left to turning right, load feedback will cause the opposite motion in the front leg’s ThC2 and ThC3 joints, and the middle leg’s ThC3 and FTi joints, and the hind leg’s FTi joint.

7.4.2 Robot Motion

Figures 7.7, 7.8, and 7.9 present the ideal stance direction and stride length as illustrated in figure 7.2, alongside the data collected from trials with the robot. In 50% of scenarios (15/30; five walking directions, three legs, two metrics), the intended
Figure 7.4: A contour of each of the front leg’s joints’ PEP is plotted as the body translation (x axis) and rotation (y axis) are varied. Each joint in the leg is in one column, with proximal joints to the left. The top row shows the exact PEP as calculated by Equation 7.4 for every joint. The second row shows the output of the network tuned by Equation 7.5. Combinations of $r$ and $\psi$ that require the joint to flex in stance phase are shaded in gray. The bottom row shows the difference between the first and second rows, in degrees for clarity.
Figure 7.5: A contour of each of the middle leg’s joints’ PEP is plotted as the body translation (x axis) and rotation (y axis) are varied. For an explanation of the layout, see the caption of figure 7.4.
CHAPTER 7. TEMPLATE FOR THE NEURAL CONTROL OF DIRECTED STEPPING GENERALIZED TO ALL LEGS OF MANTISBOT

Figure 7.6: A contour of each of the hind leg’s joints’ PEP is plotted as the body translation (x axis) and rotation (y axis) are varied. For an explanation of the layout, see the caption of figure 7.4.
stance direction or stride length is within two standard deviations of the mean of the robot data. The most common cause of error is a stride length that is too short, which occurs in 53.3% (8/15) scenarios. In addition, none of the stance directions are exactly 0 when that is the intended direction (0/3).

Despite some deviation, the same trends are observed between the intended and actual data in all the legs. Figure 7.7 shows that in the front leg, the stance direction decreases monotonically as the body rotation angle increases, as intended. The front leg’s stride length has high variance when turning at extreme angles, but the stride length increases with the body rotation for most values.

Figure 7.8 shows that in the middle leg, the robot captures the trends in the intended data. The stance direction is much more sensitive to inside turning body rotations than outside turning body rotations. This is due to the leg’s position on the body, and how the vectors $p$ and $q$ in figure 7.2 sum. This is consistent with the findings of studies of stick insects [Dürr and Ebeling, 2005, Gruhn et al., 2006] and cockroaches [Mu and Ritzmann, 2005] that the outside leg’s motion undergoes less extreme modulation than the inside leg’s motion. The stride length also changes with body rotation, with the outside leg increasing its stride and the inside leg decreasing its stride. This is also consistent with studies that show that the middle inside leg of a turn tends to take shorter steps than when walking forward [Dürr and Ebeling, 2005].

Figure 7.9 shows results from the hind leg alongside the intended motion. The stance direction matches the intended data well, although the innermost turns yielded an exceptionally large variance. This is because for this body rotation, the foot hardly moved at all, so any small errors in foot placement resulted in large fluctuations in the stance direction. The stride length varies as expected, but is about 5 cm shorter in every case.

Further insight into the robot’s performance can be gained by examining how each
CHAPTER 7. TEMPLATE FOR THE NEURAL CONTROL OF DIRECTED STEPPING GENERALIZED TO ALL LEGS OF MANTISBOT

Front Leg Foot Motion Data*

Figure 7.7: Bar graphs of the front leg’s foot motion are plotted for comparison to the intended, that is, ideal case. The left column plots the intended data, while the right column plots data taken from the robot. The top row plots the stance direction, as illustrated in figure 7.2, versus the body rotation, and the bottom row plots the stride length versus the body rotation. The number of steps collected from the robot is indicated on the robot’s stance direction plot.
Figure 7.8: Bar graphs of the middle leg’s foot motion are plotted for comparison to the intended, that is, ideal case. For an explanation of the layout, see the caption of figure 7.7.
Hind Leg Foot Motion Data*

Figure 7.9: Bar graphs of the hind leg’s foot motion are plotted for comparison to the intended, that is, ideal case. For an explanation of the layout, see the caption of figure 7.7.
joint moved in stance phase. Figures 7.10, 7.11, and 7.12 show how each joint of each leg moves in stance phase while the body heads in each direction. Plotting the data this way reveals the body direction-dependent changes to the joint kinematics.

For instance, figure 7.10 shows that changes in the direction of interjoint reflexes that were predicted in section 7.4.1 indeed occur: as the body heading varies, the ThC2 and ThC3 reverse their direction of travel in stance phase (i.e. the sign of the slope changes). This is due to the way strain information is routed in the network in section 7.3.1. Figure 7.11 shows that the middle leg’s ThC3 and FTi joint controllers also reverse the sign of load feedback, as predicted.

Figure 7.12 shows that for inside turns, the hind leg’s joints hardly rotate in stance phase. This is consistent with the observation that the inside hind leg of turning insects rarely enter swing phase, and may serve as a pivot point for rotation [Dürr and Ebeling, 2005].

7.5 Discussion

The presented work is an extension of our previous neural controller for directed stepping (Chapter 6, [Szczecinski et al., 2017a]). The same neural architecture was applied to each of MantisBot’s kinematically unique pair of legs. The same tuning method gave rise to unique motions for each leg, many of which mimic the foot and joint motions seen in insects. In addition, the tuning method is fast, requiring only 30 seconds to tune each leg, and completely automated, meaning that tuning the unique legs of MantisBot requires no additional work from the engineer.

7.5.1 Speed of the Method

The tuning method we present is deterministic and rapid, taking only 90 seconds to tune the entire robot’s controller with a i7-4770K desktop computer. These traits
Figure 7.10: Plots of each joint’s motion in stance phase, for the front leg. The x axis is normalized to the stance phase, with the time of foot touchdown as 0 and the time of foot liftoff as 0.5. The y axis is between -1 and 1 radian. The scale is the same on every graph to facilitate comparison. Each column contains data from one joint of the leg, with distal joints to the right. Each row shows data from a particular body rotation, with outside turns towards the bottom. The mean joint motion is plotted as a line, and one standard deviation is shaded above and below the mean. Changes in the sign of interjoint reflexes that occur when the body heading angle $\psi$ changes are evident in the change in sign of the slope of lines within the same column. The words “reflex direction reverses” are highlighted between body heading angles where the sign of leg strain feedback reversed. The changes in joint amplitude, including when to reverse the direction of interjoint reflexes, are encoded in the network output shown in figure 7.4.
Figure 7.11: Plots of each joint’s motion in stance phase, for the middle leg. The x axis is normalized to the stance phase, with the time of foot touchdown as 0 and the time of foot liftoff as 0.5. The y axis is between -1 and 1 radian. The scale is the same on every graph to facilitate comparison. The changes in joint amplitude, including when to reverse the direction of interjoint reflexes, are encoded in the network output shown in figure 7.5.
Figure 7.12: Plots of each joint’s motion in stance phase, for the hind leg. The x axis is normalized to the stance phase, with the time of foot touchdown as 0 and the time of foot liftoff as 0.5. The y axis is between -1 and 1 radian. The scale is the same on every graph to facilitate comparison. The changes in joint amplitude, including when to reverse the direction of interjoint reflexes, are encoded in the network output shown in figure 7.6.
are useful because they make the method broadly applicable to different legs and robots, and accessible to other engineers or modelers who wish to use them. Historically, optimization methods such as genetic algorithms have been used to tune neural circuits [Beer and Gallagher, 1992]. These methods, while certainly useful for extracting structure from nebulous data, are less ideal for problems that do possess structure, such as producing directed locomotion. Adding structure to these methods is usually helpful anyway [Haferlach et al., 2007, Izquierdo and Beer, 2013, Agmon and Beer, 2013], so we prefer to directly tune models of networks if possible (Chapter 3, [Szczecinski et al., 2017b]; Chapter 4, [Szczecinski et al., 2017c]). Such a direct approach also eliminates much of the “art” of parameter selection, potentially making the approach more applicable to new and different systems.

### 7.5.2 Eliminating Sources of Error

The robot’s foot paths did not precisely match the data generated by the kinematic model. The controller structure and tuning method was first developed for the middle leg (Chapter 6, [Szczecinski et al., 2017a]). If the middle leg were very accurate but the other legs performed poorly, one could assume that the design method does not generalize. This, however, was not the case; the front leg actually showed the fewest inaccuracies (4/10 in figure 7.7), and the same types of errors occurred in all of the legs (for instance, 0/3 legs had a stance angle of 0 when moving forward). Therefore, we must look for factors that were present in all legs, and develop a solution to apply to all of the legs.

The most likely cause of error is the limited accuracy of the maps between body transformation and the leg joints (figures 7.4, 7.5, and 7.6). In this work we assumed that these maps are planar (i.e. $\theta_{PEP} = k_r \cdot r + k_\psi \cdot \psi + c$), but in the future we could use more sophisticated maps. For instance, adding a subnetwork (Chapter 4, [Szczecinski et al., 2017c]) that multiplies the body’s translation and rotation (i.e. $\theta_{PEP} = k_r \cdot r \cdot k_\psi \cdot \psi + c$).
(θ_{PEP} = k_r \cdot r \cdot \psi + k_r \cdot r + k_\psi \cdot \psi + c) could account for some of the observed nonlinearities. This could be easily implemented by adding a \( r_i \cdot \psi_i \) column to equation 7.5, and solving for \( k_{r\psi} \). Alternatively, we could implement these maps as range-fractionalized networks, with discrete \((r, \psi)\) ordered pairs precisely coding for the corresponding \( \theta_{PEP} \). In the immediate future, we will implement the method that is the most accurate and efficient to compute. However, future studies on insects may reveal how the descending commands that control walking direction are actually encoded in the thoracic ganglia, requiring that we change our model.

Another way to improve these maps’ accuracy may be to include known reflexes that adjust motor output amplitude. Currently, each map is only a function of the body heading, that is, \( \theta_{PEP} = k_r \cdot r + k_\psi \cdot \psi + c \). It is known in insects, however, that the motion of some joints affect the motor neuron activation of others. For example, extending the FTi joint in insects increases the firing frequency of the slow depressor of the CTr joint [Hess and Büschges, 1997]. This serves as a type of feed-forward height regulation system, in which the CTr joint lowers the foot if FTi motion raises it, and vice versa. We could implement such a reflex by making the CTr joint’s PEP a function of the body heading and the instantaneous rotation of the other leg joints, that is, \( \theta_{PEP} = k_r \cdot r + k_\psi \cdot \psi + \sum_{i=1}^{m} k_i \cdot \theta_i + c \). The kinematics of the leg would inform the \( k_i \) values to keep the foot height constant throughout the step. Preliminary results suggest that this mechanism does indeed improve performance, enabling the CTr to raise and lower throughout stance phase to account for the motion of the other leg joints, and then completely raise the leg at the end of stance phase.

