A Computational Model of a Behaviour in *C. elegans* and a Resulting Framework for Modularizing Dynamical Neuronal Structures

by

Chris J. Roehrig

B.Math (1988), M.Math (1991), University of Waterloo

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

THE FACULTY OF GRADUATE STUDIES

(Program in Neuroscience)

We accept this thesis as conforming
to the required standard

The University of British Columbia

September 1998

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science / Neuroscience

The University of British Columbia
Vancouver, Canada

Date Oct 14/98

Abstract

The work presented in this dissertation grew out of a study of a physiologically based computational model of the tap withdrawal response in the nematode *Caenorhabditis elegans*. A computational model using all available anatomical and physiological data was unable to explain a dynamic property of the circuit: the ability of the behaviour to continue after the termination of the stimulus. To account for this behavioural observation, a novel approach was taken: a neuronal circuit was engineered from a set of modules each consisting of several physiologically realistic model cells. The mathematical dynamics of the resulting neuronal circuit produced an output that was similar to the behaviour observed in the intact worm and shows that neuronal network dynamics could account for the behaviour.

In the course of this study, it became clear that little is known about the modular properties of neuronal dynamics. This dissertation presents an approach for combining non-linear neuronal circuits into larger systems using dynamical modules (dymods), and a set of tools for studying dymods, and discusses a research strategy for studying the modular properties of neuronal dynamics.

Contents

List of Tables

List of Figures

List of Symbols

C_m	Membrane capacitance	17
d	Process diameter	16
E_{SYN}	Equilibrium (reversal) potential for synaptic channels	19
E_{SYNij}	Synaptic reversal potential for current I_{ij}	23
E_{ACTj}	Presynaptic activation potential for cell j	40
E_{LEAK}	Membrane leakage equilibrium potential	17
E_{RANGE}	Presynaptic activation range (mV)	22
$\bar{g}$	maximal post-synaptic membrane conductance	22
$\hat{g}_{ij}$	Gap junction conductance between cell i and cell j	18
g_{LEAK}	Membrane leakage conductance ($g_{LEAK} = 1/R_m$)	40
$g_\infty(V_{PRE})$	Steady-state post-synaptic conductance	21
$g(t)$	Synaptic conductance	19
g_{ij}	Synaptic conductance to cell i from cell j	23
I_{INJ}	Externally injected current	17
I_{INJ_i}	Externally injected current into cell i	23

w_{ij} Dimensionless weight for synapses into cell i from cell j 40

Acknowledgements

I'd like to thank my PhD supervisory and examination committees for their insightful remarks and guidance: Leah Keshet, Steve Kehl, Mark Greenstreet, Nick Swindale, and especially Jim Little and Cathy Rankin for footing the bill.

In addition, my previous advisors Robert Miura, Bob Woodham, Dinesh Pai and the Computer Science Department's Laboratory for Computational Intelligence were instrumental in the inception of this work.

I'd like to acknowledge Jennifer Enns-Ruttan, Bard Ermentrout and Randy Beer for helpful discussions in the construction of the dymod circuit of Chapter 3. Shawn Lockery provided useful advice and experimental evidence to support the physiological model, and Tom Ferrée and Tom Morse gave feedback on the model and the robotic implementation tools. I owe thanks to Steve Wicks for starting on the path to modelling the *C. elegans* tap withdrawal circuit and working with me through to its publication, and Bruce Hutcheon for his participation and insight.

Life would have been a lot tougher if not for my esteemed fellow House.ORGians: Tom Glenne, Yggy King, Brenda Mary Haggard, Hanan Elmasu, Vicki Sawyer, Sabine Tamm, Glenn Wells, and Jackie Taylor. A special thanks to Hugh Thompson for those late-night eye-scorchers that I hope will be remain the

extremes of my terror, and Fiona Miller whose friendship made the first years here bearable. To Mike Horsch whose philosophical discussions have inspired me probably more than anything else. To my family, Mom, Dad, Andy and Lisa for being there for me, and to my soulmate Christina Pechloff: your support made finishing this thing easier than I could have imagined.

CHRIS J. ROEHRIG

The University of British Columbia

September 1998

This thesis is dedicated to my supervisor, Cathy Rankin.

Without her unfailing support through many trials,

this thesis would never have been completed.

Chapter 1

Introduction

The brain has a rich dynamical structure. We already have a good idea of the complexity of its physical structure, its molecules, channels, cells, nerves, and nuclei, but we have only begun to understand the dynamical structure that emerges from the interactions among these components. In this thesis, the distinction is made between dynamical structure and dynamic structure, i.e. structure that is merely changing over time. An example of dynamic structure is an ion channel: a cell membrane protein that can change its molecular configuration to allow current to flow through. Rather, a dynamical structure may not exist as a physical entity but rather it only *emerges* from patterns of changes. For example, in pattern generators such as the lobster stomatogastric ganglion (Harris-Warrick, Nagy and Nusbaum, 1992), current flowing through ion channels can ultimately affect other connected cells and result in periodic patterns of activity — oscillations — among the cluster of cells. These oscillators are not physical structures: they exist only in the patterns of changes in the underlying physical structures.

Pattern generators like these are the simplest examples of dynamical structures, but there are likely to be many levels of more complex dynamical structures. Patterns can emerge in the way that dynamical structures themselves change — patterns of change in the patterns of change, and so on. These dynamical structures do not necessarily correspond to any physical structures and may not be readily apparent by examining physical structure, as this thesis shows. There is a growing appreciation among neuroscientists (Chiel and Beer, 1997) and cognitive and behavioural scientists (Port and van Gelder, 1995; Kelso, 1995) that understanding the dynamical structures that emerge from the brain's physical structure is the key to understanding how the brain generates behaviour.

The dynamical study of neuronal circuits is still in its infancy, and has concentrated on small systems. An interesting model system for studying small circuits is the soil nematode *C. elegans*. Its nervous system contains only 302 neurons all of which have been completely described in terms of their location and synaptic connectivity (White, Southgate, Thomson and Brenner, 1986; Hall and Russell, 1991; Achacoso and Yamamoto, 1992). Moreover, the system is amenable to laser microsurgery so that it is possible to destroy individual neurons in the living animal with little or no damage to the remaining nervous system (Chalfie, Sulston, White, Southgate, Thomson and Brenner, 1985; Wicks and Rankin, 1995). In spite of this simplicity, it has a rich behavioural repertoire and has been chosen as a model system for studying learning and memory (Rankin, Beck and Chiba, 1990) and genetics (Wood, 1988). *C. elegans* has been studied in sufficient detail that it is feasible to construct a cellular account of a non-trivial behaviour.

One such behaviour that has been studied in detail is the tap withdrawal reflex in which the nematode swims backwards in response to a vibratory tap stimulus to the agar jelly on which it locomotes (Rankin et al., 1990). The circuitry underlying this tap withdrawal response has been well-studied (Chalfie et al., 1985; Wicks and Rankin, 1995). It consists of a sensory and interneuronal subcircuit that processes potentially conflicting mechanosensory stimuli to arrive at a graded decision output (Rankin, 1991). Locomotion is mediated via two command interneuron pairs called AVA and AVB which drive two independent motor systems for reverse and forward sinuous locomotion respectively (Chalfie et al., 1985; White et al., 1986).

We have previously constructed (Wicks, Roehrig and Rankin, 1996) a detailed computer model using all available anatomical and physiological data to predict the functional polarities of the synapses in the *C. elegans* tap withdrawal circuit. These predictions were made by optimizing the behaviour of a modelled circuit when various cells are removed to the tap withdrawal behaviour of real animals when the corresponding cells are removed using laser ablation. Our model accounts for how the tap response resolves the conflicting stimuli to head and tail to arrive at a decisive response, and when neurons are removed from the model, it responds in a way that is commensurate with the response of the animal when the corresponding cell is ablated. Although this model was useful for predicting the polarities of the synapses in the circuit, it was unable to account for the continuation of the reversal behaviour when the stimulus ended.

1.1 Thesis Statement

The hypothesis of this thesis is that network dynamics could account for reversal maintenance. While the anatomical map of the nervous system is known, the connectivity alone is not sufficient to determine the dynamics of the neuronal circuitry. In-circuit recordings are not yet possible in the animal, and so the cellular patterns of activity during behaviour are still largely unknown. Therefore, a novel approach was taken: a neuronal circuit was engineered from a set of distinct modules each consisting of several physiologically realistic model cells. The circuit was designed so that its mathematical dynamics produces an output that is similar to the behaviour observed in the intact worm. The novelty of this approach is the way that a human engineer was able to design the circuit dynamics in parts to simplify the process: the dynamics of each module were designed separately using model cells, and the modules were then assembled to form the complete circuit with the desired dynamical behaviour. The term "dymod" was coined to refer to these dynamical modules.

1.2 Document Overview

This dissertation is organized into the following chapters. Chapter 2 describes the physiological derivation of the mathematical and computational model used in our polarity predictions (Wicks, Roehrig and Rankin, 1996) and in the construction of the neuronal circuit to account for the reversal maintenance behaviour (Chapter 3). Chapter 3 describes the dynamical design of the reversal maintenance

circuit model and introduces the concept of dymods. Chapter 3 also discusses a research strategy for studying the interactions between dynamical neuronal structures using dymods. Chapter 4 presents the Digital Signal Sockets (DSS) network protocol — a preliminary implementation framework for dymods that emphasizes their modular aspect by requiring dymods to have a clearly defined interface, and uses computer networks for interconnection. Finally, Chapter 5 presents another key component in the dymod research strategy: a desktop robot system for performing neuroethological experiments to investigate how dymods can be used to model complete behavioural systems.

1.2.1 A Physiological Model of *C. elegans* Cells

Chapter 2 presents the derivation of the physiologically-based computational model used in Chapter 3 and previously used (Wicks, Roehrig and Rankin, 1996) to predict the functional polarities of the synapses in the *C. elegans* tap withdrawal circuit. The model uses a simple single-compartment model for the nearly isopotential cells and presents a novel graded and tonic synaptic model. The synaptic model emphasizes the distinction between an isolated cell's leakage (resting) membrane potential and its in-circuit steady-state potential which differs because of tonic synaptic currents flowing in the circuit. The model's parameters were derived from all available anatomical and physiological data from *C. elegans* and a related nematode *Ascaris*. While the model successfully predicted the magnitude of the behavioural response in the animal, it did not account for the time course of the animal's reversal behaviour. To account for this discrepancy, a new model

that uses network dynamics to maintain the reversal was proposed, and this is the subject of the next chapter.

1.2.2 Dymods: A Framework for Modularizing Dynamical Neuronal Structures

Chapter 3 presents the design of a circuit to help explain how the nematode *C. elegans* makes transitions between forward and reverse locomotory modes and maintains its reversal for a duration much longer than the time constants of the cells controlling the behaviour. This circuit shows that the maintenance of a reversal could be the result of network dynamics and a bifurcation between two quasi-stable states governing forward and reverse locomotion.

To engineer the circuit, a novel approach was taken: a human engineer used intuition about the dynamical processes involved to hand-engineer a modular set of component dynamical systems ("dymods" — short for dynamical modules) that were assembled to form the final circuit. In spite of the non-linear interactions between dymods, the assembly was remarkably straight-forward suggesting that a human-engineered approach might not be so difficult as it seems.

The reversal dynamics circuit designed in this chapter is a single continuous dynamical system consisting of simple tonic cells. Yet it governs two discrete behaviours and the crisp transitions between them. These behavioural transitions occur as the result of bifurcations in the underlying dynamical system and raises the question of whether bifurcations can account for other behavioural transitions. A research strategy for investigating the generality of this phenomenon is discussed.

1.2.3 The DSS Network Protocol for Dymod Implementation

Chapter 4 presents the Digital Signal Sockets (DSS) network protocol as an experimental framework for interconnecting dynamical modules (dymods). Dymods are continuous dynamical systems with a defined functional interface of input and output signals designed for use in constructing complex neuronal models of behaviours for neuroethological experiments, but may have applications to other control systems. Dymods are implemented as numerical solutions to ordinary differential equations (ODEs) and DSS provides a standard mechanism to interconnect independent real-time dymod implementations together with each other and live motor and sensor signals. The DSS protocol solidifies the modular aspect of a dymod by providing an explicit definition of a dymod's interface inputs and outputs.

DSS is targeted at two network architectures: TCP/IP for inexpensive experiments with low-bandwidth systems, and IEEE 1394 "Firewire" for high-bandwidth applications with guaranteed performance. This chapter discusses the following issues for a dymod implementation: signal and time representation, signal reconstruction, connection management, time synchronization, real-time scheduling, and network transmission. The unoptimized experimental DSS implementation described here is suitable for low-bandwidth signals of less than 50 Hz and suggests the feasibility of using digital networks to interconnect real-time simulations of dynamical systems.

1.2.4 A Desktop Robot System for Experimental Neuroethology using Dymods

Chapter 5 adds to the dymod tools by presenting a robot system for neuroethology experiments. It continues the theme of the previous chapters with the goal of empowering the general neuroscience researcher, in this case to perform robot neuroethology experiments without requiring expertise in robotic engineering or real-time numerical computation.

The inexpensive LEGO robot uses a wheeled-design for simplicity and reliability. It uses a tank-like chassis with treads, which gives it the ethologically relevant ability to orient in place, unlike other car-like designs which require three-point turns. The chassis is compact and houses the battery compartment, motors and computer control system below the tank's "deck" to provide maximum flexibility for adding sensory and actuator apparatus. The control system is a tethered design: an MIT Miniboard monitors and controls the robots sensors and motors and transmits them along a cable to a host UNIX computer which performs the actual neuronal computation. This chapter also presents a Miniboard program (BINMON) that performs the control functions, a UNIX library to communicate with the robot via serial cable, and a library add-on for the popular GENESIS neuronal simulator to allow it to communicate with the robot.

Chapter 2

A Physiological Model of *C. elegans* Cells

2.1 Introduction

This chapter presents the derivation of the physiologically-based computational model used in Chapter 3 and previously used (Wicks, Roehrig and Rankin, 1996) to predict the functional polarities of the synapses in the *C. elegans* tap withdrawal circuit.

2.2 The Model

The nematode *C. elegans* reverses its locomotion and swims backwards in response to a tap stimulus (Rankin et al., 1990), and maintains this reversal for several seconds before resuming forward locomotion. The circuitry underlying this

tap withdrawal response has been well-studied (Chalfie et al., 1985; Wicks and Rankin, 1995). It consists of a sensory and interneuronal subcircuit that processes potentially conflicting mechanosensory stimuli to arrive at a graded decision output (Rankin, 1991). Locomotion is mediated via two command interneuron pairs called AVA and AVB which drive two independent motor systems for reverse and forward sinuous locomotion respectively (Chalfie et al., 1985; White et al., 1986). While the anatomical map of the nervous system is known, the connectivity alone is not sufficient to determine the dynamics of the neuronal circuitry. In-circuit recordings are not yet possible in the animal, and so the cellular patterns of activity during behaviour are still largely unknown.

The model used was a physiologically motivated one. However, in the absence of detailed physiological data from *C. elegans*, it was necessary to make a number of extrapolations from the related nematode *Ascaris lumbricoides*. These assumptions are presented in physiological rather than mathematical form to ensure that they are realistic. Furthermore, preliminary investigations suggested that polarity predictions based on the modelled circuit were more strongly determined by circuit connectivity than the exact values of parameters used. Thus, approximate ranges for these parameters rather than precise values were derived. The effects of varying some of the more uncertain parameters were then assessed by rerunning the same experiments with the values of these parameters varied over three orders of magnitude.

The circuitry was constructed by extracting connectivity data from AY's Neuroanatomy of *C. elegans* for Computation (Achacoso and Yamamoto, 1992).

This data indicated not only the presence or absence of a set of synaptic contacts between a pair of neurons, referred to here as a synaptic class, but also incorporated the actual number of documented electrical and chemical connections within that synaptic class. Each synaptic contact within a class of synapses was assigned the same reversal potential and conductance as all other synapses within that class. This enabled the simple construction of complex circuits in which all documented synapses (including all bilateral asymmetries) were included in the model. It was assumed that the functional efficacy of a synaptic class was correlated with the number of contacts observed within that synaptic class. Thus, circuits constructed in this way possessed connections with weights determined by anatomical criteria. These weights were not varied further in this model; only the reversal potential, which determined the sign of the connection, was allowed to vary. The complete connectivity of the modelled tap withdrawal circuit is shown in Figure 2.1.

The model was based on all available physiological and anatomical data from *C. elegans* and the related nematode *Ascaris*. The physiological parameters used in the derivation of the data presented in this report are shown in Table 2.1 and Table 2.2. The model was implemented in Objective-C on Intel-486, HP series 9000, and NeXT computers running NEXTSTEP software, and was integrated using fourth-order Runge-Kutta (Press, Flannery, Teukolsky and Vetterling, 1988) to an accuracy of 0.5%.

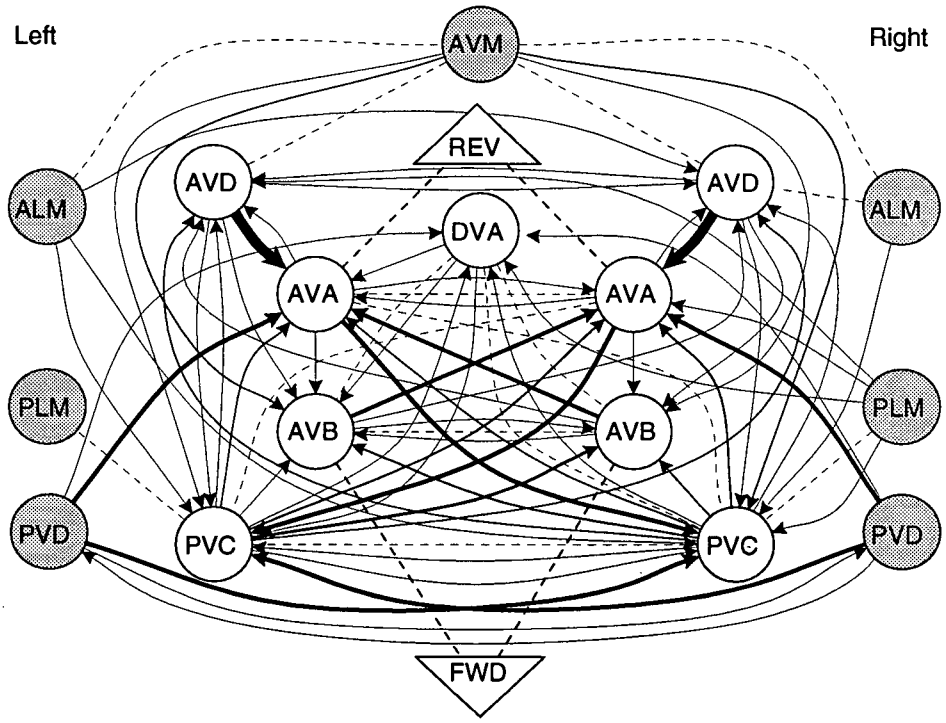


Figure 2.1: The complete connectivity of the tap withdrawal circuit (extracted from Achacoso and Yamamoto, 1992). The circuit consists of seven sensory neurons (*shaded circles*), nine interneurons (*unshaded circles*), and two motorneuron pools (not shown), which produce forward and backward locomotion (*triangles*). Chemical connections are indicated by *arrows*, with the number of synaptic contacts being proportional to the width of the *arrow*. Gap junctions are indicated by *dashed lines*. Every connection represented in this figure was also represented in the model. This representation is useful for identifying connection asymmetries which might underlie the origins of oscillations that control locomotion and are hidden in simpler views of the circuitry.

Neuron Parameters	Value	Units
Membrane resistance	see Table 2.2	Ohms
Membrane capacitance	see Table 2.2	Farads
Membrane leakage potential	-0.035	Volts
Synaptic Parameters	**Value**	**Units**
EPSP reversal potential	0.00	Volts
IPSP reversal potential	-0.048	Volts
Synaptic conductance	6.00E-10	Siemens
E_{RANGE}	0.035	Volts
Gap junction conductance	5.00E-09	Siemens
Tap Parameters	**Value**	**Units**
Pulse rest	0	Amps
Phasic pulse	1.00E-11	Amps
Start time	0.01	Sec
Duration	0.3	Sec
Tonic pulse	2.50E-10	Amps

Table 2.1: List of physiological parameters used in the four experiments run in Wicks, Roehrig and Rankin (1996) are summarized. For a more detailed discussion on the origin of these values, see the text.

2.3 Neurons

The neurons of *C. elegans* have simple morphologies which are preserved across individuals. Many neurons consist of a single unbranched process, and few have more than two branches (Wood, 1988). Electrophysiological analysis of *C. elegans* cells is still in its infancy (however, see Raizen and Avery, 1994; Avery, Raizen and Lockery, 1995; Goodman, Hall, Avery and Lockery, 1998) and little is known about the membrane characteristics of its neurons. However, electrophysiology has been done on *Ascaris lumbricoides*, a larger nematode related to *C. elegans* (Davis and Stretton, 1989a; 1989b). Its dorsal and ventral nerve chords have been reconstructed and show considerable similarity to those of *C. elegans*, and anatomical homologues of *C. elegans* motorneurons have been found in *Ascaris*

(Wood, 1988; Stretton, Donmoyer, Davis, Meade, Cowden and Sithigorngul, 1992). For our model, we used electrophysiological data from *Ascaris* to determine our model parameters.

Evidence from *Ascaris* suggests that signal propagation in *C. elegans* neurons is likely accomplished electrotonically, without classical all-or-none action potentials. Intracellular recordings of *Ascaris* motorneurons and interneurons show no evidence of action potentials, nor has it been possible to evoke them (Davis and Stretton, 1989a). Specific membrane resistance in *Ascaris* is unusually high and is within the range that would permit signal propagation without action potentials. Niebur and Erdös (1993) have used *Ascaris* data to do detailed computational studies of the electrotonic characteristics of *C. elegans* neurons and have shown that the function of *C. elegans* locomotion circuitry can be accounted for by purely electrotonic signals.

Davis and Stretton (1989a) have measured specific membrane resistances $\hat{R}_m$ and intracellular resistivity $\hat{R}_i$ in *Ascaris*. In four motorneurons, $\hat{R}_m$ varied from 61 – 251 $\mathrm{k}\Omega\,\mathrm{cm}^2$, and $\hat{R}_i$ from 79 – 314 $\Omega\,\mathrm{cm}$. We assumed that membrane properties in *C. elegans* are similar, and used an average of the four measurements: $\hat{R}_i = 180\ \Omega\,\mathrm{cm}$, and $\hat{R}_m = 150\ \mathrm{k}\Omega\,\mathrm{cm}^2$. We assumed a specific membrane capacitance of 1 $\mu\mathrm{F}/\,\mathrm{cm}^2$, a standard value for a lipid bilayer (Rall, 1989). These membrane properties were adapted to *C. elegans* anatomy by using the estimated surface area of each cell (see Table 2.2).

Each neuron's branching morphology is given in Wood (1988) and White et al. (1986). This, together with measurements of electron micrographs in White

Cell	Process Length primary (mm)	secondary (mm)	surface area (10^{-6}cm^2)	C_m (pF)	R_m (GΩ)	$\frac{V(l)}{V_0}$
ALM	0.50	0.03	9.1	9.1	16	0.89
PLM	0.48	0.06	9.1	9.1	16	0.90
AVM	0.24	0.03	5.0	5.0	30	0.97
PVM	0.50	–	8.7	8.7	17	0.89
LUA	0.10	–	1.4	1.4	107	0.99
PVD	0.74	0.22	16	16	9.4	0.78
PVC	0.96	–	16	16	9.4	0.68
AVA	0.93	–	15	15	10	0.69
AVB	0.86	–	14	14	11	0.73
AVD	0.86	–	14	14	11	0.73
DVA	0.91	–	15	15	10	0.70

Table 2.2: Summary of neuron characteristics. Branching morphology and process length were taken from Wood (1988) and White et al. (1986), assuming a standard worm length of 1 mm. An average process diameter of 0.5 μm was obtained from measurements of electron micrographs in White et al. (1976) and White et al. (1986). An average soma diameter of 5.0 μm was measured from camera lucida drawings in Wood (1988). $V(l)/V_0$ is the attenuation of a voltage clamp V_0 along the full length of the primary process according to a sealed-end cable equation (Rall, 1989), and gives an indication of a cell's isopotentiality.

et al. (1986) and White et al. (1976) was used to determine average process lengths and diameters (see Figure 2.2 for an example).

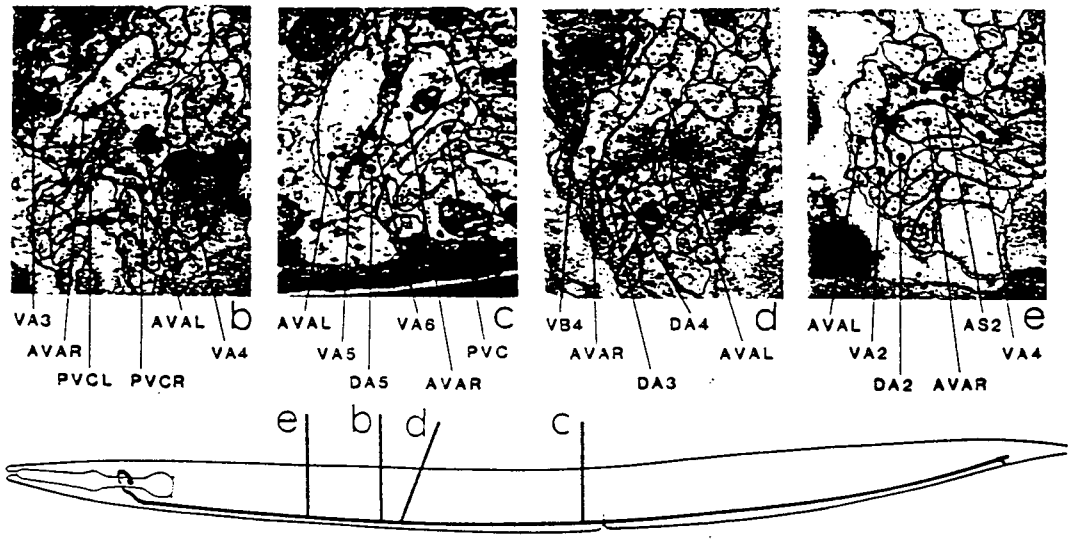


Figure 2.2: Example of neuron process length and diameters. This figure depicts a schematic of the AVA cell and electron micrographs taken at different cross sections (labelled e,b,d c). The perimeter of a process was measured at each cross section, corrected for 10% shrinkage due to fixation, and the diameter of cylinder with equivalent perimeter was computed. (Adapted from White et al., 1986).

Processes were assumed to be cylindrical and somas were assumed to be spherical. Process diameters varied from 0.2 to 1.0 μm, and an average value of $d = 0.5$ μm was used.

Process lengths were taken from diagrams in Wood (1988) assuming a standard worm length of 1 mm. Soma diameters were taken from camera lucida drawings in Wood (1988). Soma diameters varied from 2 μm to 10 μm, but since the soma contributed only a small fraction of the total surface area, we used an average soma diameter of 5 μm. From these data, a total membrane surface area for each cell was computed and the resulting total membrane capacitances and resistances

for the entire cell was derived (see Table 2.2).

