ROBUST SNAKE ROBOT CONTROL VIA A SPIKING NEURON CENTRAL PATTERN GENERATOR

by Raphael Norman-Tenazas

A thesis submitted to Johns Hopkins University in conformity with the requirements for the degree of Master of Science in Robotics

Baltimore, Maryland

May, 2021

© 2021 Raphael Norman-Tenazas

All rights reserved

Abstract

Snakes, due to their structure, are very well adapted to navigating small spaces and diverse, unstructured, and potentially amphibious terrain. In robotics, navigating these environments is difficult for conventional tracked, wheeled and legged robots, but snake robots should be well suited for it because of their connection to their biological analog. However, designing a robot to match the morphology of a snake comes with its own challenges: a high number of degrees of freedom and complex dynamics. Coordinating these many degrees of freedom to produce locomotion is challenging. Traditional control methods using models, sine waves or shapes fall short when applied to multiple environments and can be susceptible to process and sensor noise. However, neuroscience might be the key to coordinating these high degrees of freedom. In animals, their nervous system can somehow synchronize movements while being very robust across all types of environments. In particular, central pattern generators (CPGs) are a type of neural circuit found in many animals that produce rhythmic outputs for locomotion. Simulations of CPGs have been used in the past to control legged and hyper-redundant robots, but often require the tuning of a large number of parameters. In this work, we implement a neuron-based spiking CPG (SCPG) to control a snake robot in simulation. We generate our neural network from the ground up, fixing continuous parameters and optimizing over the discrete structure space. We compare our method to state of the art locomotion algorithms in environments of increasing complexity, and show that our SCPG has an increased robustness to environmental parameters.

Thesis Committee

Primary Readers

Will Gray-Roncal (Primary Advisor)
Assistant Research Professor
Department of Computer Science
Johns Hopkins Whiting School of Engineering

Alternate Readers

Ralph Etienne-Cummings
Professor
Department of Electrical and Computer Engineering
Johns Hopkins Whiting School of Engineering

Acknowledgements

I would like to acknowledge my advisor and manager, Will Gray-Roncal for his guidance on this thesis and research, Ralph Etienne-Cummings for his direction and previous work towards this endeavor, as well as my co-worker Erik C. Johnson for his ideas and inspiration. Finally, I'd like to thank my parents Raoul and Reynolds for cultivating my creativity all these years.

I'd also like to acknowledge Samuel S. White for his edits to the snake robot shell.

The design and construction of the physical robot were supported under the Johns Hopkins University Applied Physics Laboratory Internal Research and Development (APL IRAD) funding. The initial implementation of the oscillatory Hopf CPG controller was submitted by me as a final project for the class "Modeling and Design of Complex Systems" (EN.605.716), but it was an important baseline to which to compare my results, so I have included it here.

Dedication

Dedicated to my grandfather Philip S. Norman, who sparked my interest in science.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

We often turn to robots to help us explore areas that are unreachable, dangerous or otherwise not suited for humans. However, these areas are often not easy for robots to navigate, especially conventional ones that have wheels. Quadrupeds may have an advantage in navigating rubble, but they are often large and cannot fit in small spaces. Tracked robots can go over rubble or fit into tight spaces, but not both as they can only go over rubble smaller than their tracks. Therefore, there is an opportunity for unconventional robot morphologies to excel in this space.

Snakes have a very simple morphology, but are highly maneuverable. In particular, they can navigate aquatic, cluttered and tight environments. Because of this, it makes snakes prime candidates for biologically-inspired robots. This makes them well suited for tasks such as search and rescue [65], inspection [34] and surveillance. However, while snakes can easily control themselves to navigate the aforementioned environments, snake robots have a difficult time because of a large number of degrees of freedom (DOF). While snakes have evolved to easily coordinate these DOF, snake robot locomotion still has many open questions and areas to explore.

Since the design of the first snake robot from Hirose [24], their locomotion was a highly researched topic. The design itself of the first snake robot was simple: a series of

rotational joints, each offset 90 degrees about the axis of the robot. Hirose [24] modeled the locomotion of a sine wave, where the phase of each joint was based on its position along the body. For many applications, this open-loop formulation is sufficient, as it produces locomotion on flat surfaces, and by changing parameters, a variety of interesting gaits and behaviors can be achieved [60]. However, these trajectories can fail when applied to complex, unstructured terrain.

Traditional model-based dynamic or kinematic approaches have also been applied to snake robots [36][35][37], for both locomotion and obstacle avoidance. These controllers have the same problem as their simple gait predecessors: operating in unknown environments.

More recent work has shown that torque measurements at the joints themselves can be used to alter the shape of the robot around local obstacles. This allows the robot to use objects that normally would be considered obstacles as instruments for locomotion [62][61]. The controller also uses compliance to help the robot conform in three dimensions, which enables it to traverse objects much larger than itself. The controller is highly configurable to use arbitrary shapes, and can negotiate relatively complex environments without a model. Therefore, we consider it to be state of the art for model-free controllers.

Simultaneously, biology has also been used to not only influence the morphology of these robots, but also their control. In many animals, oscillating neural circuits can control regular processes such as breathing and heartbeats, and produce locomotion. In particular, central pattern generators are a type of oscillating circuit that produce patterned periodic signals. CPGs have been identified for swimming, heartbeats, respiration, walking and flight [58], in ranging from the C. elegans [45] to rats. CPGs are known for being able to produce output in the absence of external input, but many utilize external signals from modulatory projection neurons to control the activation of neurons within the network. By

doing so, the cycle frequencies and phase relationships between neurons can be changed [58]. Thus, they can be used as a control method for robots.

Central pattern generators are not a novel concept in their application to robotics. They have been used in hexapod robots[62][46][48] and fish robots [13] to produce locomotion as they help reduce the degrees of freedom for an operator or a higher level controller. They have also been used not only to produce joint trajectories, but to estimate their state as well, based on torque measurements [55]. In snake robots, central pattern generators have been implemented in a number of ways, often as a chained dynamical system [25][64][67]. More recently, Angelidis et al. [5] implemented the chained dynamical system (based off the work of Ijspeert and Crespi [25]) on neuromorphic hardware by utilizing the Neural Engineering Framework [15].

Implementation on neuromorphic hardware (or analog hardware [33][51]) is one attractive property of neuron-based central pattern generators. This can reduce controller power consumption considerably. This fact is often overlooked because the energy required to actuate the joints of a robot far outnumbers the power used for the lower-level locomotion controller. Nonetheless, advancement of energy-reduction methods for high-DOF controllers can be used for micro-robots where the power consumption of the controller can surpass that of the actuator. For example, Goldberg et al. [20] estimates that their cost to travel for their insect scale robot is an order of magnitude higher due to control and power management hardware.

In this work, we present a spiking, neuron-based, compliant central pattern generator that has been evaluated in simulation. We first explore the nature and development of this CPG as an iterated motif, focusing on stable oscillatory output and phase-locking. We also devise a method for introducing compliance and flexibility into the CPG in the form of modulatory current. Then, we apply it towards a snake robot in simulation, comparing it

to some of state of the art methods mentioned above and demonstrate its effectiveness and robustness across environments of increasing complexity.

1.1 Prior work

A non-exhaustive list of SCPGs applied to hardware can be found in Table 1.1. We discuss the prior work across multiple dimensions in more detail below.

1.1.1 On snake robot design and construction

There have been a number of snake robot designs since the the ACMIII [24]. We will focus on the ones that were used in the publications to which we are comparing our controllers to. Travers et al. [62][61] uses eighteen series-elastic modules from Pratt and Williamson [50]. Each motor was oriented such that its axis of rotation was ninety degrees from its adjacent motors. Wang, Gao, and Zhao [64] uses a custom-built ten-link snake robot with each joint having two degrees of freedom (one in the left/right plane, and one in the dorsal/ventral plane). Of note is that both of these robots were tethered, which allowed the robots to be powered and controlled externally.

1.1.2 On snake robot controllers and locomotion

Traditional, model based controllers [36][35][37][19] certainly can provide optimal joint trajectories, however, these are most limited by the need for a model of the environment and its obstacles. We recognize that some approaches have success creating partial models of the environment using sensors [28]. We are considering model-based controllers to be out of the scope of this paper since we are targeting environments where we do not know the model beforehand.

When one does not have a model of the environment, model-free approaches are

appropriate. However, state of the art model-free, shape based approaches [62][61] are limited by the fact that they are reliant on a few parameters that make assumptions about the environment. For example, the frequency and amplitude of the shape waves that define a good shape in a sparse environment might not transfer well to a more dense environment. However, for an environment for which it is well tuned, the performance is outstanding.

We also consider state of the art CPGs formed from chained dynamical systems. The CPG model from Ijspeert and Crespi [25] was the one of the first such models, and Travers et al. [62][61] compares their shape controller to it. It is a very strong candidate for model-free snake robot control because it is robust to perturbations and noise, and can use force input from the environment to change its parameters [2]. However, it is most limited by its potential gaits, and several more recent CPGs[67][64] attempt to remedy this. In our comparison, we use the CPG from Wang, Gao, and Zhao [64].

Spiking CPGs (SCPGs) are a new form factor. Angelidis et al. [5] utilized the Neural Engineering Framework, which enables the translation of arbitrary dynamical systems into spiking neural network populations, to convert the CPG from Ijspeert and Crespi [25] into a spiking neural network. However, this approach typically requires a large amount of neurons. In this particular publication, they range between 280 to 5000 neurons per oscillator, on a snake robot with eight joints.

1.1.3 On SCPGs and hardware

Typically, neuromorphic hardware has a restriction on the neuron types that are available to use on the platform. For example, the Adaptive Leaky-Integrate-and-Fire (ALIF) neurons that we use in this publication are not available on our target platform (the Intel Loihi [12]). However, Polykretis, Tang, and Michmizos [48] has a very clever approach to

overcome this limitation by defining their bursting neurons as chained leaky-integrate and fire (LIF) neurons and passive compartments. Effectively, this makes each neuron a multi-compartment neuron. By defining their neurons this way, they were able to achieve a small network size while still having the oscillatory firing patterns necessary for hexapod locomotion.

Several older publications that were written when neuromorphic hardware was still in its infancy solve this problem by using analog hardware [51] [33] or VLSI [32]. Our implementation of compliance is similar to Lewis, Tenore, and Etienne-Cummings [32], which uses inhibitory networks with excitatory tonic drives for modulation.

The aforementioned SCPGs do not take into account the difficulties of hand-tuning an entire network for robotic control. We consider two recent publications [3][16] to be state of the art in terms of spiking CPG design and parameter tuning. Espinal et al. [16] finds the structure and weights of an integrate-and-fire (IAF) spiking CPG using Christiansen grammars and an evolutionary algorithm. It is an advanced way of finding the structure and weights, but falls short since it lacks the scalability necessary for snake robot locomotion. Aljalbout et al. [3] uses a static built CPG with a pool of reservoir computing neurons to provide a task-agnostic interface. While this has a very strong to potential to work with snake robots, it was not explicitly discussed. The authors also rely on a static architecture, where they optimize weights of the synapses between the motoneurons and motors using the ReSuMe method [49].

In our publication, we will show that we can optimize this structure while maintaining static weights. We do this with a multi-step search algorithm, and validate our results compared to an oscillator CPG and a shape based controller. We show that our approach can more stable performance in environments of increasing complexity.

SCPG	Implementation	Robot morphology	Tuning method
Lewis et al. [33]	Analog	Biped	Hand
Lewis, Tenore, and Etienne-Cummings [32]	Analog	Biped	Hand
Espinal et al. [16]	Microcontroller	Quadruped	Christensen grammar + GA
Espinal et al. [16]	FPGA	Hexapod	Christensen grammar + GA
Russell, Orchard, and Etienne-Cummings [54]	Microcontroller	Biped	GA
Maruyama, Ichimura, and Maeda [40]	Analog	Quadruped	Hand
Spaeth et al. [57]	Microcontroller	Modular	Hand
Maufroy, Kimura, and Takase [41]	Simulated	Biped	Hand
Kwiatkowski and Lipson [31]	Simulated	Hexapod	Remote Supervision Method
Polykretis, Tang, and Michmizos [48]	Loihi	Hexapod	Hand
Gutierrez-Galan et al. [21]	SpiNNaker	Hexapod	Hand

Table 1.1: An non exhaustive overview of SCPGs applied to robots

Chapter 2

Theory

The design and control of snake robots is necessarily linked. A given snake robot controller will not always work with all snake robot morphologies. Therefore, it is imperative to examine the concepts behind both the controls and design of the robot we use in this publication. First, we provide an overview of other designs and configurations that have been used in prior work, then we give an overview of the mathematical concepts behind the shape-based and Hopf-oscillator based controllers we are comparing to. Finally, we examine past SCPG structures.