### 7.5.3 Making Networks Adaptable

This controller shows some flexibility by reversing the direction of interjoint reflexes as the robot’s stepping direction changes. However, it still lacks the ability to make long term changes to its behavior because of external conditions. A primary example of this
is that the CPGs oscillate at a single, slow frequency, and sensory information drives the transition from flexion to extension, and vice versa. Animals, however, entrain their nervous systems to incoming sensory information, learning to anticipate future events, such as ground contact at the beginning of stance phase [Schmitz et al., 2015]. Our recent research has shown that such adaptation can be built into the system by using a joint’s velocity feedback to reinforce the rhythm of its CPG. When a joint begins to rotate in one direction, whether actively or passively, the CPG switches to regulate the velocity of the ongoing motion. When the joint reaches its PEP angle, the CPG dynamics cause it to immediately switch to swing motion, speeding up the transition from stance and swing phase (Chapter 8, [Szczecinski and Quinn, 2017b]). Additionally, we have incorporated a searching reflex to ensure proper transition from swing to stance phase (Chapter 9, [Szczecinski and Quinn, 2017a]). The structure of the network is based on intracellular recordings from stick insects [Berg et al., 2015]. Searching is automatically activated if MantisBot depresses the leg but cannot enter stance phase, representing an extension of swing phase [Dürr, 2001], enabling it to adapt to changing substrate heights.

Another form of adaptation is modifying the amplitude of motion to adapt to terrain. Such mechanisms are locally mediated; a single local interneuron activates searching in the stick insect [Berg et al., 2015], and a single cockroach ganglion can learn to adjust motion to avoid painful stimuli [Eisenstein, 1972]. The leaky dynamics of our neuron model, and neurons in general, makes modeling these behaviors non-trivial; information about how far to depress the leg while searching, or the posture that elicits a painful stimulus, are quickly forgotten by the network. Therefore, for our controller to actively adapt to the environment, it must have memory (i.e. non-leaky integrator) units whose activity affect motion (e.g. how far to protract the leg each cycle of searching), and whose activity can be changed incrementally (e.g. protracting further each cycle as seen in stick insects, see Figure 2 in [Dürr, 2001]). Such
adaptation is crucial for locomotion in irregular or cluttered environments; stepping into holes or onto impediments require that the nervous system actively adjust motor output to maintain progress and stability. Adding memory units to the local networks enable long-term adjustments of the maps between body heading and joint motion, producing truly adaptive stepping (Chapter 9, [Szczecinski and Quinn, 2017a]).

7.5.4 MantisBot as a Model of Biological Systems

MantisBot’s controller is a hypothetical model that shows how interjoint reflexes may be modified to produce directed stepping. In [Gruhn et al., 2016], two possible mechanisms are proposed for the changing effect of leg load as the animal walks in different directions. The first is presynaptic inhibition of load pathways onto the CPG, and the second is a reweighting of premotor pathways. Our model uses both of these mechanisms to reproduce several results from insects. For instance, in [Gruhn et al., 2016], the inside front leg’s FTi joint’s response to load is unpredictable. In the context of our model, this could represent the insect heading in a direction on the boundary of the gray shaded regions in figure 7.4, in which the joint should not move in stance phase. But as the animal’s intended direction varies over time, the joint controller jumps between flexing or extending in stance phase, and thus the response is variable. For the proportions of our robot, a turn of about -0.4 radian would cause such a phenomenon in the front FTi joint. Of course, the curvature at which this happens would be different in the animal, which has different body and leg proportions.

In animals, legs may enter stance or swing at different rates because of load distribution. Specifically, the hind leg on the inside of a turn rarely enters swing, and functions much like a pivot point [Dürr and Ebeling, 2005]. Our inside hind leg functions much in the same way, only entering swing phase because the CPGs spontaneously transitioned, not because a sensory signal encouraged it to. The future
inclusion of inter-leg influences that either keep a leg supporting weight or prevent CPG transitions due to the placement of other legs [Cruse, 1990] may produce locomotion that is even more insect-like.

This leg controller may perform quite differently once MantisBot walks unsupported on all six legs. In this paper, MantisBot functions as a reduced preparation, with all but one leg amputated at a time. It is known that the range of motion of reduced preparations is reduced [Gruhn et al., 2006], in spite of showing similar changes to joint kinematics based on stepping direction. This scenario most accurately represents how we tested MantisBot in this paper.

Removing one or two legs from an otherwise intact animal immediately changes walking, generally reducing speed, increasing stride length, and making posture more sprawled, in both cockroaches [Hughes, 1957] and fruit flies [Wosnitza et al., 2013]. This may be due to the animal attempting to keep the same number of legs on the ground at the same time, which reduces the fraction of the weight that each leg must support in stance phase [Foth and Graham, 1983]. Our recent work focuses on how such speed changes may be controlled (Chapter 8, [Szczecinski and Quinn, 2017b]), enabling us to explore how a neural controller may change speed and stride length to cope with missing legs. However, in its current state, our network cannot make postural changes, such as sprawling the posture, without being retuned. In the future, we will explore possible intra- and interleg mechanisms that may lead to such rapid and effective adaptation.

MantisBot can use any leg to step in different directions without any inputs except the body’s intended heading. The system autonomously modulates interjoint reflexes as seen in animals [Akay et al., 2004, Martin et al., 2015], and adjusts motor output in animal-like ways. To truly test this controller as a model of the thoracic ganglia, it must be applied to animats with different proportions and postures. We have recently applied this methodology to a simulation of a cockroach [Szczecinski et al.,
2014], which has a much more sprawled posture than MantisBot. In spite of this difference, the animat is capable of walking at normal speeds (20 cm/s) in a straight line. Future work will apply this method to even more diverse morphologies and identify areas for improvement.

7.6 Appendix: Neural Model Details

The neural model used in this work is a Hodgkin-Huxley compartment with no spiking ion channels [Cofer et al., 2010]. Each compartment has a leak current, $I_{\text{leak}}$; $n$ synaptic currents, $\sum_{j=1}^{n} I_{j,\text{syn}}$; an applied current, $I_{\text{app}}$; and a nonlinear persistent sodium current, $I_{\text{NaP}}$, which simulates bursting dynamics without the presence of spikes [Daun-Gruhn et al., 2009]. Each compartment has three states: the membrane voltage, $V$; the sodium channel activation, $m$; and the sodium channel deactivation, $h$. Each compartment has the dynamics:

$$C \cdot \frac{dV}{dt} = G \cdot (E - V) + \sum_{i=1}^{n} g_{\text{syn}}^{i} \cdot (E_{\text{syn}}^{i} - V) + I_{\text{app}} + G_{\text{NaP}} \cdot m \cdot h \cdot (E_{\text{NaP}} - V)$$  \hspace{1cm} (7.6)

$$\frac{dm}{dt} = \frac{m_{\infty}(V) - m}{\tau_{m}(V)}$$ \hspace{1cm} (7.8)

$$\frac{dh}{dt} = \frac{h_{\infty}(V) - h}{\tau_{h}(V)}.$$ \hspace{1cm} (7.9)

Capital letters represent constant parameters, while lower-case letters represent values that change with time. $C$, $G$, and $E$ represent capacitance, conductance, and reversal potential, respectively. Terms without subscripts are parameters of the neuron’s membrane. The subscripts $\text{syn}$ and $\text{Na}$ correspond to synaptic and persistent sodium currents, respectively. $g_{\text{syn}}$ depends on the presynaptic neuron’s voltage, $V_{\text{pre}}$, and the
parameter values of the synapse itself:

\[
g_{syn}^i = G_{syn}^i \cdot \begin{cases} 
0, & V_{pre}^i < E_{lo}^i \\
\frac{V_{pre}^i - E_{lo}^i}{E_{hi}^i - E_{lo}^i}, & V_{pre}^i \geq E_{lo}^i \text{ and } V_{pre}^i \leq E_{hi}^i \\
1, & \text{else}
\end{cases} 
\]  

(7.10)

\(G_{syn}\) is the maximum conductance of the synapse, \(E_{lo}\) is the synapse’s conductance threshold, and \(E_{hi}\) is the synapse’s saturation voltage.

Both \(m_\infty(V)\) and \(h_\infty(V)\) are monotonically increasing and decreasing functions, respectively. For each variable we define a quantity \(z\):

\[
\begin{align*}
z_h & = 0.5 \cdot \exp([E_h - V]/S_h) \\
z_m & = \exp([E_m - V]/S_m),
\end{align*}
\]  

(7.11) and (7.12)

in which \(E\) and \(S\) shift and steepen \(z\), respectively. \(z\) enables us to define the values:

\[
\begin{align*}
m_\infty & = 1/(1 + z_m) \\
\tau_m & = m_\infty \cdot \sqrt{z_m} \cdot T_m \\
h_\infty & = 1/(1 + z_h) \\
\tau_h & = h_\infty \cdot \sqrt{z_h} \cdot T_h,
\end{align*}
\]  

(7.13) and (7.14) and (7.15) and (7.16)

where \(T_m\) and \(T_h\) are constant parameters that scale \(m\) and \(h\)’s time constants. Recall from (7.8) and (7.9) that \(m\), \(h\), and their time constants are all functions of \(V\).
Chapter 8

MantisBot Changes Stepping Speed by Entraining CPGs to Positive Velocity Feedback

This chapter was accepted for publication in the Proceedings of the Conference on Biomimetic and Biohybrid Systems (i.e. "Living Machines"), after peer-review. Minor edits have been made, particularly in adding references to other parts of this thesis.

8.1 Abstract

This paper demonstrates and analyzes how CPGs can entrain joints of a praying mantis robot (MantisBot) to positive velocity feedback resulting in a duration change of a leg’s stance phase. We use a model of a single leg segment, as well as previously presented design techniques to understand how the gain of positive velocity feedback to the CPGs should be modulated to successfully implement the active reaction (AR) during walking. Our results suggest that the AR simplifies the descending control of walking speed, naturally producing the asymmetrical changes in stance and swing
phase duration seen in walking animals. We implement the AR in neural circuits of a dynamic network that control leg joints of MantisBot, and experiments confirm that the robot modulates its walking speed as the simple model predicted. Aggregating the data from hundreds of steps in different walking directions show that the robot changes speed by altering the duration of stance phase while swing phase remains unaffected, as seen in walking animals.

8.2 Introduction

Insects control the velocity of their legs and joints when propelling their bodies [Cruse, 1985a, Gruhn et al., 2009]. This makes sense for smoothly controlling the body’s speed with respect to the ground (i.e. “body speed”). The duration of swing phase is generally constant and independent of body speed [Gabriel and Büschges, 2007]. Thus, changing the body speed also changes the duty cycle of stepping, and different gaits emerge. At their fastest speeds, insects walk with an alternating tripod gait, in which three legs are on the ground at once. As speed decreases, the gait changes and more and more legs are in stance phase at the same time. This has the added benefit of distributing external forces among more legs, decreasing the support and propulsion force that each leg needs to provide [Foth and Graham, 1983].

How, then, is this asymmetrical phase duty accomplished in the animal? It is known that each joint of walking insects has its own central pattern generator (CPG), which contributes to ongoing rhythmic activity during walking [Ryckebusch and Laurent, 1993, Büschges et al., 1995]. Related models of stick insect locomotor control networks accomplished this change by applying asymmetrical drive to the CPGs, changing the duration of retraction relative to protraction [Daun-Gruhn and Tóth, 2010]. Neurobiological studies with stick insects, however, suggest that descending signals that control walking speed target sensory pathways, rather than rhythm-
generating signals [Sauer et al., 1997]. More recent results from Drosophila with transfemoral amputation show that the CPGs in the cut leg do not change speed when the other legs do, but instead they oscillate at the maximum walking speed all the time [Berendes et al., 2016]. This suggests that the CPGs in the leg entrain to leg-local sensory signals by prolonging stance phase.