For simplicity we assumed that cells were isopotential. Because the length constant (Rall, 1989) — given by

$$\lambda = \frac{1}{2}\sqrt{\frac{d\hat{R}_m}{\hat{R}_i}} \approx 1\text{mm}, \tag{2.1}$$

where d is the process diameter — was generally longer than the process (data not shown), this isopotential assumption was reasonable (also see Table 2.2).

Recent electrophysiological studies in *C. elegans* by Goodman et al. (1998) support these basic assumptions. They found that the isopotential assumption holds in an identified *C. elegans* neuron ASER, as well as many other unidentified *C. elegans* neurons. However, several of the interneurons used in our model (AVA, AVB, PVC and AVD) have processes that are significantly longer than that of ASER and no direct recordings have yet been made in these cells to test the isopotential assumption. Goodman et al. (1998) also report that the neurons they studied have electrotonic properties that would enable passive signal transmission without spiking and that they failed to elicit classical Na^+action potentials from all neurons they studied. However, they did not preclude the possibility of Ca^{2+}-dependent spiking resulting from a regenerative calcium current they found in ASER.

Thus, a neuron's membrane potential, V, is governed by the usual single-compartment membrane equation (Segev, Fleshman and Burke, 1989):

$$C_m\frac{dV}{dt} = \frac{1}{R_m}(E_{LEAK} - V) + \sum I_{SYN} + I_{INJ}, \tag{2.2}$$

where C_m is the total membrane capacitance for the cell, R_m is the total membrane leakage resistance for the cell, E_{LEAK} is the equilibrium potential of the cell membrane's leakage channels, I_{SYN} is the current attributable to a synaptic input, and I_{INJ}

is any injected current. A value of -35 mV was used for E_{LEAK} in these cells (R. E. Davis, personal communication).

For the direction of current flow, this document adopts the engineering convention that is used in other computational neuroscience works (e.g. Bower and Beeman, 1995): positive current flows in the direction of positive charge. Note that this is in the reverse direction from the convention used by electrophysiologists.

2.4 Gap Junctions

The anatomical reconstruction of the nematode nervous system allowed the identification of both electrical and chemical synapses (White et al., 1986). Gap junctions were modelled as ohmic resistances where current flowing into cell i from cell j is given by

$$\hat{I}_{ij} = \hat{g}_{ij}(V_j - V_i) \tag{2.3}$$

where $\hat{g}_{ij}$ is the total conductance of the gap junction area. Niebur (1988) used a specific conductance of 1 S/cm^2 (Bennett, 1972) for a patch of membrane area, and used unpublished micrographs to determine the area of each gap junction. He reported that areas of gap junction contact ranged from 0.2 to 2 μm long and were 0.5 μm wide (Niebur, 1988). In our model, a standard gap junction length of 1 μm was assumed, with a resulting conductance of 5 nS for all gap junctions. In some experiments, this value was increased or decreased by a factor of 10 to test the sensitivity of the model's predictions to the precise value of the conductance used.

2.5 Synapses

Synaptic classes consisted of a number of individual synaptic contacts. The number of contacts in each class was extracted from an anatomical database of *C. elegans* synaptic connectivity (Achacoso and Yamamoto, 1992). The identification of chemical synapses from the anatomical reconstruction was done by identifying presynaptic specializations and inferring postsynaptic partners based on proximity; no postsynaptic specializations were evident in the electron microscope reconstruction (White et al., 1986). Any error associated with this technique would tend to overestimate the number of chemical synapses in the organism, but these errors would not appear with high frequency, and would therefore not have a large impact on the response of the circuit to stimulation.

Each modelled synapse represented a class of synaptic contacts with total synaptic conductance — the "weight" — given by the product of the number of individual contacts within the class and the individual synaptic conductance.

The synapse model used was based on the graded synapse model used by Lockery and Sejnowski (1992) in the leech local bending circuit. However, it was extended to explicitly include the synaptic reversal potential as well as the conductance. Post-synaptic current was attributable to gated channels in the post-synaptic membrane with inward current given by

$$I = g(t)\,(E_{SYN} - V_{POST}), \tag{2.4}$$

where $g(t)$ is the synaptic conductance of the postsynaptic membrane (which is related to neurotransmitter release which in turn depends on presynaptic potential), E_{SYN} is the reversal potential for the synaptic conductance, which was assumed

to be constant, and V_{POST} is the postsynaptic membrane potential. For excitatory synapses, a reversal potential of 0 mV was used. and for inhibitory synapses -45 mV was used (from *Ascaris* data; R. E. Davis, personal communication).

For simplicity, it was assumed that all synapses made by a given presynaptic cell were of the same polarity and class. This is a version of Dale's Principle (Osborne, 1983) which, although it is not always true, is often used in mathematical neurosciences to simplify models (Hoppensteadt and Izhikevich, 1997). Computationally, this reduced the number of optimized parameters — each of which possessed two possible values — to the number of neurons in the modelled circuit. It was further assumed that all modelled synapses functioned as fast ligand-gated channels. It was possible that some anatomically defined synapses were modulatory and acted via slow second-messenger systems, or that synaptic function was altered by the *milieu intérieur* (Harris-Warrick et al., 1992); however, we assumed that these modulatory effects did not affect a single tap withdrawal response.

In the leech local bending reflex, Lockery and Sejnowski (1992) observed multiple time courses in some of the postsynaptic responses and to model this, they used a fast (10 ms) and a slow (1500 ms) decaying membrane current, each governed by its own first order equation. Preliminary versions of this model used a dynamic synaptic model with a fast (10 ms) synaptic time constant, but no significant differences were noted in the results of circuits containing these synapses and simulations, which used an instantaneous synapse. To reduce the complexity of the model, we therefore used a synaptic conductance that depended only on the

presynaptic membrane potential:

$$g(t) = g_{\infty}(V_{PRE}), \tag{2.5}$$

where g_{∞} represents the steady-state post-synaptic conductance in response to a presynaptic voltage.

2.6 Synaptic Activation

No direct recordings have yet been made in *C. elegans* to determine properties of synaptic activation. In recordings made from *Ascaris* commissural motorneurons, Davis and Stretton (1989b) demonstrated that synaptic transmission is graded and transmitter is released tonically between both excitatory and inhibitory motorneurons and postsynaptic muscle and motorneurons. They found that changes in postsynaptic potential were related to presynaptic depolarizing current by a sigmoidally shaped curve and that the presynaptic resting potential lies approximately in the middle of the voltage-sensitive range of synaptic transmission.

Dynamic network simulations based on graded synaptic transmission have been described previously (Lockery, Nowlan and Sejnowski, 1992; De Schutter, Angstadt and Calabrese, 1993). We assumed that synaptic activation and transmission in *C. elegans* was similar to *Ascaris*, namely that it is graded and sigmoidally shaped with presynaptic potential, and is tonically active with the steady-state potential in the middle of the voltage sensitive range. Accordingly, we used a sigmoidal function to model the steady-state post-synaptic membrane conductance:

$$g_{\infty}(V_{PRE}) = \frac{\bar{g}}{1 + e^{K(\frac{V_{PRE} - V_{SS}}{E_{RANGE}})}}, \tag{2.6}$$

where $\bar{g}$ is the maximal post-synaptic membrane conductance for the synapse and V_{SS} is the presynaptic cell's in-circuit steady-state potential, and E_{RANGE} is the presynaptic voltage range over which the synapse activated.

Note that because of tonic synaptic input, the in-circuit steady-state potential of a cell must be determined from the fixed point of the entire system of equations governing the circuit and this was computed prior to each run.

We used a value of

$$K = 2\ln(\frac{0.1}{0.9}) = -4.3944 \tag{2.7}$$

so that the conductance changes from 10% to 90% of its maximal value over a presynaptic voltage range of E_{RANGE}. Note that because synapses were tonically active, a cell's steady-state potential was not defined solely by its membrane leakage reversal potential, but rather was determined from the steady-state solution of the entire system of equations governing the circuit. This was computed before each run in the following way.

2.7 Steady-state Potential

The assumption that tonically active synapses were active in the middle of their voltage sensitive range at the steady-state potential implied that the postsynaptic conductance $g(t)$ was one-half its maximal value $\bar{g}$ when the circuit was at steady-state.

Let V_i denote the membrane potential for neuron i and similarly for other quantities pertaining to neuron i (see Equation 2.2). Let I_{ij} denote the ligand-gated

synaptic current flowing into neuron i resulting from neurotransmitter release from neuron j across a single synapse, let E_{SYNij} denote the synaptic reversal potential for synaptic current I_{ij}, and let n_{ij} denote the total number of synaptic connections from neuron j to neuron i. Similarly, let $\hat{I}_{ij}$ denote current flow across a single gap junction where positive current is in the direction from neuron j to neuron i and $\hat{n}_{ij}$ denote the total number of gap junctions between neuron j and neuron i. Finally, let I_{INJ_i} denote the external current flow into cell i (caused by either sensory stimulation or external current injection). Then the entire system is given by:

$$R_{m_i}C_{m_i}\frac{dV_i}{dt} = E_{LEAK_i} - V_i + R_{m_i}\sum_{j=1}^{N}(I_{ij} + \hat{I}_{ij}) + R_{m_i}I_{INJ_i} \tag{2.8}$$

$$\hat{I}_{ij} = \hat{n}_{ij}\hat{g}_{ij}(V_j - V_i) \tag{2.9}$$

$$I_{ij} = n_{ij}g_{ij}(E_{SYNij} - V_i) \tag{2.10}$$

$$\frac{dg_{ij}}{dt} = \frac{g_{\infty_{ij}}(V_j) - g_{ij}}{\tau_{ij}} \tag{2.11}$$

$$g_{\infty_{ij}}(V_j) = \frac{\bar{g}_{ij}}{1 + e^{K_{ij}\frac{(V_j - V_{SS_j})}{E_{RANGEij}}}}. \tag{2.12}$$

where N is the number of neurons in the circuit and τ_{ij} is the synaptic time constant.

At steady-state, $V_i = V_{SS_i}$, and $\frac{dV_i}{dt}$, $\frac{dg_{ij}}{dt}$, and external inputs I_{INJ_i} are zero. Synaptic conductances are tonic and at their half-activation at steady-state, so that

$$g_{\infty_{ij}}(V_{SS_j}) = \bar{g}_{ij}/2. \tag{2.13}$$

After algebraic manipulation, this yields a system of linear equations that can be solved using standard Gaussian elimination to find V_{SS_i} (Press et al., 1988):

$$\mathbf{V}_{SS} = \mathbf{A}^{-1}\mathbf{b} \tag{2.14}$$

where A_{ij} is the ith row and jth column of matrix **A** and is given by

$$A_{ij} = -R_{m_i}\hat{n}_{ij}\hat{g}_{ij},\ i \neq j, \tag{2.15}$$

$$A_{ii} = 1 + R_{m_i}\sum_{j=1}^{N}(\hat{n}_{ij}\hat{g}_{ij} + n_{ij}\bar{g}_{ij}/2), \tag{2.16}$$

and **b** is a vector is given by

$$b_i = E_{LEAK_i} + R_{m_i}\sum_{j=1}^{N} E_{SYNij}n_{ij}\bar{g}_{ij}/2. \tag{2.17}$$

The computed steady-state potential of a cell varied within a physiological range of -47 mV to 0 mV, with a mean of -24 mV and a standard deviation of 13 mV. This did not vary appreciably from cell to cell, but rather depended on the circuit's polarity configuration and ablation condition (i.e. which cells were removed from the circuit).

As an aside, the system (2.14) can also be inverted to explicitly give E_{LEAK_i} in terms of V_{SS_i}:

$$E_{LEAK_i} = V_{SS_i} - R_{m_i}\sum_{j=1}^{N} n_{ij}\bar{g}_{ij}/2(E_{SYNij} - V_{SS_i}) - R_{m_i}\sum_{j=1}^{N}\hat{n}_{ij}\hat{g}_{ij}(V_{SS_j} - V_{SS_i}). \tag{2.18}$$

2.8 Synaptic Parameters

To determine values for E_{RANGE} and $\bar{g}$, the synapse model was fitted to published measurements (Davis and Stretton, 1989b) of *Ascaris* muscle cell postsynaptic response to presynaptic current injection as detailed below.

Thus, values for $\bar{g}$ and E_{RANGE} for particular *Ascaris* synapses were found. We assumed that *C. elegans* synapses activated over voltage ranges similar to *Ascaris* synapses. However, the maximal synaptic conductance $\bar{g}$ needed to be adapted

to *C. elegans*. We assumed that $\bar{g}$ represents the product of a synaptic conductance per unit area and a synaptic area. In the case of a synapse mediated by a single population of ion channels, $\bar{g}$ would be equivalent to the single-channel conductance times the total number of available channels. To adapt the $\bar{g}$ value from *Ascaris* to *C. elegans*, we assumed *C. elegans* synapses had similar unit-area conductances and accordingly scaled the *Ascaris* $\bar{g}$ by a factor to account for the presumed difference in synaptic areas. We assumed the total synaptic area between two cells was proportional to the length of the process which we estimated by the ratio of body lengths — approximately 1/250. As this represents only a gross approximation, the value of $\bar{g}$ used in these studies was varied over three orders of magnitude in different experiments (see Wicks, Roehrig and Rankin, 1996).

2.8.1 Modelling *Ascaris* Monosynaptic Response

Data from Davis and Stretton (1989b, their Figs. 13 and 14) show the post-synaptic response of an *Ascaris* dorsal muscle cell (DM) to current injected into a presynaptic excitatory motorneuron, DE1, and the response of a ventral muscle cell (VM) to current injected into a presynaptic inhibitory motorneuron, VI. Both of these response profiles were sigmoidal in shape, were centred approximately at the resting potential, and had asymptotic post-synaptic responses at the extremes of positive and negative pre-synaptic current injection. Therefore, the sigmoidal tonic synapse model presented here was well-suited to modelling these synaptic responses.

Davis and Stretton (1989a; 1989b) placed a recording electrode in a muscle cell within the output zone of the motorneuron, and an injecting electrode at the ventral end of a commissural process leading to the synapse. The measured input resistance of the motorneuron was used to obtain the resulting membrane potential at the point of injection, and an infinite cable model was used to determine the membrane potential at the presynaptic site.

Because the recordings that were used to determine input resistance were made at the same ventral end of the commissure as was injected for the synaptic response measurements (R. E. Davis, personal communication) and because the input resistance was approximately constant over the relevant range of injected current (Davis and Stretton, 1989a), it is possible to directly use the measured input resistance to determine the membrane potential at the point of current injection.

Davis and Stretton (1989a) determined the motorneuron cable properties by fitting their measurements along the commissure to an infinite cable model (Rall, 1989), and found that the length constant was unusually high ($\lambda \approx 8$mm) — roughly the same magnitude as the length of the process they were measuring. With such a large length constant, it is possible that the cable's branching morphology and sealed ends play a significant role in determining these cable properties (Rall, 1977), suggesting that a sealed-end cable model might be more appropriate. However, for consistency we used an infinite cable model with cable constants as determined by Davis and Stretton (1989a) to reproduce their measured voltage response along the commissure.

Specifically, the presynaptic depolarization in response to an injected current

is given by

$$\Delta V_{PRE} = I_{INJ}\, R_{PRE}\, e^{-L/\lambda} \tag{2.19}$$

where L is the distance from the point of current injection to the synapse, and R_{PRE} is the input resistance at the point of injection. For the DE1 – DM synapse, the distance L was 5 – 8 mm, and for the VI-VM synapse, it was 0.5 – 2.5 mm (R. E. Davis, personal communication). We used the mean of 7 mm and 1 mm, respectively. The input resistances for DE1 and VI were reported to be 6 MΩ and 17 MΩ respectively (Davis and Stretton, 1989a).

According to the sigmoidal tonic synapse model, the steady-state plateau response of the post-synaptic muscle is expressed as:

$$V_{POST} = E_{LEAK} + R_{POST}\, n\, g_\infty(\Delta V_{PRE})(E_{SYN} - V_{POST}), \tag{2.20}$$

where E_{LEAK} and E_{SYN} pertain to the post-synaptic muscle cell, R_{POST} is the post-synaptic cell's input resistance, $g_\infty(\Delta V_{PRE})$ is the steady-state synaptic conductance, and n is the number of synapses between the motorneuron and the muscle cell. Note that here, the presynaptic potential is written as ΔV_{PRE} and is taken relative to the presynaptic cell's steady-state potential since the data points taken for Figure 2.3 and Figure 2.4 were taken relative to steady-state and not to any absolute voltage reference (Davis and Stretton, 1989b).

The input resistance of ventral and dorsal muscle cells was measured to be 0.18 – 0.50 MΩ, with a mean of 0.3 MΩ(R. E. Davis, personal communication). However, it is worth noting that since muscle cells exhibit spiking behaviour, this input resistance may not be truly constant. We used $E_{LEAK} = -35$ mV and $E_{SYN} = 0$ mV for excitatory and $E_{SYN} = -45$ mV for inhibitory reversal potentials (Davis,

personal communication).

A light microscope study of dye-injected muscle cells in *Ascaris* suggested that the DE1 motorneurons make 5 – 10 synapses to each dorsal muscle cell and the VI motorneurons make 8 – 16 synapses to each ventral muscle cell (J. Donmoyer, personal communication).

Equation (2.20) can be arranged to give V_{POST} explicitly in terms of V_{PRE}:

$$V_{POST} = \frac{E_{LEAK} + E_{SYN}\, R_{POST}\, n\, g_\infty(\Delta V_{PRE})}{1 + R_{POST}\, n\, g_\infty(\Delta V_{PRE})} \tag{2.21}$$

where

$$g_\infty(\Delta V_{PRE}) = \frac{\bar{g}}{1 + e^{K \frac{\Delta V_{PRE}}{E_{RANGE}}}}. \tag{2.22}$$

The change in postsynaptic potential is therefore given by

$$\Delta V_{POST} = V_{POST} - V_{SS}, \tag{2.23}$$

where V_{SS} is the steady-state potential of the postsynaptic cell under unstimulated tonic synaptic input, and is given by

$$V_{SS} = \frac{E_{LEAK} + E_{SYN}\, R_{POST}\, n\, \bar{g}/2}{1 + R_{POST}\, n\, \bar{g}/2}. \tag{2.24}$$

Equations (2.19) and (2.21) — (2.24) define a non-linear function for post-synaptic membrane potential in terms of presynaptic injected current and contains two unknown parameters, $\bar{g}$ and E_{RANGE}. Levenberg-Marquardt's method (Press et al., 1988) was used to fit this function to the data from Davis and Stretton (1989b).

We obtained good results to the fit for the VI-VM synapse (see Figure 2.3). This fit included the reversal potential E_{SYN} as a fit parameter; this improved the fit substantially without significantly changing the reversal potential (-48 mV as

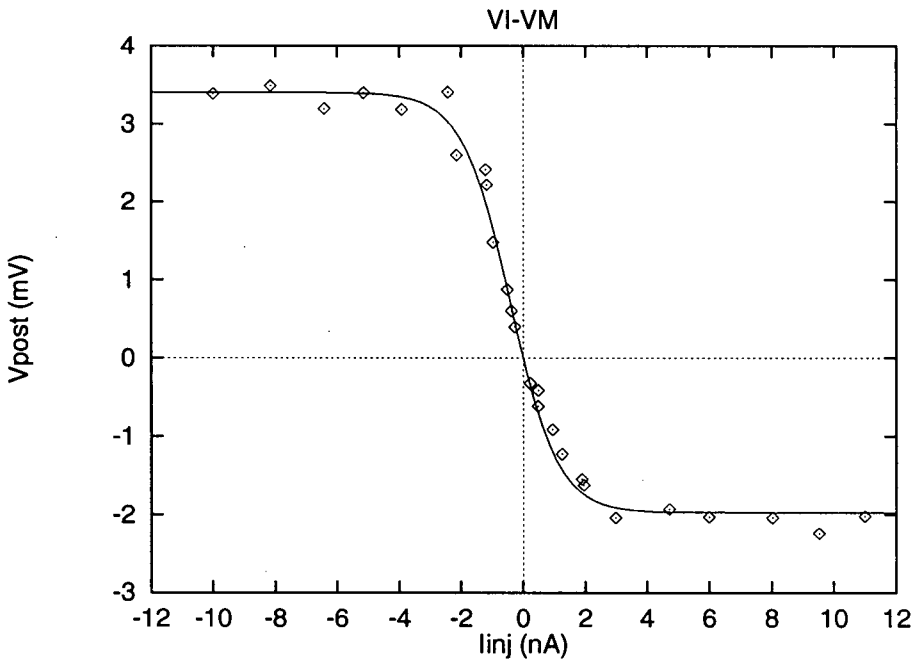


Figure 2.3: Fit for the VI-VM synapse. The diamonds indicate individual measurements made of the change in postsynaptic membrane potential in a muscle cell VM in response to a current injected into a presynaptic inhibitory motorneuron VI (experimental data taken from Davis and Stretton (1989b), their Fig. 14). The curve is the fit by equations (2.19) — (2.24) produced by Levenberg-Marquardt's method, with results of $\bar{g} = 150$ nS, $E_{RANGE} = 52$ mV, and $E_{SYN} = -48$ mV.

opposed to -45 mV). The fit results were $\bar{g} = 150$ nS, $E_{RANGE} = 52$ mV, $E_{SYN} = -48$ mV and were stable under various initial conditions.

The DE1-DM fit was less accurate since the steady-state conductance was not precisely a symmetric sigmoid. Therefore, to find approximate ranges for E_{RANGE} and $\bar{g}$, a number of parameter values were explored manually to fit each branch of the sigmoid separately, and we found that E_{RANGE} ranged from 13 – 20 mV, and $\bar{g}$ ranged from 50–150 nS (see Figure 2.4).

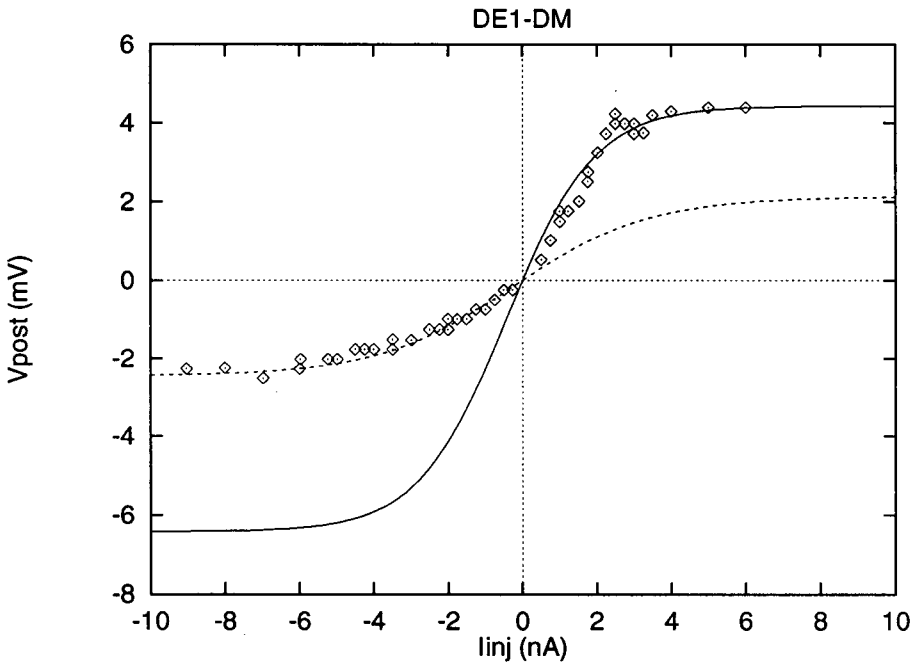


Figure 2.4: Fit for the DE1-DM synapse. The diamonds indicate individual measurements made of the change in postsynaptic membrane potential in a muscle cell DM in response to a current injected into a presynaptic excitatory motorneuron DE1 (experimental data taken from Davis and Stretton (1989b), their Fig. 13). Because the data do not form a perfect sigmoid, a good fit was not obtained to both branches simultaneously. The solid curve shows a fit to the upper branch ($\bar{g} = 150$ nS; $E_{RANGE} = 13$ mV; $E_{SYN} = 0$ mV), and the dashed curve shows a fit to the lower branch ($\bar{g} = 50$ nS; $E_{RANGE} = 20$ mV; $E_{SYN} = 0$ mV). In both cases, the curves were fit by hand.

To adapt these results to *C. elegans* we took a rough average of the two E_{RANGE} estimates to get an activation range of -35 mV, and scaled the $\bar{g}$ estimate from the VI-VM synapse by the 1/250 ratio of body lengths to obtain a maximal conductance of 0.6 nS for an individual *C. elegans* synapse.

2.9 The Gearbox

This model does not explicitly incorporate nematode locomotion; these issues have been dealt with adequately elsewhere (Niebur and Erdös, 1991; Niebur and Erdös, 1993). Rather, this report concentrates specifically on sensorimotor integration. However, because a behavioural variable was used to optimize the modelled output, it was necessary to rigorously define the relationship between the animal's locomotion and activation of the circuitry that controls that behaviour. This issue was addressed with simple assumptions, which were consistent with work on the modelling of nematode locomotion (Niebur and Erdös, 1991; Niebur and Erdös, 1993; Stretton et al., 1992) and current theories of tap withdrawal circuit function (Chalfie et al., 1985; Wicks and Rankin, 1995). The output of the tap withdrawal circuit was assumed to control locomotory behaviour primarily through the action of the interneurons AVB and AVA. These two interneurons make electrical connections with motorneurons all along the ventral cord of the worm. The AVA interneurons make gap junctions with the motorneurons AS, VA, and DA, which are presumed to excite backward locomotion; the AVB interneurons form gap junctions with the motorneurons VB and DB, which are presumed to excite forward locomotion. Ablation of these cells almost completely destroys an animal's ability to move

forward (in the case of AVB ablations) or backward (in the case of AVA ablations) (Chalfie et al., 1985; Wicks and Rankin, 1995). Thus, it was simply assumed that the degree to which an animal reversed was proportional to the depolarization of the AVA interneuron and inversely proportional to the depolarization of the AVB interneuron. Forward locomotion in response to tap was also proportional to this value; a lower propensity to reverse was equivalent to a higher propensity to accelerate. The exact nature of this proportionality was not defined because *in vivo* it will be modulated by a number of neural, hydrostatic, and physical forces that are beyond the scope of this endeavour. The gearbox, i.e., the transformation equation that was used to convert depolarization of AVA and AVB into behaviour, was simply:

$$\text{Propensity to Reverse} \propto \int (V_{AVB} - V_{AVA})\, dt. \tag{2.25}$$

The integration was calculated from the time of the tap stimulation until either the end of the simulation or until the integrand changed sign. Additionally, the test for a change of integrand sign was suppressed for a grace period of 100 ms to allow for initial transients after the tap. The tap stimulus was modelled as a phasic depolarization of the sensory neurons PLM, ALM and AVM which have been shown to mediate the response to tap in the intact animal (Wicks and Rankin, 1995).