2.1 Design

2.1.1 Kinematic approach

Most snake robots are an example of hyper-redundant serially-linked robots. In many designs, the robots are composed of serially-linked rotational joints. Sometimes, joints are divided into two alternating orthogonal planes of rotation (dorsal/ventral and left/right) such that each joint is rotated 90° from its preceding and following joints [62][24]. Others have all their joints rotating in the left/right plane [66][10][56]. Finally, some have two axes of rotation at each joint [34][27]. A good review of robot morphologies can be found in Liu, Tong, and Liu [38].

2.1.2 Exterior

In biology, snakes move on flat surfaces because their scale layout and structure provides anisotropic friction, where their forward friction is much less than their perpendicular friction and backward friction [17]. In snake robots, some have attempted to emulated this by adding wheels [56][10], while others have solved it by adding "fins" to the robot [34], or by minimizing frictional effects and using obstacles [62][61]. Nonetheless, friction is important to consider, especially in a flat 2D plane. Crespi, Ijspeert, and F [9] notes that changing their frictional coefficient in simulation reduces their speed by 24%. Since the applications of snake robots often revolve around inspection and search, most of the designs typically have a camera or image sensor of some sort [34][53][27]. One places a bladed saw at the end of the robot for purposes of cutting a material[42], although a physical version of this is not built to our knowledge.

2.2 Control

The control of snake robots can generally be broken into two categories: model-based control, where a model of the robot and environment is used to generate trajectories, and model-free control, which produces trajectories without knowledge of the environment.

2.2.1 Model-based

Model-based approaches rely on a dynamic or kinematic model of the environment and robot to accomplish their task, which means they are limited by how accurate that model is. This is particularly difficult in novel or complex terrain, where it is next to impossible to have a model *a priori* or have a constructed model good enough to execute on. Since the dynamics model of snake robots are highly nonlinear, model-based approaches are also difficult to implement. Several dynamics models are elaborated in Liljebck et al. [37].

These models include 2D and 3D models, with varying friction constraints. Nonhoff et al. [44] utilizes a simplified model from this to produce an "economic" model predictive control for a snake robot.

2.2.2 Model-free

Model-free control produces trajectories in the joint-space that do not require knowledge of the environment nor a dynamic model of the robot. However, assumptions are made about the robot's form. The reason they are used is because they produce a "net desired behavior"[62].

2.2.2.1 Centralized

Centralized controllers generate a trajectory for all the joints at once using a single model. In traditional snake robot literature, this is usually defined as a sine wave for every joint, nominally:

$$\theta = \kappa + A\sin(\eta s - \omega t) \tag{2.1}$$

This equation is referred to as the serpenoid curve in the literature. Here, $\theta = [\theta_1, \theta_2 ... \theta_n]$ is the vector of desired angles at each of the joints s at time t, $\kappa = [\kappa_1, \kappa_2 ... \kappa_n]$ is an offset vector which can allow turning, η is the spatial frequency parameter that defines the curve along the snake robot body, ω is the temporal frequency pattern that defines how fast the robot oscillates.

Typically, by modifying these parameters, one can define a set of different gaits for a snake robot that can be of use in varying terrain. In particular, when a snake robot has two alternating orthogonal planes of rotation as described before, one can define two serpenoid curves for each plane, with linked temporal frequency parameters. This enables the robot to have gaits not inspired by snakes, such as corkscrewing and linear progression[60]

2.2.2.2 Shape based

Shape-based control is somewhat of an extension of the centralized control. In general, a shape function can be defined as any function $h(\sigma) \rightarrow \mathbb{R}^N$, where $\theta \in \mathbb{R}^N$ and $\sigma = [\sigma_1, \sigma_2 ... \sigma_m]$ are the shape parameters. For example, the serpenoid curve (2.1) is a shape equation where $\sigma = \{\kappa, A, \eta, \omega\}$.

Another shape equation is defined in Travers et al. [62] that allows shapes to progress down the body of the snake robot. This is useful for using objects in terrain to aid in movement, as the snake robot can form shapes around these objects and push off of them. The following shape and compliance equations have been directly taken from Travers et al. [62] and Travers, Whitman, and Choset [61].

They define shapes as an amplitude modulation of the serpenoid curve i.e

$$\theta = A(s,t)\sin(\eta s - \omega t) \tag{2.2}$$

The amplitude modulation signal itself is a sum of Gaussian activation windows:

$$A(s,t) = \sum_{i=1}^{W} A_i \exp(-\frac{(s-\mu_i(t))^2}{2\psi^2}) \tag{2.3}$$

Here, W is the number of activation windows, A_i defines the activation of the ith window, $\mu_i(t)$ defines the center of the ith window, and ψ defines the width of the windows. As a shape equation, the shape parameters are $\sigma = A_1, A_2 ... A_W$. The serpenoid curve in Equation (2.1) acts as a carrier wave for the amplitude modulation signal in that it moves the shape down the body of the snake.

The amplitude modulation signal itself also moves as a function of time due to the moving window centers. In writing the equation of $\mu(t)$, one can either choose a infinite number of windows that propagate at the same speed or choose a rule to recirculate them such that the number of window centers remains constant. For simplicity, the latter case is

used. If one keeps the spatial frequency η and number of windows W constant, you can define a rule to recirculate the window centers as

$$\mu_i(t) = \mu_i(0) + mod(\frac{\omega}{\eta}t, \frac{\pi}{\eta}) \tag{2.4}$$

Here, the initial window centers $\mu_i(0)$ is constrained such that all the window centers have a constant distance.

$$\mu_{i+1}(0) = \mu_i(0) + \frac{\omega\pi}{\eta} \tag{2.5}$$

The end result is that each window center is recirculated when it reaches the original location of the following window center.

Compliance is accomplished by utilizing torque measurements at each joint. The value of compliance is that it allows the snake robot to form its body around its environment via proprioceptive forces. The way that these torques are used depends greatly on the controller.

Compliance in a shape based controller is done as a modulation of the shape parameters. In particular, a compliant shape controller modifies its shape parameters from nominal values σ_0 to desired values σ_d, which are modified based on the proprioceptive forces mentioned above. This is done using a second order Dynamic Motion Primitive (DMP) controller of the form

$$M\ddot{\sigma_d} + B\dot{\sigma_d} + K(\sigma_d - \sigma_0) = J\tau_{ext} \tag{2.6}$$

Where M, B, K are parameters that control the response of the system and τ_{ext} is the external torque at each of the joints. J is the Jacobian that maps the joint-space into the shape space based on the shape-function $h(\sigma)$.

$$J = \frac{\partial h(\sigma)}{\partial \sigma} \tag{2.7}$$

Intuitively, this means when a robot joint moves in a direction and encounters an object, it reduces its desired joint angle, as the torque is higher than in free motion. The gravitational forces also provides torque to the non-planar joints, which allows the robot to conform in three dimensions.

2.2.3 Central pattern generators

Central pattern generators (CPGs) are model-free controllers that are typically found in biological systems. They typically produce rhythmic output based on non-rhythmic input that is thought to be used for repetitive movement like slithering, swimming and walking. Some biological examples can be found in snakes, eels, lampreys, mice and even humans. The use of CPGs as locomotion controllers is not a novel idea, and many works utilize something similar [9] [14] [4] [67] [23].

Models of CPGs can be divided into two general categories – dynamical systems based and spiking neuron based.

Oscillator-based models simulate differential equations over time in order to have individual joint angles approach a limit cycle. This means they are naturally robust to small changes in initial conditions. However, since they are mostly utilized as model-free feed-forward controllers, their robustness is underutilized for robotic control. Their advantage to sinusoidal feed forward controllers is that they enable smooth changes in desired motor angles when parameters are changed. In Wang, Gao, and Zhao [64], they define their CPG using oscillators implemented as a two dimensional system.

$$\begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} -\omega u - \lambda(\frac{u^2+v^2}{A^2} - \sigma)v \\ \omega v - \lambda(\frac{u^2+v^2}{A^2} - \sigma)u \end{pmatrix} + s(t) \tag{2.8}$$

Equation 2.8 can be written in a state space representation as:

$$\dot{x} = f_H(x;, \omega, A) + s(t), x = (u, v)^T \tag{2.9}$$

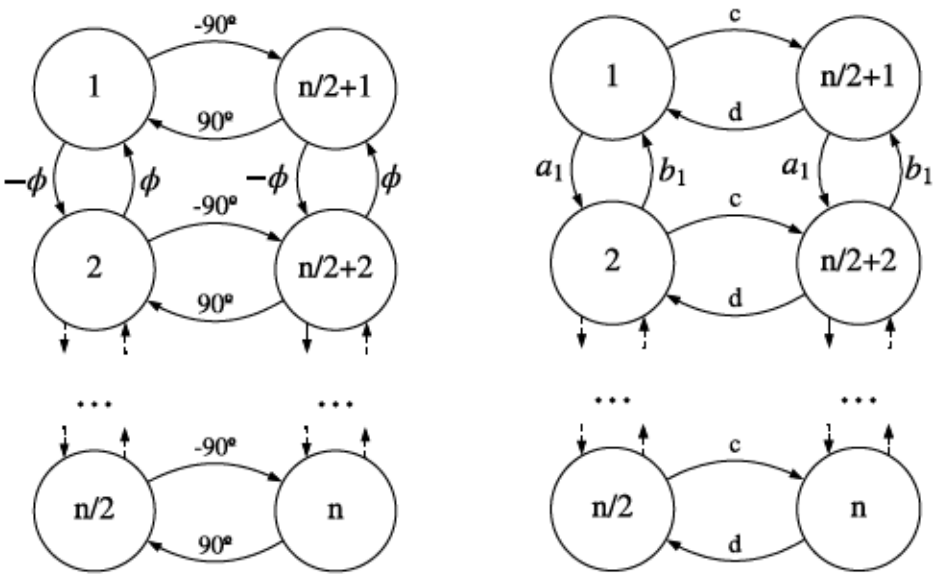
Here, ω and A are, as defined in the serpenoid equation, the temporal frequency and amplitude of the limit cycle. λ defines the convergence rate of the system to the limit cycle, and σ is a Hopf bifurcation parameter. $s(t) \in \mathbb{R}^2$ is a coupling input that allows oscillators to synchronize with each other to enable coordinated locomotion. This coupling input can have several different configurations based on the desired effect. Wang, Gao, and Zhao [64] defines this input for the ith oscillator as:

$$s_i(t) = -\sum_{j \neq i}^{n} w_{i,j}(x_i(t) - \frac{A_i}{A_j} R(\phi_{i,j}) x_j(t) \tag{2.10}$$

where n is the number of oscillators, $w_{i,j}$ and $\phi_{i,j}$ are the weight and phase between oscillator i and oscillator j, and $R(\phi) \in SO(2)$ is the 2D rotation matrix about ϕ.

The weights and phases of the system are dependant on the topology of the oscillator chain. Wang, Gao, and Zhao [64] considers multiple topologies, but the one they consider best is in Figure 2.1, which has been adapted from their paper. Each side of the chain corresponds to a different plane of the robot: the left side (oscillators 1 to $n/2$) of corresponds to the dorsal/ventral joints and the right side of the chain (oscillators $n/2$ to n) corresponds to the left/right joints.