One such leg-local sensory phenomenon that may contribute is the active reaction (AR). The AR is a positive-feedback reflex in which flexing the FTi joint causes excitation of the flexor muscle, followed by excitation of the extensor [Bässler, 1988]. In reduced preparations, the AR is elicited by a combination of movement information from the femoral chordotonal organ (fCO), which monitors the FTi joint’s rotation, and load information from the femoral campaniform sensilla (fCS) [Akay and Büschges, 2006]. However, stimulating the fCO and fCS separately reduces the reliability of the reflex. Stimulating the fCO alone sporadically elicits an AR, while stimulating the fCS alone reliably elicits a delayed AR. The authors of [Akay and Büschges, 2006] suggest that loading information may increase the gain of positive velocity feedback. This hypothesis led to the network design in this paper, which is tuned to reproduce results like those from the animal. We believe the AR is the CPG entraining to local positive feedback, and thus refer interchangeably to these phenomena throughout the text.

This paper uses previously presented design tools (Chapter 3, [Szczecinski et al., 2017b]) to analyze how the AR affects the stability of walking at a continuum of body speeds, and how it should be tuned. Section 8.3 presents our methods, including the neuron and synapse models, the control network connectivity, a simulation of a single leg segment from MantisBot, and how it enabled us to tune the AR. Section 8.4 presents our results, showing that the AR enables MantisBot (Figure 8.1) to step at a continuum of speeds, while the duration of swing phase remains constant. Section 8.5 discusses the implications of these results for the study of the neural control of
walking in both insects and robots.

8.3 Methods

8.3.1 Neuron and Synapse Models of the Synthetic Nervous System (SNS)

MantisBot is controlled by a Synthetic Nervous System (SNS), a continuous-time dynamical neural simulation. SNS dynamics are computed with the neuromechanical simulator AnimatLab 2 [Cofer et al., 2010]. The AnimatLab Robotics Toolkit enables the SNS to issue motor commands to a robot and receive sensory feedback over a serial connection. This system performs real-time neural control of MantisBot. The neural model is a simplified Hodgkin-Huxley model [Cofer et al., 2010, Daun-Gruhn and Tóth, 2010, Hodgkin et al., 1952]. The details can be found in Chapter 7, but a summary is provided here. Each neuron has three dynamical variables: $V$, the membrane voltage, $m$, the activation of a persistent sodium channel, and $h$, the deactivation of the same sodium channel. This model does not generate spikes. The dynamics of the model are as follows:

$$C_{\text{mem}} \cdot \frac{dV}{dt} = G_{\text{mem}} \cdot (E_{\text{rest}} - V) + g_{\text{syn}} \cdot (E_{\text{syn}} - V) + G_{\text{NaP}} \cdot m \cdot h \cdot (E_{\text{NaP}} - V)$$  \hspace{1cm} (8.1)
$$\frac{dm}{dt} = \frac{m_\infty(V) - m}{\tau_m(V)}$$  \hspace{1cm} (8.2)
$$\frac{dh}{dt} = \frac{h_\infty(V) - h}{\tau_h(V)}.$$  \hspace{1cm} (8.4)

where $C$ is capacitance, $G$ is a static conductivity, $E$ is a static reference voltage (i.e. reversal potential), $\tau$ is a time constant, and subscript $\text{mem}$ stands for membrane, $\text{syn}$ stands for synaptic, $\text{NaP}$ stands for persistent sodium, and $\infty$ stands for steady-state.
Figure 8.1: MantisBot secured to a frame. The left middle leg is taking steps on a block. The green line on the block is 15 cm long, for scale.
Both $m_\infty(V)$ and $h_\infty(V)$ are sigmoidal functions, but $m_\infty(V)$ increases monotonically with $V$, and $h_\infty(V)$ decreases monotonically with $V$. In addition, $\tau_m \ll \tau_h$, which means that initial rises in voltage are first positively reinforced by $m$, and then resisted as $h$ decreases over time. Neurons communicate via synapses by changing their conductivity:

$$G_{\text{syn}} = \begin{cases} 
0 & \text{if } V < E_{lo}, \\
G_{\text{max}} \cdot \frac{V - E_{lo}}{E_{hi} - E_{lo}} & \text{if } V \geq E_{lo} \text{ and } V \leq E_{hi}, \\
G_{\text{max}} & \text{if } V > E_{hi}.
\end{cases}$$ (8.5)

where $V$ is the presynaptic neuron’s voltage, and $G_{\text{max}}$, $E_{hi}$, and $E_{lo}$ are properties of the synapse.

This model has several advantages for robotic control and animal modeling: 1) the persistent sodium channel causes bursting dynamics, in which the membrane voltage $V$ rapidly increases, plateaus, and diminishes over time. This is necessary for the robust oscillations of central pattern generators [Selverston and Moulins, 1985]; 2) the leaky integrator dynamics enable the construction of dynamic networks such as differentiators and integrators (Chapter 4, [Szczecinski et al., 2017c]); 3) the simplicity of the dynamics enables direct tuning of parameter values (Chapter 3, [Szczecinski et al., 2017b]).

### 8.3.2 Single Joint Model

We built a model of a single leg segment (i.e. the tibia) and its control network to examine how positive velocity feedback may affect the CPG’s phase, and thus generate an active reaction. The network is shown in Figure 8.2. It is mostly the same as our previously-presented network (Chapter 6, [Szczecinski et al., 2017a]), so we will focus our explanation on the novel components. New descending pathways are shown in green, and new sensory pathways are shown in cyan. Descending
CHAPTER 8. MANTISBOT CHANGES STEPPING SPEED BY ENTRAINING CPGS TO POSITIVE VELOCITY FEEDBACK

Joint Controller Commands Servomotor Velocity and Uses Positive Velocity Feedback to Entrain CPG to Walking Speed

Figure 8.2: Schematic of a control network for one joint of MantisBot. The overall function of the network is described in Chapter 6 [Szczecinski et al., 2017a]. New
Figure 8.2: (contd.) descending pathways are drawn in green. New sensory feedback pathways are drawn in cyan. A full description is in the text.

pathways affect motion by changing the local processing of sensory feedback [Berendes et al., 2016, Martin et al., 2015]. In our robot, the servomotor can control its speed by altering how negative velocity feedback is processed, so issuing a motor speed command via the “Comm. Speed” neuron effectively does the same thing. Comm. Speed receives excitatory synaptic input from the “Range of Motion” neuron, such that each joint’s speed is scaled by its range of motion. This ensures that in swing phase, each servomotor returns to the AEP in the same amount of time. When an insect slows its body speed, it only slows the rotation of its joints when in stance phase [Foth and Graham, 1983, Gabriel and Büschges, 2007]. The “Leg Speed” neuron reduces the sensitivity of the Comm. Speed neuron via a modulatory disinhibitory pathway (Chapter 4), which is only active if the leg is in stance phase (i.e. “Leg Strain” is active). This reduces the servomotor’s commanded speed.

Tests with our model showed that if the servomotor is commanded to move more slowly but the CPG is not also slowed down, then the servomotor cannot achieve its full range of motion, resulting in incorrect stepping (See Figure 8.4Bii). Therefore, the CPG must also receive feedback about the motion of the joint, so it can entrain with the motion of the leg. Experiments with stick insects showed that velocity feedback alone entrains the CPG about 50% of the time, but load feedback alone or simultaneous load and velocity feedback entrain the CPG every time [Akay and Büschges, 2006].

The authors of [Akay and Büschges, 2006] suggested that the load feedback may increase the gain of the velocity feedback. In order to cause an AR without load feedback, the velocity feedback must have some “baseline gain”. This is provided by the “Baseline Gain” neuron in Figure 8.2. When the leg is loaded, this neuron is further excited, increasing the amplification of the velocity signal, and evoking an
AR. In order to cause an AR without velocity feedback, the “Ext. Velocity” and “Flx. Velocity” neurons must also receive some depolarizing input from the strain sensors. The “Strain Increasing” neuron, which primarily computes the rate of change of the leg strain, also supplies a small tonic current in steady state, which is sufficient to entrain the CPG. Because the duration of swing phase is the same no matter an insect’s body speed [Gabriel and Büschges, 2007], the swing phase velocity pathway is inhibited by the “Ext. PEP” or “Flx. PEP” neuron, depending on the direction of stepping, such that only the duration of stance phase is affected by velocity feedback.

8.3.3 δ and Tuning the Active Reaction

How can we tune the synaptic parameter values to obtain the insect-like CPG entrainment described in the previous section? The AR and other positive feedback mechanisms stabilize stance phase by ensuring that the leg does not enter swing until the leg is no longer propelling or supporting the body. Therefore, our sensory input must be strong enough to temporarily push the CPG into a stable stance phase (network shown in Figure 8.3A). Simulations show that a weak stimulus will affect, but not halt oscillation, failing to produce a stable stance phase (Figure 8.3Bi). However, a stronger stimulus will halt oscillation, and then cause an immediate transition once the stimulus disappears (Figure 8.3Bii). This is precisely what is needed, but how do we quantify this effect? Previously we identified δ, a bifurcation parameter of our CPG that dictates its stability (Chapter 3, [Szecinski et al., 2017b]). When δ > 0, the CPG oscillates, and when δ < 0, the CPG cannot oscillate. We can quickly calculate δ as a function of $V_{\text{input}}$, which dictates how large of a sensory input will stabilize the stance phase (Figure 8.3C). For the network shown, δ < 0 if $V_{\text{input}} > -57.1 \text{ mV}$, meaning that we can use the positive feedback’s amplitude to determine if it will stabilize stance phase.

Based on experiments with stick insects, the AR should occur 50% of the time.
Figure 8.3: Demonstration of how feedback may stabilize CPG dynamics. A) A schematic of a four-neuron CPG model with one input neuron. Bi) Weak sensory input will cause asymmetrical CPG bursting, but will not lock the phase. Bii) Stronger sensory input will lock the CPG’s phase. C) Computing $\delta$ as a function of $V_{\text{input}}$ reveals how strong the input must be to lock the CPG’s phase.
without load feedback. We used our suite of design tools to run a large batch of AnimatLab simulations of our single joint model to see how the leg velocity and baseline gain affected its ability to entrain to velocity feedback with and without load feedback. Figure 8.4Ai shows $V_{input}$ plotted as a contour plot. If the CPG entrained with the sensory input, the contour is plotted as a solid line. If it failed to entrain, it is plotted as a dashed line. The boundary between these two regimes intersects with the command for 50% leg speed when the gain is 22.75%, meaning that the resting potential of the “Baseline Gain” neuron can be determined. Finally, the tonic activity of the “Load Increasing” neuron must be determined such that the AR always occurs if load feedback is present, whether there is velocity feedback or not.

Figure 8.4Aii shows another contour plot of the same simulation when load feedback is provided when the joint flexes. In this case, $V_{input}$ no longer depends on the baseline gain, and is always large enough to stabilize stance phase. The plots in Figure 8.4B illustrate this. In Figure 8.4Bi, the servomotor is commanded to move fast in stance phase, and the velocity feedback alone is enough to ensure the joint rotation reaches the PEP. In Figure 8.4Bii, the servomotor is commanded to move slowly in stance phase, and the velocity feedback does not surpass the entrainment threshold, and the range of motion is incomplete. In Figure 8.4Biii, however, load feedback is enabled, and the same speed commanded in Figure 8.4Bii entrains the CPG, and the joint rotation reaches the PEP. Note that “swing phase”, or positive joint rotation, is the same no matter the conditions.