One consequence of the gearbox assumption is that, because of uncertainty regarding the exact nature of the proportionality between the output of the AVA and AVB interneurons and the magnitude of the evoked behaviour, comparisons of model data and empirical data must be limited to *relative* changes in response magnitude. Thus, such comparisons were made between data profiles that had been

normalized about the mean of that polarity configuration's response level (i.e. the value of the integral (2.25) for the model data, and the living animal's reversal magnitude for the empirical data. See Wicks, Roehrig and Rankin, 1996) This measure detected changes in the levels of responding to tap produced by an ablation series, without being sensitive to the absolute response magnitude of a particular circuit configuration — information which in any case is meaningless in the context of the gearbox assumption.

2.10 Results and Conclusions

This model was successfully used (Wicks, Roehrig and Rankin, 1996) to predict the functional polarities of synaptic connections in the tap withdrawal circuit in the following manner. Because the synaptic weights were determined by anatomical data and other model parameters were determined from physiological criteria, the only free parameters in the model were the synaptic polarities. The output of the model was then tested against the behaviour of the living animal under various conditions of degradation involving the removal of one or more cells from the circuit (using laser ablation to kill the cell in the animal). For each ablation condition, all possible synaptic polarity configurations of the model were exhaustively enumerated and tested to find the best overall fit to the behavioural data. The model provided statistically significant predictions for 13 of the 15 neuron classes and accurately reproduced the response of the animal to ablations (see Figure 2.5). For full details on the polarity determination strategy and results, the reader is referred to Wicks, Roehrig and Rankin (1996).

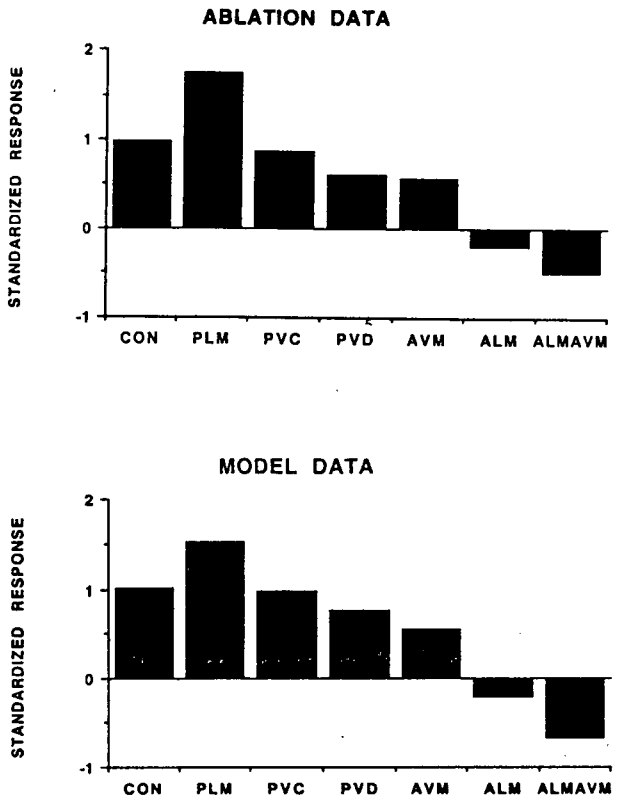


Figure 2.5: Comparison of model data to behavioural data. The top figure shows the reversal response of the animal, normalized to the control (CON) response. The bottom figure shows the response of the model to the same set of ablations when the predicted synaptic polarities are used. (Adapted from Wicks, Roehrig and Rankin, 1996).

A novel contribution of this model is the tonic and graded synaptic model it uses and the resulting distinction that it makes between a cell's resting membrane potential and its in-circuit equilibrium potential. Graded (non-firing) synaptic models have been proposed previously (Lockery et al., 1992; De Schutter et al., 1993), but our assumption that synapses were tonically active forced us to consider whether they were tonically active in an isolated cell or while in a circuit. In the latter case, the half-activation threshold of the cell is its in-circuit equilibrium potential which differs from its leakage membrane potential due to the tonic synaptic currents flowing in the circuit. The preceding derivation contains a treatment of the equilibrium vs. resting potential and formulas to convert from one to the other.

For the purposes of this dissertation, the interesting consequence of this model is that the output of the model (the gearbox), while successfully predicting the magnitude of the behavioural response in the animal, did not follow the same time course as the animal's behaviour. Following a transient tap stimulus lasting from 100–600 ms, the nematode continues to swim backwards for up to several seconds before resuming forward locomotion. The gearbox output of the model, on the other hand, decays quickly after the termination of the stimulus — with a time constant equivalent to the average cell membrane time constant, approximately 150 ms. To account for this discrepancy, a new model that uses network dynamics to maintain the reversal was proposed, and this is the subject of the next chapter.

Chapter 3

Dymods: An Approach to Modularizing Dynamical Neuronal Structures

The physiological model of the tap withdrawal circuit we developed in Chapter 2 successfully predicted the behavioural response, but it did not explain how the reversal locomotion was maintained for several seconds after the completion of the transient tap stimulus which lasts for only a few hundred milliseconds. The duration of a reversal response is much longer than both the stimulus and the characteristic time constants of the neurons that participate in the response (Wicks, Roehrig and Rankin, 1996; Goodman, Hall, Avery and Lockery, 1998). Moreover, the crispness of the change from forward locomotion to reverse and back again to forward suggests that it is an active transition between two states rather than a decay of an impulse that is responsible for maintaining and terminating the reversal.

One possibility is that the maintenance of a reversal could be the result of network dynamics and a bifurcation between two quasi-stable states governing forward and reverse locomotion. To test this hypothesis, physiologically-based models of *C. elegans* cells were used to engineer a neuronal circuit to account for the reversal maintenance response. Because the goal was to see if network dynamics could account for the behaviour and because so little is known about cellular activity in *C. elegans* the design was accomplished without the connectivity constraints of the anatomical data. The way that the resulting circuit might fit into the anatomical map is addressed in the discussion.

3.1 Methods and Results

The initial conceptual design of this circuit consisted of a pair of cells to drive forward and reverse locomotion. The tap was assumed to cause the forward drive cell to inactivate, the reverse drive cell to activate, and a charging circuit to begin a charging cycle. When the charging circuit reached a threshold, it was assumed to cause the reversal cell to inactivate and the forward cell to reactivate. Because there is some evidence that an oscillatory signal is needed to drive locomotion (Niebur and Erdös, 1991), the circuit was assumed to exhibit oscillatory activity. A current input pulse lasting 100 ms was used as an approximation to the tap stimulus (Wicks, Roehrig and Rankin, 1996).

The resulting neuronal circuit model for maintaining reversals is a dynamical system and it was constructed in a modular way from three dymods, each themselves a dynamical system: a bistable switch dymod, a bistable oscillator dy-

mod and a charger dymod. These dymods were designed independently and interconnected to form the complete circuit. The dymods were designed using the XPP phase plane analysis software for non-linear systems of differential equations (Ermentrout, 1998). All graphs were produced from the output of XPP computations.

3.1.1 The Cell Model

C. elegans neurons are simple. Although electrophysiology in *C. elegans* is in its infancy, current results suggest that signalling in *C. elegans* neurons is accomplished passively without spiking (Goodman et al., 1998). Direct recordings from some neurons (Goodman et al., 1998) and estimations based on data from a related nematode species, *Ascaris* (Wicks et al., 1996; Davis and Stretton, 1989a) suggest that neurons are nearly isopotential and can be approximated by a single membrane equation, equivalent to a passive RC circuit (Wicks et al., 1996; Koch and Segev, 1989). Synapses in *C. elegans* are also simple: evidence from *Ascaris* (Davis and Stretton, 1989b) suggests that neurotransmitter is released tonically (even when the cell is at "rest"), so that positive or negative changes in the cell's activity (i.e. its membrane potential) result in proportional changes in synaptic transmitter output. In the cell model, the synaptic activation function describes how much neurotransmitter is released at a given presynaptic cell membrane potential. Based on data from *Ascaris* (Davis and Stretton, 1989b), this function was assumed to be sigmoidally shaped: it is approximately linear at the cell's resting potential and saturates smoothly at both extremes. As in our previous study (Wicks, Roehrig

and Rankin, 1996), a simple equivalent channel model was used for synaptic input: current flows into the postsynaptic cell with a driving force equal to the difference between the post-synaptic cell's membrane potential and an effective channel reversal potential. This reversal potential determines the synaptic polarity (whether it is excitatory or inhibitory). For full details about the physiological derivation of the model and its assumptions, see Wicks, Roehrig and Rankin (1996) or Chapter 2.

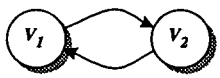


Figure 3.1: A coupled pair of cells.

The design of the dynamical system for maintaining reversals was begun by examining the dynamics of a pair of two reciprocally connected cells (Figure 3.1). This system has two types of behaviour. The first is simple: each cell's membrane potential tends toward a stable steady-state voltage that is dependent on the synaptic efficacies (weights). The second is more interesting: a pair of cells can operate as a switch that can toggle between two different stable states in response to an external transient input.

The voltage response of a pair of coupled cells is described by the following system:

$$\frac{dV_1}{dt} = \frac{-V_1 + J_{12}(V_2)\ (E_{SYN12} - V_1)}{\tau_1} + I_{INJ1} \tag{3.1}$$

$$\frac{dV_2}{dt} = \frac{-V_2 + J_{21}(V_1)\ (E_{SYN21} - V_2)}{\tau_2} + I_{INJ2} \tag{3.2}$$

where V_i is the membrane potential of cell i ($i = 1$ or 2) taken with respect to the cell's leakage (resting) potential, τ_i is the membrane time constant for cell i, E_{SYNij} is the synaptic reversal potential for connections into cell i from cell j, taken with

respect to the resting potential of cell i (the postsynaptic cell). The sign of E_{SYNij} determines whether the synapse is excitatory (positive) or inhibitory (negative). The derivation of this model is presented in Wicks, Roehrig and Rankin (1996) and Chapter 2, but note that voltage quantities here are expressed relative to the cell's leakage potential, rather than as the absolute quantities used in Chapter 2.

$J_{ij}(V_j)$ is the synaptic input to cell i from cell j and is given by

$$J_{ij}(V_j) = w_{ij}\ s(\frac{V_j - E_{ACTj}}{E_{RANGEj}}), \tag{3.3}$$

where E_{ACTj} is the synaptic half-activation potential for cell j, and E_{RANGE_j} is the voltage range over which cell j activates.

The weight coefficient w_{ij} is the dimensionless synaptic coupling for connections into cell i from cell j. In physiological terms, w_{ij} is the ratio of the fully-activated postsynaptic conductance (g_{SYN}) to the postsynaptic cell's leakage conductance (g_{LEAK}), multiplied by total number of synapses.

The sigmoidal synaptic activation proportion is given by $s(x) = \frac{1}{1+e^{Kx}}$, where $K = -2\ln(\frac{0.9}{0.1}) = -4.3944$ so that the synapse activates from 10% to 90% over a voltage range of E_{RANGE}.

The external input to cell i is expressed as I_{INJ_i} which is the change in the cell's membrane potential induced by an injected current I across the cell's input resistance, R_{INPUT}, and is expressed in mV/s: $I_{INJ} = I \times R_{INPUT}$, where the input resistance was taken to be a constant 10 GΩ for the short duration of the pulse input used in the experiments (Wicks et al., 1996).

3.1.2 The Switch Dymod

The possible behaviours of this system can be deduced by studying the nullclines of the system in the (V_1, V_2) phase-space. (The nullcline is the set of points where the membrane potential of the cell does not change.) At an intersection of the two nullclines, both cells are unchanging so this defines a steady-state or fixed point of the system (Strogatz, 1994).

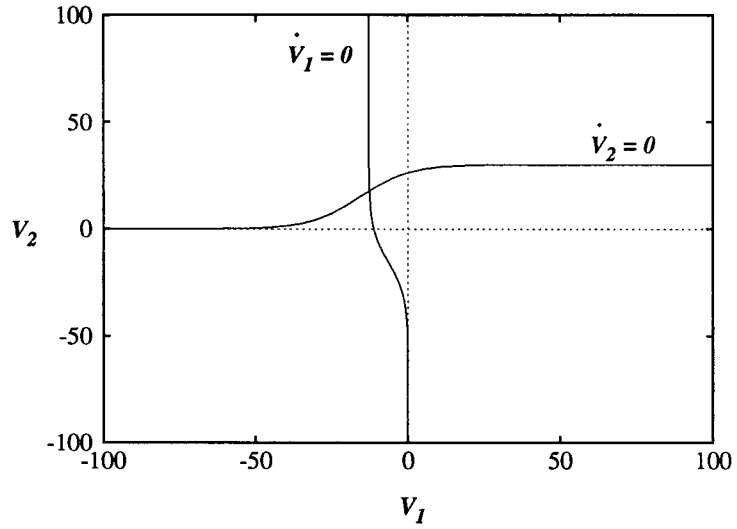


Figure 3.2: Nullclines for a coupled pair of cells. In this case, cell V_1 makes an excitatory connection to cell V_2 ($E_{SYN21} > 0$), and cell V_2 makes an inhibitory connection to cell V_1 ($E_{SYN12} < 0$).

The nullclines of the system (3.1) and (3.2) are given by $\frac{V_1}{dt} = 0$ and $\frac{V_2}{dt} = 0$, and can be written as:

$$V_1 = \frac{J_{12}(V_2)}{1 + J_{12}(V_2)} E_{SYN12}, \; V_2 = \frac{J_{21}(V_1)}{1 + J_{21}(V_1)} E_{SYN21} \tag{3.4}$$

In the (V_1, V_2) phase plane, these nullclines are sigmoidal in shape and have asymptotes parallel to their respective axes (see Figure 3.2). In the simple case,

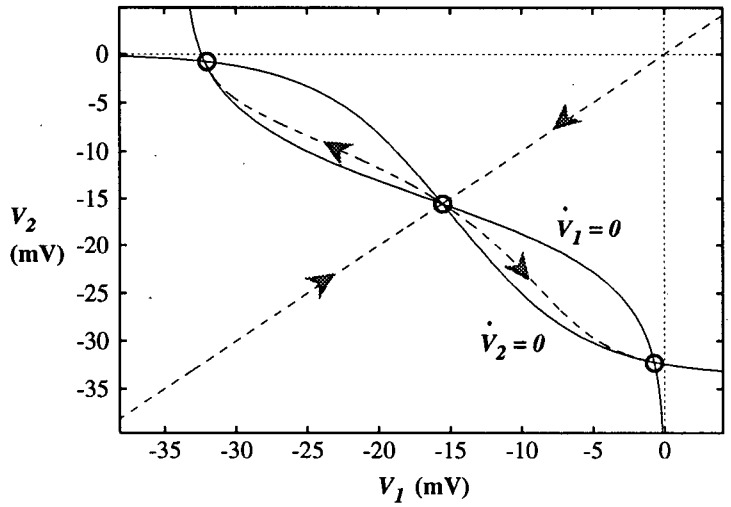


Figure 3.3: Fixed points for an inhibitory switch. In this case, both cells make inhibitory connections with one another ($E_{SYN} < 0$), and the synaptic weights and activation ranges have been adjusted so that the nullclines $\dot{V}_1 = 0$ and $\dot{V}_2 = 0$ intersect at three fixed points (circles). Stable and unstable manifolds for the hyperbolic middle fixed point are shown in dashed lines. See text for parameter values.

the two nullclines intersect at a single point which defines the steady-state point for the system. That this point is stable can be deduced from the sign of $\frac{V_i}{dt}$ in each quadrant. It is also possible for the two sigmoids to intersect at more than one point and this underlies the second, more interesting behaviour of the system.

The sigmoids can have multiple intersections in the third quadrant when both connections are inhibitory (Figure 3.3), or in the first quadrant when both connections are excitatory. In both cases, there are three intersections corresponding to three equilibrium states of the system. The outermost two fixed points are stable, and the middle one is an unstable hyperbolic point. The excitatory case has stable points when the cells are either both activated or both inactive. The inhibitory case has stable points when only one is active and this is more suitable for the design

since only one of the forward and reverse drive cells can be active at a time.

Figure 3.3 also shows the stable and unstable manifolds for the middle hyperbolic point. Trajectories that start precisely on the stable manifold will tend toward the middle hyperbolic point. All others will tend toward one of the outer stable fixed points, so that the stable manifold is a separatrix which separates (V_1, V_2) space into two basins of attraction for their respective outer stable fixed point. Figure 3.4 and Figure 3.5 show how this circuit can operate as a neural switch.

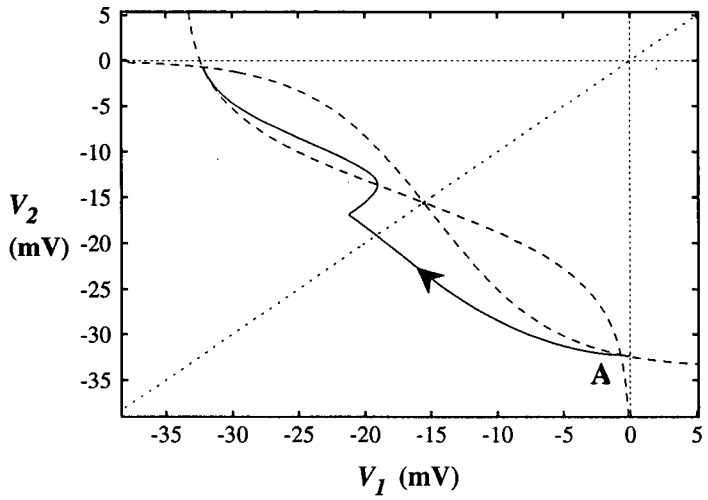


Figure 3.4: Inhibitory switch turning on. The trajectory starts at stable fixed point A with cell 1 turned on (0 mV), and cell 2 off (-32 mV). At $t = 200$ ms, an inhibitory current pulse of -2.5 pA is injected into cell 1 for 100 ms, causing the trajectory to cross the threshold (Figure 3.4), after which the system settles into its other stable state.

This behaviour is not singular and occurs within the physiological ranges of parameter values (see Wicks, Roehrig and Rankin, 1996). Parameter values for this example are shown in Table 3.1. In subsequent design phases, the cells were taken to have these default parameter values, unless the design necessitated a change.

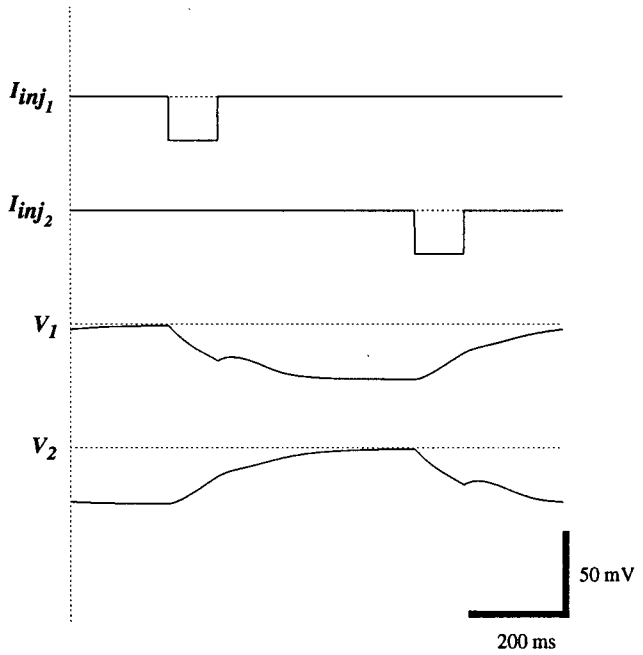


Figure 3.5: Simulated recording traces for an inhibitory switch. The trajectory starts at a stable fixed point (Figure 3.4) with cell 1 turned on (0 mV), and cell 2 off (-32 mV). At $t = 200$ ms, an inhibitory current pulse of -2.5 pA is injected into cell 1 for 100 ms, causing the trajectory to cross the threshold (Figure 3.4), after which the system settles into its other stable state. At 700 ms, an inhibitory current pulse is injected into cell 2 and the system switches back to its original state.

Note that $E_{ACT} = 0$ mV corresponds to the assumption that synapses are tonically active at their resting potential.

Model Parameters	Value
τ	75 ms
E_{RANGE}	20 mV
w_{ij}	25
E_{ACT}	0 mV
E_{SYN}	-35 mV
I_{INJ}	-2.5 pA $\times$ 10 GΩ = 25 mV/s

Table 3.1: Default model parameters used for all cell models unless otherwise noted.

The switch is a dymod that forms the basis for the reversal maintenance circuit. It has two outputs: V_1 which drives forward locomotion, and V_2 which drives reverse locomotion. It has two hyperpolarizing pulse inputs I_{INJ1} and I_{INJ2} which are used to toggle the switch on and off.

3.1.3 The Oscillator Dymod

The switch dymod has two stable output states which are suitable for switching between forward and reverse locomotion, but they are static. To satisfy the requirement of an oscillatory output, the next step was to investigate the possible oscillatory behaviour of these model cells.

With these model cells, oscillations are not possible with only two cells (although see the discussion following). Oscillations can emerge when a third cell is added to the circuit. In three dimensions, the nullclines become null surfaces, and it is quite difficult to visualize the ways that three null surfaces might intersect and the kinds of trajectories that might arise. Optimization techniques such as genetic al-

gorithms have been used to construct neuronal dynamical systems in the past (Beer and Gallagher, 1992; Yamauchi and Beer, 1994), but because it was difficult to express a suitable objective function for the desired behaviour, a direct engineering approach was taken.

The oscillator dymod was built up from the two-cell inhibitory switch by adding an third cell (Figure 3.6). The basic idea was that the third cell would be made to "pull" the trajectory in its opposite direction, but operating with a delay to result in oscillations.

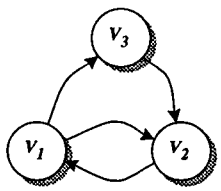


Figure 3.6: Oscillator circuit

Because the two switch cells already made inhibitory connections with one another, they were assumed to be restricted to making inhibitory connections with the third cell. This is the same application of Dale's Principle as used in Chapter 2 and it was applied with the understanding that it is not always true. Because of this inhibitory connection, cell 3 would therefore become activated as cell 1 turns off, and by making an inhibitory connection to cell 2, it would "pull" cell 2 off in the desired manner.

When the third cell was added, this system produced a new stable fixed point instead of oscillations because the activation of cell 3 was not delayed with respect to cell 1. A delay was added by reducing cell 3's activation range to 5 mV, but leaving its half-activation potential the same as the other cells (to preserve the tonic

synapse assumption). The increased steepness of the activation curve gave cell 3 an effective activation delay compared to cell 1 (see Figure 3.7). (This "delay" is not a "hard" delay in the sense of $\Delta_\tau(t) = t - \tau$, but rather an apparent delay induced by different activation characteristics.) The activation range was chosen to be steep enough so that cell 3's synaptic transmission only occurred when cell 3 was almost maximally activated, resulting in a pulse-like activation pattern (see the s_3 trace in Figure 3.8). Note that the assumption that cells are tonically active at their resting (leakage) potential does not necessarily imply that they are active at their in-circuit steady-state potential. In fact, in the oscillator, cells 2 and 3 never attain their leakage potential (0 mV in the simulated traces). See Section 2.7 or Wicks, Roehrig and Rankin (1996) for a treatment of steady-state potential versus leakage potential.

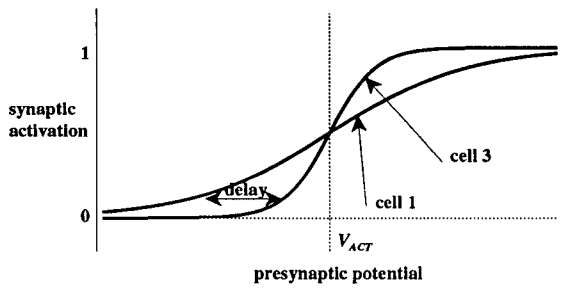


Figure 3.7: Delay due to increased steepness of activation curve. Cell 3 has steeper activation characteristics than cell 1. Because both are centred at E_{ACT} (i.e. they are tonically active), cell 1 will begin to activate earlier at a lower presynaptic potential than cell 3.

Adding this delay did indeed result in oscillations which tended toward a limit cycle (Figure 3.9). As with the switch, an inhibitory current pulse of 2.5 pA into cell 1 was used to start the oscillations and the same pulse into cell 2 was used

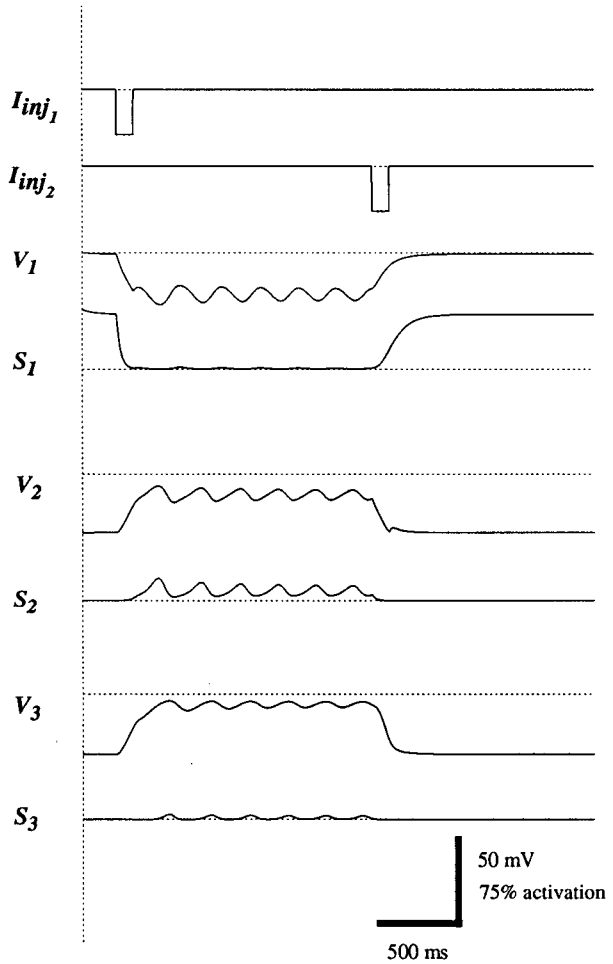


Figure 3.8: Simulated recording traces for an oscillator. The s_i curve is the synaptic activation for cell i ($0 \leq s_i \leq 1$) given by $s_i = s(\frac{V_i - E_{ACT_i}}{E_{RANGE_i}})$.

to stop the oscillations (Figure 3.8).

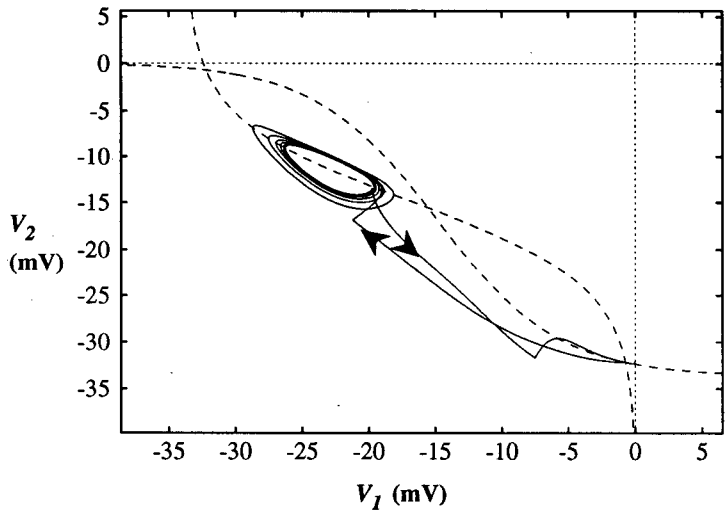


Figure 3.9: Phase portrait of an oscillating switch. This graph shows the projection of the phase space trajectory onto the (V_1, V_2) plane. The trajectory starts at the lower right-hand fixed point and settles onto a limit cycle before an inhibitory pulse returns it to its stable state.