The left and right sides of the chain have a constant phase offset of 90° between them, which is related to the physical rotation of adjacent joints. They also share a constant weight. Between adjacent in-plane oscillators, there is a global variable phase offset ϕ that



(a) Phase **(b)** Weights

Figure 2.1: Chained oscillator topology for CPG defined in [64]. 2.1a shows the phase offsets between different oscillators and 2.1b shows the weight configuration

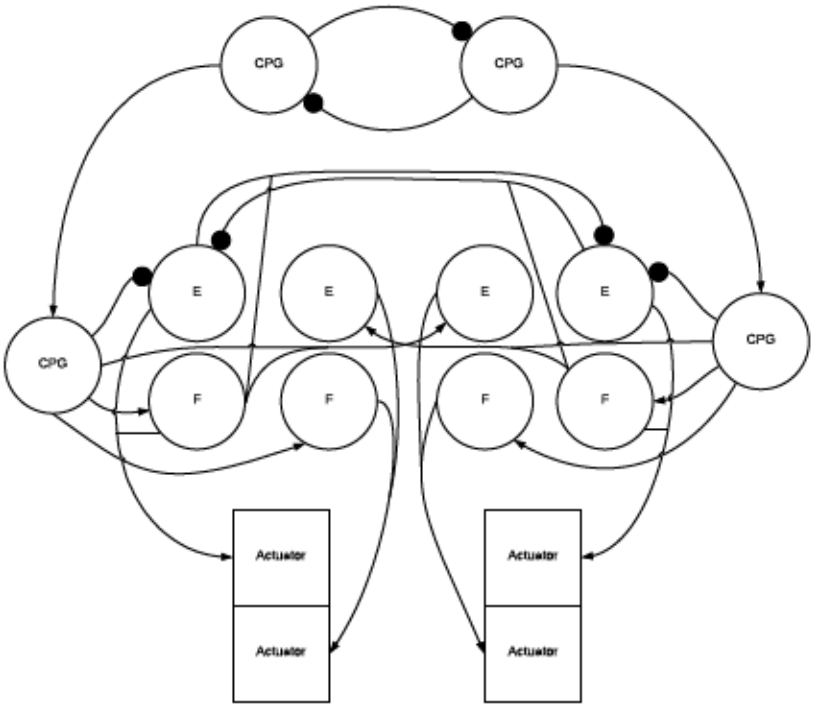
corresponds to the spatial frequency parameter η in the serpenoid equation.

Neuron-based CPGs are typically implemented as a recurrent spiking neural network. The implementation and structure is highly dependant on the morphology of the robot. For example, in Lewis et al. [33], an integrate and fire (IAF) model was used with the dynamics:

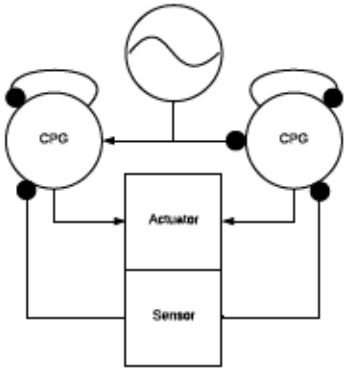
$$C_i \frac{dV_i}{dt} = J - G_i^T I_{dis} + I_0 \left(\sum_j W_{ij} G_j \right)$$

Where C_i is the capacitance of the neuron membrane, I_{dis} is the discharge current, $V(t)$ is voltage, $I(t)$ is current, W_{ij} is the weight between i and j, $G_j(t)$ is the spike train of neuron j and J is an external current. In Aljalbout et al. [3] and Espinal et al. [16], a discrete time IAF model is used with essentially the same equations. The notation here has been unified to be consistent the rest of this paper. We show a comparison of the various forms that

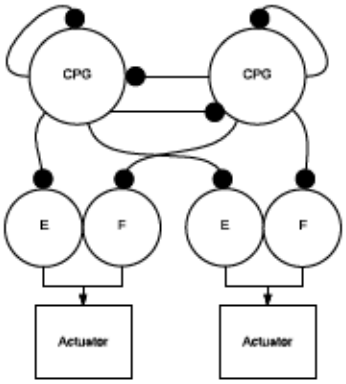
SCPGs have taken in Figure 2.2. None of the publications claim that their designs are bio-fidelic, but rather that they are inspired from biology. Most of them revolve around innervating motoneurons to mimic flexion and extension muscles controlling a joint. The figures have been adapted and modified from their original publications.



(a) Espinal et al. [16]



(b) Lewis et al. [33]



(c) Lewis, Tenore, and Etienne-Cummings [32]

Figure 2.2: Some examples of SCPG motifs from prior work. The full networks are often larger. Nodes labeled "CPG" represent neurons crucial for oscillatory behavior, while "E" and "F" represent extension and flexion motoneurons. (a) shows a relatively complex system to control two tibia and coxa of a hexapod. Network size is made of three motifs.(b) shows two bursting neurons with a pacemaker to control a single hip of a biped with feedback. The full network is two motifs. (c) shows scheme to control a single knee and hip of a biped. The full network is two motifs.

Chapter 3

Methodology

In this chapter, we describe our approach in designing the robot. We then show our methods to recreate the robot in simulation as well as the environments we used to test the controllers. Next, we detail the methods to implement the comparison controllers and our custom CPG controller, including how we introduce compliance. Finally, we discuss the metrics that we use to compare all three controllers across multiple environments.

3.1 Physical robot design

Our designed snake robot, which we named "Neurally Adaptive Graph Agent" (NAGA) is designed to be built using commercially obtained parts and a 3D printer, and to not require custom electronics or metal parts. An image of NAGA is shown in Figure 3.1. It is designed to be modular, allowing any number of links to operate in various configurations. Each link can be attached to the next link such that it rotates in the same axis, or the perpendicular axis. In this work, all joints rotate in the same axis since the environments are all planar and flat. Each joint is actuated by a single servo, a Dynamixel AX-12A. The Dynamixel joints have a clamped proportional controller that controls their angle. It should be noted that the AX12-A does not contain an integrated torque sensor, and one is not included on the physical robot.

A round, 3D printed shell surrounds each of the joints in order to minimize friction and to add protection to the wires and electronics. A "head" link which carries an IR distance sensor, a battery and motor controller is attached to the first joint. A tail, using the same design as the head is included as well on the physical robot, and contains another battery. Both the segment design and head design are described in Figure 3.3. The joints are wired together in a daisy chain fashion and communicated using a half-duplex serial connection. The controller for the robot is an Arbotix-M board which has a plot for an X-Bee wireless connectivity module that enables communication with a laptop running ROS. The information flow is shown in Figure 3.2.

3.2 Simulation design

The simulated robot matches the physical robot in most ways. However, because the simulated controllers that our chosen robot simulator, Gazebo, offers do not include a proportional controller, a simulated PID controller is used. We also include a simulated torque sensor at each of the joints. We chose to not include the tail segment for simplicity, but found that the head segment was necessary for movement since it had a curved dome-like head that could not get stuck on obstacles. Finally, we also do not include the distance sensor on the head of the robot, as it unused in all of the controllers.

3.2.1 Physical modeling

The simulated model was created using XACRO [22], an XML macro language. At first, we used direct CAD models of the snake robot from SolidWorks exported as stereolithography mesh files and imported directly into Gazebo using XACRO. However, we found that the simulation ran extremely slow due to the unnecessary detail. We solve this by reducing the geometry of the mesh using Meshlab [8].

Figure 3.1: The Neurally Adaptive Graph Agent (NAGA) robot

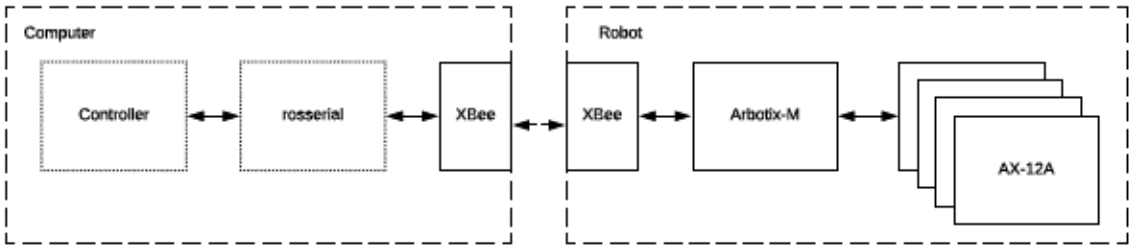
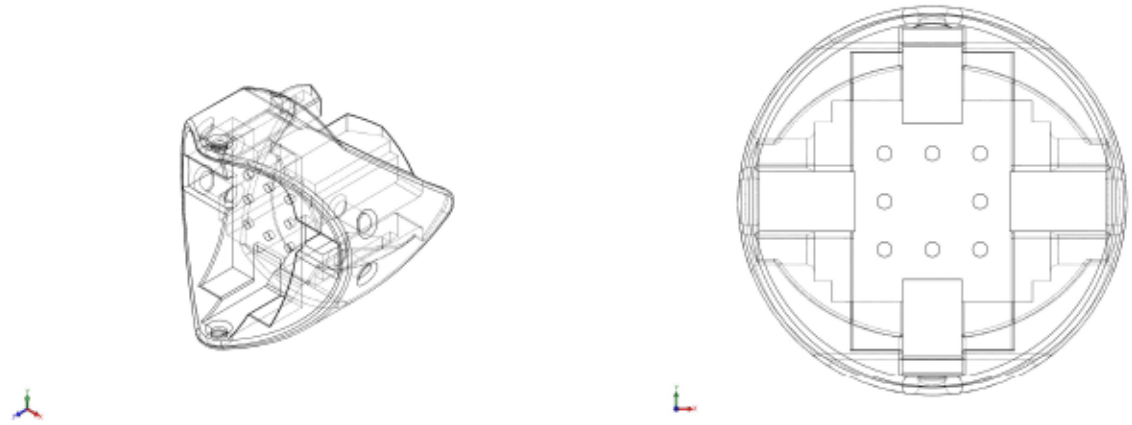
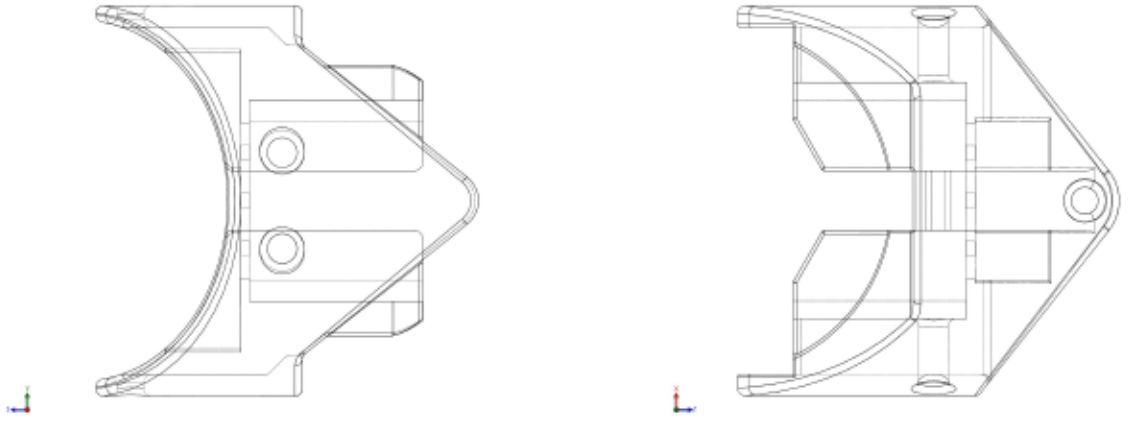


Figure 3.2: Information flow in the physical robot. Solid lines represent hardware nodes, while small dashes represent software nodes.



(a) Isomorphic view

(b) Front view



(c) Side view

(d) Top view

Figure 3.3: Design of a single shell segment. (c) shows holes on the side provided screw access for assembly, (b) shows windows through the front for wire access.