8.3.4 Robot and Experiments

The data in Figure 8.5 and Figure 8.6 were collected from experiments with our robot, MantisBot (Figure 8.1). MantisBot’s thorax was bolted to a stand that supported its weight. The left middle leg was then commanded to walk on a slippery block for 60-second trials. This is similar to the reduced preparation used to study insect
CHAPTER 8. MANTISBOT CHANGES STEPPING SPEED BY ENTRAINING CPGS TO POSITIVE VELOCITY FEEDBACK

CPG Entrainment Properties and Biological Data Determine the Strength of Velocity Feedback Necessary for Entrainment.

Figure 8.4: Single-segment model informs the strength of positive velocity feedback to the CPGs. Ai) To determine how strong velocity feedback should be to evoke an AR in 50% of cases, the magnitude of the velocity feedback, $V_{input}$, was calculated as the commanded leg velocity and baseline gain were changed. The contour plot reveals which parameter values produce stable entrainment. The baseline gain value that causes entrainment for 50% of commanded velocity values was chosen. Aii) Another contour plot of the same system, but with load feedback during stance, verifies that the system always entrains to the sensory feedback. Bi) High joint velocity and no
Figure 8.4: (contd.) load stably entrains the CPG. Bii) Low joint velocity and no load feedback cannot stably entrain the CPG. Biii) Low joint velocity plus load feedback can entrain the CPG.

locomotion [Bässler, 1993]. In each trial, the leg was commanded to walk in a different direction (−30, −15, 0, 15, or 30 degrees of body yaw per step) and at a different speed (15, 11.25, 7.5, 5.625, or 3.75 cm/s, which correspond to the stance phase foot speed that is 100%, 75%, 50%, 37.5%, and 25% of swing phase foot speed). Force feedback from a strain gage on the trochanter, and speed and position feedback from the motors coordinate the motion of the separate servomotors into directed locomotion (Chapter 6, [Szczecinski et al., 2017a]).

8.4 Results

Figure 8.5A shows data from single leg walking trials with MantisBot. They show the same data as in Figure 8.4B. These plots show that the joints on MantisBot entrain to sensory input as in the single-joint model. No matter the commanded joint velocity, the CPG is entrained by sensory input to prolong stance phase and ensure the joint rotates all the way to its PEP. In contrast, swing phase’s duration is constant. Figure 8.5Bi and Bii each show plots of every step that MantisBot took in a particular direction. Each commanded speed is plotted in a different color. Each trace begins with the end of stance phase. Swing phase takes 1.4 s on average to complete, shaded in gray. Stance phase takes place from the end of swing phase to the dotted line of the matching color. For instance, the stance speed for traces in blue is 100% of the swing speed, meaning that stance phase should end 1.4 s after swing phase. One can see that the blue traces end near to the blue dotted line. Each speed’s number of steps is listed in the key at the bottom of Figure 8.5B. Note that each joint’s range of motion is unaffected by the commanded speed, that is, that traces have the same amplitude, regardless of color.
CHAPTER 8. MANTISBOT CHANGES STEPPING SPEED BY ENTRAINING CPGS TO POSITIVE VELOCITY FEEDBACK

CPG Entrainment Properties Enable MantisBot to Asymmetrically Change the Speed of Stance Phase

Figure 8.5: The results from the single segment model apply directly to MantisBot, enabling it to change speed while the duration of swing phase remains constant. A) Data from the robot, formatted as in Figure 8.4B. The same entrainment occurs. B) Data from all steps taken in specific directions. Each step is plotted, starting in swing phase at 0 s. The average swing phase duration is shaded in gray. Each walking speed
Figure 8.5: (contd.) is plotted in a different color. Dotted vertical lines indicate when lines of that color should end, based on the commanded walking speed. Note that each joint’s range of motion does not depend on its speed in stance phase.

**MantisBot Changes Speed by Changing the Duty Cycle of Stepping**

A. 

![Graph showing stance duration vs. commanded stance speed.](image)

B. 

![Graph showing stance duration vs. commanded stance speed.](image)

Error bars are ± 1 standard deviation.

Figure 8.6: All 279 steps taken by the robot were aggregated to analyze gross trends in the data. A) No matter the stance phase speed, the duration of swing phase is constant. At the robot’s fastest speed, stance phase and swing phase have equal durations. As the robot walks more slowly, the duration of stance phase increases, as seen in animals. B) The fraction of time spent in stance phase changes as intended with walking speed.

Figure 8.6 aggregates all of the steps that MantisBot took (N = 279), bins them by commanded stance speed as a fraction of swing speed. Figure 8.6A shows that the duration of swing phase is constant no matter the body speed. This is critical for the emergence of the continuum of gaits seen in insects [Foth and Graham, 1983]. Figure 8.6B shows the duty cycle and the theoretical value, that is, if swing phase were exactly constant over all speeds, and the foot moved at exactly the commanded speed. The robot data is within one standard deviation for all stance speeds above 37.5% of the swing speed. This suggests that the robot steps as intended in a gross sense, adjusting the proportion of time spent in stance phase as the commanded walking speed is changed.
8.5 Discussion

In this paper we used our suite of SNS design tools to tune a positive velocity feedback reflex to model the active reaction (AR). The behavior of the AR was used as the intended behavior of a single-segment model, and sensory pathways were tuned to produce this result. These pathways actually stabilize the CPG, ensuring that stance phase does not end unless the leg stops moving or is unloaded. These pathways were added to MantisBot’s stepping controller, enabling MantisBot to change its walking speed without changing its range of motion or the duration of swing phase. The resulting duty cycle of stepping was very close to what should be theoretically expected, showing that positive velocity feedback can indeed entrain the CPGs and stabilize stance phase over a continuum of body speeds.

The asymmetrical change in stance and swing duration as body speed changes is critical to producing the continuum of gaits seen in insects [Foth and Graham, 1983]. This continuum contributes to supporting external forces, since placing more legs on the ground at once distributes the forces over more legs. A robot must be able to react to external conditions, including external forces. For example, an agricultural robot’s weight may increase as it harvests more crops, or its weight may decrease as it depletes fuel. Adapting to these changes is critical to real-world operation, and the work in this paper is a first step towards solving this problem via gait modification.

The network in this paper may also serve as a hypothetical structure underlying several neurobiological findings. It is known that the descending pathways that control an insect’s walking speed and direction primarily affect sensory processing [Berendes et al., 2016, Martin et al., 2015, Sauer et al., 1997]. Our speed-control descending pathways limit the velocity of the servomotors; the servomotor, in turn, accomplishes this change by modulating negative velocity feedback in its controller. Ultimately, both the animal and MantisBot may change speed the same way, by increasing the strength of sensory feedback that resists motion in stance phase.
In this paper we called the AR a form of positive feedback. This phrasing is what is used in the neurobiological literature, because initiating stance phase motion, such as flexing the FTi joint or loading the leg, initiates and reinforces stance phase [Bässler, 1988]. This is not the same as continuous positive velocity feedback acting on the motor neurons directly, which is also reported in the literature [Schmitz et al., 1995]. This mechanism, termed Local Positive Velocity Feedback (LPVF), can be used to minimize internal torques in an insect or robot with multiple legs on the ground at once [Schmitz et al., 2008]. In practice, this is implemented by adding the current position of the limb, passed through a high-pass filter, to the motor command. This is precisely what the “Flx. Velocity” and “Ext. Velocity” neurons calculate in our network (see Figure 8.2). LPVF may be readily implemented in the future when MantisBot walks freely, with the legs mechanically coupled through the ground.

Our results highlight the importance of local sensory feedback for coordinating rhythmic motion. Recent findings in Drosophila show that the CPG that controls leg levation and depression will oscillate at its fastest possible speed (i.e. a 50% duty cycle) if the distal portion of the leg is amputated [Berendes et al., 2016]. This suggest that in the animal, local sensory feedback is critical for prolonging stance phase when walking at slower speeds. Our results show how sensory feedback may bifurcate the CPG system into a stable state, ensuring that stance phase is the proper duration for the speed commanded by the brain. We believe that this work is more flexible and biologically accurate than providing asymmetrical drive to the different halves of the CPG [Daun-Gruhn and Tóth, 2010], especially when considering that speed-controlling descending commands appear to target sensory feedback processing, not the CPG networks themselves [Sauer et al., 1997].
Chapter 9

Leg-Local Neural Mechanisms for Searching and Learning Enhance Robotic Locomotion

This chapter is currently under review in the journal *Biological Cybernetics*.

9.1 Abstract

Adapting motor output based on environmental forces is critical for successful locomotion in the real world. Arthropods use at least two neural mechanisms to adjust muscle activation while walking based on detected forces. Mechanism 1 uses negative feedback of leg depressor force to ensure that each stance leg supports an appropriate amount of the body’s weight. Mechanism 2 encourages searching for ground contact if the leg supports no body weight. We expand the neural controller for MantisBot, a robot based upon a praying mantis, to include these mechanisms by incorporating leg-local memory and command neurons, as observed in arthropods. We present results from MantisBot transitioning between searching and stepping, mimicking data from animals as reported in the literature.
9.2 Introduction

Insects and crustaceans are agile, adaptable walkers. Their nervous systems use central patterns [Büsschges et al., 1995] and sensory information [Akay et al., 2004, Hess and Büsschges, 1999, Zill et al., 2004] to coordinate them and allow them to adapt to their environment (for a review, see [Buschmann et al., 2015]). We believe their nervous systems are useful models for controlling walking robots. A walking animal or robot must be able to manage the forces applied to its body, while it changes its walking pattern when stepping onto obstacles, into holes, or missing the ground completely.

Arthropods measure forces with strain organs, and use the forces to adjust motion [Zill and Seyfarth, 1996]. A much-studied topic is how signals from strain organs coordinate walking systems [Akay et al., 2004, Noah et al., 2004, Zill et al., 2004], and several models have been constructed to further investigate it [Daun-Gruhn, 2010, Ekeberg et al., 2004, Szczecinski et al., 2014]. Our hexapod robotic model of a praying mantis, MantisBot, already uses leg strain signals to coordinate stepping (Chapter 6, [Szczecinski et al., 2017a]). However, to walk adaptively through cluttered environments, arthropods use additional mechanisms that also modulate the amplitude of motion.

When a leg enters the stance phase of its stepping cycle, it should support an appropriate share of the body’s weight. When walking in a cluttered environment, this is unlikely to occur. If a leg steps onto an obstacle and attempts to depress its leg to the joint angles typical for horizontal terrain, it will cause excessive force and possibly damage the limb or tip the body over. If a leg steps into a hole and does not reach the ground, then it must search forward or downward for a foothold so that it can support the body’s weight. The literature reports that arthropods have two mechanisms for handling these scenarios, which we will refer to as “Mechanism 1” and “Mechanism 2” in this paper. Mechanism 1 uses negative feedback of leg strain...
to adjust muscle force, reducing depressor activation when leg strain is too great [Libersat et al., 1987], or when walking on vertical surfaces [Noah et al., 2001]. These experiments suggest that the nervous system has a context-dependent expectation of how much force a leg should support. Mechanism 2 encourages a leg to search for a foothold when the animal reaches a gap in the walking substrate [Bläsing and Cruse, 2004] or when it attempts to walk but cannot make ground contact [Berg et al., 2015].