3.1.4 The Charger Dymod

The next step in constructing the reversal maintenance circuit was to design the circuit to automatically generate the inhibitory pulse into cell 2 that stops the oscillation. This was done using a charger dymod that received input from the oscillator and gradually increased its activity over a few seconds before exceeding a threshold and delivering the inhibitory pulse to cell 2.

A single cell did not easily work as a charger, since it needed excitatory input in order to charge to a threshold, and so far the circuit design only had inhibitory neurons in it. So again, the two-cell inhibitory switch was chosen as the

starting point for the charger dymod. The charger dymod was designed separately as an independent module, with a periodic pulse train test input. The input cell that received the pulse train was labeled cell 4 and its partner, cell 5.

When used as input to the switch, a periodic pulse train induced forced oscillations which can be clearly seen in (V_4, V_5) phase space (Figure 3.10). There are three different qualitative behaviours that can occur, depending on the strength of the pulse input. For small pulse inputs, the switch is driven into oscillations which settle onto a limit cycle around the nullclines near the initial fixed point. For large inputs, the switch is driven first into its other state and then into small stable oscillations near the other fixed point.

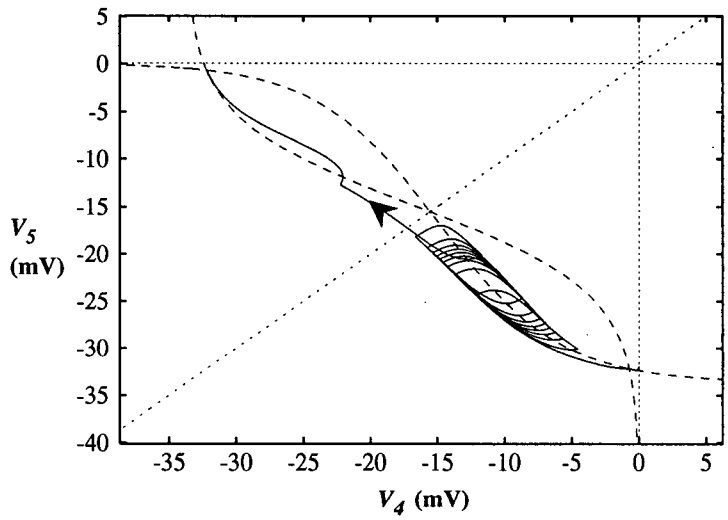


Figure 3.10: Phase portrait of charging behaviour. The input pulse train kicks the trajectory a bit further each cycle until it finally crosses the threshold. (Input strength of $14.7\text{mV}/s$).

Between these extremes is a fairly narrow, continuous range where the forced oscillations slowly creep along the nullclines until the trajectory nears the

switching threshold at which point it is driven across and the switch changes state (Figure 3.10). This is the desired charging behaviour. The charging time depends on the input level, and is determined by the number of oscillations that occur before the trajectory crosses the threshold. Figure 3.11 shows the simulated recording traces for the charging behaviour. The synaptic activation of Cell 5 is suppressed until the trajectory crosses the threshold, at which point it becomes activated. Cell 5 was therefore used as the output of the charger dymod to deliver the inhibitory "turn-off" signal to the oscillator.

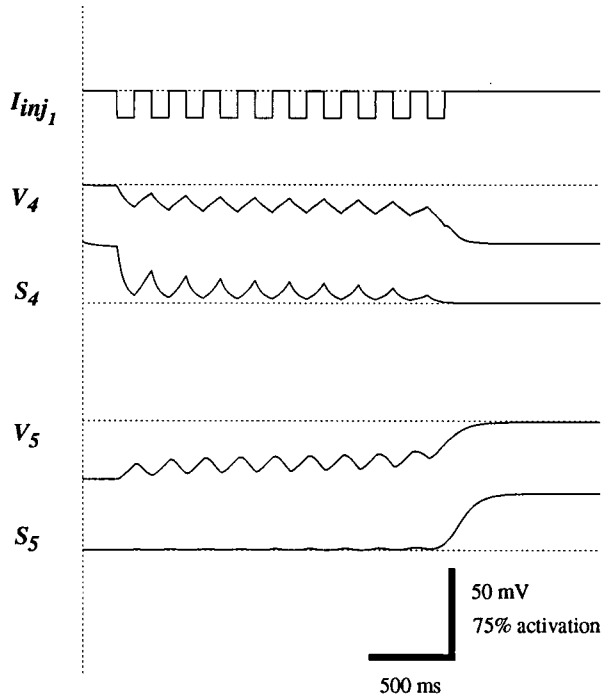


Figure 3.11: Simulated recording traces of charging circuit. The s_i curves are as defined in Figure 3.8.

This charging behaviour, although quite sensitive to the input level, is not a singular occurrence and can occur whenever the nullclines have 3 intersections

to form a switch. However, the appropriate input range for charging is sensitive to the precise configuration of the nullclines, and therefore depends on the other cells' parameters. In the example in Figure 3.10, the charging behaviour occurred over an input range of about 0.1 mV/s.

3.1.5 Dymod Assembly: The Reversal Maintenance Circuit

The final stage in the construction of the reversal maintenance circuit was to connect the charger dymod to the oscillating switch dymod to turn it off (Figure 3.12).

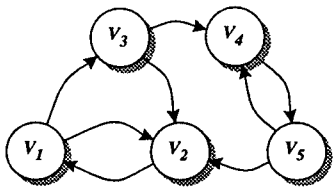


Figure 3.12: Complete reversal maintenance circuit.

Cell 3 was chosen as input to the charger dymod because its synaptic output was pulse-like (see s_3 in Figure 3.8). The charging behaviour of the circuit was first tuned with its output disconnected by adjusting the input coupling strength to cell 4 (w_{43}) until the appropriate charging behaviour was manifest. This was done by observing the charging trajectory in (V_4, V_5) phase space as in the design of the charger. Cell 5 was then connected to Cell 2 to deliver the inhibitory "turn-off" signal, and its coupling strength (w_{25}) was gradually turned up until the oscillator turned off. During this process, the input coupling strength to cell 4 needed to be adjusted to compensate for the interactions between the oscillator and the charger and maintain the appropriate charging behaviour. The process was straight-forward

and required only a few iterations of adjustments. The simulated recording traces of the complete circuit are shown in Figure 3.13.

Note that the shape of the charger traces (V_4, V_5) in Figure 3.13 is more rounded than those of Figure 3.11. This could be because a more rounded pulse (s_3 in Figure 3.13) was used as charger input (compared to the square pulse of Figure 3.11), but it might also be due to some non-linear interactions between the oscillator and charger dymods.

3.2 Discussion

In this chapter a circuit was designed to help explain how the nematode *C. elegans* makes transitions between forward and reverse locomotory modes and maintains its reversal for a duration much longer than the time constants of the cells controlling the behaviour. The circuit was designed using the novel approach of combining separate dynamical modules ("dymods") governed by non-linear ordinary differential equations. In spite of the non-linear interactions between dymods, the assembly was remarkably straight-forward suggesting that a human-engineered approach might not be so difficult as it seems.

The system is depicted in Figure 3.14 and consists of three multi-cell dynamical modules (dymods): a switch, an oscillator, and a charger. The switch bifurcates between two stable states in response to a tap stimulus: a stable forward state where V_1 drives forward locomotion and V_2 is off, and a reverse state where V_1 is off and V_2 drives reverse locomotion. The oscillator was constructed out of the switch by adding an "oscillator helper" cell V_3 to "pull" V_2 against its trajectory, but with a

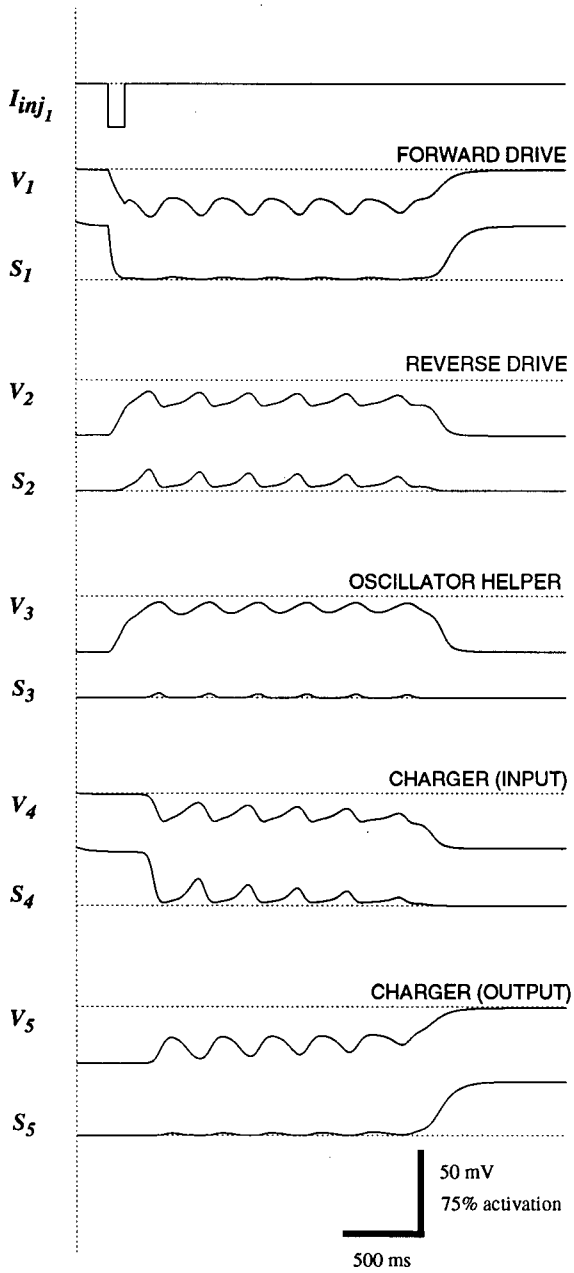


Figure 3.13: Simulated recording traces for the complete reversal circuit. The s_i curves are as defined in Figure 3.8. Non-default parameter values: $E_{RANGE3} = 5$ mV; $w_{25} = 5$; $w_{43} = 30$. See Table 3.1 for other parameter values.

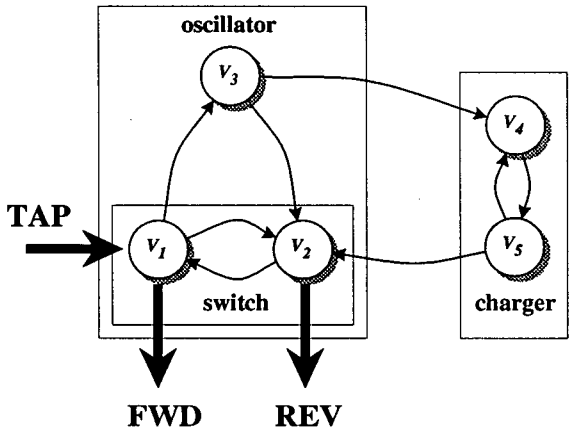


Figure 3.14: The dymod view of the reversal maintenance circuit.

"soft" delay (as opposed to a "hard" $t - \tau$ delay) which was induced by adjusting the synaptic activation characteristics of V_3. The oscillator's job was to generate an oscillatory locomotion signal for reverse locomotion. (Note that the forward signal is not oscillatory, but it can be made so by connecting it to another oscillator.) The reverse locomotion is turned off by a signal from the charger (V_5). The charger is a variant of the switch which gradually approaches its switching threshold in response to an oscillatory input from V_3. When the threshold is reached, it switches state and V_5 delivers its "turn off" signal to end the reversal. The circuit's operation can be seen in Figure 3.13.

3.2.1 The Locus of *C. elegans* Reversal Dynamics

The reversal maintenance circuit constructed here was based on physiologically realistic cell models, but did not take into account the known anatomical connectivity of the nervous system. The function of the two switch cells V_1 and

V_2 bear a striking resemblance to that of two pairs of ventral cord interneurons: V_1 corresponds to the bilateral pair AVBL and AVBR which connect to the forward motorneuron pool and V_2 corresponds to the pair AVAL and AVAR which connect to the reverse motorneuron pool (White et al., 1986; Chalfie et al., 1985). However, neither AVAL nor AVAR make direct reciprocal connections with either AVBL or AVBR (White et al., 1986) which suggests that they do not function as a switch of the form V_1, V_2. Another interneuron pair, PVC, makes strong reciprocal connections with AVA and is involved in the response to tail-touch (Wicks and Rankin, 1995). However, laser ablation of PVC does not affect the duration of spontaneous reversals (Wicks and Rankin, 1997), which suggests that it is not involved in the reversal maintenance dynamics. As no other interneurons from the tap withdrawal circuit make any strong reciprocal connections (Wicks and Rankin, 1995; Chalfie et al., 1985; White et al., 1986), this suggests that if reversal dynamics is governed by a circuit like that of Figure 3.12, the circuit likely does not lie in the interneuronal circuitry and may instead lie downstream in the motorneuron circuitry or may involve other processes than network level dynamics.

Mechanical interactions may play a role in reversal dynamics. When a reversal is initiated, the body goes into a deep bend which is thought to activate a stretch-related current (Tavernakis, Shreffler, Wang and Driscoll, 1997; Corey and Garcia-Anoveros, 1996) which could modulate neuronal function. Niebur and Erdös's (1991) model of *C. elegans* locomotion circuitry suggests that interactions between the interneurons, muscle cells, body wall and the medium in which the nematode swims all may play a role in determining the locomotion dynamics. It

may be possible, however, to rule out an external component in the reversal dynamics. When the worm is placed in water, thus removing the mechanical resistance to body movement, it thrashes in an oscillatory manner during periods of forward locomotion and bends into an unmoving hoop during periods of spontaneous reverse locomotion before resuming thrashing If the duration of this "reversal" period is unchanged in water, it would suggest that the reversal maintenance dynamics is generated entirely internally without interaction with the medium in which it swims.

It is also possible that reversal dynamics is governed by more diffuse processes which modulate synaptic activity such as neuropeptides (Schinkmann and Li, 1992). A slowly increasing neuropeptide concentration might be a more plausible charging mechanism than the charger presented in this chapter which has very sensitive charging regime and would need some kind of precise regulation.

In order to reduce the complexity of the model, gap junctions were ignored during the design process. However, there are strong gap junctions connecting the tap withdrawal interneurons to the motorneurons governing locomotion (Chalfie et al., 1985) and they may also play a role in the reversal maintenance dynamics.

Further experiments in the animal could narrow down the possibilities for the locus of the reversal dynamics. Chalfie et al. (1985) and Wicks and Rankin (1995) have successfully used laser ablations to determine the functional roles of neurons in the reversal response to mechanosensation. If the charging component of the reversal dynamics was performed by a neuronal circuit like V_4 and V_5 in Figure 3.12, one would expect that a suitable ablation would destroy the charging behaviour and the reversal would continue without stopping. This has not been the case for the

ablations performed on any of the tap withdrawal interneurons (Wicks and Rankin, 1995; Wicks and Rankin, 1997). It may be possible to perform an ablation study of the locomotory motorneurons to find such a cell, but if the charging mechanism is a modulatory peptide rather than a neuronal circuit, no such cell would be found.

Modelling efforts in *C. elegans* are frustrated by the inability to record neuronal activity during behaviour. It appears unlikely that the usual electrophysiological recording methods will ever be possible *in vivo* in *C. elegans* since its external cuticle is under hydrostatic pressure and the worm explodes when it is punctured. However, new genetic techniques have been developed to engineer *C. elegans* strains that express green fluorescent proteins (GFP) in specific cells (Chalfie, Tu, Euskirchen, Ward and Prasher, 1994) and it may be possible in the future to obtain activity information from voltage-sensitive fluorescence resonance energy transfer (FRET) imaging (Gonzalez and Tsien, 1995) using fluorescent proteins expressed in the cells of interest. For the quantitative studying of locomotory behaviour, the most interesting cells are AVB and AVA, the command interneurons that play a key role in mediating forward and reverse locomotion, respectively. Knowing the activity patterns of these cells during a reversal — whether they have transient activity, prolonged activity or oscillatory activity — is crucial to further modelling efforts.

3.2.2 Oscillations

Niebur and Erdös (1991) found in their computer model that an oscillatory signal was needed to drive the locomotion circuitry and so the model circuit was assumed to exhibit oscillatory behaviour. While this is weak evidence of the existence

of a neuronal oscillator, other evidence of intrinsic rhythmic signals in the animal (e.g. the defecation cycle (Thomas, 1990), and side-to-side head oscillations) led us to consider how oscillations might arise from these tonic cells. Additionally, in the initial concept of the charger it was considered that a rhythmic pulsed input would allow for a longer charging time and less sensitivity to the input strength than if a continuously increasing input signal was used, but this has not been verified.

In the model, oscillations were obtained by adding a third "delay" cell into the switch circuit (Figure 3.1). This delay was accomplished within the constraints of the simple cell model by using a narrow synaptic activation range (Figure 3.7). However, there may be other ways to obtain oscillations. The dynamic processes underlying synaptic transmission could provide a sufficient delay for oscillations. The cell model used here did not include synaptic dynamics but instead considered synapses to act instantaneously. In earlier work with these model cells (Wicks et al., 1996), we found that a simple model of synaptic dynamics that used a first-order relaxation equation and a 10 ms time constant made no appreciable difference to our simulations. However this synaptic model also did not include a hard delay and it may be possible to obtain oscillations with a synaptic model that included hard delays.

Beer (1995) has shown that oscillations can arise in a 2-cell circuit when the cells have self-connections. The cell models he used are continuous time recurrent neural network (CTRNN) cells (Funahashi and Nakamura, 1993) and are very similar to the *C. elegans* cell model used here, differing only in that CTRNN cells use a synaptic driving force that is assumed to be constant. Although cells in *C. elegans*

are typically unbranched cylindrical processes and are unlikely to form anatomical self-connections (White et al., 1986), Goodman et al. (1998) have recently discovered a regenerative calcium current in a *C. elegans* neuron which could achieve the same effect as a self-connection.

A vast amount of work has been done on neuronal oscillations. Some related modelling work in other biological systems include the central pattern generator (CPG) of the lobster stomatogastric ganglion (Marder and Selverston, 1992; Abbott, Marder and Hooper, 1991) which uses a bursting "pacemaker" cell to generate rhythmic oscillations, and the *Tritonia* sea slug locomotory CPG which generates oscillations at a network level (Getting, 1989). Jung, Kiemel and Cohen (1996) have undertaken a bifurcation analysis of the dynamics of a computational model of the lamprey locomotor CPG. Considerable theoretical work has been done on the dynamics of neuronal oscillations (Hoppensteadt and Izhikevich, 1997; Ermentrout and Kopell, 1990; Beer, 1995), and its physiological mechanisms (Skinner, Kopell and Marder, 1994). Campbell and Wang (1998) have analyzed the dynamics of neuronal oscillators that use hard delays. Yang and Dillon (1994) have proven using cells similar to the *C. elegans* model that 2-cell oscillations not possible without self-connections, and Yang (1995) has analyzed a 3-cell network oscillator similar to the one derived here.

It is also interesting to note that *C. elegans* cells may have some generality as model cells. They are almost the same as CTRNN cells which have been shown to be universal dynamics approximators (Funahashi and Nakamura, 1993). Although *C. elegans* neurons do not exhibit classical spiking behaviour, passive cell models

can be used to model spiking systems by considering cell activity to represent the firing frequency (Bialek, Rieke, de Ruyter van Steveninck and Warland, 1991; Jung et al., 1996). However, this should be tempered with evidence that temporal spiking patterns can encode essential information (Hopfield, 1995).

3.2.3 The Forward-Engineering Approach

In this chapter a forward-engineering approach was used to design a circuit whose operation resembles the reversal behaviour observed in the worm. It is natural to ask why this artificial system should tell us anything about reversal mechanism in the worm or give insight into biological systems in general, especially when there are so many other possible candidate mechanisms.

In the worm, this forward-engineered model elucidates the key concepts in the reversal mechanism: the active transition between two locomotory states and a charger that builds to a threshold, and it provides quantitative evidence that this transition could be accomplished by network dynamics alone, even using simple and physiologically plausible tonic cells. Even if such a mechanism is ultimately not found in the worm, this result has a useful generality. The model also provides a quantitative framework for adding new data about cellular and intra-cellular activity in *C. elegans* as it becomes available, and it suggests an approach using dynamical phase-plane analysis that could be used in subsequent models. Similar forward engineering approaches have been used to model the leech local-bending reflex (Lockery and Sejnowski, 1992), cockroach locomotion and escape (Beer and Chiel, 1993), and chemotaxis in *C. elegans* (Morse, Ferrée and Lockery, 1998). In

these studies, optimization techniques were used to train neural networks to produce behaviour which approximates that of the animal. Although there is no guarantee that the resulting circuits actually exist in those animals, those artificial models provide insight into the issues involved when there is insufficient biological data to construct anatomically realistic models.

Where artificial forward-engineered systems have an advantage over strict biological models is in their effectiveness for producing generalizable understanding of how behaviour can arise from neuronal systems. Ultimately, we are not particularly interested in how the worm locomotes — we study it because we hope to understand general principles about neuronal systems that we can apply to other biological organisms including humans. There is no guarantee that neuronal circuitry or dynamical mechanisms found in the worm will also be found in other organisms, and in this sense an artificial forward engineered system is on the same footing as a real biological system for producing generalizable knowledge. However, detailed biological studies are extremely time consuming and arduous endeavours: for example, nearly 30 years of research on *C. elegans* circuitry still has failed to provide an adequate cellular account of a simple behaviour in a simple organism. In contrast, artificial systems are relatively cheap and quick to construct and analyze.

The selection of a model biological system involves an intuition about the tractability of its analysis and the generality of its mechanism. The same intuition can be applied to the selection of an artificial system in order to ensure that it gives insight that generalizes to biological systems. This knowledge transfer can be improved by using models based on physiological mechanisms and expressed

in physiological units so they can be readily interpreted by biological researchers working on other systems.

3.2.4 The Human Engineered Approach

Previous approaches to engineering non-linear dynamical systems used computer optimization techniques such as recurrent backpropagation (Pearlmutter, 1989), genetic algorithms (Beer and Gallagher, 1992), or simulated annealing (Morse et al., 1998). A computer optimized approach has the advantage of finding solutions when the solution space is large and there is little human intuition for guidance. However, this is also the weakness of a computer-generated approach when the goal is to develop an intuition for how behaviour can arise from network dynamics. Unless the dynamics of resulting circuit is mathematically analyzed (as in Beer and Gallagher, 1992), the mechanism remains mysterious and the designer misses out on the intuition that might be gained by understanding its function. Dynamical systems become extremely difficult to analyze as they increase in size (see Beer (1995) for some of the difficulties involved) and as we progress to larger systems, a computer-generated solution may not be amenable to mathematical analysis.

In this chapter, a novel approach was taken instead: a human engineer used intuition about the dynamical processes involved to hand-engineer a modular set of component dynamical systems that were assembled to form the final circuit. As far as the authors know, this is the first attempt to engineer a non-trivial continuous dynamical system in this way. A human engineered approach forces the designer to identify the issues and develop an intuition about the problem. This chap-

ter suggests that in spite of the complexities of non-linear interactions, a human-engineered approach might not be so difficult as it seems.

To facilitate a human-engineered approach, the design of the dynamical system was done in steps by building a simple component (the switch) and using it to build more complex components (the oscillator and charger) which were assembled to form the complete system. To simplify discussions, the term "dymod" — short for dynamical module — was coined to describe a continuous dynamical system that is used as a component in the design of a larger dynamical system (which itself is a dymod that could be used to build still larger systems). The term was intended to be reminiscent of terms describing computer programming modules such as objects, functions and procedures.

It is remarkable that a complex, interdependent, non-linear dynamical system can be designed in parts and recombined in this way. The design process proceeded according to plan from the original sketch to the final assembly of the dymods into a working system in a surprisingly straight-forward manner. It is interesting to speculate whether it is merely a fluke that in this case the non-linear interactions did not destroy the functional properties of the separate modules, or whether this approach is generalizable: are dymods a useful general concept for building non-linear dynamical systems? How scalable is this approach?

Because there is no convenient unifying superposition principle for non-linear systems as there is for linear systems, there is no guarantee that it is possible in all cases to combine dymods without destroying their function. The important point here is that we may not necessarily need a guarantee for *all* cases. It may

be sufficient to know how a dymod can be used in a finite set of cases and how to correct for any non-linear interactions in those cases.

There are at least two industries that operate on a similar principle to produce extremely large and complex non-linear dynamical systems: the electronics industry, and computer software engineering. They have no appreciable theoretical framework or guarantees (though there is considerable work being done in this direction). Instead of a unifying principle, these fields have a vast collection of techniques for solving specific problems (e.g. Knuth, 1997; Horowitz and Hill, 1989). Each technique has a great deal of lore associated with it: when it is appropriate, what pitfalls to watch for, modifications for specific cases, and the success of a solution depends in a large part upon the intuition and experience of the designer who has accumulated this lore. The overwhelming success of these industries suggests that a pragmatic, informal approach to combining non-linear modules might be a good starting point in the absence of a unifying theory.

If a modular approach to constructing non-linear dynamical systems is possible in a general way, it has a number of advantages over optimization techniques which generate an entire solution at once. Optimization algorithms require the specification of an objective function which characterizes the desired behaviour as a single real-valued "score". This can be sufficient for simple behaviours, but it is uncertain whether complex behaviours or interactions between multiple behaviours can always be characterized so simply. A human-engineered design is limited only by the intuition of the designer and has no requirements that its behaviour be characterized in this way.

A modular approach may be more scalable than an all-at-once technique since different dymods can be designed and implemented independently. A modular solution is also easier to understand since it provides a decomposition in terms of higher-level building blocks (e.g. compare the dymod view of the reversal circuit in Figure 3.14 with the "flat" cellular view of Figure 3.12). Finally, forward-engineered dymods might suggest higher-level building blocks to use in understanding biological circuits.

Related work is being done in the emerging field of hybrid systems (Lygeros, 1996) which also seeks to combine non-linear, continuous dynamical systems in a modular way. However, in a hybrid system, the interactions between distinct continuous systems are discrete automata so the result is a hybrid discrete/continuous system. By contrast, the dymod approach seeks to combine continuous dynamical systems into larger continuous dynamical systems and hybrid systems theory does not provide any framework for doing this.

3.2.5 Conclusions and Future Directions

The reversal dynamics circuit designed in this chapter is a dynamical system that governs two discrete behaviours: a persistent forward locomotion and a temporary reversing locomotion. These two behaviours have different underlying mechanisms and the transitions between them are crisp: there is no blending of the two behaviours. Yet both behaviours are governed by a single continuous dynamical system consisting of simple tonic non-spiking cells. These behavioural transitions occur as the result of bifurcations in the underlying dynamical system. How

general is this phenomenon? Can bifurcations explain other forms of behavioural transitions?