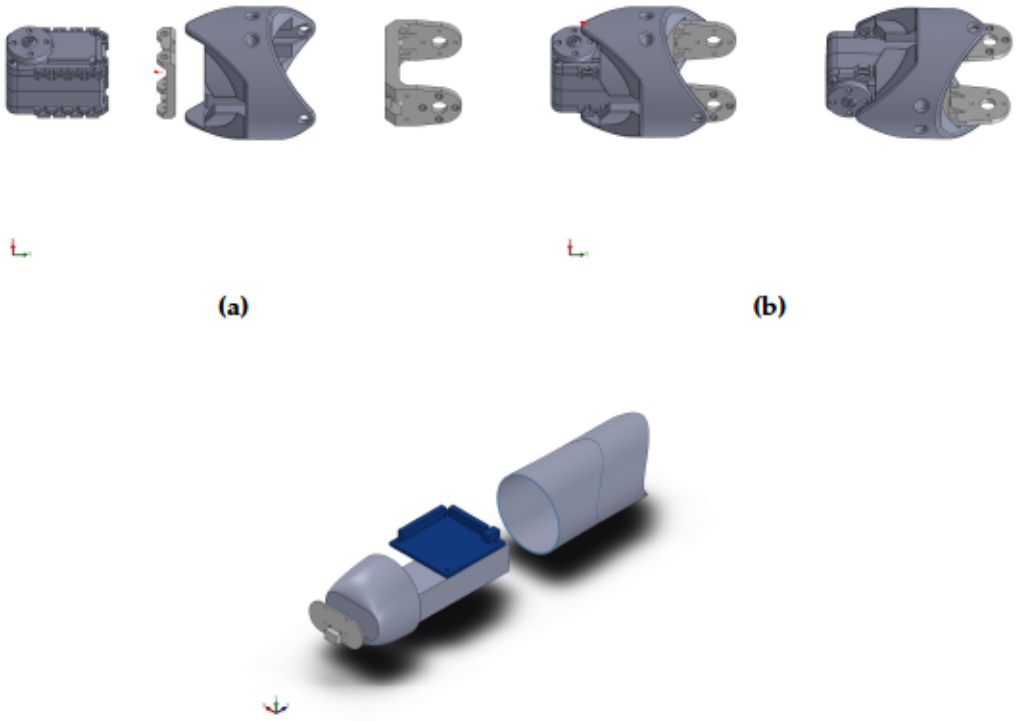


Figure 3.4: Assembly for two links of the robot. (a) shows all parts (not including screws and wires) for a single link. (b) shows two links chained together serially, using the planar configuration. (c) Shows the configuration of the head, including an unused distance sensor (front), the battery, and the control board (blue).

Because the links are geometrically complex, they potentially have interesting inertial properties. We originally found the inertial matrix via Meshlab, but this resulted in too small values for simulation using the Open Dynamics Engine (ODE), the default physics engine of Gazebo. To solve this problem, we model each link as a cylinder. Then the inertial matrix is calculated from

$$I = \begin{bmatrix} i_{xx} & 0 & 0 \\ 0 & i_{yy} & 0 \\ 0 & 0 & i_{zz} \end{bmatrix} \tag{3.1}$$

where

$$i_{zz} = \frac{1}{2}mr^2$$

$$i_{yy} = i_{xx} = \frac{1}{12}m(3r^2 + h^2)$$

Where m,r,h are the measured mass, radius and height of the links after physical construction. Each of the robot links has a collision mesh that is also approximated as a cylinder. This is done for two reasons: one, to simplify the load on the physics engine, and two, to reduce gaps in the robot links from getting caught on the pegs. The coefficients of the cylinder are set to be 0.3 for along the length, and 0.5 across. Practically, this does not affect the movement of the robot when not colliding with obstacles.

3.3 Environments

Each environment is composed of a circular array of pegs, similar to the environment testing in Travers et al. [62]. Parameterized environments are generated via a Python script. Modifiable parameters include the thickness and height of the pegs and their positional density. Randomness can be added to all parameters and as we increase the randomness

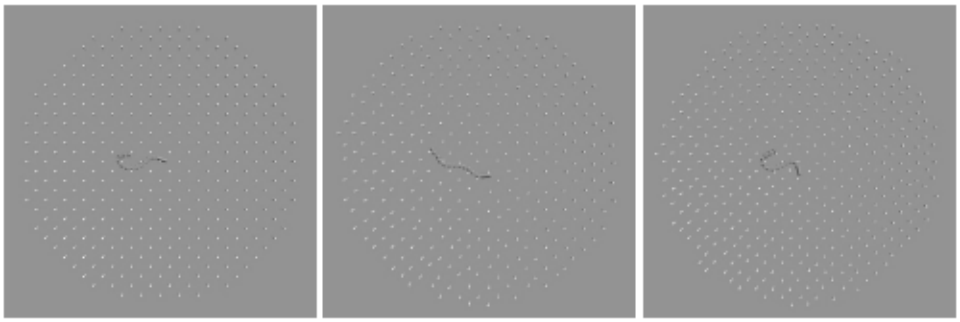
all of these parameters, we can start to approach a real world environment. For example, a very dense peg world with random heights, thicknesses and positions can resemble rocky terrain. We chose two parameters to vary, and generated three "worlds". These are shown in Table 3.1. Although these worlds do not represent real-world conditions, we start with the easiest world and attempt to make it more complex. We hope to compare robustness across environments by tuning the controllers on a single environment and analyzing how performance is affected by a more complex world.

World #	σ_r	ρ (pegs/m^2)
1	0	80
2	0.5	80
3	0.5	100

Table 3.1: Environments ("worlds") of increasing complexity. σ_r is the randomness factor, ρ is the density

All environments have pegs with 0.01m radius and are 0.2m tall. The pegs are generated in a grid with a given density, and then all pegs outside the circle with radius 2m are pruned. The number of total pegs is approximately $n = 2\rho\pi r^2$, so $\sqrt{n}$ rows and columns are generated. Next, pegs are pruned at every other position, alternating between rows such that no pegs in adjacent rows are adjacent to each other. This means that the practical density is approximately half of the specified density. This is done in an attempt to remove "channels" in which the robot can get stuck in. However, this does not eliminate them, and is a cause of some interesting behavior (see Subsection 4.4.1).

When introducing randomness, we do not completely randomize the placement of the pegs since this results in non-homogenous local density, which forms local pockets of empty space that all controllers can get stuck in. Since it is outside the scope of this work to analyze this effect, we choose to first generate a locally-homogeneous world and apply randomness to the peg positions, which alleviates this problem. Randomness is added to the pegs' coordinates drawing from a Gaussian distribution $N\left(0, \left(\sigma_r/\sqrt{n}\right)^2\right)$.



(a) (b) (c)

Figure 3.5: Screenshots of the gazebo environment, with parameters corresponding to Table 3.1

Screenshots from the worlds in Gazebo can be found in Figure 3.5.

3.4 Controllers

3.4.1 Shape-based compliant controller

The shape-based compliant controller[62][61] is implemented based on the equations discussed in Subsection 2.2.2.2. A controller with three windows is used. Following the advice of the authors [62], two extra windows that are centered outside of the linkages of the robot are used to smoothly transition the angle values during the window re-circulation. First, testing was done outside of the Gazebo simulation to verify that the trajectory generated by the feed-forward portion was correct. Then, a controller was designed in the Gazebo simulation, where torque was measured from the simulated joints and used as an input into the DMP defined in Equation 2.6.

3.4.2 Oscillator-based CPG

The oscillator based CPG is implemented based on Wang, Gao, and Zhao [64], following the details in Subsection 2.2.3. 11 oscillators are used, one mapped to each joint. In order to add compliance, as noted in the comparisons of Travers et al. [62], the exact same DMP

that was used the shape controller is used to modulate the parameter A of the oscillators. Since in the original formulation, A was a single parameter that all the oscillators shared, it was pulled out to be an individual parameter that could be tuned. Therefore, the equations are modified from Equation 2.8 to be:

$$\begin{pmatrix} \dot{u_i} \\ \dot{v_i} \end{pmatrix} = \begin{pmatrix} -\omega u_i - \lambda(\frac{u_i^2+v^2}{A_i^2} - \sigma)v_i \\ \omega v_i - \lambda(\frac{u_i^2+v^2}{A_i^2} - \sigma)u_i \end{pmatrix} + s_i(t) \tag{3.2}$$

for each oscillator i.

Here, since the DMP operates on the shape function $H(A_i) = \begin{pmatrix} \dot{u_i} \\ \dot{v_i} \end{pmatrix}$, the inputs and outputs of the DMP do not need to mapped to the joints via the joint-shape Jacobian (or, rather, the joint-shape Jacobian is the identity).

3.4.3 Neuron-based SCPG

In this work, we use an adaptive leaky-integrate and fire model, with neuron dynamics

$$\tau_V \frac{dV(t)}{dt} = R(I(t) - n(t)) - V(t) \tag{3.3}$$

$$\tau_n \frac{dn(t)}{dt} = -n(t) \tag{3.4}$$

$$I_i(t) = b_i + J_i(t) + \alpha_i \sum_{j \in N} w_{ji}(A * G_j)(t) \tag{3.5}$$

Where $V(t)$ and $I(t)$ are the membrane potential and input current at time t, $n(t)$ is the adaptation parameter, R is the membrane resistance, and τ_V and τ_n are the time constants for membrane voltage and adaptation, respectively. Input current $I(t)$ is calculated using synaptic connections where i,j, are neurons in the circuit N, b_i is the bias current into i,α_i is

the neuron gain, w_{ij} is the weight between j and i, $G_j(t)$ is the spike train of j. $A(t)$ is the synaptic filter. $J(t)$ is an external current that can be used to provide modulatory signals.

When $V(t) > V_{th}$, the neuron spikes, and $V(t)$ is set to V_{reset} and $n(t)$ is incremented by n_{inc}. Nengo abstracts the units of their simulations such that $V_{th} = 1$ and $V_{reset} = 0$.

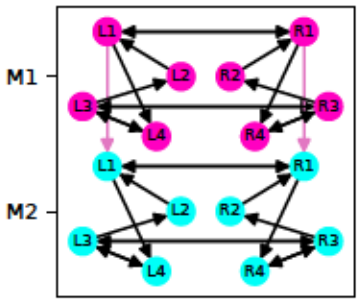
We chose this model since it can produce a "bursting" spike response when a constant current is applied. Since it is a single-compartment model, it is relatively easy to simulate but is not particularly bio-fidelic.

The bias and alpha current is constant for each of the neurons, such that $b_i = \alpha_i = 5$. These numbers are selected such that the neurons begin to fire with no input. Along with negative weights, this means that the entire network is competitive. This is similar to the choice of a tonic excitation in Lewis, Tenore, and Etienne-Cummings [32] and Espinal et al. [16], except the tonic current is provided internally to the network in the form of bias. An alternative, equivalent method would be to introduce this bias current along with modulatory signals for compliance. Furthermore, we could have included self-recurrent connections like in Aljalbout et al. [3], but it was determined that self-recurrent connections add an additional layer of complexity and tuning, particularly because they require a tunable weight that will be inherently different from the other neurons in the network. This could make it incompatible with our other assumptions made in Subsection 3.4.3.1.

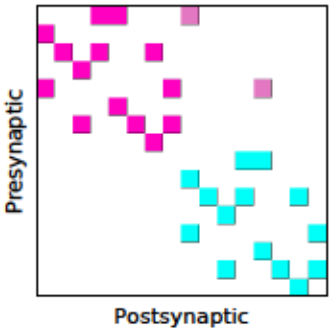
In our implementation, the synaptic filter is an alpha filter with impulse response

$$A(t) = \frac{t}{\tau_A^2} e^{\frac{-t}{\tau_A}} \tag{3.6}$$

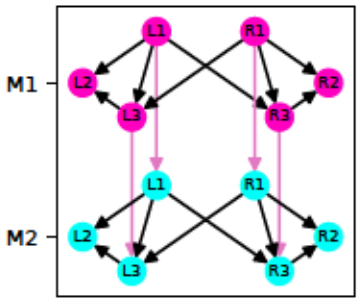
For time constant τ_A. The transfer function is given below



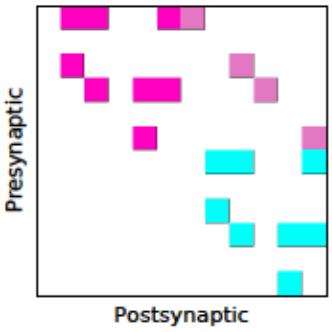
(a) Leech heartbeat SCPG



(b) Leech heartbeat adjacency matrix



(c) C. elegans SCPG



(d) C. elegans adjacency matrix

Figure 3.6: Biological examples of SCPGs. 3.6b and 3.6b shows the leech heartbeat CPG from Ambroise et al. [4] . MC1 is colored green, and MC2 is colored orange. Intermotor connections are colored pink. Note that the biological SCPG is only a single MC. 3.6c and 3.6d show the CPG in C. elegans identified by [30] [45].

$$H(s|\tau_A) = \frac{1}{\tau^2 s^2 + 2\tau s + 1} \tag{3.7}$$

Overall, the SCPG has a layout of 12 motor circuits (MCs), that are each composed of a symmetric left and right side, each containing 4 or less neurons. Each MC has the exact same layout and weights, and are connected serially to each other. Each neuron within each MC will be referenced with the form "Mn[LR]m", e.g "M1R1" means MC 1, Right neuron 1. Examples can be found in Figure 3.6.