We have implemented dynamical neural models of both of these mechanisms and fully integrated them into the stepping controller of MantisBot (Figure 9.1). These additions are necessary for MantisBot to adaptively walk in real-world environments. They were added to the robot’s synthetic nervous system (SNS) without dismantling its existing locomotion controller; now, there are context-dependent alteration of pathway strength and local learning mechanisms to adjust motion over time [Eisenstein, 1972]. While other impressive robots exist that can produce the same adaptive stepping [Dasgupta et al., 2015], our implementation directly exploits neural dynamics to implement onboard learning, and we believe it is simpler and less computationally expensive.

In this paper, we present two mechanisms that enable MantisBot to adapt its stepping to changing conditions. Mechanism 1 allows the leg to adjust its depression to obtain intended force. Mechanism 2 encourages searching for ground contact when the leg depresses and no strain is detected. Section 9.3 presents experimental and modeling methods. Section 9.4 presents results from Mechanism 1. Section 9.5 presents results from Mechanism 2. Section 9.6 presents results from both mechanisms collaborating, and Section 9.7 discusses how these results contribute to the neurobiology of arthropod locomotion and future directions for research.
9.3 Methods

9.3.1 Experimental Methods

All data presented in this paper was collected from tests with one leg of our hexapod robot, MantisBot. It was set up like the “reduced preparation” of insects, in which the body is fixed and all but one leg are removed [Berg et al., 2015]. The setup is shown in Figure 9.1. The thorax was fixed to a support frame, preventing translation and rotation of the body. An adjustable height step with a slippery surface was placed under the left middle leg. When the robot stepped, it dragged its foot over the surface similar to oiled-plate [Noah et al., 2001] or treadmill [Berg et al., 2015] experiments with insects.

In our previous work, we presented a framework for controlling walking direction with simple descending commands from the head (Chapter 6, [Szczecinski et al., 2017a]). This is based on data from cockroaches showing that activity in the central complex modifies joint-level reflexes to change the walking direction [Martin et al., 2015]. These descending pathways were used to collect the data presented in this paper. In section 9.4, MantisBot is commanded to walk forward with a 15 cm stride length. In section 9.5, MantisBot is commanded to turn to the left, rotating the body 10 degrees with each step. This produces the inward stepping observed in the “reduced preparation” of stick insects [Berg et al., 2015]. These commands correspond to the brushing of the abdomen frequently used in insect studies to elicit walking behavior from this reduced preparation [Berg et al., 2015]; the brushing may cause the brain to activate the thoracic networks that control walking, or may elicit an escape motion, although Phasmids tend to use catalepsy to avoid predators.
Figure 9.1: Photograph of our hexapod robot, MantisBot. The thorax is fixed in place, while the robot uses its left middle leg to step on the slippery substrate. The green stripe on the step is 15 cm long, for scale.

9.3.2 Modeling Methods

MantisBot’s neural controller is implemented in AnimatLab 2, a neuromechanical simulator [Cofer et al., 2010]. The AnimatLab Robotics Toolkit enables the user to build closed-loop, real-time neural controllers by using motor neuron membrane voltages to control actuators, and sensor readings to inject current into sensory neurons (Chapter 8, [Szczecinski et al., 2015a]). MantisBot’s controller is built from the “nonspiking neuron” type, which is a Hodgkin-Huxley compartment without ion channels for generating spikes [Cofer et al., 2010]. One such neuron can serve as the actual membrane voltage of a nonspiking neuron, or the mean activity of a pool of spiking neurons.

Each neuron’s membrane voltage, \( V \), is a dynamic function of currents flowing through the membrane,

\[
C_m \dot{V} = I_{\text{leak}} + I_{\text{syn}} + I_{\text{app}}. \tag{9.1}
\]
These currents have the forms

\[ I_{\text{leak}} = G_m \cdot (E_r - V) \]  \hspace{1cm} (9.2)

and

\[ I_{\text{syn}} = \sum_{i=1}^{n} G_{s,i} \cdot (E_{s,i} - V), \]  \hspace{1cm} (9.3)

Where \( C_m \) is the membrane capacitance, \( G \) is the membrane conductance, \( E_r \) is the resting potential, \( G_{s,i} \) is the conductance of the \( i^{th} \) incoming synapse, and \( E_{s,i} \) is the reversal potential of the \( i^{th} \) incoming synapse. \( I_{\text{app}} \) is a tonic stimulus.

\( G_{s,i}(V_{\text{pre}}) \) is a piecewise function of the presynaptic neuron’s membrane voltage, \( V_{\text{pre},i} \):

\[ G_{s,i}(V_{\text{pre}}) = \begin{cases} 0 & \text{if } V_{\text{pre},i} < E_{\text{lo}}, \\ g_{s,i} \cdot \frac{V_{\text{pre},i} - E_{\text{lo}}}{E_{\text{hi}} - E_{\text{lo}}} & \text{if } E_{\text{lo}} < V_{\text{pre},i} < E_{\text{hi}}, \\ g_{s,i} & \text{if } V_{\text{pre},i} > E_{\text{hi}}. \end{cases} \]  \hspace{1cm} (9.4)

\( E_{\text{lo}} \) is the synapse’s threshold for conductance, and \( E_{\text{hi}} \) is its conductance saturation. This piecewise linear function approximates a sigmoid activation function, but simplifies analysis when \( E_{\text{lo}} < V_{\text{pre},i} < E_{\text{hi}} \).

Additional dynamics, such as the persistent sodium channels that generate CPG bursting, are explored in (Chapter 3, [Szczecinski et al., 2017b, Daun-Gruhn et al., 2009]).

### 9.3.3 Leg Control Network

Figure 9.2 shows part of the leg control network for MantisBot’s middle leg. Details about the structure, function, and tuning of this network can be found in (Chapter 6, [Szczecinski et al., 2017a]; Chapter 7, [Szczecinski and Quinn, 2017c]). Here we describe the elements relevant to the results in this paper.

Figure 9.2 is divided into nine regions (A1, A2, etc.). Each region performs a
different function. Neurons in row 1 are descending neurons from higher command centers, which control the walking direction and the amount of force that the leg can carry in stance phase. Neurons in row 2 are leg-level neurons that implement inter-segmental reflexes that coordinate walking. Neurons in row 3 are joint-level neurons that control the velocity of the servomotors and generate rhythms for walking. Column B controls the motion of the CTr joint, and column C controls the motion of the FTi joint. The ThC joints also have networks identical in structure to that in section C3.

The new networks added for this paper are in column A. Section A1 contains a neuron whose activity encodes the amount of force that the leg should support, called Ref. Strain. Answering the question, "what determines Ref. Strain's activity" is beyond the scope of this study. It could be innervated by sensors from adjacent legs [Libersat et al., 1987, Zill and Seyfarth, 1996], descending commands based on body orientation [Noah et al., 2001], or a combination. In our network for this reduced preparation, Ref. Strain's activity is set by a tonic stimulus. In a more complete animal model or robot controller, this question would need to be addressed.

Ref Strain's activation is compared to the amount of force the leg is actually supporting, encoded by the strain in the leg, in A2. These two values are compared, and if too much force is detected, the depressor's position in stance phase is elevated by inhibiting neuron 1 in section A3. If too little force is detected, the depressor's position in stance phase is depressed by exciting neuron 1. Neurons 1 and 2 form a memory network, which enables the leg to adjust the degree of leg depression from step to step until the intended force is supported. This mimics results from walking crabs, in which excessive dactyl (i.e. foot) strain reduces the activation of depressor muscles in stance phase [Libersat et al., 1987, Zill and Seyfarth, 1996]. The dynamics and stability of this network are analyzed in Section 9.3.4, and results from the robot are presented in Section 9.4.
Section A3 of Figure 9.2 also shows nonspiking interneuron I4, which has been identified as a command neuron for searching behavior [Berg et al., 2015]. I4 is thought to activate interneurons (INs) “downstream”, because straining the leg stops searching, even when I4 is strongly depolarized. To mimic this result, I4 disinhibits Search IN. This neuron’s membrane voltage is proportional to the difference between the CTr’s actual and intended depression. However, it is strongly inhibited by leg strain information, meaning that Search IN is only active when the CTr has fully depressed and no load was detected on the leg.

How, then, does the Search IN induce searching? First, it causes FTi flexion when the CTr is fully depressed without load. This is necessary because leg load causes this transition in walking insects [Akay et al., 2001, Schmitz et al., 2015], but the FTi only flexes once the CTr fully depresses in searching legs [Bässler et al., 1991, Berg et al., 2015]. Second, it provides CPG drive to strengthen the rhythmic motion needed for searching. Finally, it incrementally increases CTr depression, as seen in a ditch-crossing paradigm [Cruse, 1976]. This helps the body maintain its pitch and roll as one leg steps into a ditch. Results from this network are presented in Section 9.5.

9.3.4 Tuning Mechanism 1: Learning Proper Leg Depression

The linear dynamics of this model enable us to use analytical methods to understand the function of small networks (Chapter 4, [Szczecinski et al., 2017c]). The novel contribution of this work is to use a marginally-stable memory network to make incremental changes to the CTr servomotor’s range of motion based on the measured ground reaction force. If the foot makes contact with the ground, it must use onboard learning to adjust the CTr motion to obtain the intended force. If the foot does not make contact, it must initiate a searching mechanism, during which it lowers the foot to find the ground. To function properly, we must prove 1) that our memory subnetwork is marginally stable, and that 2) the magnitude of our step-to-step adjustment
Figure 9.2: Diagram of part of our leg control network. Column A contains the novel networks in this paper. Column B contains the CTr servomotor’s control network.
Figure 9.2: (contd.) Column C contains the Fti servomotor’s control network. Row 1 contains high-level commands, row 2 contains inter-leg signals, and row 3 contains local networks. Synapses of the Mechanism 1 network are green. Synapses of the Mechanism 2 network are blue. Descending neurons are red, sensory neurons are orange, and CPG neurons are yellow. For a full description of the network that controls walking, please see (Chapter 6, [Szczecinski et al., 2017a]; Chapter 7, [Szczecinski and Quinn, 2017c]).

to the CTr’s motion results in stable learning.

**Stability of Memory Subnetwork**

Two mutually inhibitory neurons can form a memory subnetwork when tuned properly (see Chapter 4.5.2). This subnetwork is built into the CTr joint controller (neurons 1 and 2 in Figure 9.2A3) such that the servomotor’s commanded position in stance phase (Stance Depression neuron) is the sum of descending signals and the memory subnetwork. The descending signal is dependent on the direction of travel. The memory subnetwork’s activity is changed depending on the difference between the average force during stance phase and the intended force. The next section analyzes the stability of this process.

**Stability of Mechanism 1**

Mechanism 1 compares the leg strain throughout stance phase to a commanded value, and adjusts the leg’s depression until the commanded strain is obtained. Initial experiments suggested that this mechanism’s stability is sensitive to the neural and synaptic parameter values used. In this section, we model the leg strain at the end of stance phase as a discrete differential equation, changing from step to step depending on the difference between the intended and actual leg strain. The stability and robustness of this model is used to tune the network, and results from the robot are presented in Section 9.4.