My further research lies in exploring this question by expanding upon this human-engineered dymod approach. I am constructing a robotic implementation of an artificial creature that exhibits several interacting behaviours, starting with the obvious choices of locomotion, mechanosensation and chemotaxis (using light instead of chemical sensation, in the same manner as Morse et al., 1998). While these behaviours have been well studied in a variety of organisms, I do not seek to mimic the mechanism of any specific organism, but rather to develop mechanisms that make neuroethological sense within the desktop environment and tracked tank-like locomotion of the robot, and relate them to the mechanisms found in other organisms to extract their salient features.

This artificial creature approach borrows heavily from the pioneering work of Brooks (1986a), Cliff (1991), Beer (1997), and others and combines the modular behaviour approach of Brooks's (1986b) subsumption architecture with Beer's (1997) dynamical systems approach. The goal of the project is to engineer each behaviour in a modular way, but so that the complete control system is a single continuous dynamical system of biologically plausible cellular mechanisms. This project will also be a good test case for the validity of the human engineered dymod approach since I intend not to resort to automated trial-and-error methods until my intuition completely fails.

It is my firm belief that the accumulation of lore and intuition by individual researchers is a necessary precursor to the development of any theoretical frame-

work for understanding the cellular dynamics underlying behaviour. In fact, it is my suspicion that the brain may not yield to any unifying theory: the accumulated lore may be all we get.

The dymod research strategy proposed here is a pragmatic, generative one based on experimentation to determine the conditions under which dynamical systems can be decomposed into modules. To facilitate this strategy, the next chapter presents a framework for implementing and interconnecting dymods in the form of a digital networking protocol called DSS that solidifies the separation of interface from implementation that characterizes dymods.

Chapter 4

The DSS Network Protocol for Dymod Implementation

4.1 Introduction

A dynamic module (dymod) is a non-linear continuous dynamical system together with a qualitative description of its inputs, outputs, and functional characteristics. Dymods are an approach to constructing and understanding large non-linear dynamical systems in term of smaller modules. They were originally conceived to describe neuronal dynamics in nervous systems (Roehrig and Rankin, 1998) and for experimentally testing their behavioural characteristics (for example, see Figure 4.1), but they may also apply to other control systems governed by continuous dynamical laws.

This chapter presents an experimental implementation framework for dymods in the form of a network protocol called Digital Signal Sockets (DSS). Dy-

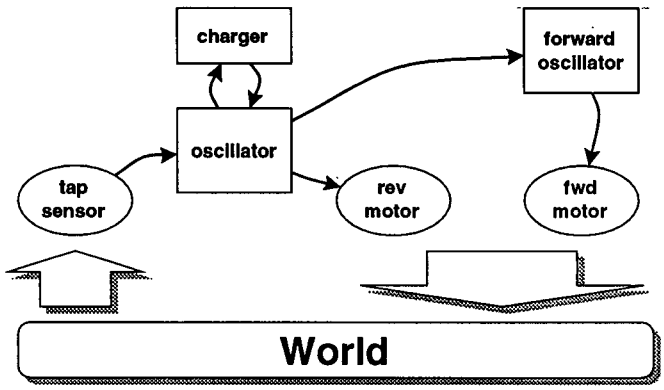


Figure 4.1: A neuronal control system for the reversal maintenance behaviour of the nematode *C. elegans* (Roehrig and Rankin, 1998). It consists of several independent dymods that are interconnected with sensors and actuators to form a complete behavioural system. The small arrows depict dymod interconnections representing continuous signals. The squares indicate dymods which are governed by non-linear ODEs and the circles represent sensory and motor transducers

mods are governed by systems of ordinary differential equations (ODEs) and are implemented using numerical integration techniques. DSS allows dymods to be interconnected with continuous signals and to operate asynchronously using independent time steps for their internal computation. It also allows these numerical simulations to incorporate live signals from sensors and actuators. Most importantly, the DSS protocol solidifies the modular aspect of a dymod by providing an explicit definition of a dymod's interface inputs and outputs.

This work arose out of a prototype neuronal simulator software developed by the author to allow neuronal simulations to be constructed in a modular fashion (i.e. using dymods). The prototype simulator was used to numerically solve the system in Chapter 2, and it handled the explicit definitions of a dymod's input and output interface as well as the mechanisms to send signals between dymods and manage their interconnections. As that work progressed, it became clear that a cleaner and

more general approach could be taken by separating the task of numerical simulation from the task of managing the interface and communication of signals between different dymods. It also became clear that there was a natural correspondence between dymod interconnections and computer network connections and that for a modest extra effort, a dymod interconnection implementation framework could be cast into a networking protocol to allow large simulations to be distributed over multiple computers, but also to bypass the network if the simulation was entirely contained on a single computer.

4.2 Design Goals

The design goals of DSS were:

- to solve the problem of interconnecting independent real-time simulations which are numerically computed solutions to ODE initial value problems,
- to interconnect simulations using band-limited continuous signals that represent any one-dimensional physical quantity,
- to allow live signals to be incorporated into simulations,
- to handle all resampling issues that arise when two interconnected simulations use unrelated integration step sizes, and provide predictable error bounds,
- to multiplex signals over digital computer networks,

- to operate with workstations and networks that lack hard real-time facilities (e.g. typical UNIX workstations and TCP/IP networks used by neuroscience researchers), but also to be efficient and have guaranteed performance when these facilities do exist.

- to scale well in both number of connections and signal bandwidth. For neuronal systems, dymods typically have internal computation step sizes in the order of microseconds and inter-dymod signals have bandwidths in the 1 Hz – 1 kHz range. The DSS implementation framework is intended for these low bandwidth, high accuracy signals, but was also designed to scale to higher bandwidth signals as computer and network hardware permit, allowing for the implementation of arbitrarily large dymod systems.

4.3 Review of Existing Frameworks

Several existing frameworks were considered as candidates for dymod implementation. Quantitative studies of neuronal dynamics are done using numerical neuronal simulators such as Neuron (Hines, 1993) and GENESIS (Bower and Beeman, 1995). Although parallel and distributed versions are available (Pittsburgh Supercomputing Center, 1998a; Pittsburgh Supercomputing Center, 1998b), these simulators do not support real-time operation and interaction with live signals[1]. Real-time neuronal studies have been done using silicon artificial neural network (ANN) chips (Hammerstrom, 1995) and a simple bus protocol exists for intercon-

[1] although the DSS architecture can be readily incorporated in a GENESIS add-on package to add real-time and distributed support in the same straight-forward manner as the `mblib` GENESIS add-on package (see Chapter 5).

necting chips for modular, distributed operation (Mahowald, 1992; Northmore and Elias, 1997). However, these chips typically use limited or inflexible biological models and are complex and expensive to produce. The DIS protocol (IEEE, 1993; Schug, 1995) for performing distributed interactive simulation over the Internet is designed for real-time operation, but it is event-based and intended for military exercises and not suitable for the graded, continuous signals that realistic neuronal simulations require. There are several protocols for distributed digital audio production (Yamaha, 1998; Young Chang RDI, 1998; Peak Audio Inc., 1998) that are suitable for real-time interconnections of continuous signals, but they are either immature or proprietary, and are not well-suited to low-bandwidth signals or TCP/IP networks. Other real-time, distributed networking protocols exist for industrial automation and embedded systems (CAN, 1998; Profibus, 1998) and consumer and professional electronics (1394TA, 1998). These protocols do not provide the required functionality for a dymod implementation framework, but instead may serve as suitable network transmission layers for the higher-level DSS protocol. Of these, IEEE 1394 "Firewire" (1394TA, 1998; IEEE, 1995) is particularly appealing as a network layer because of its high bandwidth, real-time delivery guarantees, and the promise of cheap and widespread availability in the near future. For the TCP/IP networks, there are emerging standards such as RTSP (Real Networks, Inc, 1998) and multicast IP (Comer, 1995), to better handle real-time multimedia transmission over the Internet, and these are also suitable as DSS network layers.

The DSS design is independent of the underlying networking layer, but was specifically targeted at two network transport protocols: TCP/IP which is the

foundation of the Internet and the desktop workstations used by neuroscience researchers, and IEEE 1394 for high-performance dymod implementations with guaranteed performance.

4.4 Theory of Operation

Dymod interconnections represent a shared quantity between two continuous dynamical systems. For example, in the coupled system

$$\begin{aligned} \dot{v_1} &= F_1(v_1, v_2, t) \\ \dot{v_2} &= F_2(v_1, v_2, t), \end{aligned} \tag{4.1}$$

the variables v_1 and v_2 represent the shared quantities that couple the system and t represents time. This system is depicted in Figure 4.2 as two dymods with two interconnections. $T[v_1]$ and $T[v_2]$ represent the transmitted, reconstructed versions of v_1 and v_2 respectively.

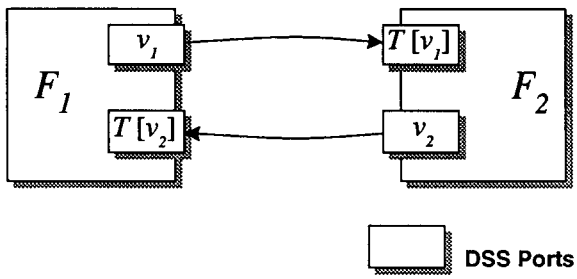


Figure 4.2: Dymod interconnections in a coupled two-dimensional system.

The DSS dymod implementation framework maintains a notion of simulation time that is distinct from the actual passage of time, but requires that computation be scheduled and synchronized to the actual *wall-clock* time. For example,

the discrete values $v_1(t_1), v_1(t_2), \ldots$ produced by the F_1 dymod must be delivered to the F_2 dymod within some interval of the actual times $t_1, t_2, \ldots$ which are taken relative to some starting time. There are several distinct problems to be addressed in a dymod framework: signal and time representation, signal reconstruction, connection management, time synchronization, real-time scheduling, and network transmission. The DSS protocol deals with signal and time representation, signal reconstruction and connection management. Solutions to the other problems are also discussed in the following sections.

4.4.1 Signal and Time Representation

DSS version 1 is experimental and deals with only one type of signal: a continuous real-valued signal that is bandlimited (containing no frequency components above a highest frequency). The Nyquist sampling theorem (Oppenheim and Schafer, 1989; Nyquist, 1928), states that bandlimited signals can be exactly represented by a discrete series of samples taken at a sampling rate of at least twice the highest frequency. This minimum sampling rate is called the signal's Nyquist frequency. DSS version 1 represents these signals by a discrete set of 32 bit values, but in future versions, this can be expanded to include other signal encodings.

DSS uses an absolute time reference for signal data, and each signal sample is timestamped with the absolute time at which that sample was taken. Using a timestamp allows a receiving dymod to recover the time the sample was taken without needing to account for the latencies and variabilities of network transmission. By using an absolute time reference, the DSS protocol is simplified because

the time synchronization problem can be solved by well-known time synchronization protocols (Mills, 1992). This lightens the requirements for the DSS network transmission mechanism so that DSS messages can easily be bridged between different network types (such as TCP/IP networks and the higher-speed IEEE 1394 isochronous network that has its own intrinsic clock) without requiring network-level clock synchronization.

In DSS version 1, the timestamp is a 64 bit fixed-point quantity representing the time in seconds since the epoch of midnight January 1, 1970 Coordinated Universal Time (UTC) and is computed according to the POSIX.1 `gettimeofday` system call (IEEE, 1996). Note that the value returned by this call is computed from the current time of day and ignores leap seconds. While this is adequate for experimental purposes, is discontinuous whenever a leap second is inserted into UTC (roughly every two years) and future DSS versions may incorporate a different encoding.

For a more efficient and compact representation, each DSS connection maintains a 32 bit reference time in whole seconds — the connection epoch — as the zero point for transmitted signal timestamps. The absolute timestamp is obtained by adding the sample's transmitted timestamp to this epoch. The transmitted timestamps are also represented in 32 bits and can only represent a fixed interval of time: in DSS version 1, the timestamp represents microseconds since the connection epoch, and will overflow the 32-bit quantity approximately 72 minutes after connection establishment. Before this occurs, the connection epoch is renegotiated as described in Section A.1.2. Future DSS versions may incorporate higher

resolution timestamps.

4.4.2 Signal Reconstruction

Dymod implementations operate independently and asynchronously, typically with internal computation rates that are higher than the signal interconnection sampling rates. Incoming signal samples must therefore be used to reconstruct the continuous signal in order to avoid step discontinuities between sample values. This reconstructed continuous signal is then resampled at the appropriate intervals for computation in the receiving dymod.

In Figure 4.2, the transmitted versions of a signal v is given by $T[v]$. The transmission functional T involves two undesirable components: delay and approximation. It can be written as:

$$T[v](t) = v(t - \delta) + \epsilon(t) \tag{4.2}$$

where δ is the delay incurred by signal transmission and reconstruction, and $\epsilon(t)$ is the error in the reconstruction.

There is a tradeoff between the amount of delay and the amount of interpolation or prediction error. For instance, in the system (4.1) it is possible to use a zero delay by fitting a polynomial to the samples of v_1 for $t < t_n$ and use it to predict values for v_1 in the interval $[t_n, t_{n+1}]$. This is a similar approach to the predictor-corrector numerical method for ODE integration (Gear, 1971), except for the corrector step: because the solution to (4.1) is distributed, the correction of v_1 via the computation of F_1 cannot be applied immediately to the computation of $v_2(t_{n+1})$, but must wait instead until the computation of $v_2(t_{n+2})$ when a

new v_1 sample has been received from the F_1 dymod. For smooth functions, the predictor-corrector method is highly accurate and this modification may still yield good results. However, the formal error analysis would be complex and this modified predictor-corrector approach is left for a future endeavour.

In some cases a delay is acceptable. If the physical system includes delays, a model can incorporate them so that the dymod interconnection delay plays a part in the model. This is the case for neuronal systems when synaptic transmission and axonal spike propagation involve delays.

If a delay is used, no prediction is required and the problem is a simpler one of interpolating an intermediate signal value from data samples on either side. For bandlimited signals, there is a well-established theory and a wealth of efficient numerical techniques for doing this (Crochiere and Rabiner, 1983). A bandlimited signal can be reconstructed exactly from its sequence of samples using the well-known sinc interpolation function (Oppenheim and Schafer, 1989)

$$v(t) = \sum_{n=-\infty}^{\infty} v_n \operatorname{sinc}(\pi(t - t_n)F_s), \tag{4.3}$$

where F_s is the signal's Nyquist frequency, t_n is the time value for sample n, $v_n = v(t_n)$ are the sample values, and $\operatorname{sinc}(x) = \sin(x)/x$. In practice, there are different techniques for truncating this doubly-infinite sum and implementing it using efficient digital reconstruction filters (Crochiere and Rabiner, 1983). For example, a k-tap finite impulse response low-pass digital filter uses a symmetrically truncated sum of k terms and incurs a delay of $\frac{k}{2F_s}$ in the signal pathway.

If an incoming signal has a higher bandwidth than the internal computation rate of the receiving dymod, the signal should be filtered to remove the higher fre-

quency components which would cause *aliasing* — a form of noise caused when higher frequencies obscure the low frequency trends when sampling the signal. In this case, F_s in (4.3) should be set to the Nyquist frequency of the receiver's bandwidth capabilities.

The reconstruction accuracy is determined by the design and width of the reconstruction filter. Better accuracy is obtained by using more input samples with a wider interpolation filter kernel, but at the expense of a longer delay. Figure 4.3 shows how reconstruction accuracy changes with filter width and delay.

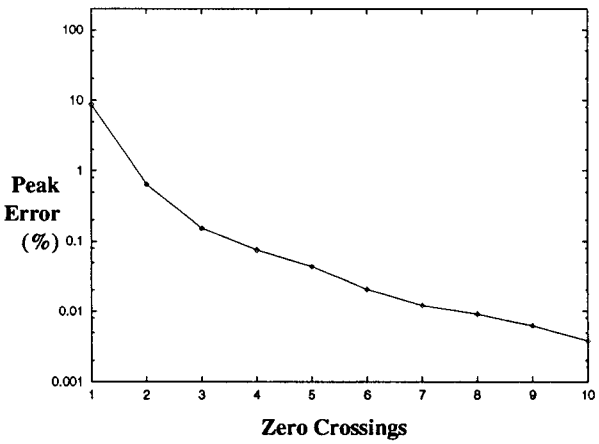


Figure 4.3: Reconstruction Error vs Filter Width. This shows typical peak interpolation errors in upsampling a spectrally-white test signal to 14.3 times its original sampling frequency F_s (n=1000 interpolations). The upsample ratio of 14.3 was chosen to avoid a simple rationally related sampling rate change, but these interpolation errors are typical of other ratios. Filter width is given as the number of zero crossings on either side of the sinc reconstruction kernel included in a Hann-windowed filter design. The test signal was bandlimited to $F_s/4$ so that it was in effect oversampled by a factor of 2. The filter delay can be computed by $delay = zero\ crossings/F_s$, and the approximate number of samples used by $\#\ samples = 2 \times zero\ crossings + 1$. The average errors were less than 0.00002% when at least 3 zero crossings were used.

In this experimental DSS version, a simple Hann-windowed filter design

(Oppenheim and Schafer, 1989) that included three zero crossings on either side of the sinc interpolation kernel was used, with a resulting delay of three sample periods. The reconstruction filter was computed using double-precision floating point arithmetic and was not optimized for minimum error or performance. Windowed filter designs such as this one yield good frequency-domain characteristics and efficient implementations, but do not produce optimal time-domain interpolations. Filter design techniques to minimize the maximum or least-squares error in the time domain are provided in Crochiere and Rabiner (1983). For a more efficient fixed-point arithmetic interpolation filter that supports non-uniform resampling and is suitable for hardware or embedded DSP applications, see Smith and Gossett (1984).

Reconstruction accuracy is also affected by "lost samples" which occur when a sample has not been received in time to be used in the signal reconstruction. This can happen for three reasons: real-time scheduling problems, time synchronization errors, and network delay.

4.4.3 Real-time Scheduling

Accurate signal reconstruction is critically dependent upon accurate real-time scheduling of computation. If a receiver's computation begins too early, the samples required for computation may not have arrived; if it begins too late, the computation might not complete in time resulting in a delay (and more lost samples downstream). These lost samples result in a reduction in the signal reconstruction accuracy and possibly exceeding the error tolerance of the application. This can be

mitigated by oversampling the signal (sampling it at a rate greater than its Nyquist frequency) so that even if occasional samples are lost, the Nyquist criterion is still satisfied.

A dymod's computation of a value $v(t_i)$ at time t_i must be scheduled to commence at time t_i in order to incorporate any live signal values at time t_i. In order to keep up with real time, this computation must be completed by time $t_{i+1} = t_i + dt$ where dt is the dymod's internal step size. If δ_C is the time taken for internal computation, then δ_C/dt is the dymod's "real-time load".

When $v(t)$ is transmitted to another dymod, its transmitted version is $T[v](t) = v(t-\delta)+\epsilon(t)$. In order to ensure that no samples are lost, the receiver must schedule the computation that makes use of $T[v](t)$ to occur in the interval $[t, t + T_s]$ where T_s is the sampling interval of the transmitted signal v and, in addition, the signal delay δ must satisfy

$$\delta > \delta_R + \delta_C + \delta_N, \tag{4.4}$$

where δ_R is the reconstruction delay imposed by the reconstruction filter, δ_C is the computation time taken to produce samples of $v(t)$, and δ_N is the network transmission time. Note that δ_R is determined by the reconstruction filter design characteristics while δ_C and δ_N are empirical quantities determined by the speed of the computer and network hardware. Faster hardware can reduce δ_C and δ_N, but only a design change in the reconstruction filter can reduce δ_R.

In our experimental DSS version 1, the filter delay δ_R is negotiated at connection time based on the sender's and receiver's sample rate and filter width, and the signal's delay δ is set to δ_R with no provision for δ_C and δ_N. This does not

present a problem providing $\delta_C + \delta_N < T_s$, in which case at most one sample will be lost per reconstruction. In our experiments, we compensated for this by over-sampling by 2 times. The signal delay is made available to the application via the DSS API (`dss_getinfo`, Section A.2) Future versions of DSS will provide the means to establish and monitor δ_C and δ_N in addition to δ_R.

Embedded systems typically have excellent real-time facilities, but UNIX-based dymod implementations are more problematic. The Posix `setitimer` system call is typically used to schedule real-time activity on UNIX systems, but real-time guarantees vary widely between UNIX implementations. Sun's Solaris and SGI's IRIX are considered real-time operating systems (Real-Time Magazine, 1998) with guarantees on real-time performance, but that is somewhat moot considering the lack of any real-time guarantees for TCP/IP networks. A UNIX and TCP/IP dymod implementation is intended for experimenting with the qualitative behaviour of systems and should not be relied upon for guaranteed numerical accuracy. However, the performance of UNIX systems and TCP/IP networks can be quite adequate for behavioural experiments providing that the system limitations are tested and determined in advance. Some UNIX implementations — on PC hardware in particular — have poor real-time clock support with resolutions in the tens of milliseconds, and any more precise scheduling is impossible. An embedded DSP (digital signal processing) implementation using a real-time network such as IEEE 1394 can be used for guaranteed numerical accuracy. For a good practical starting point on real-time computation issues, see the online Real-Time Encyclopaedia (Real-Time Magazine, 1998).

4.4.4 Time Synchronization

DSS requires accurate time synchronization between sender and receiver, but does not provide a synchronization mechanism and instead relies on an external protocol. On TCP/IP-based UNIX implementations, the NTP protocol (Mills, 1992) is used to synchronize the workstation clock to a reference signal obtained by radio or the Internet. In embedded implementations using isochronous networks, a hardware clock synchronization mechanism can be used. In order for an embedded implementation to interact with a TCP/IP implementation, the embedded network clock master should be synchronized to Universal Coordinated Time.

The accuracy of time synchronization limits the real-time scheduling accuracy and therefore the signal reconstruction accuracy as described in the previous section. Most typical UNIX installations can maintain a synchronization accuracy to within a few tens of milliseconds using NTP (Mills, 1990). This time synchronization error is the major factor that limits signal bandwidth when using DSS in a UNIX and TCP/IP environment. Sub-millisecond accuracy is possible using special kernel patches and extra hardware (Mills, 1994).

In our experimental test network, system clocks were synchronized with an accuracy of approximately 10 milliseconds and we experienced nominal reconstruction errors when the sampling interval was substantially greater than the time synchronization error, but the reconstruction errors increased dramatically to 100% as the sampling interval approached the time synchronization error and the resulting real-time scheduling errors caused samples to be lost from the reconstruction (Figure 4.4).

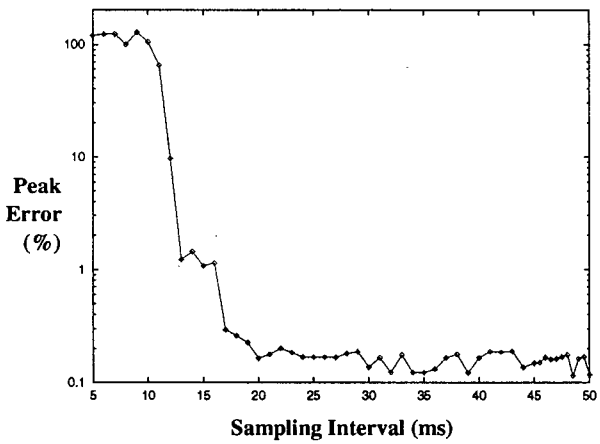


Figure 4.4: DSS Reconstruction Errors vs Sampling Interval. This shows how the peak interpolation error changes as the sampling interval approaches the time synchronization error (approx 10 ms) and the resulting real-time scheduling errors cause samples to be lost from the reconstruction. A spectrally-white test signal that was bandlimited to $F_s/4$ was used and it was upsampled at the receiver by a factor of 14.3. Three zero crossings were used for the reconstruction filter resulting in nominal reconstruction errors of 0.2% (cf. Figure 4.3) when no samples were lost.

4.4.5 Connection Management

DSS signal connection endpoints are called DSS ports and they are identified by a DSS address (see Section A.1.1). DSS connections carry unidirectional signals and therefore ports must be of either input or output type. This directionality only refers to signal data transmission; all ports can send and receive DSS control messages which are used to establish and remove connections between DSS ports. Both ports must exist and be active before the connection can be made. The use of DSS control messages for connection management allows dymods to be implemented independently of any controlling user interface: for instance, a dymod implementation can be a featureless black box with a network jack, and a DSS-enabled web browser could be used to control its behaviour and to connect it to other dymods. To facilitate connection management, the DSS protocol includes a name service to associate symbolic names with DSS ports. This allows a user interface to specify connection endpoints by name rather than their DSS addresses. In future versions, the name service may also provide additional information such as the physical quantity and units the signal represents.

4.4.6 Network Transmission

The DSS protocol does not include a network transmission mechanism and is meant to be layered on top of an additional networking protocol (see Figure A.1). DSS requires two kinds of network transport services: an unacknowledged *isochronous* datagram service for delivering sample data at regular rates, and an *asynchronous* acknowledged datagram service for control messages. This experimental

DSS version was implemented on top of TCP/IP using UDP (Comer, 1995) for both isochronous and asynchronous services. Future versions may be implemented using the emerging real-time streaming protocol (RTSP) (Real Networks, Inc, 1998) for TCP/IP.

Typical 10 Mbit Ethernet LANs can saturate at 4 Mbit/s data rates. The DSS isochronous signal messages are 20 bytes (Section A.1.2), the IP and UDP headers are 28 bytes and the ethernet frame overhead is 26 bytes (Comer, 1995), yielding a maximum total signal bandwidth of a few kHz, which is sufficient for simple neuronal simulations.

IEEE 1394 provides isochronous and asynchronous services intrinsically and is designed with scalable data transfer rates with current rates of 100, 200 and 400 Mbit/s. With a more compact 12 byte isochronous frame overhead (IEEE, 1995), IEEE 1394 can theoretically support a total DSS signal bandwidth of over 1 MHz.

The DSS protocol specification and application programmer interface (API) is presented in Appendix A.

4.5 Conclusions

The experimental DSS implementation described here suggests the feasibility of using digital networks to interconnect continuous dynamical systems modules (dymods) for distributed real-time simulations. DSS solidifies the dymods' separation of interface from implementation by providing an explicit mechanism to define the interface inputs and outputs via DSS input and output ports. DSS provides a

standard mechanism to interconnect dymods ports with band-limited continuous signals, and also provides a name service to allow a remote connection manager to identify dymod ports by name.

The DSS protocol is targeted at two network architectures: TCP/IP for low-cost, low-bandwidth simulations and IEEE 1394 for high-bandwidth simulations with guaranteed performance. With UNIX and TCP/IP implementations, the total signal bandwidth is theoretically limited to a few kHz on typical 10 Mbit ethernet networks, but in practice this is limited by the accuracy of time synchronization across the network. In order to use signals with bandwidths of greater than about 50 Hz, special UNIX kernel modifications and hardware are required for accurate time keeping and network time synchronization.