Since an initial state where the left and right side is a stable fixed point, the initial state is set such that the voltage for the left side of the network is set to be 1, while the right side is set to be 0. These are chosen to be constants in order to improve reproducability.

The motor output is calculated as follows:

$$\tau \frac{dm_i}{dt} = -m_i(t) + A_m * G^i_{L1}(t) - A_m * G^i_{R1}(t) \tag{3.8}$$

$$\theta_i = A_n m_i \tag{3.9}$$

where $m_i(t)$ is the motor circuit output at time t and $G^i_n(t)$ is the activation of the nth neuron in the ith motor circuit. A_m is a alpha filter, described in Equation 3.6 and τ is a time constant. θ_i is the desired value fed into the motor PID controllers and A_n is the scaling amplitude into the range of the physical motors. Actual values used can be found in Table 3.3.

3.4.3.1 Exploration of the motif-space

At first, we attempted to tune the leech heartbeat SCPG from [4], chained together as in Figure 3.6a for our initial simulations. However, we wanted to see if there was another structure out there that had better performance and stability. Therefore, we needed to search through possible connectivity graphs that would produce oscillations and could provide sufficient output for snake robot locomotion.

This is done in two steps. First, we examine possible single oscillators and simulate them. Next, we select the best ones based on how well they oscillate and then attempt to chain copies together. The evaluation procedure is then done again to select the best SCPG structure.

Since the search space for even a single oscillator is very large, a number of rules and assumptions are made to reduce the complexity of the problem. It is assumed that:

1. (Binary synapses) Within an MC, neurons have the same inhibitory synapse and weight value. Across MC, they have an excitatory synapse with a smaller weight. Therefore, connections are binary (either they exist, or do not).

2. (Forward connections) Connections between MC are adjacent, forward only, and only connected to the same index i.e M2R1 can only connect to M3R1 and not M1R1, M4R1, M3R2, etc.

3. (Symmetric connectivity) The left and the right of each MC is symmetric, and L-R connectivity is the same as R-L connectivity.

4. (No autapses) No neuron connects to itself.

5. (Sparse connectivity) Connectivity between the left and right side of each MC is "sparse" (having two or less connections).

6. (No isomorphisms) No graphs that are isomorphic on R1/L1 are allowed to pass to the next step.

Most of these assumptions were inspired from connectivity data found in biology, notably in the leech heart [4] and the C. elegans locomotion CPG [30] [45], but most of the assumptions are not bio-fidelic. For example, in many biological CPGs, autapses do exist, and connections are certainly not binary. However, these assumptions are made to reduce the complexity of the search.

Table 3.2 shows the reduction of the search space for graphs of size 8. The search space starts out infinite, since any weight can have an infinite number of values.

The search is done by generating adjacency matrices for the connectivity of one motor, simulating it, and fitting the output m_0 from the motor (see Equation 3.8) to a sine wave (below) using a non-linear least squares method.

Assumption	Parameters eliminated	Resulting search space
Binary synapses	∞	2^{128}
Forward connections	2^{60}	2^{68}
Symmetric connectivity	2^{32}	2^{36}
No autapses	2^4	2^{32}
Sparse connectivity	$2^{16} - \binom{16}{2}$	$\frac{2^{32}}{2^{16}-\binom{16}{2}}$

Table 3.2: Reduction of the search space. Note: isomorphism exclusion is not included since it takes place between steps and is non-trivial to calculate the parameters eliminated

$$\hat{y}(t) = \alpha \sin(\omega t + \rho) + \phi \tag{3.10}$$

Where α,ω,ρ, and ϕ are the parameters that are solved for and represent the amplitude, frequency, phase, time-shift, and offset, respectively. A sine wave is chosen since it is a good approximation of oscillatory behavior.

Next, an objective function for fitness is calculated as:

$$e_{s1}(y, \hat{y}, \alpha, \gamma) = |\gamma| + \frac{1}{\alpha}\sum_{i=0}^{N}(y - \hat{y})^2 \tag{3.11}$$

Top 100 performers are selected, and then inter-MC connection matrices are generated. Each top performer is connected to a copy of itself using a generated inter-motor matrix. These networks are then simulated, and the outputs for each MC are fit to the same curve as in Equation 3.10. The fitness of the entire circuit is then evaluated using the below error function.

$$e_{s2}(\rho_1, \rho_2, \hat{\rho}, e_1, e_2) = \sigma_1|(|\rho_1 - \rho_2| \mod 2\pi) - \hat{\rho}| + \sigma_2(e_1(y_1...) + e_1(y_1...) \tag{3.12}$$

Here, ρ_1, ρ_2,e_1,e_2, are the phase and step 1 fitness (calculated from 3.11) for the first and

second MC. $\hat{\rho}$ is the desired phase. σ_1, σ_2 are scaling parameters that allowed the control of which aspect of the fitness was more important. σ_1 prioritizes finding a desired phase difference $\hat{\rho}$, while σ_2 finds circuits that fit better to the sine waves.

Here, we prioritize the phase difference, so we set $\sigma_1 = 10, \sigma_2 = 1$. Our desired phase is $\hat{\rho} = \pi/6$, which corresponds to one full wave down the length of the twelve link robot.

3.4.3.2 Incorporation of compliance

In essence, the previous models of compliance [62] reduce the amplitude of the oscillation proportionally to the torque measured at the motor. The DMP implementation is essentially a second-order system around this operation. It follows that to introduce compliance into the SCPG, we must do something similar. In biology, CPGs modulate their output via the introduction of currents into the neurons involved in the CPG. Typically, this is done by a diffuse release of neurotransmitters into a targeted area. Here, we can do something similar: by introducing a modulatory current into neurons of the SCPG, we can make it utilize its environment. The general approach in introducing external currents is to alter the firing rate in each neuron such that the amplitude of the oscillation is changed based on the external torque.

In particular, we try a few different methods of introducing torque into the circuit. The first is to introduce a proportional current into the L1 and R1 neurons.

$$J_L(t) = J_R(t) = \lambda_\tau \tau_{ext}(t) \tag{3.13}$$

Where J_L, J_R are the modulatory currents into all the L1 and R1 neurons for the MC, λ_τ was a scaling factor that scaled the torque into the motor into the circuit, and τ_{ext} is the torques measured at each motor. In this way, we do not have to apply the Jacobian to

convert from motor space into motor circuit space.

However, due to the neuron dynamics, a constant current might be too low to alter the firing rates at the timescales necessary. Therefore, we also try modifying the current such that depending on the polarity of the motor derivative, either the left or right neuron would be excited, while the contralateral neuron would be suppressed. This can be written as follows:

$$\begin{bmatrix} J_{Li}(t) \\ J_{Ri}(t) \end{bmatrix} = \left\{ \begin{array}{l} \begin{bmatrix} h(t|\sigma) * \lambda_\tau \tau_{ext}(t) \\ h'(t|\sigma') * \lambda_\tau \tau_{ext}(t) \end{bmatrix}, \frac{\delta m(t)}{\delta t} > 0 \\ \begin{bmatrix} h'(t|\sigma') * \lambda_\tau \tau_{ext}(t) \\ h(t|\sigma) * \lambda_\tau \tau_{ext}(t) \end{bmatrix}, \frac{\delta m(t)}{\delta t} <= 0 \end{array} \right\} \tag{3.14}$$

For synaptic filters h, h' with parameters σ, σ'.

However, we only want this contralateral excitation/inhibition for a short while, or else the mean of the oscillation would drift away from zero, which would result in the robot turning. Therefore, we need filter that would produce large positive or negative currents with the initial introduction of the torque, but quickly return to the nominal torque value. Basing it off the synaptic filter in Equation 3.6, we choose a transfer function that had a 1st order term in the numerator.

$$H(s|\alpha, \tau) = \frac{\alpha s + 1}{\tau^2 s^2 + 2\tau s + 1} \tag{3.15}$$

Beyond compliance, this modulatory input allows us to do a few more things by choosing where we inject current and for how long. For example, we can change the mean of the oscillation center (essentially adding an offset similar to the parameter κ in the serpenoid equation 2.1). This can result in the robot turning. Furthermore, we can move this offset through the various oscillators, giving us control of a shape that moves along

the robot body. However, an analysis of these types of control is outside the scope of this work.

3.5 Comparison

We compare the different controllers across three different environments with increasing complexity. Each controller is run for 10 trials. The position of all the links, the torques measured at each joint, and the commanded joint positions are all recorded. The robot is spawned at 1 meter above the pegs, and dropped. This is done to give the robot a starting position within the pegs. Damage is not simulated. If the first or last link exits the peg array or the controller surpasses 10,000 iterations, it is re-spawned in the original starting position, and the trial is iterated. 10,000 iterations is chosen because when a controller got to that point, typically it means it is stuck. The controller is not reset when the robot is re-spawned, and it keeps its joint positions.

3.5.1 Metrics

The distance/period metric is measured by calculating the positive radial distance traveled of the head link and dividing by number of positive peaks in the commanded position over the same time period.

The periods/trial metric is calculated by counting the number of peaks until the position is reset. This represents how fast a given controller leaves the environment, which is correlated with the directness of the snake path. Since the robot is not given any turning signals, an ideal controller would immediately navigate to the edge without any turns (besides navigating around obstacles). The failure rate is calculated as the number of times that the position is reset due to surpassing 10,000 iterations over the total number of trials.

Specifically, wall time is not used as a metric, since each controller time step is different

depending on the set frequency, as well as the execution time of the program, which can vary for a number of reasons.

It follows that a good controller would have a high distance/cycle, a low cycles/trial and a zero failure rate. We compare the means and standard deviations of each controller's cycles/trial and distance/cycle using a Student's T-test to produce a p-value with respect to the null hypothesis that the cross-controller metrics were the same.

We also use survival analysis[39] to determine controller fitness in each environment. In general, a survival function is defined as

$$S(t) := Pr(T > t) \tag{3.16}$$

and denotes the probability of an event occurring after time t. A hazard function is defined as

$$h(t) := \lim_{\Delta t \to 0} \frac{P(t \leq T < t + \Delta t | T \geq t)}{\Delta T} \geq 0 \tag{3.17}$$

which gives the approximate probability that an event occurs in the interval $[t; t + \Delta t]$. Finally, the cumulative hazard function is defined as

$$H(t) = \int_0^t h(u)du \tag{3.18}$$

which gives the approximate rate of an event occurring at time t.

The advantage of a survival analysis versus traditional regression is that it can take into account partial data observations. Here, we want to analyze the number of cycles it takes

to complete a trial, so our "event" is a trial completion. A partial observation (i.e right censured data) is defined as the trial timing out after 10,000 iterations. Since we know that the robot did *not* exit before 10,000 iterations, we can use this analysis to better improve our estimate of the performance. In order to complete this analysis, we use the python package scikit-survival [47] using the Kaplan-Meier estimator[29] to estimate the survival function and the Nelson-Aalen estimator[43] to estimate the cumulative hazard function. We then compared survival distributions using a K-sample Log-Rank comparison [18], which produces a two sided chi-squared p-value with respect to the null hypothesis that two survival distributions are equal.

3.6 Code layout and implementation

The code is divided into separate packages. A ROS metapackage named "snakebot" contains all the code for running simulations and communicating with the physical robot. Within this metapackage, the "worm" package is for the simulation code, including launch files, mesh files, and a XACRO description of the robot and environment generation. The "snakebot" package contains code for interfacing with the physical robot using and "rosserial" for communication through the XBee (see 3.2). The "joint2dynamixel" package is a small package that enabled communication with the Dynamixel servos via the Dynamixel SDK. A separate ROS package outside of the "snakebot" metapackage named "cpg_control" contains all the controllers for the snake robot. It is designed generally enough to work with any ROS-compatible snake robot. Beyond the ROS packages, the "motif_search" package provides code for the search through the possible motor motifs.

Controllers are implemented in Python 2.7, and simulations are completed in ROS Kinetic with Ubuntu 16.04 and Gazebo 7. Nengo [6] is used for spiking neuron simulations.

3.7 Identified parameters

The parameters we use are shown in Table 3.3. As stated previously, these are hand-tuned in the first environment, and then applied towards the second and third environments. Additional parameters for the Hopf controller were taken directly from [64].