When the leg is in stance phase, neuron 1 accumulates the activity of the Too
Much and Too Little strain neurons, and decreases or increases CT$r$ depression, respectively. When the leg is in swing phase, neuron 1’s remembers the CT$r$ depression for the beginning of the next step. Thus, the mean leg strain changes from step to step, because the strain in the current stance phase controls the rate at which CT$r$ depression changes for the next step. This can be modeled by the discrete differential equation

$$V_{s,i+1} - V_{s,i} = k \cdot (V_s^* - V_{s,i}), \quad (9.5)$$

where $V_s$ is the voltage of the Leg Strain neuron, $i$ is the step counter, and $k$ is the time constant. Collecting terms from Equation 9.5 gives

$$V_{s,i+1} = (1 - k) \cdot V_{s,i} + k \cdot V_s^*. \quad (9.6)$$

This system is stable if $a = (1 - k) \in (-1, 1)$. The response is overdamped if $a \in (0, 1)$, underdamped if $a \in (-1, 0)$, and critically damped if $a = 0$. If the system is critically damped, i.e. $k = 1$, the leg would learn the CT$r$ depression to produce the intended leg strain in one step, which is ideal for adapting to irregular terrain.

Knowing that $k \in (0, 2)$ places a constraint on the parameter values of the network for stable learning. The change in the Stance Depression neuron’s voltage depends on 1) amplification of the comparison between the Ref. Strain and Leg Strain neurons, $k_{comp}$, 2) the amplification of the Too Much and Too Little neuron voltages, $k_{int}$, and 3) how quickly neuron 1 integrates the input, $k_{mem}$. These three factors combine to control how quickly the Stance Depression neuron’s voltage, $V_\theta$, changes.

$$V_{s,i+1} - V_{s,i} = \Delta V_s$$

must be related to $\Delta V_\theta$, which can be done via the chain rule. $\Delta V_s = dV_s/dV_\theta \cdot \Delta V_\theta$, where

$$\frac{dV_s}{dV_\theta} = \frac{dV_s}{dF} \cdot \frac{dF}{d\theta} \cdot \frac{d\theta}{dV_\theta}. \quad (9.7)$$
CHAPTER 9. LEG-LOCAL NEURAL MECHANISMS FOR SEARCHING AND LEARNING ENHANCE ROBOTIC LOCOMOTION

$F$ is the 10-bit integer representation of the leg strain that the microprocessor reads, and $\theta$ is the rotation of the CTr joint. All of these values are empirically determined based on the robot and its sensors. Appendix 9.8 gives the values used.

Combining all of these effects, Equation 9.5 can be expressed as

$$V_{\theta,i+1} - V_{\theta,i} = \frac{k_{\text{comp}} \cdot k_{\text{int}} \cdot k_{\text{mem}}}{dV_s/dF \cdot dF/d\theta \cdot d\theta/dV_\theta} \cdot (V_s^* - V_{s,i}),$$

(9.8)

where $k_{\text{eff}} = (k_{\text{comp}} \cdot k_{\text{int}} \cdot k_{\text{mem}})/(dV_s/dF \cdot dF/d\theta \cdot d\theta/dV_\theta) \in (0, 2)$. If this condition is met, the robot stably adjust its CTr depression to obtain the commanded leg strain. This corresponds to the shaded regions in Figure 9.4. Results from the robot are presented in the next section.

9.4 Results from Mechanism 1: Learning Proper Leg Depression

MantisBot can actively adapt its stepping motion to obtain the commanded leg strain. Figure 9.3 shows two thirty-second trials of robot stepping, and how the CTr depression changes to achieve the intended leg strain in stance phase. When $0 < k_{\text{eff}} < 1$, the leg learns the proper height, but the response is overdamped, slowly approaching the intended force from one side. When $1 < k_{\text{eff}} < 2$, the leg also learns the proper height, but the response is underdamped, overshooting the intended value before settling. From these plots, it is also evident that changes in the Stance Depression neuron’s voltage correspond to changes in Leg Strain, and that the Stance Depression neuron’s voltage does not change at all during swing phase. The plots in the bottom row summarize the data by plotting each subsequent step’s mean leg strain in staircase form, illustrating how the leg strain approaches the commanded value as the robot steps.
$k_{eff}$ accurately predicts the dynamics of learning the proper leg depression. Figure 9.4 illustrates how our simple model predicts the learning dynamics should change as $k_{eff}$ and the Reference Strain neuron’s voltage are varied. Isoclines for $G_{syn}$, which is proportional to $k_{int}$, are drawn. They curve because as the Reference Strain neuron’s voltage increases, the Too Much and Too Little neurons’ effective conductance increases from the synaptic input, making them less sensitive to inputs (Chapter 4, [Szczecinski et al., 2017c]).

Six parameter value combinations were tested, labeled with points A-F. For each, the simple model’s prediction (i.e. Equation 9.8) is plotted above the data from the robot. In every case, the simple model captured the dynamics of the robot’s behavior. When the model predicts stable learning, the leg strain evolves almost identically (Figure 9.4A-D). When the model predicts instability, the leg strain does not evolve identically. When marginally stable, the leg strain changes apparently at random, presumably because a marginally stable control system exerts no “restoring force”. When unstable, the leg strain grows with each subsequent step, although it does not grow as quickly as the model predicts. The simple model enables us to constrain parameter values to produce stable and predictable onboard learning. This system for stable learning enables MantisBot to actively adapt its locomotion to the environment. Figure 9.5 shows data from two 60 second trials of robot stepping. In Figure 9.5A, the leg sought the same leg strain with each step, as the height of the ground changed. In the first step after the ground height was changed, the force is smaller or larger than commanded, but it quickly converges with the subsequent step. In Figure 9.5B, the ground height was constant, but the reference strain for each step was altered. As in A, the CTr depression adjusts to produce the intended leg strain in stance phase.
Figure 9.3: Two examples of how leg strain evolves as MantisBot steps. A) If the learning mechanism is overdamped, then the leg strain slowly approaches the intended value as the robot steps. B) If the learning mechanism is underdamped, then the leg strain oscillates about the intended value as the robot steps. The learning is stable in both A and B because $k_{eff} \in (0, 2)$. The bottom row shows plots used in Figure 9.4 and how they are produced from raw data.
CHAPTER 9. LEG-LOCAL NEURAL MECHANISMS FOR SEARCHING AND LEARNING ENHANCE ROBOTIC LOCOMOTION

Figure 9.4: Our simple model of leg strain learning predicts results from Mantis-Bot. The top plot shows isoclines of learning speed synapse strength $G_{syn}$ (labeled in Figure 9.2A3) as the stability criterion (x axis) and intended force (y axis) change. Shading corresponds to different qualitative response types. Points A and B exhibit overdamped dynamics, slowly approaching the intended force. Point C exhibits critically damped dynamics, approaching the intended force in one step. Point D exhibits overdamped dynamics, which stably oscillate about the intended force. Point E exhibits marginally stable dynamics, showing no apparent attraction to the intended force.
Figure 9.4: (contd.) Point F exhibits unstable dynamics, with the error growing each step. These results validate our use of the simple model to predict the speed and stability of the robot’s force learning.

9.5 Results from Mechanism 2: Searching for Ground Contact

Our model of a leg-local command neuron for searching produces successful searching in MantisBot, and is largely consistent with what is known about how insects control searching behavior. Figure 9.6 shows data from the robot showing how the I4 neuron controls searching. At $t = 5$ s, the robot’s leg is suspended in the air. It is not in a walking state. At $t = 15$ s, descending commands are given to step sideways. This corresponds to the brushing of the abdomen used to get stick insects to walk or search for a foothold [Berg et al., 2015]. This causes the CTr to depress (positive rotation), disinhibiting the Search IN neuron. Because no leg strain was detected, the Search IN neuron remains depolarized, adding drive to the CPG and causing FTi flexion (negative rotation). At $t = 20$ s, searching is halted by hyperpolarizing the I4 neuron. In our model, I4 disinhibits the Search IN, so inhibiting I4 desensitizes Search IN to inputs. Thus, the CPG drive decays, halting oscillation. At $t = 40$ s, the hyperpolarizing current stops, and searching behavior continues. This is consistent with experiments with stick insects which found that I4 hyperpolarization always halts searching, but depolarizing may either increase the likelihood or cause searching [Berg et al., 2015].

9.6 Results from Mechanisms 1 and 2

Mechanisms 1 and 2 work together to produce robust stepping in extreme scenarios. Figure 9.7 shows data taken from MantisBot as it transitions between searching and walking as the ground height is changed. At $t = 5$ s, the intended CTr depression
Figure 9.5: MantisBot can adapt its locomotion to obtain the intended leg strain. A) MantisBot can adjust CTr depression to obtain the commanded leg strain as the ground height is changed under the leg, suggesting that the robot can adapt to irregular terrain. B) MantisBot can adjust CTr depression to obtain the commanded leg strain as the command changes, suggesting that the robot can alter its posture to accomplish tasks that change leg strain, such as climbing.
Figure 9.6: Our searching network replicates neural data from searching stick insects. Initially the leg is suspended above the ground. When the command to walk is given (triangles on top), the leg begins to search for ground contact. When the I4 neuron is hyperpolarized, searching stops. When depolarized, searching resumes.
is very small, meaning that the foot remains high above the ground when it expects to enter stance phase. Therefore, the I4 neuron is activated, and the Search IN is disinhibited. This neuron ensures that the CPGs receive enough drive to oscillate, coordinates the FTi flexion at the end of CTr depression, and increases the Stance Depression voltage, lowering the foot toward the ground with each cycle.

At approximately $t = 17$ s, the foot makes ground contact and walking begins. After two steps, at $t = 20$ s, the ground height is raised, increasing the leg strain, and causing the Stance Depression neuron’s voltage to decrease. After two steps, at $t = 27$ s, the ground is drastically lowered. This again activates searching. At approximately $t = 35$ s, the foot again makes ground contact, and begins normal walking. The leg strain immediately converges to the intended value. This demonstrates that these two mechanisms work together to ensure that the leg supports its share of the body weight, whether or not it initially makes ground contact.

9.7 Discussion

In this work, we use our praying mantis model robot, MantisBot, to demonstrate two leg-local force-regulating mechanisms seen in arthropods. Mechanism 1 uses a local learning mechanism to adjust a leg’s depression force based on the strain sensed in the leg segment [Libersat et al., 1987, Zill and Seyfarth, 1996]. Mechanism 2 activates a searching behavior if no ground contact is detected after depressing the leg [Berg et al., 2015, Bläsing and Cruse, 2004]. Together, they enable the robot to adjust to changing terrain by modifying motion based on detected forces.

The strength of Mechanism 1 is derived from our analysis of its stability. First, we proved that we could construct a memory network to remember the leg depression from step to step. Such memory is necessary for a learning process like Mechanism 1. Second, we constructed a simple model of the learning process, and showed that the
Figure 9.7: Both mechanisms work together to produce adaptive stepping. The leg begins above the ground, and searches for ground contact. Once contact is made, CTr depression is adjusted until the intended force is obtained. When the ground height is raised, depression changes again. When the ground is lowered, the leg searches until ground contact is made. The stepping leg rapidly adjusts CTr depression to obtain the intended force.
stability of the simple model predicted the stability of the robot’s learning. Then, we used properties of the simple model to tune connections within the network, enabling MantisBot to adjust its stepping motion to produce the commanded leg strain, i.e. ground reaction force, within one step.