The current experimental implementation requires a hard delay along dymod interconnections which limits its utility for solving general systems of ODEs such as the *C. elegans* dymod example (Roehrig and Rankin, 1998). However, an extrapolation algorithm may be used to implement a modified predictor-corrector numerical method for solving general ODE initial value problems without delays. The next step in the development of the DSS dymod implementation framework is to perform the error analysis and implementation for this modified predictor-corrector method.

DSS provides only a basic implementation framework on which to build other dymod tools such as neuronal simulators, and robots for experiments in computational neuroethology using dymods. The next chapter presents an inexpensive robot system to be used for dymod experimentation.

Chapter 5

A Desktop Robot System for Experimental Neuroethology using Dymods

5.1 Introduction

The dymod approach of Chapter 3 is intended for constructing complex neuronal models of behaviours and to ultimately allow neuroscientists to create complex behavioural models using high-level building blocks that are grounded in a physiological implementation, but without having to understand all the physiological details. The DSS network protocol of Chapter 4 provides the means to interconnect dymod implementations using typical UNIX workstations and TCP/IP networks, scales to support arbitrarily large dymod systems, and provides an explicit mechanism to define a dymod's interface.

This chapter adds to the dymod tools by presenting a robot system that the author has constructed in order conduct neuroethology experiments. It continues the theme of the previous chapters with the goal of empowering the general neuroscience researcher, in this case to perform robot neuroethology experiments without requiring expertise in robotic engineering or real-time numerical computation.

5.2 Design Goals

The robot was designed to meet the following goals. To be accessible to general neuroscience researchers, the robot needs to be inexpensive and easy to construct. It should not require any special expertise in electronics or robotics design. It should be easy to use so that neuroscientists can quickly experiment with ideas with a minimal of complication by irrelevant technical issues. The robot should be flexible and adaptable to different neuroethological tasks in order to appeal to a wide neuroscience audience. For instance, a chemotaxis simulation could use simple infrared or optical sensors mounted to the robot and use lights of different colours to represent different chemicals. Finally, to allow easy comparison between different levels of behaviour models, the robot control system should allow for testing of both simple neuronal models and quite complex and realistic models.

5.3 Chassis Design

5.3.1 Locomotion

A wheeled design was chosen because it is simple, robust, and requires few motors and mechanical components. A disadvantage of a traditional wheeled car chassis is that the vehicle must perform a 3-point turn to reorient it self in place. This is a fairly complex task that doesn't have much neuroethological relevance and would require a substantial neuronal control system. To avoid this problem, two approaches were considered.

The first was a tricycle design with a tail-dragger wheel (see Figure 5.1).

Two motors drove the main left and right wheels, and a third free-rotating wheel at the tail provided support and proprioceptive feedback. The two drive motors operated independently to move the vehicle or turn it left or right. The rear wheel was mounted on a swing arm which pivoted as the robot turned. The swing arm was mounted on the shaft of a 360° precision servo potentiometer to provide proprioceptive feedback on the tail's angular orientation. The rotational speed of the rear wheel was to provide proprioceptive feedback as to the vehicle's motion, but it was difficult to construct a sensor on the small tail-wheel swing arm as well as rig the wiring across the freely-rotating shaft it was mounted on.

This design was unnecessarily complex and was abandoned in favour of a tank-like design (Figure 5.2) which suggested itself by the presence of tank treads in a LEGO Dacta kit. The tank chassis is much simpler, requiring only the two drive motors and is able to orient itself in place by driving its two treads in opposite

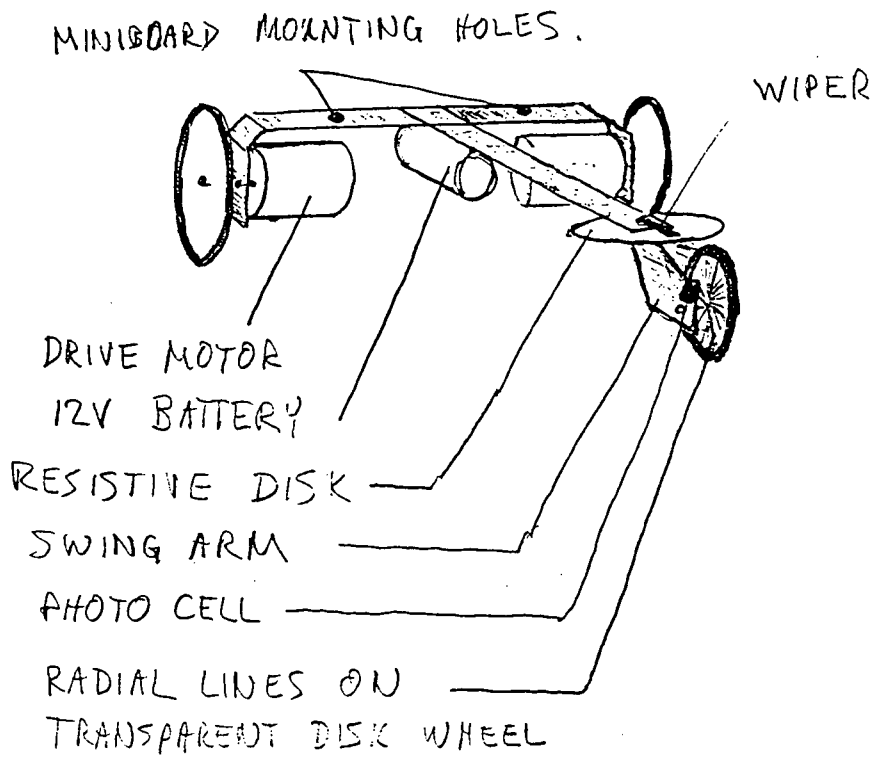


Figure 5.1: Preliminary Tail-dragger Robot Design

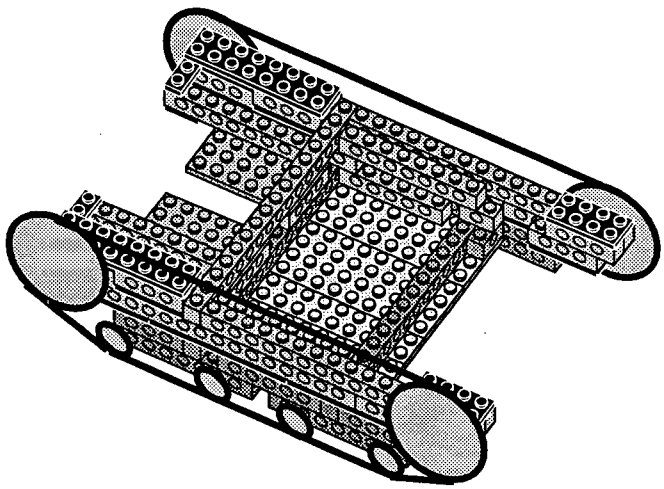


Figure 5.2: LEGO Tank Robot Design

directions.

5.3.2 Proprioception and Sensors

Proprioception is an important part of a control system. Most biological motor control operations require sensory feedback from stretch receptors to operate properly. Robotic limbs also use sensors to provide information on the joint angles and stresses. In a wheeled vehicle, proprioception does not have a direct analogue in the biological world, but is nevertheless important. If the robot is driving a motor in order to locomote, it is important to know whether the motor is actually turning. The robot may have run against an obstacle, or encountered more difficult terrain requiring more motor power. More importantly, if power is being delivered to a motor but the motor is not turning, it draws considerably more power than if it were turning and can uselessly drain valuable battery energy.

There are two possible signals that can be used as proprioceptive feedback from a wheeled vehicle: the angular position of each wheel, and the rotational speed of each wheel. The latter was chosen as being generally more ethologically relevant since the wheel's absolute position is typically unimportant. However, there may be some tasks involving slow or precise positioning where absolute wheel position is more important.

Several ways of encoding wheel speed were explored. A small DC motor from a slot-car racing car was geared to the wheel. As the wheel turned, the DC motor generated a voltage proportional to the rotation speed. However, the motor needed to be geared up from the wheel in order to provide a sufficient voltage and the geared-up inductive resistance of the motor was sufficient to prevent the wheel from rotating as the vehicle moved. A larger, heavier vehicle might be able to overcome this frictional resistance, but it would not be as suitable for a desktop robot.

Optical shaft encoders use a photocell to detect a light beam passed through a wheel containing slits. By measuring the time difference between on and off light pulses, the rotational speed of the wheel can be determined. The existing design does not use an optical shaft encoder, but rather a resistive one (manufactured by ALPS) that was cannibalized from an old Microsoft mouse. However, it has proven to be noisy and somewhat unreliable and is now difficult to obtain. The next revision will use Bourne optical sensors.

Additional sensors can be used depending on the neuroethological experiments to be done.

5.3.3 Construction

The chassis was constructed from LEGO Dacta (Technics) using kits 9605, 8826, as well as some other assorted LEGO parts.

The chassis has a compact gearing mechanism to gear the motors down by a ratio of 25:1 to provide adequate torque to the treads. The design also houses both the microcontroller and a battery compartment below the tank's deck, providing a clean base for additional sensory and actuator mechanisms for various neuroethological experiments. The battery compartment holds 4 C-cell batteries which can deliver a typical maximum load of 500 mA (with 2 motors fully activated) for 14 hours (using alkaline batteries).

The construction diagrams are given in Appendix B.

5.4 Controller

To allow for arbitrarily complex and realistic neuronal control systems, the controller was designed so that the detailed neuronal simulations would be computed by outboard high-speed computers that are connected to the robot with a wire tether, while an inexpensive onboard microcontroller handles the simpler task of managing the sensory data acquisition, motor control, and host communication.

This approach was taken because there are two different tasks that need to accomplished: detailed, timing-sensitive control of sensors and motors, and high-speed numerical computation. These tasks have different computational requirements and there are inexpensive generic solutions to each problem, but a single

solution to both would require special-purpose hardware. The limitation of having to tether the robot to a computer was not seen as a significant one for a desktop system.

While the recent availability of inexpensive high-performance 32-bit microcontrollers makes it feasible to build stand-alone autonomous robots which perform all neuronal computations onboard, a tethered solution has the advantage of being able to use a workstation to monitor, analyze and modify the neuronal operating parameters during behaviour.

5.4.1 Overview

Fred Martin's MIT Miniboard (version 2.1) was chosen for providing on-board sensor and motor control. (Martin, 1995) This is a compact, low-power, inexpensive ($50) board based on the E2 variant of the popular Motorola 68HC11 microcontroller chip, and fits nicely into the chassis of the LEGO robot. The 68HC11E2 has 2 Kbytes of on-chip read-only program memory and 256 bytes of RAM which is sufficient for the control and communication program (but would be wholly inadequate for numerical computation).

A communication protocol was designed to allow communication of sensor and motor signals between the Miniboard and the host computer over a serial cable. The protocol was implemented in the control program of the Miniboard as well as in a portable C library for use on the host computer. This allows any standard C program to interact with the robot, and gives maximum flexibility in implementing computational neuronal models for use with the robot. An add-on module for the

popular GENESIS neuronal simulator was created to allow the use of GENESIS simulation for real-time control of the robot.

The Miniboard provides the robot with the capability for eight analog (graded) sensors, eight digital (on/off) inputs or outputs, four higher-power motor outputs, and two timer inputs for response to time-sensitive events.

5.4.2 BINMON: The Miniboard Control Program

The miniboard control program (called BINMON) is responsible for several tasks.

- It reads and records the robot's analog and digital sensor values every millisecond.
- It maintains an accurate millisecond clock to use as timestamps.
- It implements the host communication protocol to communicate sensor and motor signals.
- It modulates the motor output signals to provide different motor speeds using pulse-width modulation (PWM).
- It decodes the timer input signals to implement a simple shaft-encoder support.

The control program was based on Fred Martin's original HEXMON code, but was substantially rewritten to support the communication protocol and resistive shaft-encoder support. Full details are included in the `mblib` package (see below).

Communication Protocol

The communication protocol provides a mechanism for the host computer to communicate with the Miniboard. This protocol operates in two modes: an "ASCII" (human-readable) mode which can be used for diagnostics and a binary mode for efficient communication of sensor and motor signals.

It is important for the sensor data to be accurately timestamped in order to maximize numerical accuracy when simulating the neuronal controller model. Rather than try to synchronize the Miniboard's clock with the host computer's clock, a simpler approach was taken. The Miniboard operates in a passive mode, recording sensor data every millisecond, but not initiating any communication. When it receives a request from the host computer, it responds by transmitting the sensor data. The host computer can timestamp the sensor data when it receives it, and can correct for the transmission delay to accurately determine the sensor acquisition time.

This mechanism fits well with existing ASCII command-response mode of the original HEXMON code. The original ASCII commands were preserved mostly unchanged for debugging purposes. The ASCII commands return a human-readable response which is terminated by a command-prompt '>'. The ASCII commands are summarized as follows:

s Perform a reset and resynchronize to the host.

r*mmmm* Read byte at location *mmmm*.

w*mmmmdd* Write byte *dd* at location *mmmm*.

q*mmmm* Read word at location *mmmm*.

z*mmmmdddd*

Write word *dddd* at location *mmmm*.

v Print monitor version.

d ASCII dump of state.

The binary communication protocol is initiated by the host by a `b` command. This causes the Miniboard to receive a frame of control commands and once it has been received, to return a frame of sensory data. Full-duplex operation (i.e. transmitting the sensory data simultaneously with receiving the control commands) is not possible due to the specifics of the Miniboard's serial communication hardware implementation (see (Martin, 1995)). The control command frame consists of 11 bytes: a motor control byte (4 bits on/off, 4 bits direction), 4 motor speed words (each consisting of a 16-bit PWM mask), a data direction control byte for the digital input/output port, and a data byte for that port. The sensory data frame consists of 11 bytes: one byte for each of the eight analog inputs, a data byte for the digital input/output port, and a byte for each of the two shaft encoder inputs. The transaction is terminated by the command prompt character.

Timing Discussion

Since computer operations occur many orders of magnitude faster than the communication time over a serial cable, it is necessary to correct for serial transmission time to obtain an accurate timestamp for the sensory data. The binary transmission time can be computed by the number of bits of information transmitted

divided by the bit rate. At the current bit rate of 9600 baud, command transmission (12 bytes of 8 bits with 1 stop bit) takes 11.25 milliseconds. Because of peculiarities of the Miniboard's serial communication hardware implementation, all transmitted bytes are echoed back to the host. In addition to these bytes, the return frame consists of 11 bytes for a total of 23 bytes that must be received by the host (there is no reason to wait for the command prompt). This reception takes 21.56 milliseconds (but because of the hardware echo, this happens concurrently with transmission of control information).

Therefore, the Miniboard will respond with motor commands with a 11 ms delay from the issuing of the motor commands, and the sensory data from the Miniboard will only be able to be used by the numerical simulation 22 ms after it is actually acquired. (Because the Miniboard only updates at a rate of 1 kHz, these delays are only accurate to within 1 ms.)

DSS can be used for the numerical simulation by embedding the Miniboard sensor values into DSS packets. If this is the case, the DSS timestamps can be corrected for this delay and the Miniboard sensory signals can be processed accurately. However, as in the case of DSS network transmission latencies, this delay imposes a bandwidth restriction on the signals which can only be overcome by faster transmission time. (A 22 ms delay means a maximum signal bandwidth of about 20 Hz.) See below for a brief discussion on how to increase the Miniboard's serial transmission speed.

Miniboard Host C Library

A library of routines was implemented in the C programming language to allow a host computer to interact with the Miniboard using the communication protocol. The C library maintains a copy of the miniboard's state on the host computer and periodically synchronizes this state with the Miniboard.

The library application programmer interface (API) is given in Appendix B.

The `mblib` Package

The `mblib` package contains implementations of the BINMON Miniboard control program and Miniboard C Library, together with the following additional components:

`mbview` An interactive miniboard viewer to allow viewing and manipulation of the Miniboard. It has a simple keyboard interface and a text-based display to allow it display on any VT100 compatible terminal. It also displays various timing measurements to analyze serial port latencies.

`dlm11` A UNIX port of the Miniboard program downloader.

GENESIS Miniboard Library

This is a library module for the popular GENESIS neuronal simulator to allow it to synchronize the simulation to real-time and interact with live signals from the Miniboard.

The `mblib` package can be obtained from http://www.crispart.com/mblib.

5.5 Future Work

There are many avenues for continued work on this robot project. At the forefront, is the implementation of a Miniboard DSS interface to encapsulate the Miniboard signals into DSS packets. This will allow the Miniboard to interoperate with other DSS modules. The implementation would be in the form of a daemon process that runs a continuously updating Miniboard on a serial port and maintains a collection of DSS ports for each Miniboard signal.

In addition, a GENESIS DSS module needs to be implemented which would allow the GENESIS simulator to interoperate with any DSS signal.

The BINMON Miniboard Control Program needs to be modified to support optical shaft encoders. The Bourne Model ENC1J-D28-L00128 shaft encoder is a compact low-friction shaft encoder that can be used with appropriate modifications to the BINMON program. Modifications to the LEGO robot to accommodate this encoder also need to be done. In addition, the binary communication protocol can be made more compact. Currently, the motor PWM masks are 16-bits which means 8 bytes are transmitted every frame. However, these PWM masks only encode 16 different speeds according to a table kept on the host. This table could be moved to the Miniboard and the 4-bit motor speeds could be transmitted instead for a savings of 6 bytes (6 ms at 9600 baud).

To substantially improve the signal bandwidth between the Miniboard and the host, the serial speed needs to be increased. The maximum standard speed of the Miniboard is 9600 baud using the standard 8 MHz clock, but this clock could be replaced by a 4.9152 MHz crystal to obtain higher standard baud rates (such as

38.4 Kbaud) at the expense of reducing the Miniboard CPU's bus frequency from 2 MHz to 1.2 MHz. However, the Miniboard is not performing any computationally intensive work, and this loss of CPU speed will likely not be a problem. At 38.4 Kbaud with the savings from the PWM modifications, the host-Miniboard update time would be reduced from its current 22 ms to 4 ms, with signal bandwidth of 125 Hz.

5.6 Conclusions

This chapter presents a desktop robot system the author has constructed for conducting neuroethology experiments using dymods. The inexpensive LEGO robot uses a wheeled-design for simplicity and reliability. It uses a tank-like chassis with treads, which gives it the ethologically relevant ability to orient in place, unlike other car-like designs which require three-point turns. The chassis is compact and houses the battery compartment, motors and computer control system below the tank's "deck" to provide maximum flexibility for adding sensory and actuator apparatus. The control system is a tethered design: an MIT Miniboard monitors and controls the robots sensors and motors and transmits them along a cable to a host UNIX computer which performs the actual neuronal computation. This chapter also presents a Miniboard program (BINMON) that performs the control functions, a UNIX library to communicate with the robot via serial cable, and a library add-on for the popular GENESIS neuronal simulator to allow it to communicate with the robot.

Currently, the LEGO robot is fully mobile and contains proprioceptive sen-

sors for tread motion, and the microcontroller's other 6 sensory inputs and 2 motor outputs are not utilized and available for expansion. The support software has been developed to the point where a simple single-compartment neuron simulation running under GENESIS was able to successfully control locomotion in the robot.

Chapter 6

Conclusions

This dissertation introduced the dymod concept to help account for a previously unexplained behaviour in the nematode *C. elegans*: its ability to continue swimming backwards for a period of time after the end of a tap stimulus. The dymod approach provides building-blocks for constructing a dynamic behaviour out of a set of simpler dynamical structures.

Current brain theories (Kelso, 1995) speculate that understanding dynamical structures is the key to understanding how the brain generates abstract behaviours like language, planning and perception. However, so far we have no building blocks to quantitatively describe dynamical structures except at the most detailed level of the physiological components: cells, channels, etc. It is unlikely that we will be able to understand as complex a behaviour as perception in terms of cells and channels without some form of higher-level abstract building blocks. The dymod approach is a step towards those building blocks.

The purpose of the dymod framework is to describe a dynamical system in

terms of a set of simpler dynamical modules. This dissertation illustrates the dymod concept with a simple example and lays out a research programme to investigate the generality and usefulness of the dymod approach. The research programme is based on the generative computational neuroethology approach pioneered by Beer, Cliff and others (Beer, 1997; Harvey, Husbands, Cliff, Thompson and Jakobi, 1997) using robots to ground neuronal models in a complete behavioural system that includes its environment as part of the system. The dissertation also presents a preliminary set of tools for embarking upon this research programme: the DSS dymod implementation framework, and a general-purpose desktop robot platform.

6.1 Future Directions

The next step in the research programme that was laid out in Chapter 3 is to combine the locomotory circuit of Chapter 3, the implementation framework of Chapter 4 with the robot of Chapter 5 (see Figure 6.1)

To complete the next step, the following projects need to be completed.

- The DSS protocol must implement the modified predictor-corrector numerical method (Section 4.4.2) to allow signal reconstruction without hard delays. The tonic and graded synaptic model for *C. elegans* (Section 2.6 and Section 3.1.1) does not use hard delays — indeed many neuronal dynamical systems do not use hard delays — and the current DSS implementation cannot be used for these models which substantially limits its utility. The implementation of the modified predictor-corrector numerical method would require a fair amount of detailed numerical analysis to determine convergence

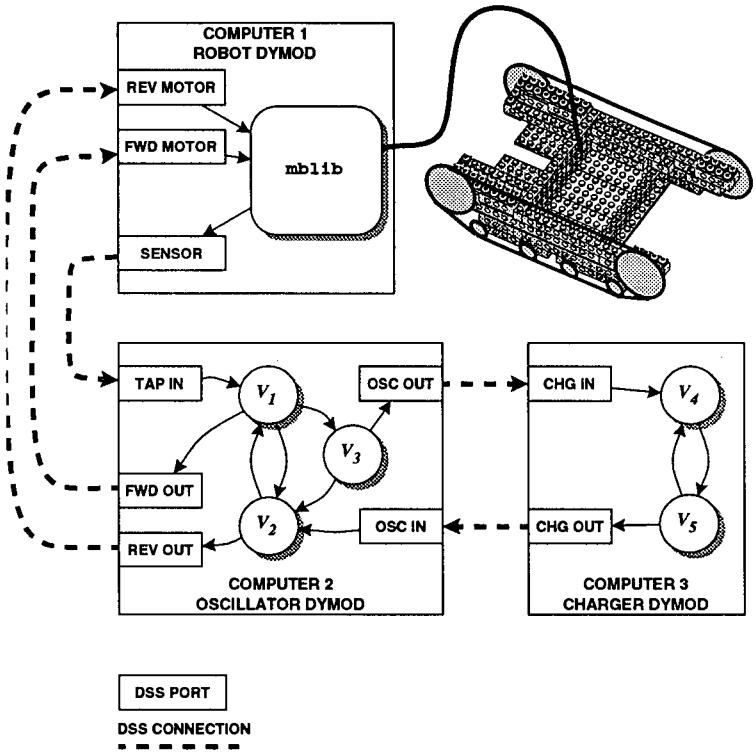


Figure 6.1: The next step in the research programme is to combine the locomotory circuit with the DSS implementation framework and the robot. A robot with a tap sensor is to be tethered to a computer running a DSS-enabled `mblib` (Section 5.4.2) to implement a dymod representing the robot's sensors, motors and its environment. It will be connected to a second computer implementing the oscillator dymod which would in turn be connected to a third computer implementing the charger dymod. The dymods can be implemented using a DSS-enabled version of the GENESIS neuronal simulator (Section 5.4.2).

properties and error bounds.

- DSS-enabled implementations of `mblib` and GENESIS need to be completed. Both are relatively straight-forward modifications of the existing implementation presented in Section 5.4.2.

6.2 Novel Contributions

The novel contributions of this dissertation are summarized as follows:

1. A physiologically detailed cellular model of the nematode tap withdrawal circuit (Chapter 2).

2. A graded and tonic synaptic model and a treatment of the distinction between a cell's resting potential and its in-circuit steady-state potential (Section 2.7).

3. A cellular account of how the nematode might continue its reversing after the end of a stimulus using neuronal network dynamics (Chapter 3).

4. Dymods: a way of decomposing dynamical neuronal structures into modular subcomponents (Chapter 3).

5. DSS: an implementation framework for dymods that is designed to scale to handle large, complex systems of dymods (Chapter 4).

6. A hybrid numerical method for solving ODE initial value problems in real-time using digital signal reconstruction filters as part of the numerical method (Section 4.4.2).

7. An inexpensive, general-purpose desktop robot system intended to allow neuroscience researchers to experiment with the dynamical basis of behaviour without requiring expertise in robotics or numerical solutions to systems of ODEs (Chapter 5).

8. A research programme for investigating the modular properties of dynamical neuronal structures and how they interact, together with a preliminary set of tools for embarking upon that programme.

Bibliography

1394TA (1998). IEEE 1394 trade association. URL: ⟨http://www.1394ta.org⟩ [accessed Apr 7, 1998].

Abbott, L., Marder, E. and Hooper, S. (1991). Oscillating networks: Control of burst duration by electrically coupled neurons, *Neural Computation* **3**(487–497).

Achacoso, T. B. and Yamamoto, W. S. (1992). *AY's Neuroanatomy of* C. elegans *for Computation*, CRC Press, Boca Raton, FL.

Avery, L., Raizen, D. and Lockery, S. (1995). Electrophysiological methods, *in* H. Epstein and D. Shakes (eds), Caenorhabditis elegans*: Modern Biological Analysis of an Organism*, Academic Press, San Diego, pp. 251–269.

Beer, R. D. (1995). On the dynamics of small continuous-time recurrent neural networks, *Journal of Adaptive Behavior* **3**(4): 469–509.

Beer, R. D. (1997). The dynamics of adaptive behavior: A research program, *Robotics and Autonomous Systems* **20**: 257–289.

Beer, R. D. and Chiel, H. J. (1993). Simulations of cockroach locomotion and escape, *in* R. D. Beer, R. E. Ritzmann and T. McKenna (eds), *Biological Neural Networks in Invertebrate Neuroethology and Robotics*, Academic Press, pp. 267–285.

Beer, R. D. and Gallagher, J. C. (1992). Evolving dynamical neural networks for adaptive behavior, *Adaptive Behavior* **1**: 91–122.

Bennett, M. V. L. (1972). A comparison of electrically and chemically mediated transmission, *in* G. D. Pappas and D. P. Purpura (eds), *Structure and Function of Synapses*, Raven Press, New York, pp. 221–256.

Bialek, W., Rieke, F., de Ruyter van Steveninck, R. and Warland, D. (1991). Reading a neural code, *Science* **252**: 1854–1857.

Bower, J. M. and Beeman, D. (1995). *The Book of Genesis*, Springer-Verlag.

Brooks, R. A. (1986a). Achieving artificial intelligence through building robots, *A.I. Memo 899*, M.I.T.

Brooks, R. A. (1986b). A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation* **2**(1): 14–23.

Campbell, S. and Wang, D. L. (1998). Relaxation oscillators with time delay coupling, *Physica D* **111**: 151–178.

CAN (1998). KVASER's controller area network (CAN) introduction. URL: ⟨http://www.kvaser.se/can/initial.htm⟩[accessed Apr 7, 1998].

Chalfie, M., Sulston, J. E., White, J. G., Southgate, E., Thomson, J. N. and Brenner, S. (1985). The neural circuit for touch sensitivity in *Caenorhabditis elegans*, *Journal of Neuroscience* **5**(4): 956–963.