Parameter	Value
A_n	0.04
τ	0.05
λ_τ	1
ω	1

(a) SCPG parameters

Parameter	Value
A_n	1
λ	5
σ	1
ω	2

(b) Hopf parameters

Parameter	Value
A_n	0.7
η	0.75
ω	2
ϕ	0
ψ	0.4

(c) Shape parameters

Parameter	Value
M	0.001
B	0.5
K	1

(d) DMP parameters

Table 3.3

Chapter 4

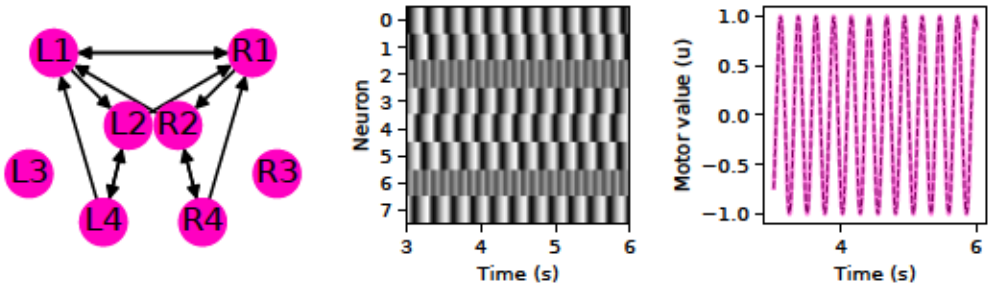
Results

In order to show a the validity of SCPGs for snake robot control, we examine the difference between the SCPG controller that we design and the state of the art snake robot control methods described in Chapter 1. However, as we note in Chapter 1, the tuning of SCPGs is a significant endeavor and several publications have proposed methods to approximate the best parameters. We propose another approach in Chapter 3, where instead of optimizing a continuous space of multiple weights, filter values and neuron properties, we fix that set of parameters and find an optimal structure for the SCPG. We then do our comparison versus state of the art in increasingly difficult environments to show that the SCPG has an increased robustness to environmental complexity. Below, we present our results from the SCPG structure search and simulations from the three environments.

4.1 Motif search

The structure of a recurrent SNN is crucial for its dynamics, such as oscillations and fixed points. In order to find a good structure for the SCPG, we first conduct a search across possible structures ("motifs") for a single oscillating motor circuit (MC), and then across two MCs chained together with the best motifs from the first step.

The motif search is run in two steps as described in Chapter 3.4.3.1. The first step

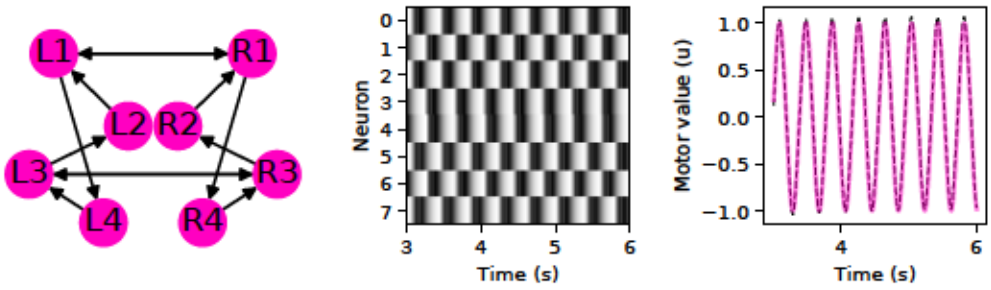


(a) Structure (b) Spikes (c) Motor output

Figure 4.1: Output of the top performer in motif search step one. a shows the structure. b shows the filtered spikes into the motor node. c shows the motor node output. The dashed line is the output from the motor node, while the magenta line is the fit sine wave from Equation 3.10

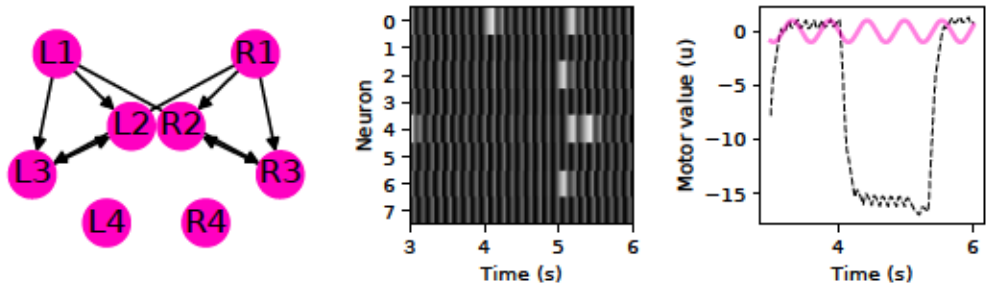
generates 107,848 motifs. The top performing motif is shown in Figure 4.1. Motifs where the output could not be fit well to a sine wave are excluded. Other isomorphs of the top motif are also ranked with the same fitness, but were discarded before moving to the next step.

In total, 77,420 graphs are eligible for step 2, but only the top 100 are chosen to iterate on. The output from the biological leech heart SCPG is shown in Figure 4.2 and ranks 1,434 in the first step. Due to its high rank, it is excluded from the next step, despite its fitness function being only slightly higher. For comparison sake, we include an example of the outputs of a chained leech heart (from Figure 3.6a) in Supplementary Figure A.12, which shows that the chained leech heart is not viable for movement. We notice that the first oscillator works well, but the second chained oscillator does not oscillate at nearly the same amplitude/frequency. Including motifs with rankings up to the leech heart would have increased the search space by a factor of 14. The C. elegans CPG is easily excluded with a rank of 43,950 (see Figure 4.3). A plot of fitness function vs rank is shown in Figure 4.4. In order to accurately calculate the fitness function, graphs that produce a fit wave with extremely low amplitudes (less than 0.01) are excluded.



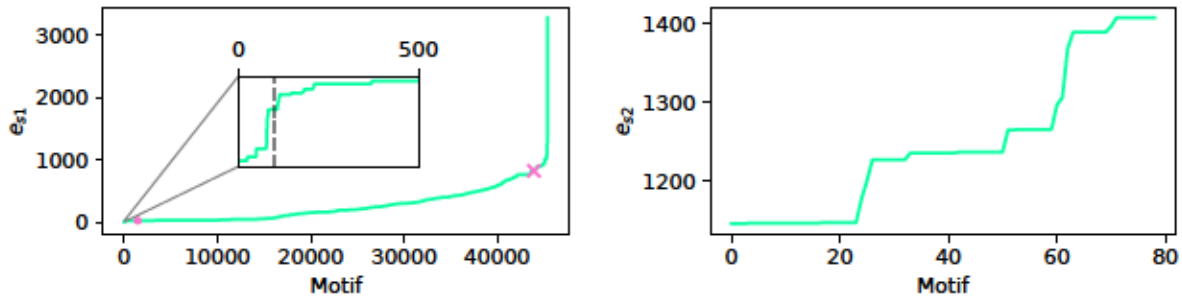
(a) Structure **(b)** Spikes **(c)** Motor output

Figure 4.2: Output of the leech heart SCPG in motif search step one. Its fitness ranked 1,434.



(a) Structure **(b)** Spikes **(c)** Motor output

Figure 4.3: Output of the C. elegans SCPG in motif search step one. Its fitness ranked 43,950.



(a) Step one **(b)** Step two

Figure 4.4: Sorted fitness from steps one and two in the motif search. (a) shows that step one produced many viable motifs, but we sub-selected the top 100 (the black dashed line). The magenta dot and "x" show the fitness of the leech heart and C. elegans CPGs, respectively. (b) shows that the first few motifs of step two were the only viable graphs. We used the top performer for our calculations.

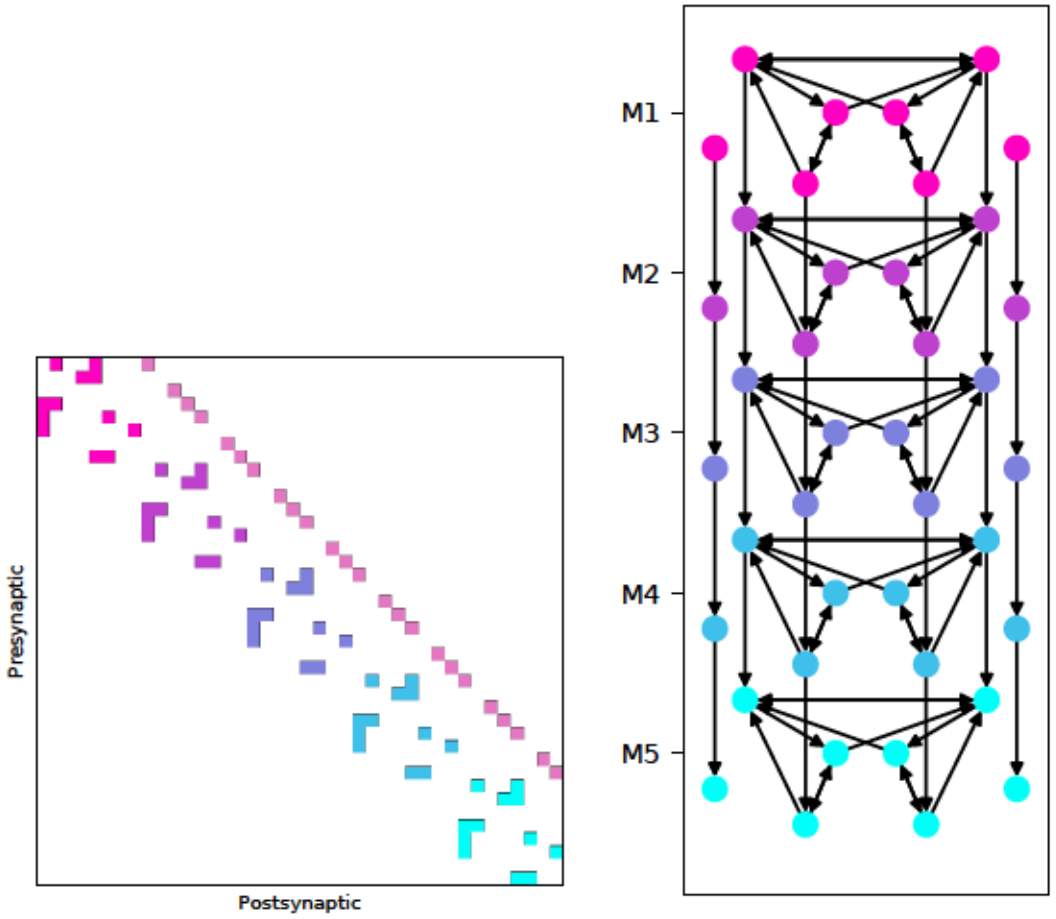
Figure 4.4b shows the top performer of the second motif search step. Notice the phase difference between M1 and M2 is not quite $\frac{\pi}{6}$, but it is the closest value. The next ranking graphs all have phase of $\frac{\pi}{2}$. The sorted fitness values for this step can be found in Figure 4.4b. The final best performer can be found in Figure 4.5. The torque-free outputs of the first five MC for the top perfomer are shown in Figure 4.6. The output of each MC is colored to match the plots in the output graph. The solid magenta line corresponds to the first MC, while the other four are slightly transparent. Note that the amplitudes are not the same, and gradually increase over MC number. This also can be noted in Figure 4.7b. This effect is most likely caused by the excitatory connections between the MCs, which causes the firing rate to be increased (much like a positive compliance input, which causes the oscillation amplitude to increase). It could potentially be solved by a introducing a slight, decreasing bias current as a function of MC number.

4.2 Compliance

In order for the snake to properly conform to its environment, it must utilize some sort of sensor information to understand its surroundings and take inputs from the world. In our case, we utilize our torque sensor measurements at the motors to introduce a external current into our SCPG neurons. The goal is to modify the amplitude of the motor oscillation by changing the number of spikes from L1 and R1 in each MC. We try a few different methods. The first is to make the current proportional to the torque.