Mechanism 2 derives its strength from detailed study of individual nonspiking interneurons (NSIs) in the thoracic ganglia of stick insects [Berg et al., 2015]. We implemented an NSI whose function mimics I4, an inhibitory NSI that functions as a leg-local command neuron for searching behavior. Like the actual I4 neuron, our command neuron’s membrane voltage oscillates in phase with joint rotation, hyperpolarizes in stance phase, and remains persistently depolarized during searching. Hyperpolarizing it prevents the leg from searching, even if environmental conditions should cause searching. Having such a detailed blueprint from the animal enables our control network to implement searching behavior like that seen in the animal, producing adaptive locomotion. In addition, our detailed network model helps refine neurobiological hypotheses; the original researchers suggested that leg strain may inhibit other neurons in the searching control network downstream of I4 [Berg et al., 2015], which our network and results show is plausible.

Both of these mechanisms were implemented as small (fewer than 10 neurons) additions to the locomotion framework we have presented previously (Chapter 6, [Szczecinski et al., 2017a]). The structure we have implemented is based both on previous neurobiological [Akay et al., 2004, Büschges et al., 1995, Hess and Büschges, 1999] and modeling [Knops et al., 2012, Toth et al., 2012, Szczecinski et al., 2014] studies. The work presented in this paper demonstrates the flexibility of this structure, since much of the network is reused, for example, for searching motions. It is also simpler than some alternative approaches, which use large, randomly connected networks to produce the same behavior [Dasgupta et al., 2015]. As more neurobiological data becomes available, we will continue to refine this model.
9.7.1 Source of commanded leg strain

In this work, the reference leg strain was set directly by a tonic stimulus to the Ref. Strain neuron. Where might this signal come from in the animal? In crabs, increasing the apparent force on one foot causes the ipsilateral legs to increase their depression [Libersat et al., 1987]. There may be a thoracic mechanism to distribute the total detected weight among all of the legs.

Another possibility is that force distribution is centrally mediated. Drastic changes in posture, such as climbing over obstacles or walking on vertical surfaces [Noah et al., 2001], change the force that the legs should expect to support, and thus, the motor output. The activity of populations in the central complex of insects precedes changes in walking speed, heading, and body pitch. Stimulating these areas modify joint-level reflexes [Martin et al., 2015]. It is possible that descending commands from the brain anticipate the leg loading necessary to perform a specific locomotion task, and modulate the reference strain accordingly. Our future work will investigate whether such a centralized mechanism is necessary, or if the same performance can be accomplished purely with local mechanisms.

9.7.2 Where does CPG drive come from?

CPG drive is often provided directly by the brain in computational models [Daun-Gruhn et al., 2009, Knops et al., 2012, Szczecinski et al., 2014, Toth et al., 2012]. This drive bifurcates the CPG from a bistable element to an endogenous oscillator. Such a model is logical, because brain centers control the rate of stepping [Bender et al., 2010]. However, if local interneurons can function as command neurons and halt oscillation, the system must be more complicated than the brain activating the CPGs. In an earlier version of the presented model, CPG drive was provided directly by descending commands. In this case, however, hyperpolarizing I4 did not halt searching as seen in the insect [Berg et al., 2015]. This is not only biologically inaccurate, but
also prevent I4 from acting as a true command neuron in our network.

In the presented version, we resolved this issue by using feedback from the Leg Strain neuron to set the CPG drive, since feedback from load sensors is known to contribute to CPG drive in the posterior legs of walking stick insects [Borgmann et al., 2009], and dissected insects could not produce full searching motor output [Berg et al., 2015]. Using leg strain to drive the CPGs prevented them from oscillating unless the foot could reach the ground. The Search IN also contributed to CPG drive, enabling stable oscillation when the foot could not reach the ground and the leg was searching. Hyperpolarizing I4, however, deactivated this pathway, and the CPG drive decayed to zero, halting oscillation and searching. This enabled us to directly replicate the results from the animal.

With this in mind, it seems plausible that descending commands about body motion modulate sensory pathways that determine CPG drive. This seems necessary for true “entrainment” of the animal’s nervous system with its body’s dynamics and the environment, and has been observed in CPGs in locusts [Möhl, 1988]. In the future, we plan to use a memory network like that presented in Section 9.3.4 to set the CPG drive. We will use the memory network to count the rising edges of strain signals and CPG signals, and use the difference to tune CPG drive to the ongoing motion. In this way, the CPG is actively predicting when the next sensory signal will arrive, and can activate corrective reflexes if it does not arrive. Descending commands will gate the incoming sensory information, enabling the CPG to shut itself off when sensory rising edges are gated.

9.7.3 Searching with synergies

In this work, the controller for the CTr joint (i.e. the levator-depressor) adapted its range of motion to support a specific amount of body weight. This approach, however, led to some awkward leg postures, since the range of motion of the other leg joints
assumes a specific CTr motion. Although the details are not well understood, animals appear to simplify motor control by grouping muscles into synergies, which move the leg or foot in a particular direction when coactivated. For example, straining the campaniform sensilla on the tarsi of insects activates a muscle synergy that supports body weight [Zill et al., 2015]. Our leg controller uses the body’s intended linear and rotational speed to adjust the range of motion of the joints, producing de facto muscle synergies. In the future, we plan to add a synergy for controlling body height, in which case the leg can activate motor neurons throughout the leg until the commanded strain is achieved.

9.7.4 Walking with all legs

The presented work deals only with intra-leg control. This work will need to be expanded to all of the legs to enable MantisBot to walk freely. Even though some of our work regarding intra-leg control has been successfully applied to all of MantisBot’s legs (Chapter 7, [Szczecinski and Quinn, 2017c]), future work will require coordinating the timing of stepping between the legs, adding ascending pathways to halt walking if one leg is searching for ground contact, and distributing forces between the legs.

Proper timing of the stepping of each leg is necessary for effective locomotion. Many behavioral rules that coordinate the timing of inter-leg stepping have been described, although the underlying neural mechanisms are unknown [Cruse, 1990]. Neuromechanical models have suggested that inter-leg CPGs may communicate via direct communications, gated by sensory feedback [Daun-Gruhn, 2010]. We could try a similar structure for MantisBot, and incorporate other behavior rules as necessary.

Some inter-leg neural pathways have been discovered, although these appear to control stepping frequency rather than stepping phase between the legs. If the front leg of a stick insect walks on a treadmill, then rhythmic activity is elicited in the mesothoracic ganglion, and motor neuron voltages generally increase in the metatho-
racic ganglion [Borgmann et al., 2009]. This suggests that the legs coordinate walking speed with a network like we implemented in this paper, where load feedback changes CPG drive, or like we proposed earlier in the discussion, in which CPG drive is adjusted based on the difference in stepping speed. Future work will attempt to find the simplest network that produces proper coordination, and will serve as a hypothesis for future neurobiological experiments.

Proper inter-leg coordination is especially important when legs stop walking to search for ground contact. The front legs of stick insects play a large role in determining where the others legs should step, frequently disrupting the walking gait to find footholds [Bläising and Cruse, 2004, Grabowska et al., 2012]. This has two implications for control: first, each leg must be able to halt the body’s forward progress if it is searching for ground contact; and second, to minimize time spent searching, each leg must be able to share information about successful steps with posterior legs.

Recent work in Drosophila has shown that descending and ascending pathways work together to control walking direction [Bidaye et al., 2014]. It is possible that a searching leg alters ascending information to prevent the other legs from walking until it has found a foothold. Such searching can be reduced if the ganglia share information about successful steps with one another. It is known that cockroach ganglia can share such information, even when two ipsilateral hemiganglia are separated from the rest of the nervous system [Eisenstein, 1972]. Such a structure reduces dependence on descending signals, and makes the control system more robust to changing environmental conditions.

Proper locomotion in a changing environment requires that the legs share information about loading. As discussed above, individual legs of crabs adjust their depression force based on the strain in all of the legs [Libersat et al., 1987]. We suspect that local rules that change the posture to minimize leg strain subject to kinematic constraints may replicate this result, using only leg-local mechanisms. Models of stick
insect posture have shown that local rules are sufficient to minimize the forces on all of the legs [Lévy and Cruse, 2008], suggesting that this approach may work.

9.8 Appendix

In our system, $dV_s/dF = 20 \text{ mV}/2^{10} = 19.53 \times 10^{-3} \text{ mV}$; $dF/d\theta = 326 \text{ rad}^{-1}$ (empirical); and $d\theta/dV_\theta = 1.972 \text{ rad}/20 \text{ mV} = 98.61 \times 10^{-3} \text{ rad/mV}$. 
Chapter 10

Conclusion and Future Work

10.1 Conclusion

In Chapter 3, I presented a serial design process with associated tools for selecting parameter values for a posture and locomotion controller for a simulation of a robot. The controller was constructed from dynamic neuron and synapse models. Rather than using stochastic or evolutionary techniques to design a control network, individual components were designed based on biological networks and analyzed via engineering analysis. The result is a network whose components have guaranteed stability and robustness, and whose structure has biological meaning, rather than a black-box network.

The parameter value design process was intended to run on a desktop computer, and has three steps. First, the tool feedbackDesign uses classical control methods to find neural and synaptic parameter values that stably and robustly control servomotor output. Next, the tool cpgDesign generates bifurcation diagrams and phase response curves for the CPG model. Finally, to understand sensitivity to parameters and how descending commands affect a leg’s stepping motion, my tool simScan runs batches of neuromechanical simulations with specified parameter values, which is useful for
searching the parameter space of a complicated simulation. These design tools were demonstrated on a simulation of a robot, but are applied to MantisBot’s controller throughout the rest of this thesis.

In Chapter 4, I presented a methodology for tuning functional subnetworks for SNSs, which can be assembled into larger networks. Specifically, I presented networks that perform addition, subtraction, multiplication, division, differentiation, and integration of incoming signals. Starting with the operating range of neural activity in the network, $R$, the gain of the operation, $k$, and bounds based on biological values, parameters could be constrained to produce one functional set of values. Assembling large networks from functional subnetworks that are tuned analytically underpins the results from the rest of this thesis (Chapters 6 through 9), and further demonstrates how my methods of SNS assembly enable the neurobiologist to infer the function of networks based on their structure.

In Chapter 5, I presented Mantisbot robot with which I tested the SNS models in this thesis. This chapter summarized the mechanical, electrical, and software design of the robot, including the proportions of the body, actuators, sensors, and control software.

In Chapter 6, I presented the basic SNS model for controlling the leg joints that produces directed stepping, on which all of the subsequent chapters built. In addition to low-level networks, I used the literature on mantis prey tracking to design a hierarchical, distributed neural controller that establishes the goal, coordinates different joints, and executes prey-tracking motion. The descending commands were simple, indicating only 1) whether the robot should walk or stand still, and 2) the intended direction of motion. Each joint’s controller used the descending commands differently to alter sensory-motor interactions, changing the sensory pathways that coordinate the joints’ CPGs into one cohesive motion. Experiments with one leg of MantisBot showed that visual input produces simple descending commands that alter walking
kinematics, change the walking direction in a predictable manner, change the direction of reflexes when necessary, and can control both static posture and locomotion with the same network. This chapter is evidence that such structure may underlie the adaptive posture and locomotion observed in insects. In addition, this chapter demonstrates that a large, flexible, and capable low-level network is a logical basis for the development of a brain controller, which modulates the function of the low-level networks to alter behavior.