Chalfie, M., Tu, Y., Euskirchen, G., Ward, W. and Prasher, D. (1994). Green fluorescent protein as a marker for gene expression, *Science* **263**: 802–805.

Chiel, H. J. and Beer, R. D. (1997). The brain has a body: Adptive behavior emerges from interactions of nervous system, body and environment., *Trends in Neuroscience* **20**: 553–557.

Cliff, D. (1991). Computational neuroethology: A provisional manifesto, *in* J. A. Meyer and S. W. Wilson (eds), *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behaviour*, MIT Press, pp. 29–39.

Comer, D. E. (1995). *Internetworking with TCP/IP, Volume I, Third Ed.: Principles, Protocols, and Architecture*, Prentice-Hall.

Comer, D. E. and Stevens, D. L. (1993). *Internetworking with TCP/IP, Volume III, BSD Socket Version: Client-Server Programming and Applications*, Prentice-Hall.

Corey, D. P. and Garcia-Anoveros, J. (1996). Mechanosensation and the DEG/ENaC ion channels, *Science* **273**: 323–324.

Crochiere, R. and Rabiner, L. (1983). *Multirate Digital Signal Processing*, Prentice-Hall.

Davis, R. E. and Stretton, A. O. W. (1989a). Passive membrane properties of motorneurons and their role in long-distance signaling in the nematode *Ascaris*, *Journal of Neuroscience* **9**(2): 403–414.

Davis, R. E. and Stretton, A. O. W. (1989b). Signaling properties of *Ascaris* motorneurons: Graded active responses, graded synaptic transmission, and tonic transmitter release, *Journal of Neuroscience* **9**(2): 415–425.

De Schutter, E., Angstadt, J. and Calabrese, R. (1993). A model of graded synaptic transmission for use in dynamic network simulations, *Journal of Neurophysiology* **69**: 1225–1235.

Ermentrout, B. (1998). X-windows phaseplane plus auto. URL:http://mrb.niddk.nih.gov/xpp.

Ermentrout, G. and Kopell, N. (1990). Oscillator death in systems of coupled neural oscillators, *SIAM J. Appl. Math.* **50**: 125–146.

Funahashi, K. and Nakamura, Y. (1993). Approximation of dynamical systems by continuous time recurrent neural networks, *Neural Networks* **6**: 801–806.

Gear, C. W. (1971). *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall.

Getting, P. (1989). A network oscillator underlying swimming in *Tritonia*, *in* J. Jacklet (ed.), *Neuronal and Cellular Oscillators*, Marcel Dekker Inc, New York, pp. 215–236.

Gonzalez, J. and Tsien, R. (1995). Voltage sensing by fluorescence resonance energy transfer in single cells, *Biophys. J.* **69**: 1272–1280.

Goodman, M. B., Hall, D. H., Avery, L. and Lockery, S. R. (1998). Active currents regulate sensitivity and dynamic range in *C. elegans* neurons, *Neuron* **20**: 763–772.

Hall, D. H. and Russell, R. L. (1991). The posterior nervous system of the nematode *Caenorhabditis elegans*: serial reconstruction of identified neurons and complete pattern of synaptic interactions, *Journal of Neuroscience* **11**(1): 1–22.

Hammerstrom, D. (1995). Digital VLSI for neural networks, *in* M. A. Arbib (ed.), *Handbook of Brain Theory and Neural Networks*, MIT Press, pp. 304–309.

Harris-Warrick, R., Nagy, F. and Nusbaum, M. (1992). Neuromodulation of stomatogastric networks by identified neurons and transmitters, *in* R. Harris-Warrick, E. Marder, A. Selverston and M. Moulins (eds), *Dynamic Biological Networks: The Stomatogastric Nervous System*, MIT Press, pp. 87–137.

Harvey, I., Husbands, P., Cliff, D., Thompson, A. and Jakobi, N. (1997). Evolutionary robotics: The Sussex approach, *Robotics and Autonomous Systems* **20**: 205–224.

Hines, M. (1993). The NEURON simulation program., *in* J. Skrzypek (ed.), *Neural Network Simulation Environments*, Kluwer Academic Publishers, Norwell, MA.

Hopfield, J. (1995). Pattern recognition comutation using action potential timing for stimulus representation, *Nature* **376**: 33–36.

Hoppensteadt, F. and Izhikevich, E. (1997). *Weakly Connected Neural Networks*, Springer, New York.

Horowitz, P. and Hill, W. (1989). *The Art Of Electronics*, second edn, Cambridge University Press.

IEEE (1993). IEEE standard for information technology: Protocols for distributed interactive simulation applications. IEEE Std. 1278-1993.

IEEE (1995). IEEE standard for a high performance serial bus. IEEE Std.1394-1995.

IEEE (1996). The POSIX system application program interface. IEEE Std. 1003.1-1988.

Jung, R., Kiemel, T. and Cohen, A. (1996). Dynamic behavior of a neural network model of locomotor control in the lamprey, *Journal of Neurophysiology* **75**: 1074–1086.

Kelso, J. A. S. (1995). *Dynamic Patterns. The Self-Organization of Brain and Behavior*, MIT Press.

Knuth, D. (1997). *The Art Of Computer Programming, Vol 1: Fundamental Algorithm*, third edn, Addison-Wesley.

Koch, C. and Segev, I. (1989). *Methods in Neuronal Modeling: From Synapses to Networks*, MIT Press, Cambridge, MA.

Lockery, S. R., Nowlan, S. J. and Sejnowski, T. J. (1992). Modelling chemotaxis in the nematode *C. elegans*, *in* J. Bower and F. Eechman (eds), *Computation and Neural Systems*, Kluwer Academic Publishers, Norwell, MA.

Lockery, S. R. and Sejnowski, T. J. (1992). Distributed processing of sensory information in the leech. III. a dynamical neural network model of the local bending reflex, *Journal of Neuroscience* **12**(10): 3877–3895.

Lygeros, J. (1996). *Hierarchical Hybrid Control of Large Scale Systems*, PhD thesis, Department of Electrical Engineering, University of California, Berkeley.

Mahowald, M. A. (1992). Evolving analog VLSI neurons, *in* T. McKenna, J. Davis and S. F. Zornetzer (eds), *Single Neuron Computation*, Academic Press.

Marder, E. and Selverston, A. I. (1992). Modeling the stomatogastric nervous system, *in* R. M. Harris-Warrick, E. Marder, A. I. Selverston and M. Moulins (eds), *Dynamic Biological Networks: The Stomatogastric Nervous System*, MIT Press, pp. 161–196.

Martin, F. (1995). Mini board 2.0 technical reference. URL: ⟨http://fredm.www.media.mit.edu/people/fredm/papers/mb⟩[accessed Apr 27, 1998].

Mills, D. (1990). On the accuracy and stability of clocks synchronized by the Network Time Protocol in the Internet system, *ACM Computer Communication Review* **20**(1): 65–75.

Mills, D. (1994). Precision synchronization of computer network clocks, *ACM Computer Communication Review* **24**(2): 28–43.

Mills, D. L. (1992). Network Time Protocol (version 3) specification, implementation and analysis. IETF Network Working Group Report RFC-1305.

Morse, T., Ferrée, T. and Lockery, S. (1998). Robust spatial navigation in a robot inspired by chemotaxis in *Caenorhabditis elegans*, *Adaptive Behavior* **6**: 393–410.

Niebur, E. (1988). *Théorie du système locomoteur et neuronal des Nématodes*, PhD thesis, University of Lausanne.

Niebur, E. and Erdös, P. (1991). Theory of the locomotion of nematodes: Dynamics of undulatory progression on a surface, *Biophysical Journal* **60**: 1132–1146.

Niebur, E. and Erdös, P. (1993). Theory of the locomotion of nematodes: Control of the somatic motor neurons by interneurons, *Mathematical Biosciences* **118**: 51–82.

Northmore, W. W. D. P. M. and Elias, J. G. (1997). Neuromorphic synapses for artificial dendrites, *Analog Integrated Circuits and Signal Processing* **13**: 167–184.

Nyquist, H. (1928). Certain topics in telegraph transmission theory, *AIEE Trans.* pp. 617–644.

Oppenheim, A. V. and Schafer, R. W. (1989). *Discrete-Time Signal Processing*, Prentice Hall.

Osborne, N. (1983). *Dale's Principle and Communications Between Neurones*, Pergamon Press, New York.

Peak Audio Inc. (1998). CobraNet. URL: ⟨http://www.peakaudio.com/public/CobraNet/index.html⟩[accessed Apr 7, 1998].

Pearlmutter, B. (1989). Learning state space trajectories in recurrent neural networks, *Neural Computation* **1**: 263–269.

Pittsburgh Supercomputing Center (1998a). NEURON. URL: ⟨http://www.psc.edu/general/software/packages/neuron/neuron.html⟩ [accessed Apr 7, 1998].

Pittsburgh Supercomputing Center (1998b). Parallel GENESIS at PSC. URL:⟨http://www.psc.edu/general/software/packages/pgenesis/project_docs%⟩ [accessed Apr 7, 1998].

Port, R. F. and van Gelder, T. (eds) (1995). *Mind as Motion: Explorations in the Dynamics of Cognition*, MIT Press.

Press, W., Flannery, B., Teukolsky, S. and Vetterling, W. (1988). *Numerical Recipes in C*, Cambridge University Press, Cambridge.

Profibus (1998). PROFIBUS. URL: ⟨http://www.profibus.com⟩[accessed Apr 7, 1998].

Raizen, D. and Avery, L. (1994). Electrical activity and behavior in the pharynx of *Caenorhabditis elegans*, *Neuron* **12**: 483–495.

Rall, W. (1977). Core conductor theory and cable properties of neurons, *in* E. R. Kandel (ed.), *Handbook of Physiology: The Nervous System, Vol. 1*, American Physiological Society, Bethesda, MD.

Rall, W. (1989). Cable theory for dendritic neurons, *in* C. Koch and I. Segev (eds), *Methods in Neuronal Modelling: From Synapses to Networks*, MIT Press, pp. 9–62.

Rankin, C. H. (1991). Interactions between two antagonistic reflexes in the nematode *Caenorhabditis elegans*, *J. Comp. Physiol. A* **169**: 59–67.

Rankin, C. H., Beck, C. D. O. and Chiba, C. M. (1990). *Caenorhabditis elegans*: a new model system for the study of learning and memory, *Behavioural Brain Research* **37**: 89–92.

Real Networks, Inc (1998). Realtime streaming protocol resource center. URL: ⟨http://www.real.com/rtsp⟩[accessed Apr 7, 1998].

Real-Time Magazine (1998). Real-time encyclopaedia. URL: ⟨http://www.realtime-info.be⟩[accessed Apr 7, 1998].

Roehrig, C. J. and Rankin, C. H. (1998). Dymods I: A framework for modularizing dynamical neuronal structures. (submitted).

Schinkmann, K. and Li, C. (1992). Localization of FMRFamide-like peptides in *Caenorhabditis elegans*, *J. Comp. Neurol.* **316**(251–260).

Schug, K. H. (1995). DIS NG – a flexible protocol for all simulation applications, *Proc. 13th DIS Workshop on Standards for the Interoperability of Distributed Simulations*, Orlando, Florida, Sep. 18-22.

Segev, I., Fleshman, J. W. and Burke, R. E. (1989). Compartmental models of complex neurons, *in* C. Koch. and I. Segev (eds), *Methods in Neuronal Modelling: From Synapses to Networks*, MIT Press, pp. 63–96.

Skinner, F., Kopell, N. and Marder, E. (1994). Mechanisms for oscillation and frequency control in reciprocally inhibitory model neural networks, *Journal of Computational Neuroscience* **1**: 69–87.

Smith, J. O. and Gossett, P. (1984). A flexible sampling-rate conversion method, *Proceedings of the IEEE Conference on Acoustics, Speech and Signal Processing, San Diego*, Vol. 2, pp. 19.4.1–19.4.4.

Stretton, A. O. W., Donmoyer, J. E., Davis, R. E., Meade, J., Cowden, C. and Sithigorngul, P. (1992). Motor behavior and motor nervous system function in the nematode *Ascaris suum*, *J. Parasitol* **78**: 206–214.

Strogatz, S. H. (1994). *Nonlinear Dynamics and Chaos with Applications to Physics, Biology, Chemistry, and Engineering*, Addison-Wesley.

Tavernakis, N., Shreffler, W., Wang, S. and Driscoll, M. (1997). Unc-8, a DEG/ENaC family member, encodes a subunit of a candidate mechanically gated channel that modulates *C. elegans* locomotion, *Neuron* **18**: 107–119.

Thomas, J. H. (1990). Genetic analysis of defecation in *Caenorhabditis elegans*, *Genetics* **124**: 855–872.

White, J. G., Southgate, E., Thomson, J. N. and Brenner, S. (1976). The structure of the ventral cord of *Caenorhabditis elegans*, *Philos. Trans. R. Soc. Lond. (Biol.)* **275**: 327–348.

White, J. G., Southgate, E., Thomson, J. N. and Brenner, S. (1986). The structure of the nervous system of the nematode *Caenorhabditis elegans*, *Philos. Trans. R. Soc. Lond. (Biol.)* **314**: 1–340.

Wicks, S. R. and Rankin, C. H. (1995). Integration of mechanosensory stimuli in *Caenorhabditis elegans*, *Journal of Neuroscience* **15**(3): 2434–2444.

Wicks, S. R. and Rankin, C. H. (1997). The effects of tap withdrawal response habituation on other withdrawal behaviors: The localization of habituation, *Behavioral Neuroscience* **111**: 342–353.

Wicks, S. R., Roehrig, C. J. and Rankin, C. H. (1996). A dynamic network simulation of the nematode tap withdrawal circuit: Predictions concerning synaptic function using behavioral criteria, *Journal of Neuroscience* **16**(2): 4017–4031.

Wood, W. B. (1988). *The Nematode* Caenorhabditis elegans, Cold Spring Harbor Laboratory, Cold Spring Harbor, NY.

Yamaha (1998). mLAN: Audio and music data transmission protocol. URL: ⟨http://www.yamaha.co.jp/tech/1394mLAN/⟩[accessed Apr 7, 1998].

Yamauchi, M. M. and Beer, R. D. (1994). Sequential behavior and learning in evolved dynamical neural networks, *Adaptive Behavior* **2**: 219–246.

Yang, H. (1995). Intrinsic oscillations of neural networks, *Proc. ICNN'95*, Vol. 6, pp. 3044–3047.

Yang, H. and Dillon, T. (1994). Exponential stability and oscillation of hopfield graded response neural network, *IEEE Trans. on Neural Networks* **5**: 719–729.

Young Chang RDI (1998). Presto digital audio interface. URL: ⟨http://www.ycrdi.com/presto⟩[accessed Apr 7, 1998].

Appendix A

The DSS Protocol Specification and API

A.1 The DSS Protocol Specification

The DSS protocol implements layers 5 to 7 of the ISO 7-layer reference model (Comer, 1995) and relies on an underlying network transport protocol such as TCP/IP or IEEE 1394.

The protocol has three main components (Figure A.1). The DSS Signal component deals with messages that carry the signal data. This component is responsible for presenting a continuous signal abstraction to the receiver application, and performs the necessary signal reconstruction and resampling as required by the receiver. It uses an unacknowledged isochronous datagram transport service. The DSS Control component is responsible for establishing, managing and breaking DSS connections. It uses an acknowledged datagram transport service. The

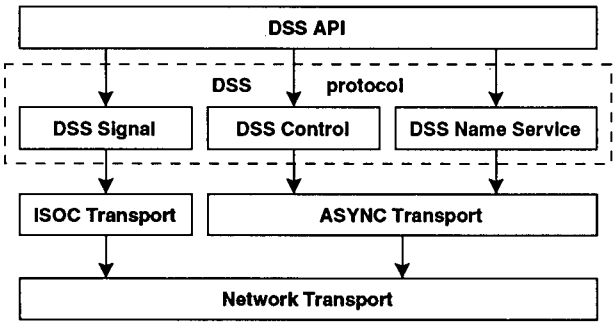


Figure A.1: DSS Architecture

DSS Name Service provides a name registry to allow DSS signal connections to be specified by a symbolic name rather than numerical addresses.

Much of the nomenclature used here has been adopted from the TCP/IP protocol (Comer, 1995). An octet refers to an 8-bit quantity (i.e. a byte on most computers). All numerical quantities appearing in DSS messages are in network byte ordering (i.e. MSB first), and are aligned on type boundaries.

A.1.1 DSS Port Addresses

DSS port addresses have two components: a 32-bit local address to uniquely identify the port on a single host computer or device, and an 8-octet transport address to uniquely identify the host. The transport address consists of a single octet to identify the transport family, plus a 7-octet transport-dependent address (see Figure A.2). The transport family is 1 for TCP/IP using UDP, and 2 for IEEE 1394.

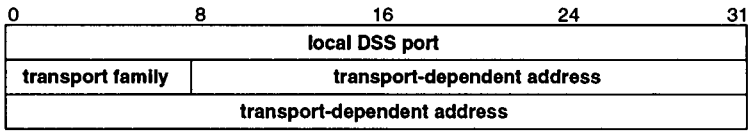


Figure A.2: DSS Address

In the TCP/IP UDP implementation of DSS version 1, the 7-octet transport address octets are in order: a zero-pad octet, a 2-octet UDP port number in network byte order, and a 4-octet IP address in network byte order.

A.1.2 DSS Messages

DSS messages are an integral number of quadlets (4-octets) and consist of a DSS header followed by the message contents. DSS messages can be one of two types: isochronous (ISOC) or asynchronous (ASYNC). ISOC messages are unacknowledged and used for carrying the time-critical signal data. ASYNC messages are used for control and name service functions and are acknowledged using a sequence number to match acknowledgement responses to requests. To allow for efficient embedded implementations and mappings onto transport layers such as IEEE-1394, all DSS messages are guaranteed to have a size of 200 octets or less. The two message types were designed to be efficiently mapped onto the asynchronous and isochronous services provided by the underlying transport layer. In the case of IEEE-1394, these would be the similarly named services. DSS version 1 was implemented on TCP/IP using UDP datagrams to provide both services.

The DSS Header

The DSS message header is shown in Figure A.3.

The four high-order bits of the header is used to identify the version of the DSS protocol, encoded in reverse bit order. The current version is 1 and it is encoded as `0x8`. Bits 12–15 of the first header quadlet contain message flags as shown

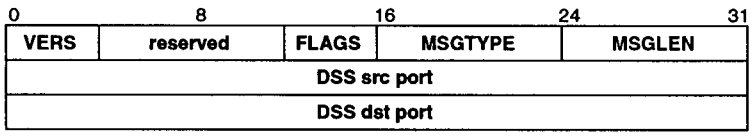


Figure A.3: DSS Message Header

in Table A.1:

flag	value	description
DSS_ACKREQ	0x1	requires an a acknowledgement
DSS_ACK	0x2	this is an acknowledgement
DSS_TIMEOUT	0x4	request timed out
DSS_FAIL	0x8	request failed

Table A.1: DSS Message Header Flags

The MSGTYPE octet encodes the DSS message type as described below. The MSGLEN octet encodes the size of the DSS message contents in octets, not including the DSS header. The DSS source port and destination port identify the endpoints of the DSS connection. To form a complete DSS address, they must be combined with the transport-dependent address which must be obtained from the transport layer. Since DSS messages are encapsulated in transport-layer datagrams, these datagrams must include the source transport address if the full DSS source address is to be recoverable. This is the case for TCP/IP (Comer, 1995) and IEEE-1394 (IEEE, 1995).

ISOC Message (MSGTYPE=1)

An isochronous message (Figure A.4) carries signal data. The sample timestamp is the number of microseconds since the connection epoch. In DSS version 1, the signal data is a single sample encoded as an integer scaled up by a factor of

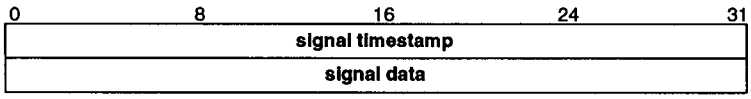


Figure A.4: DSS Isochronous Message

10^6. This allows a representation of signal values in the range -2147 to 2147 with a precision of 10^{-6}. Future DSS versions will specify the signal and timestamp format at connection time and may encode multiple samples in a single message.

ASYNC Message (MSGTYPE=2)

A generic asynchronous message is shown in Figure A.5. This message is

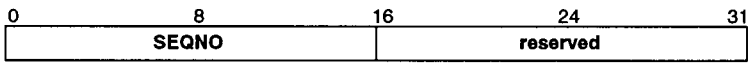


Figure A.5: DSS Asynchronous Message

used for simple acknowledgments and to "ping" ports to see if they are alive. For acknowledgements, the `DSS_ACK` bit is set in the header and SEQNO identifies the original message that requested the acknowledgment. For pings, the `DSS_ACKREQ` bit is set in the header flags, and the destination port returns an ASYNC acknowledgement message. A 16-bit data field can be used to return error codes or other data.

CONNECT Message (MSGTYPE=3)

A connection request message is shown in Figure A.6. This message is sent by an output port to a target input port to request its connection. The DSS source address is the originating output port of the signal. FORMAT indicates the

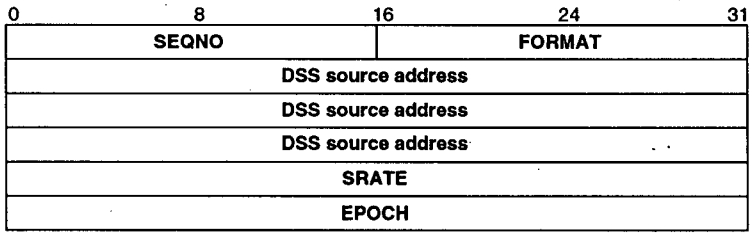


Figure A.6: DSS Connection Request Message

data format for the signal and is currently unused in DSS version 1. In future versions, this will indicate the encoding format and scaling factor for the signal data. SRATE is the nominal sampling rate (i.e. twice the bandwidth) of the signal in thousandths of Hertz. SRATE is used to determine the filter cutoff frequency of the reconstruction filter. EPOCH is the reference time for the connected signal's timestamps in seconds since midnight, January 1, 1970, GMT. This value is added to the signal's timestamps to give their actual absolute time reference. An ASYNC acknowledgement packet is returned to the sender upon completion.

DISCONNECT Message (MSGTYPE=4)

The DISCONNECT message (Figure A.7) is sent to a target DSS input port by an output port to request its disconnection. The DSS source address is the orig-

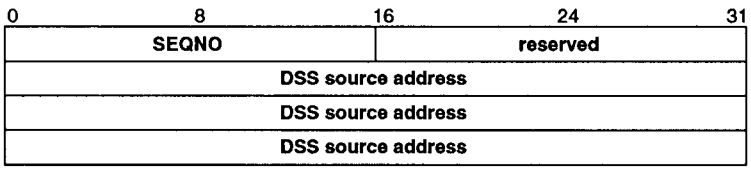


Figure A.7: DSS Disconnection Request Message

inating output port of the signal. An ASYNC acknowledgement packet is returned to the sender upon completion.

ADD TARGET Message (MSGTYPE=5)

The ADD TARGET message (Figure A.8) is sent to a DSS output port to request its connection to a target input port. The DSS destination address is the

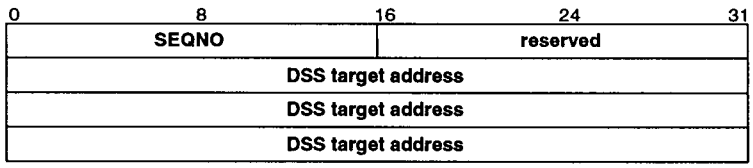


Figure A.8: DSS Add Target Message

address of the target input port to be connected. An ASYNC acknowledgement packet is returned to the sender upon completion. If the output port was unable to contact the target port, the `DSS_TIMEOUT` bit is set in the acknowledgement packet.

DEL TARGET Message (MSGTYPE=6)

The DEL TARGET message (Figure A.9) is sent to a DSS output port to request its disconnection from a target input port. The DSS destination address is the

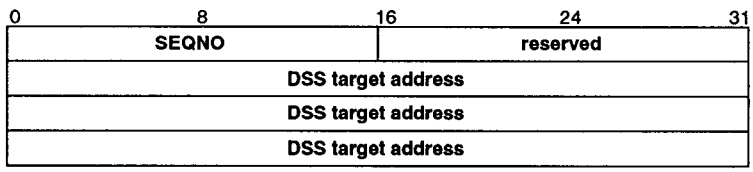


Figure A.9: DSS Delete Target Message

address of the target input port to be disconnected. An ASYNC acknowledgement packet is returned to the sender upon completion. If the output port was unable

to contact the target port, the DSS_TIMEOUT bit is set in the acknowledgement packet.

NAME REGISTER Message (MSGTYPE=7)

The NAME REGISTER message (Figure A.10) is sent to a DSS name server to register the name with a DSS address. NAMELEN is the length in octets of the

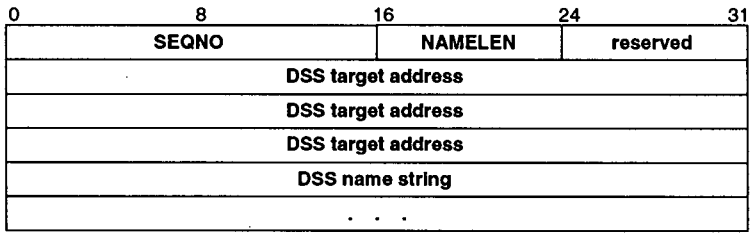


Figure A.10: DSS Name Register Message

name string to be registered, not counting any padding octets. DSS address is the address to be registered with the name. DSS name string is a sequence of ASCII octets, padded to a quadlet boundary. An ASYNC acknowledgement packet is returned to the sender upon completion.

NAME QUERY Message (MSGTYPE=8)

The NAME QUERY message (Figure A.11) is sent to a DSS name server to lookup a DSS address by name. NAMELEN is the length in octets of the name

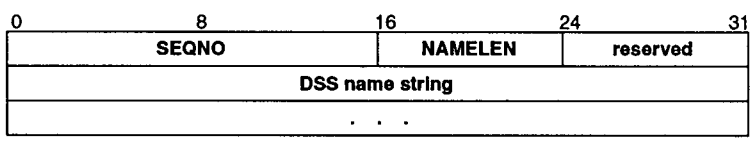


Figure A.11: DSS Name Query Message

string to be looked up, not counting any padding octets. DSS name string is a sequence of ASCII octets, padded to a quadlet boundary. A NAME RESPONSE acknowledgement packet is returned is returned to the sender upon completion. If the lookup was unsuccessful, the `DSS_FAIL` flag is set in the header of the NAME RESPONSE packet.

NAME RESPONSE Message (MSGTYPE=9)

The NAME RESPONSE message (Figure A.12) is an acknowledgement packet containing the response to a NAME QUERY message. SEQNO identifies

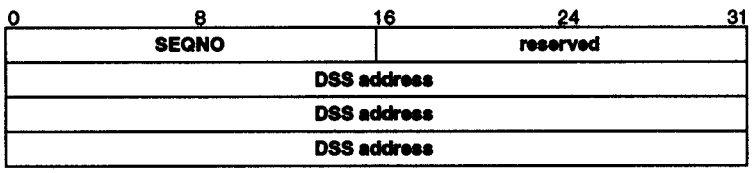


Figure A.12: DSS Name Response Query

the original NAME QUERY message that requested the name lookup. If the `DSS_FAIL` flag is not set, the DSS address is the address that is registered with the name.