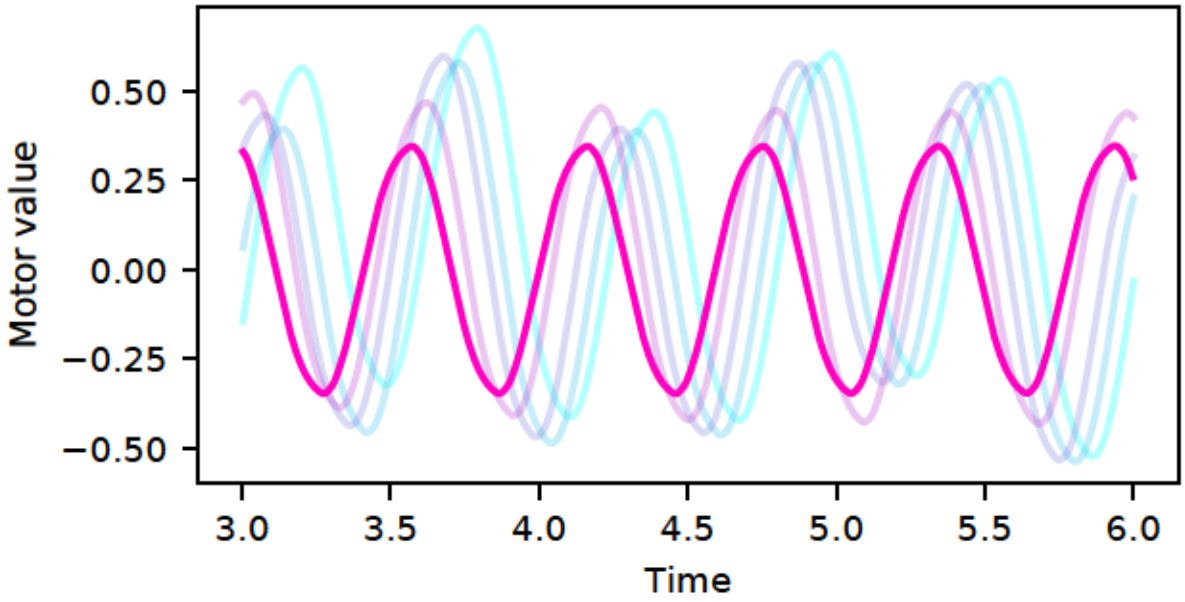
While this is successful in reducing the amplitude of the oscillations, this is not sufficient due to the slower frequency of the oscillation compared to the rate at which input inhibitory current changes the firing rate of the oscillations. This can be seen in Figure 4.7.

Here, an inhibitory bias current of magnitude 1 is introduced at t=4.5 to the first oscillator, but we do not see the amplitude of the oscillation decrease until a half second

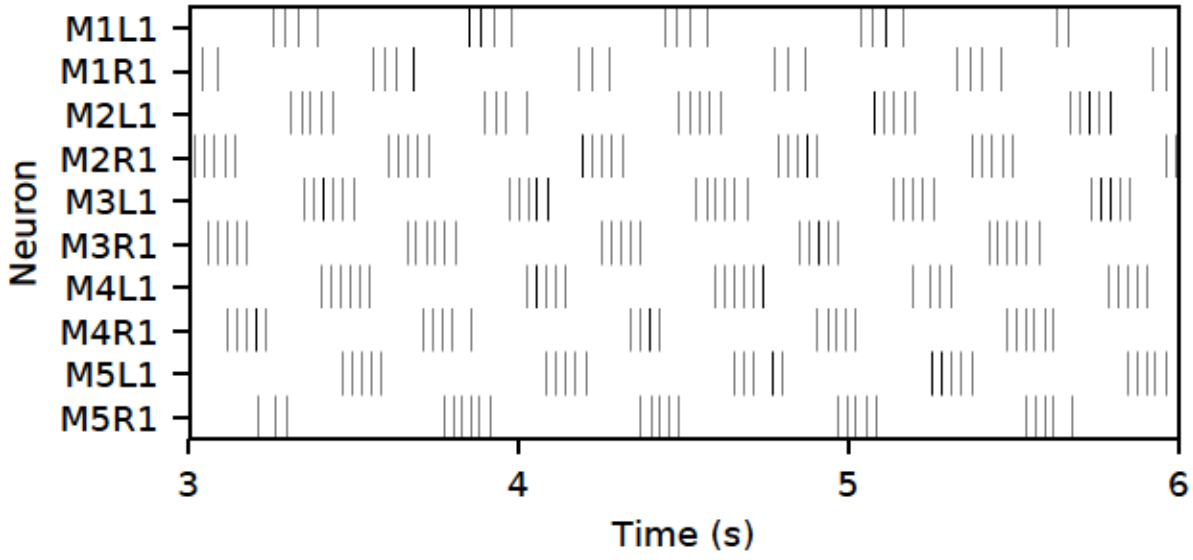


(a) Best performer adjacency matrix **(b)** Best performer graph matrix

Figure 4.5: Full graph and adjacency matrix of the best performer. First five MC are colored.

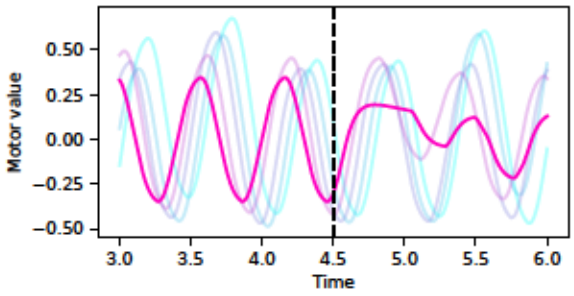


(a)

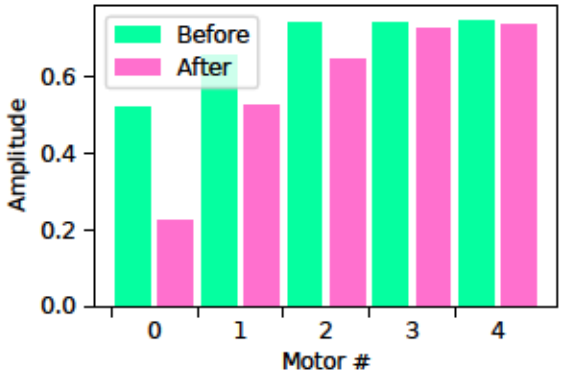


(b)

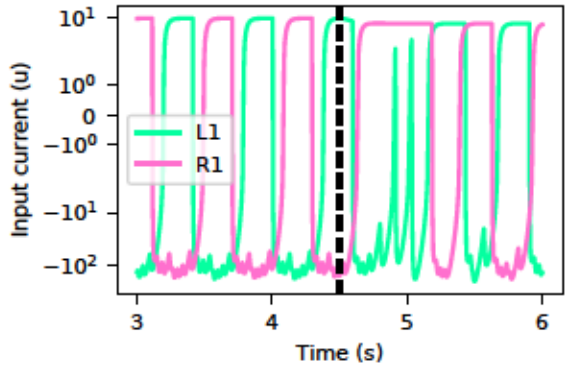
Figure 4.6: (a) Torque-free output from the first 5 MC of the best performer in Figure 4.5. The colors here match the colors on the graph and adjacency matrix. The first MC (green) is highlighted to show individual MC performance. (b) shows the spikes of the first 5 modules



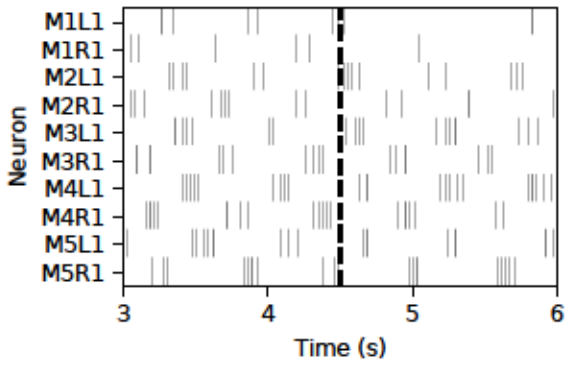
(a) Output of the first five MC



(b) Amplitude of the first five MC, before and after an inhibitory current is introduced



(c) Input into M1L1 and M2L2



(d) Spikes of the first 5 MC

Figure 4.7: Introducing an inhibitory current of -0.4 into the into M1R1 and M1L1 at t=4.5 results in a smaller amplitude

later. In robotic context, this would mean the robot would not respond to an obstacle until the body wave was already oscillating in the opposite direction. To analyze this effect, we looked at the input currents into the R1 and L1 neurons, and found that the inhibitory current was not enough to stop the current firing neuron while allowing the other contralateral neuron to take over.

Next, we utilize the contralateral excitation method with a synaptic filter described in Equation 3.14. The step responses to the filter can be seen in Figure 4.8, along with outputs after applying the same torque value (0.4) to the first MC. Of note is that as the amplitude decreases, the frequency increases. Nevertheless, the selection of $\alpha = \pm 0.1, \tau = 0.05$ has dynamics similar to what we are looking for.

4.3 Simulation

To quantify the difference in robustness between our SCPG controller and the other state of the art controllers, we simulate them in environments of increasing complexity. Each controller has its parameters hand tuned in the first environment, and is evaluated in the following environments. The change in performance can be considered a measure of robustness. A controller with better robustness would perform similarly in all environments. Following Chapter 3, the controllers are run for ten trials in an environment with pegs, where each trial runs until the head or tail of the robot reaches the edge of the pegs or 10,000 controller steps have passed, whichever comes first. The density and randomness of the peg placements are increased with each sequential "world."

4.3.1 World 1

(See Figure 4.9) The first world consists of a grid with the density of 80 pegs/m^2 and no randomness. We see in Figure 4.9a that the SCPG moves faster towards the exit than

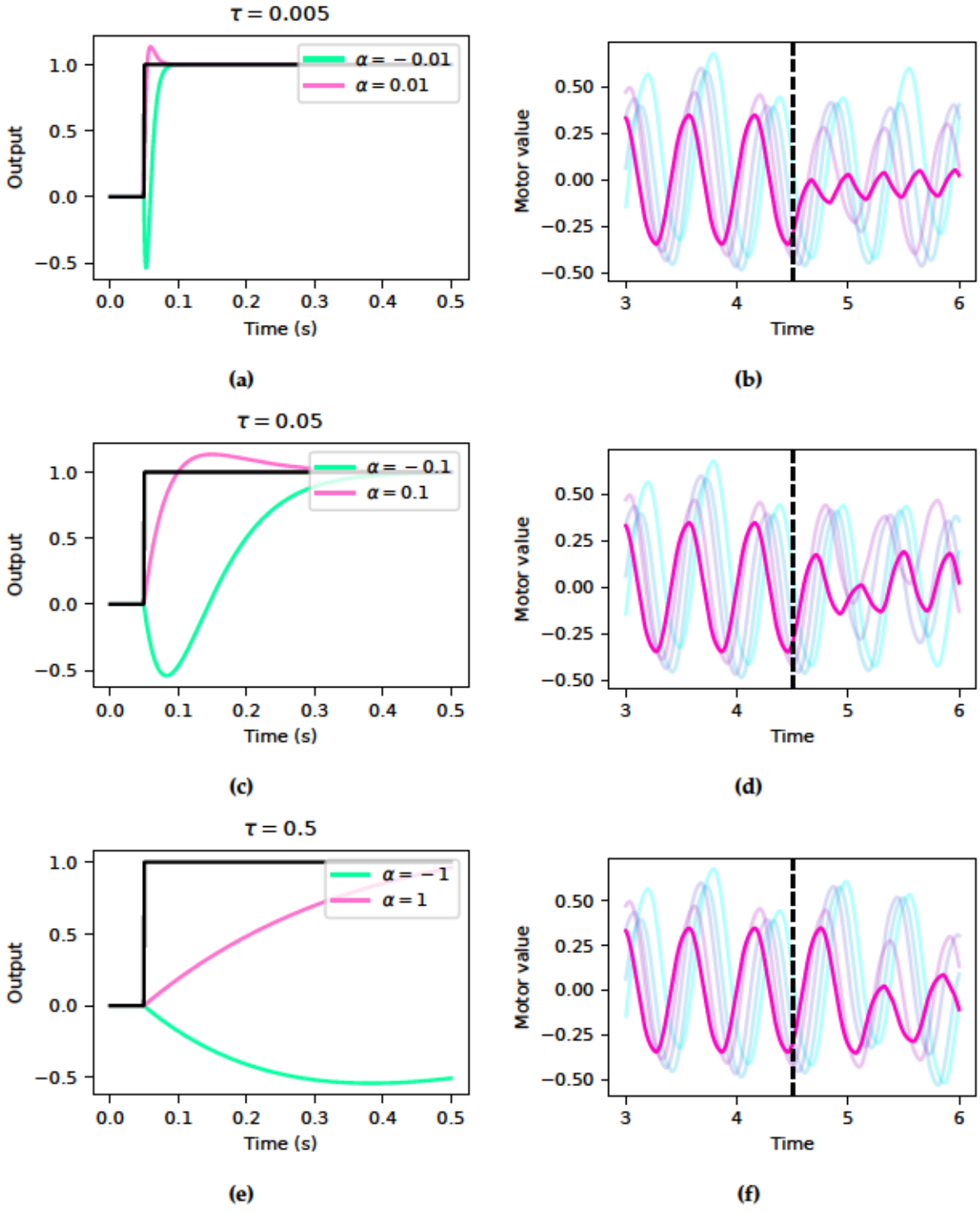


Figure 4.8: Step response to the designed synaptic filter with varying τ and α values. Outputs of the first 5 MC are shown on the right after the filter is applied. Note the different timescales. Amplitude comparison can be found in Appendix A