In Chapter 7, I successfully applied an automated method to tune the control network for all of MantisBot’s legs. This process is completed only once, after which the legs can direct stepping into whatever direction the brain specifies. Each leg had a controller like that in Chapter 6, tuned to the morphology of that leg. This chapter both suggests neural mechanisms underlying observations from walking insects, and demonstrated the broad applicability of this control method for robots.

In Chapter 8, I demonstrated and analyzed how CPGs could entrain MantisBot’s joints to positive velocity feedback resulting in a change of a leg’s stance phase duration. I used a model of a single leg segment, as well as previously presented design techniques to understand how the gain of positive velocity feedback to the CPGs should be modulated to successfully implement the change in walking duty cycle. These results suggest that the feedback simplifies the descending control of walking speed, naturally producing the asymmetrical changes in stance and swing phase duration seen in walking animals. I implemented the AR in neural circuits of a dynamic network that controlled the leg joints of MantisBot, and experiments confirmed that the robot modulated its walking speed as the simple model predicted it should. Aggregating the data from hundreds of steps in different walking directions showed that the robot changed speed by altering the duration of stance phase while swing phase remained unaffected, as seen in walking animals. This chapter is yet another example of how a more competent low-level controller simplifies the descending commands.
Finally, in Chapter 9, I implemented two leg-local neural mechanisms with MantisBot, enabling it to adjust motor activation while walking based on detected forces. Mechanism 1 used negative feedback of leg depressor force to ensure that each stance leg supported an appropriate amount of the body’s weight. Mechanism 2 encouraged searching for ground contact if the leg supported no body weight. Adding these mechanisms only required adding new network components to the basic stepping controller. These additional pieces enabled each joint to adjust how it interprets descending commands, depending on sensory information in the leg. This represents the first step toward a truly adaptive SNS, in which segments of the system (i.e. a single leg) uses a combination of descending commands and sensory information to make decisions about what to do next. Such a structure exploits the distributed network to its fullest extent, enabling each leg to perform its own computations, subject to both body-level commands and leg-level sensory information.

**Importance of Morphological Approach**  In the introduction I contrasted morphological and functional models of the nervous system as controllers for robots. Specifically, I spurned the functional approach for producing black-box controllers that do not provide any insight into neurobiology. However, if the goal is to control a walking robot in the field in the short term, then the functional approach may yield results faster.

In the long term, it is hoped that continuing the approach presented in this thesis will lead not only to increasingly capable robots, but also a deeper understanding of how the nervous system works. Indeed, this work intertwines these goals. The networks in Chapters 6 through 9 have clarified how the thoracic ganglia may process descending and sensory information by showing that they can produce directed stepping that adapts its speed and force based on sensory feedback, while incorporating as
many networks from the literature as possible. Therefore, continuing to build robots with SNSs may directly contribute to both neurobiology and robotics.

Indeed, using a biological network to control movement in an agent like a robot is a critical test of a model. The more realistic the model, the more thorough the test is. A possible series of models, in order of increasing realism, could be 1) Open-loop actuation of a simple model, 2) Closed-loop actuation of a mechanical model, 3) Closed-loop actuation of a mechanical agent in a lab setting, 4) Closed-loop actuation of a mechanical agent in a real-world setting. Many models, such as those from Silvia Daun-Gruhn and Einat Fuchs only complete levels 1 or 2, depending on the study. All of my results, however, come from level 3. This has introduced many more problems not present in a mathematical model, such as uncertain mechanical parameter values, delay, noise, and vibration. Confronting these issues have required that I incorporate adaptation and memory, filters, and efference copy. Working with a robot has also introduced opportunities for exploiting these issues, such as force-based modulation of motor output. These have led to more interesting experiments and conclusions about the details of motor control, as presented in Chapters 8 and 9. In order to better understand additional adaptive mechanisms in the future, SNSs will need to continue to be tested onboard robots.

Changes in Network Structure Between Chapters Throughout this thesis, diagrams of network structure change (compare Figures 3.1, 6.5B, 7.1B, 8.2, and 9.2). As the network became more sophisticated, more basic structures remained, while additional components were added. For example, Figure 3.1 has a joint with a CPG, and a MN for driving the joint in either direction. Figure 6.5B expands this by adding structures to change the amplitude and timing of the joint’s rotation (lines i. to iv.). Figure 8.2 reintroduces positive velocity feedback to enable the CPG to asymmetrically alter its duty cycle. Finally, Figure 9.2 adds structures for force
control and searching, which modify how the basic network functions. Just as in biological evolution, the most essential network structures were preserved, but more components were added to achieve more specific adaptations.

Figure 10.1 clarifies this point by showing the joint control network from Figure 6.5 with color-coded boxes around functional subnetworks from Chapter 4. Thus, functional segments of the network can be identified in other network figures. The top of the network maps the body heading into the language of the joint, establishing the PEP and AEP (black rectangle). The PEP and AEP are compared to one another (blue squares), and the comparison is used to control the gain of the load pathway (violet rectangles) to the CPG (orange square). The CPG’s output is also modulated by the mapping network, controlling the amount of extension and flexion (violet pentagons). The resulting motion is compared to the PEP and AEP to tell the other joints in the leg where this joint is (blue octagon). The other versions of this network simply add more subnetworks from Chapter 4 to accomplish the goal of that chapter. This modular approach helps infer the meaning of neurobiological structures, and provides concrete examples of the types of networks to look for in animals.

10.2 Future Work

This thesis may serve as the launch point for several projects in neurobiology, computational modeling, and robotics.

10.2.1 Neurobiology

The leg controller is insect-like, and can be expanded to have more capabilities, like in Chapter 9. We may be able to add more such behaviors with similar, small changes. For instance, we could implement an elevator reflex to step over obstacles. We could use a network similar to that in 9, such that whenever ThC motion is impeded in
Figure 10.1: Joint control network from Figure 6.5 with functional subnetworks boxed, to emphasize the functional structure of the network.
swing phase, the CTr levation is increased by some amount, and saved in an integrator network. Repeated collisions with an obstacle would result in higher and higher levation, eventually clearing the obstacle.

Such adjustments could then be shared with posterior legs via intersegmental connections. We could test foot-placement reflexes [Eisenstein, 1972, Cruse, 1990], in which the protraction or levation height of the anterior leg is fed to memory units in the posterior leg, enabling it to adjust its stepping without going through the same adjustments. Also, descending commands get sent through the prothoracic ganglion, meaning that the front legs are the most volitional [Grabowska et al., 2012], and sensory information from the front legs controls how the posterior legs move [Borgmann et al., 2009]. Because our framework is so insect-like, and we have tools for directly tuning functional networks, we can rapidly test hypothetical networks to reproduce these biological results.

Another form of intersegmental communication is the control of step timing (i.e. gait). Neural data from reduced preparations with stick insect suggest posteriorly-directed communication in which sensory feedback from the anterior leg excites locomotion centers in the posterior leg [Borgmann et al., 2009]. Sensory feedback from the leg itself then interacts with feedback from the anterior leg, adjusting its stepping phase into a “useful” pattern (i.e. not in phase with the anterior leg) [Borgmann et al., 2011]. These observations are not in line with behavioral data from insects and other arthropods, which suggest a number of anteriorly-directed coordination rules [Cruse, 1990]. Is it possible that the networks uncovered by Borgmann et al. can be tuned to produce behavior like that observed by Cruse? To what degree do the mechanics of the robot, that is, the changing strain in stance-phase legs while other legs enter swing phase, affect coordination [Zill et al., 2011]? With this model in place, we can continue to make small additions and build hypothetical networks anchored in what is known about the neurobiology, and tuned to reproduce behavioral observations in
animals.

10.2.2 Modeling and Network Tuning

The next step for testing this thesis as a model of insect locomotion is to use the techniques to design controllers for other robots or insect models of varying morphology. The kinematicOrganism toolbox was written to automate the construction and tuning of the controller in Chapters 6 and 7. My mentee Scott Rubeo has used kinematicOrganism to produce a cockroach model capable of walking at cockroach-like speeds after only making a couple changes to the program. However, some issues did arise, especially to interface between the CPGs and muscles. Before applying this method more broadly to model other animals, kinematicOrganism needs to be enhanced with muscle feedback pathways that can enforce asymptotically stable position control, with variable velocity control, like MantisBot’s servomotors. This will likely require expanding the SNS to include a subesophageal ganglion, which controls muscle tonus and reflexes [Roeder, 1937, Ridgel and Ritzmann, 2005, Mu and Ritzmann, 2008], as well as common inhibitory motor neurons to modulate contraction force [Pfüger and Sillar, 2013].

More detailed control of thoracic networks will likely require a brain as well. There is an ever-increasing base of knowledge about the connectivity schemes of the central complex (CX) [Heinze and Homberg, 2008, Pfeiffer and Homberg, 2014, Wolff et al., 2015, Turner-Evans and Jayaraman, 2016]. The CX itself only has around 3,500 neurons [Heinze and Homberg, 2008], a large network to tune, but maybe manageable if it were assembled from functional subnetworks as in Chapter 4. Other studies have had success in tuning large networks when they included smaller networks that performed specific roles, such as comparison or memory [Haferlach et al., 2007, Pasemann et al., 2001]. Would it be possible to establish functional subnetworks, such as individual rows and columns in the CX, connect them according to data that
exists [Pfeiffer and Homberg, 2014, Wolff et al., 2015], and tune these connections to produce descending signals for navigation [Webb and Wystrach, 2016]? Such a process would be a huge undertaking, and would benefit from the collection of species-specific behavioral and neural data. Starting with the work in this thesis would also benefit such a model, because now the modeler could make detailed hypotheses about how CX activity results in descending commands that modulate thoracic network activity.

10.2.3 Robotics

One of the potential benefits of using an SNS to control a robot is its parallelization. Our tests so far have suggested that running the SNS is not computationally demanding (Chapter 7.3.1). It should be possible to use a small single-board computer to run this system, especially if it is parallelized across several boards. We are currently setting up AnimatLab to run on an Odroid XU4 single-board computer. The XU4 is ideal for mobile parallelized computing, because it has an 8-core CPU and an 8-core GPU, fits inside a 83 mm by 58 mm by 20 mm envelope, and only uses 60 W of power at peak consumption. Such a small package, combined with the network tuning methods from Chapters 3 and 4, may make this SNS technology a practical method for controlling walking robots in the future.

Before MantisBot can walk freely, more detailed posture control must be implemented. It is known that insects can alter the balance of negative velocity and negative position feedback at their joints to switch from active, position-based posture to more passive, velocity-based posture [Sauer et al., 1997]. Such a mechanism would enable a robot to distribute its weight over all of its legs on an uneven surface, which would be important for a robot in the real world. When walking, positive velocity feedback helps insects avoid internal joint torques, which arise because of their redundant, multi-legged posture [Schmitz et al., 2008]. Because our network already calculates joint rotation velocity, adding such a reflex should not be difficult.
However, before implementing such a reflex on MantisBot, the stability of the system would need to be analyzed by revisiting the calculations in Chapter 3.