EPOCH Message (MSGTYPE=10)

The EPOCH message (Figure A.13) is used to renegotiate the connection's epoch as described below. EPOCH is encoded in the same format as in the CON-

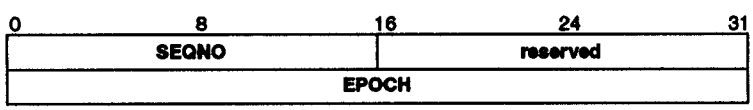


Figure A.13: DSS Epoch Query

NECT message. An ASYNC acknowledgement message is returned to the sender upon completion.

The epoch renegotiation proceeds as follows. Let E_0 be the current connection epoch, and let `MAX_TIMESTAMP` be the maximum timestamp interval encoded in 32 bits (this is 2^{32} microseconds in DSS version 1, but in future versions may depend on the encoding format). The relative timestamp is difference between the absolute timestamp and the current connection epoch. When the time is such that the relative timestamp exceeds `MAX_TIMESTAMP/2`, the connection is in an epoch renegotiation phase. Timestamps relative to the old epoch E_0 can be identified by a 1 in the high-order bit. A new epoch E_1 is established by the signal output port and sent to the input port which records and acknowledges it. When the output port has received the acknowledgement, it begins using the new E_1 epoch for its timestamp references. Figure A.14 depicts the process.

This mechanism ensures that any out-of-order samples will still have their timestamps computed correctly. This method assumes that epoch renegotiation takes less than `MAX_TIMESTAMP/2` and that packet transmission delays are less than `MAX_TIMESTAMP`.

A.2 The DSS Application Programmer Interface (API)

The DSS application programmer interface (API) is a set of C program functions that provides an application with the capability to interact with DSS signals. The interface abstraction is similar to the BSD sockets interface (Comer and Stevens, 1993) and consists of the following functions.

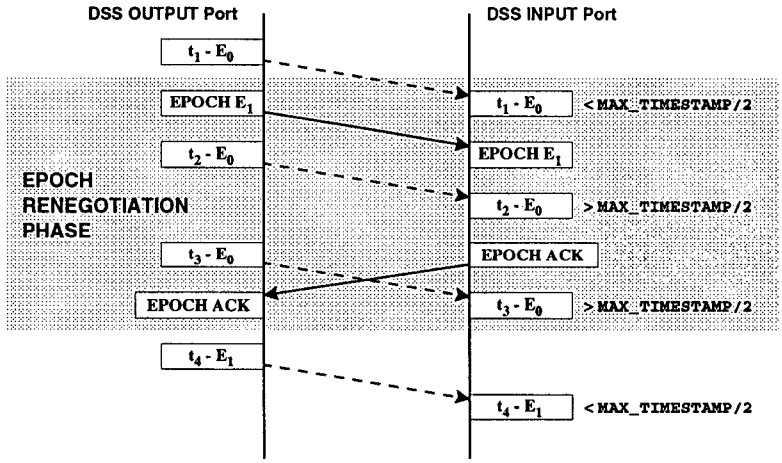


Figure A.14: Epoch renegotiation. The boxes represent DSS messages sent between a DSS output port and a DSS input port and time flows downwards. The dashed arrows show isochronous messages carrying signal data, and the solid arrows show the control messages involved in epoch renegotiation. The relative timestamp contained in an isochronous message is shown as $t_i - E_n$ where t_i is the absolute timestamp of that sample datum, and E_n is the epoch in effect when that message was sent. The shaded area represents the epoch negotiation phase (see text).

void dss_init(char *nameserver)

Initializes the DSS subsystem. The *nameserver* is the transport-dependent name string that identifies the host functioning as the DSS name registry.

void dss_fini()

Terminates the DSS subsystem.

dss_socket dss_open_input(char *name, double rate)

Creates an input socket called *name*, registers it with the name server and returns it. The *rate* is the expected rate at which samples will be read from the socket and is used to determine the interpolation filter characteristics.

dss_socket dss_open_output(char *name, double rate, int format)

Creates an output socket called *name*, registers it with the name server and returns it. The *rate* is the expected rate at which samples will be written to

the socket and is used to determine the interpolation filter characteristics of the receiver. In DSS version 1, *format* is unused; in future versions it will specify the signal data format.

void dss_close(dss_socket s)

Closes socket *s*. Disconnects all open connections and removes its name from the registry.

int dss_lookup(char *name, struct dss_addr *addr)

Looks up the *name* in the DSS registry and returns its full DSS address in *addr*. Returns 1 if successful, 0 otherwise.

int dss_ping(struct dss_addr *addr)

Pings a DSS port to see if it is alive. Returns 1 if successful; 0 if the ping times out.

double dss_read(dss_socket s, double t)

Interpolates and returns the signal's value from input socket *s* for the time *t*. This function performs the signal reconstruction from the currently available samples and will therefore only be accurately reconstructed for t values in the interval $[t, t + dt]$ where dt is the sampling interval of the received signal. Therefore, it should be scheduled in the following manner:

```
while( wallclock() < t )
    usleep(SLEEP_INTERVAL);     /* wait */
x = dss_read( s, t );
```

where `SLEEP_INTERVAL` is a suitably small fraction of `dt`. If the system supports better real-time scheduling facilities, they should be used instead of `usleep`.

int dss_getinfo(struct dss_info *info)

Gets information associated with input socket *s*:

```
struct dss_info{
    double delay;
    int    bufsize;
    int    samples;
};
```

The `delay` is established at connection time and is determined by the interpolation filter width. The `bufsize` is the size of the signal input buffer and `samples` is the number of samples used in the most recent interpolation (invocation of `dss_read`).

int dss_addtarget(dss_socket s, struct dss_addr *addr)

Adds the DSS port with address *addr* to the list of targets for output socket *s*. Returns 1 if successful; 0 otherwise.

int dss_deltarget(dss_socket s, struct dss_addr *addr)

Removes the DSS port with address *addr* from the list of targets for output socket *s*. Returns 1 if successful; 0 otherwise.

void dss_write(dss_socket s, double t, double val)

Writes the signal's value *val* to the output socket *s* with timestamp *t*. The caller is responsible for ensuring that the sampling interval is sufficient to represent the signal's bandwidth as declared in the *rate* parameter of the `dss_open_output` call, and for scheduling calls to `dss_write` within the interval $[t, t + dt]$ where dt is the sampling interval. Otherwise, the receiver may not receive the necessary samples for accurate reconstruction.

In DSS version 1, `dss_addtarget` operates on an open output socket. In future versions, the `dss_addtarget` function will take two DSS addresses and

will not require an open socket. This will allow connections to be established and managed remotely by a connection manager.

The following example illustrates how to set up a DSS output port to deliver a signal computed by `f(t)` to a destination DSS input port with name `listener`. The sampling interval is `dt`.

```
dss_addr dst;
dss_socket *me;
dss_init();
me = dss_open_output("sender", 1.0/dt, 0);
if(dss_lookup("listener", &dst))
    dss_addtarget(me, &dst);
for(t=wallclock(); t<MAX_T; t+=dt){
    while( wallclock() < t )
        usleep(SLEEP_INTERVAL);     /* wait */
    dss_write(me, t, f(t));
}
```

The following example illustrates how to set up a DSS input port that reads a signal and compares it to a computed version. Its sampling interval is `dt`, which is unrelated to the sampling interval of the transmitted signal.

```
dss_socket *me;
dss_init();
me = dss_open_input("listener", 1.0/dt);
delay = dss_delay(me);
for(t=wallclock(); t<MAX_T; t+=dt){
    while( wallclock() < t )
        usleep(SLEEP_INTERVAL);     /* wait */
    x = dss_read(me, t);
    printf("difference = %g\n", x-f(t-delay));
}
```

Appendix B

Robot Assembly Instructions and `mblib` API

B.1 LEGO Tank Robot Construction

The assembly diagrams for the the LEGO tank robot of Chapter 5 are shown in Figure B.1 – Figure B.5.

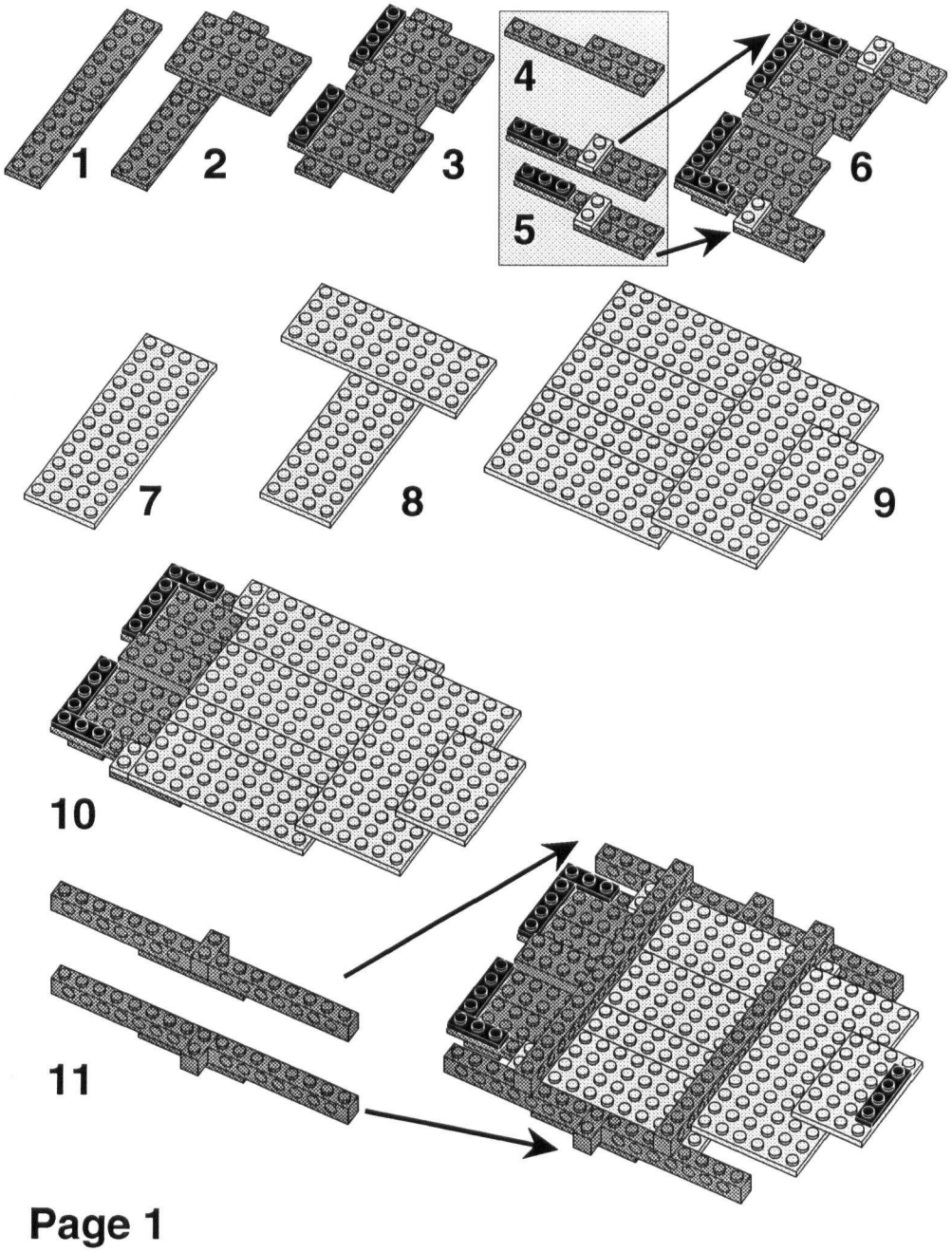


Figure B.1: Robot Tank Chassis LEGO Assembly, Part 1

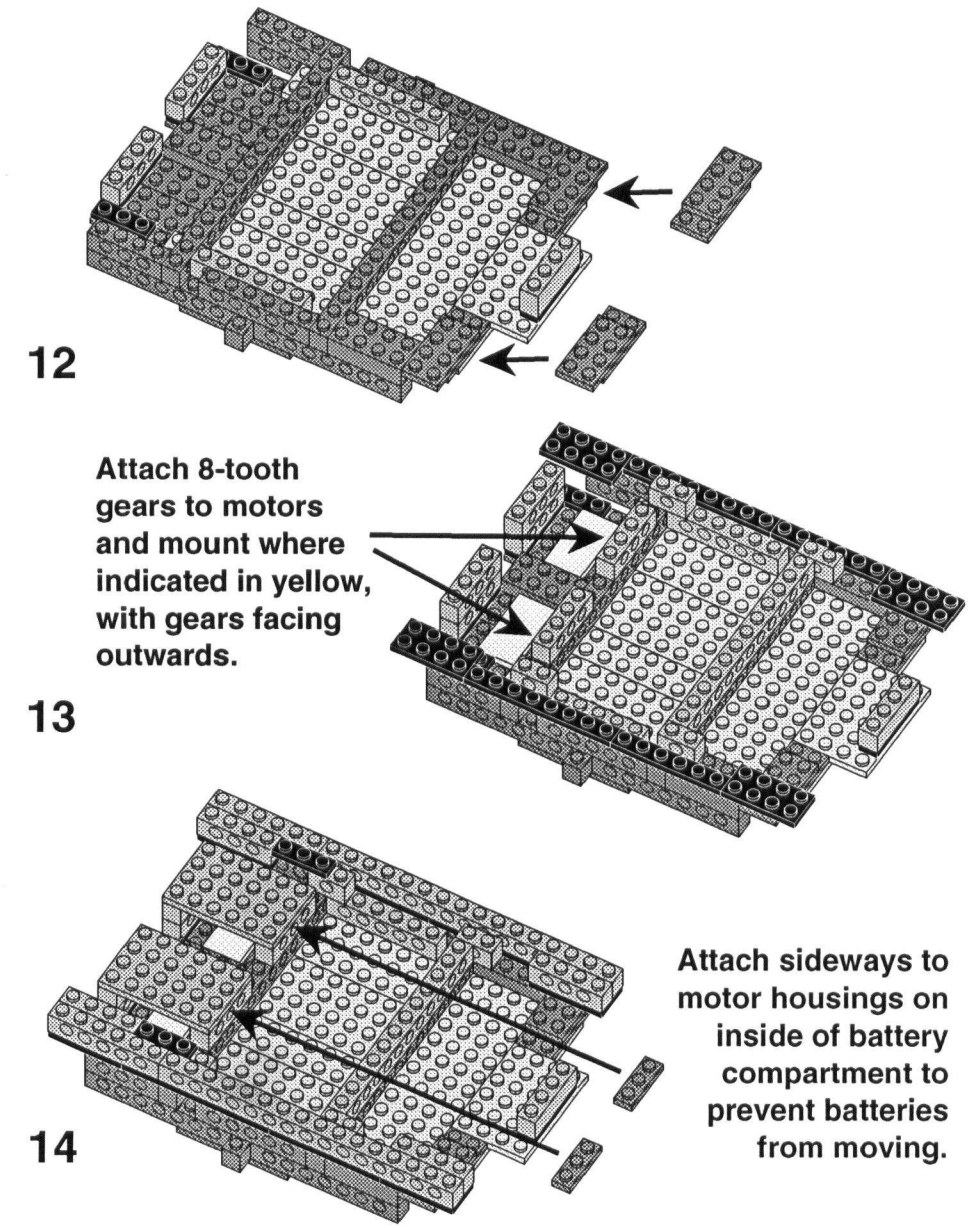


Figure B.2: Robot Tank Chassis LEGO Assembly, Part 2

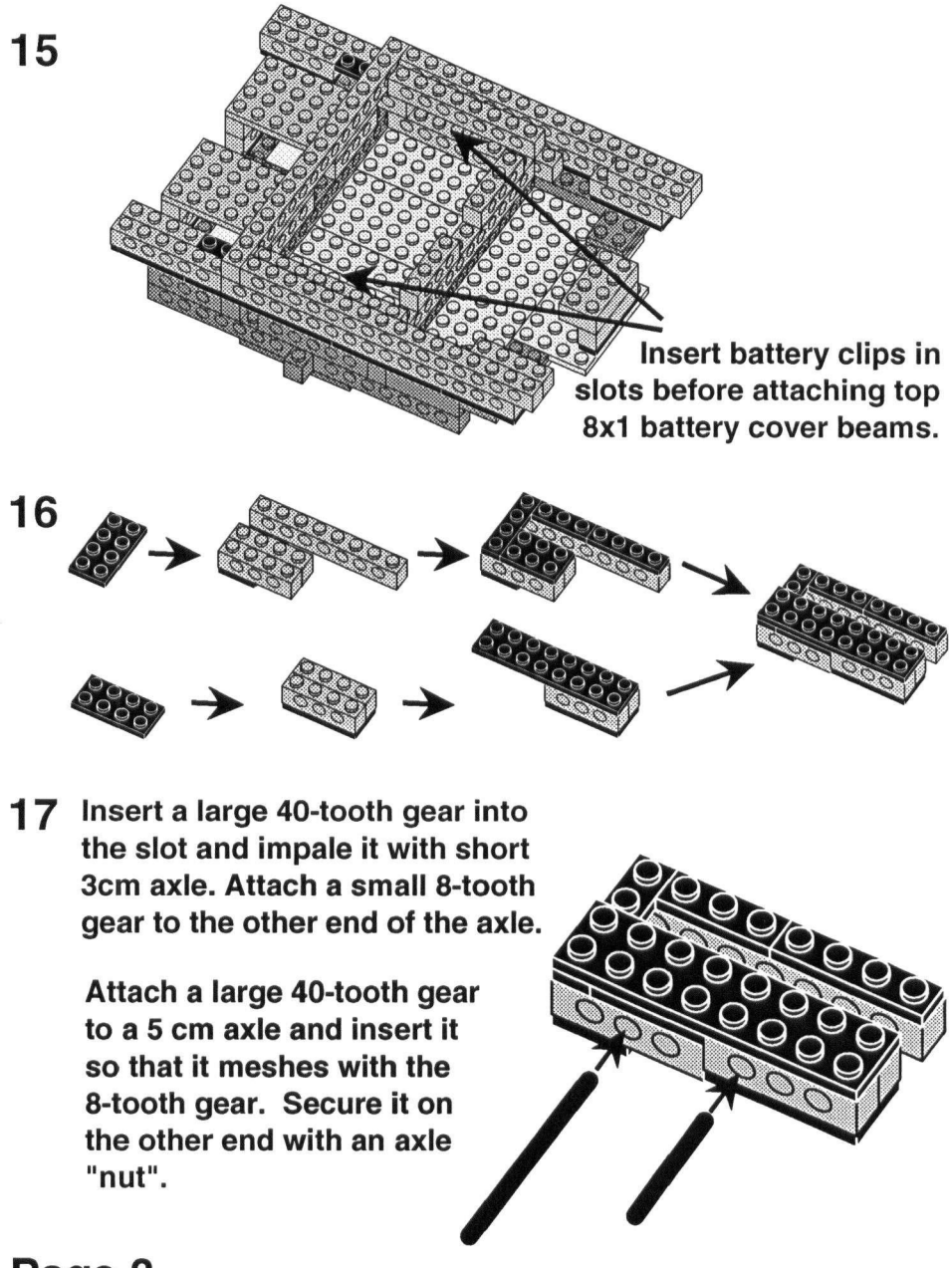


Figure B.3: Robot Tank Chassis LEGO Assembly, Part 3

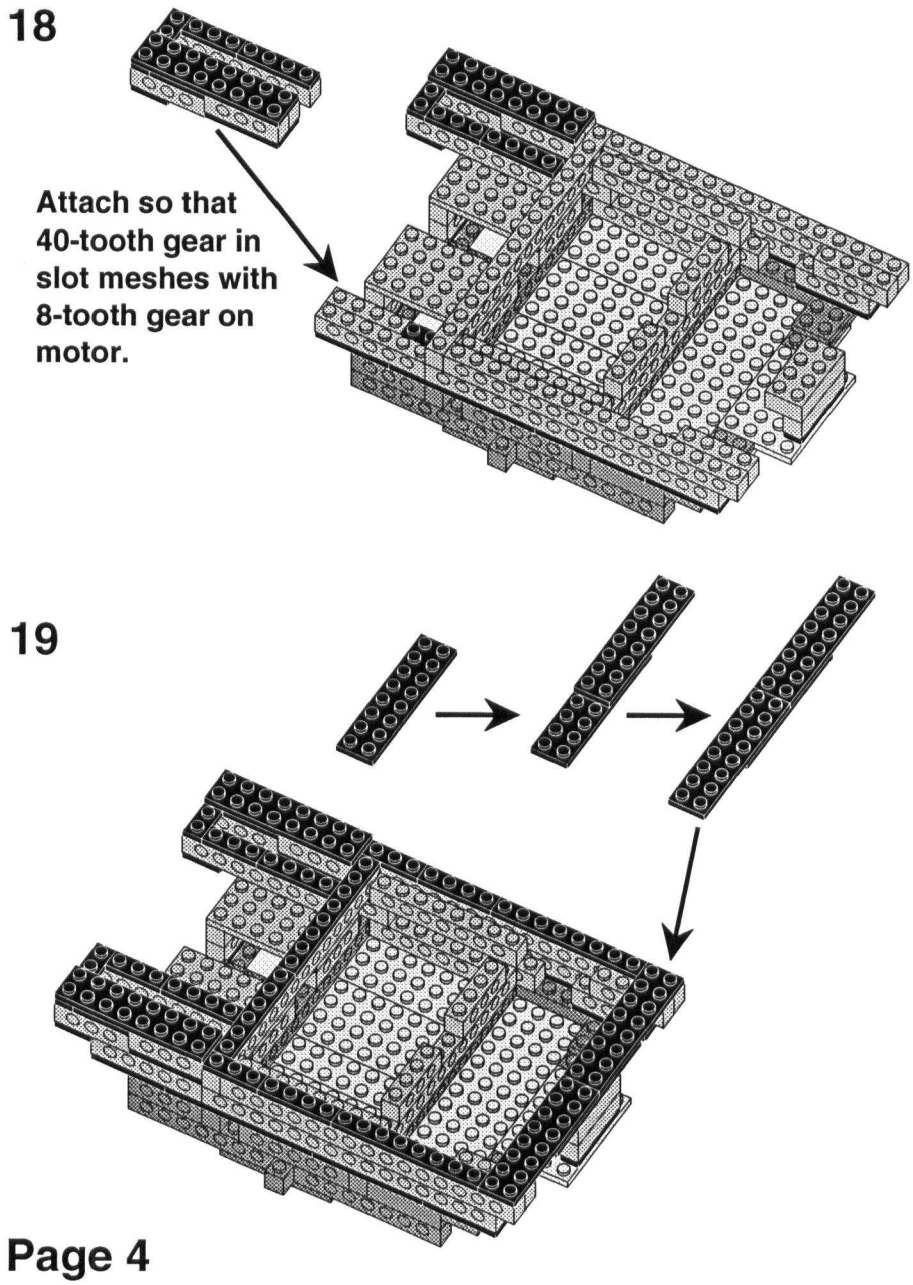
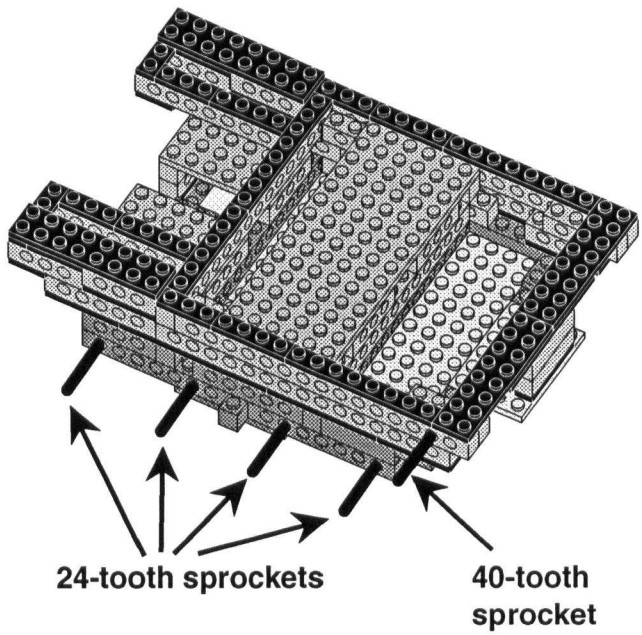


Figure B.4: Robot Tank Chassis LEGO Assembly, Part 4

20 Sprocket Positioning

Attach freewheeling sprockets at indicated positions. You'll need to use snap-in axles for the middle pair of small sprockets because there is no clearance on the other side of the hole. Snap-in axles aren't found in kit 9605, but can be found in other Technics kits.

With a little modification, the two rear-most sprockets can be mounted on shaft-encoders. The rear engine space was designed to accommodate a modified ALPS "press brush contact" type rotary encoder from an old (circa 1990) Microsoft mouse.



Link together 74 chain links to form each tread.

Page 5

Figure B.5: Robot Tank Chassis LEGO Assembly, Part 5

B.2 Miniboard Host C Library

The `mblib` host library consists of the following API:

`mb_open(port)`

Open and return a miniboard descriptor using the given serial port.

`mb_close(mb)`

Close the given miniboard descriptor.

`mb_cleanup()`

Clean up the miniboard system after an error.

`mb_run(mb,interval,userfunc,userparm`

Start periodically updating the Miniboard at the specified interval. After each update, the user supplied function is called with the provided parameter.

`mb_stop(mb)`

Stops the periodic Miniboard updating.

`mb_set_motor_pwr(mb,n,on)`

Turns motor `n` on or off.

`mb_motor_pwr(mb,n)`

Returns the on/off state of motor `n`.

`mb_motor_set_dir(mb,n,dir)`

Sets the direction for motor `n`.

`mb_motor_dir(mb,n)`

Returns the direction for motor `n`.

`mb_motor_set_speed(mb,n,speed)`

Sets the speed for motor `n`.

`mb_motor_speed(mb,n`

Returns the speed for motor `n`.

mb_set_digital_dir(mb,n,dir)

Sets the direction for digital I/O bit n.

mb_digital_dir(mb,n)

Returns the direction for digital I/O bit n.

mb_set_digital(mb,n,value)

Sets the output value for digital I/O bit n.

mb_digital(mb,n)

Returns the state of digital I/O bit n.

mb_switch(mb,n)

Returns the complement of the state of digital I/O bit n (more intuitive when using switches).

mb_analog(mb,n)

Returns the value of analog input n.

mb_dshaft(mb,n)

Returns the number of milliseconds between edges on the digital shaft n.

mb_read_version(mb)

Reads the BINMON version string.

mb_read_byte(mb,addr)

Reads the byte at Miniboard address addr.

mb_write_byte(mb,addr,data)

Writes the given byte of data at Miniboard address addr.

mb_read_word(mb,addr)

Reads the 16-bit word at Miniboard address addr.

mb_write_word(mb,addr,data)

Writes the given word of data at Miniboard address addr.

`mb_reset(mb)`

Resets the Miniboard.

For a full description of this API, see the `miniboard.h` file in the `mblib` package.