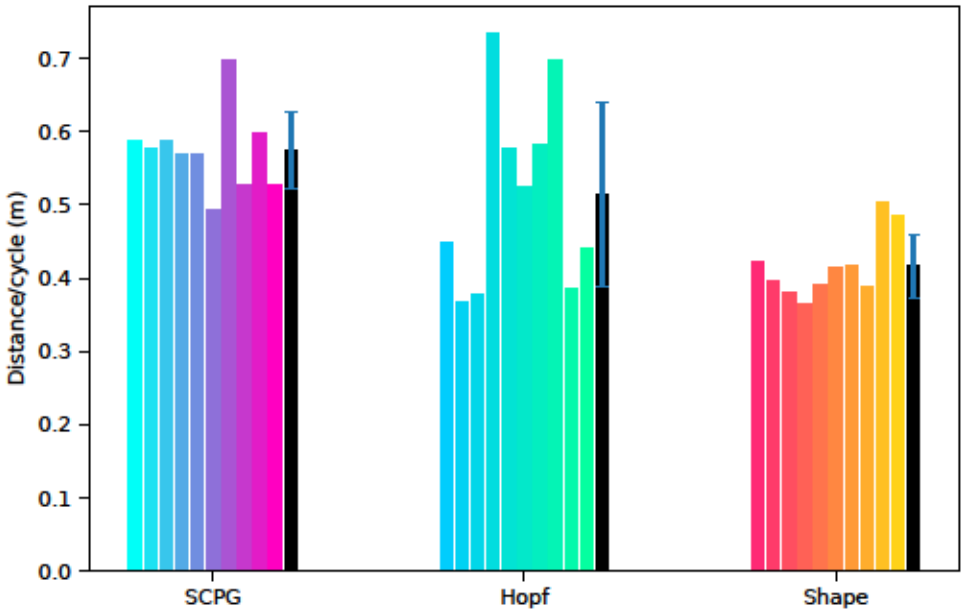
the shape controller ($p < 10^{-6}$), but results are similar for the number of cycles to exit. However, the SCPG is very consistent in the number of cycles it takes to exit, while the other two controllers vary. The distance/cycle is correlated with the cycles/exit. Of note is the Hopf controller's two best trials, which navigate out of the area in just five and six cycles. Similarly, the SCPG controller has three such trials, navigating out in 6 or 7 cycles each time. This is evidence that the tuned parameters for the Hopf and SCPG controller are well chosen.

4.3.2 World 2

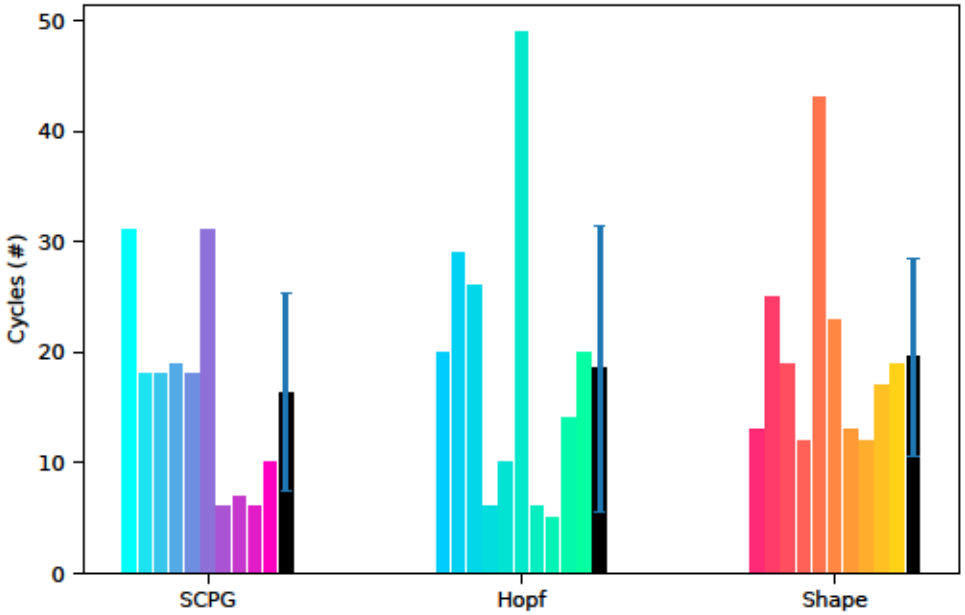
(See Figure 4.10) The second world is a grid similar to World 1, except each peg has noise added to it according to Section 3.3, with $\sigma_r = 0.5$. Here, the advantage of the SCPG is more notable, with significant maneuverability (vs Hopf $p < 0.02$, vs shape $p < 0.003$) with significantly lower cycles/exit than the shape controller ($p < 0.007$) and the Hopf controller ($p < 0.02$). It should be noted that the Hopf controller fails to exit three times, so the mean is biased towards its best trials since we do not exclude the failures from the mean and standard deviation calculation. The SCPG here also tends to do better than itself in the first world, despite using the exact same parameters and increased randomness in the peg position. The shape controller noticeably performs worse, while the Hopf controller's successful trials maintain the same performance.

4.3.3 World 3

(See Figure 4.11) This world is 25% more dense than World 2, with the same randomness in the peg positions. Here, every controller fails to exit at least once. The SCPG fails once, the shape controller fails twice, and the Hopf controller fails in seven out of ten trials.

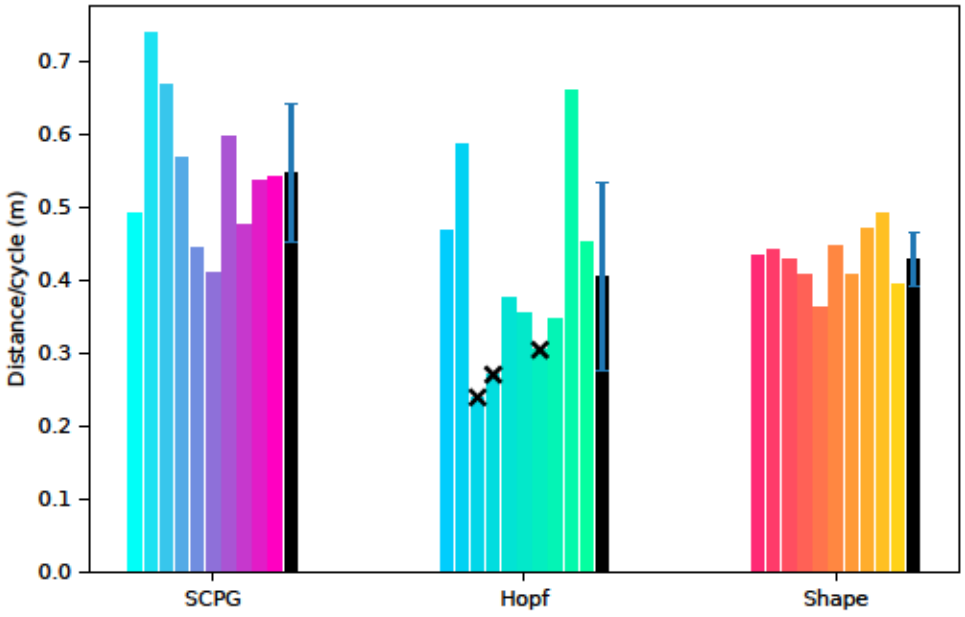


(a)

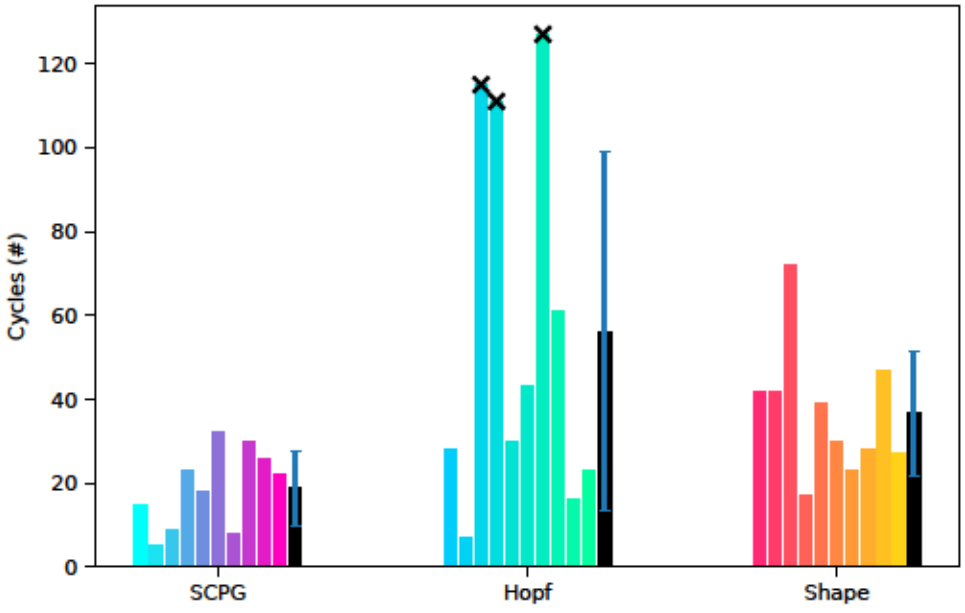


(b)

Figure 4.9: World 1 ($\rho = 80, \sigma_r = 0$) controller comparison. Each color is a different trial, matching the trial colors in Appendix A. Each trial is run until either the head or tail segment exits the peg area, or the controller is run for 10,000 timesteps, whichever comes first. (a) shows the average speed towards the edge per cycle of the controller. (b) shows the total number of cycles to reach the edge.

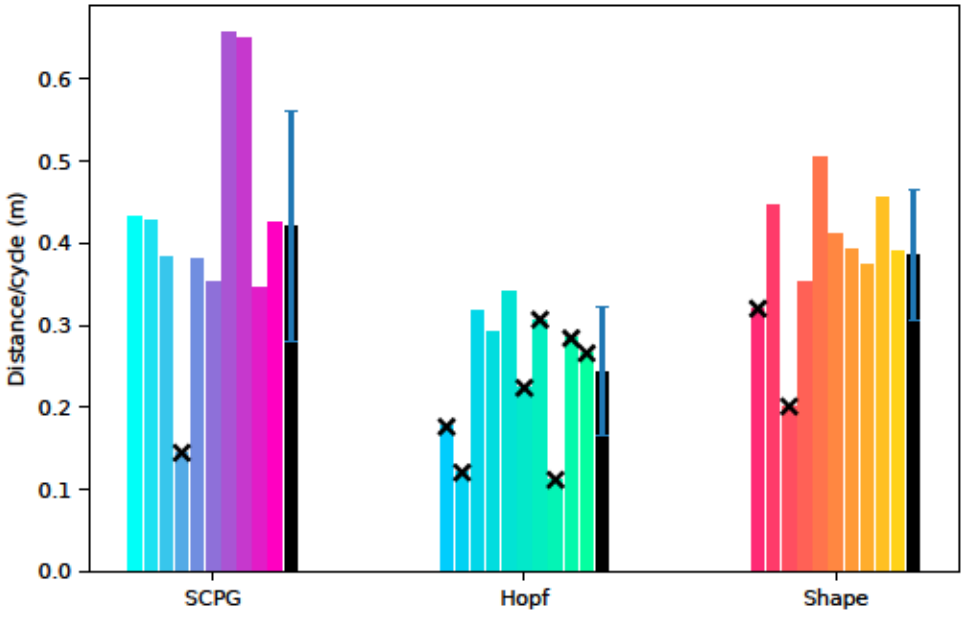


(a)

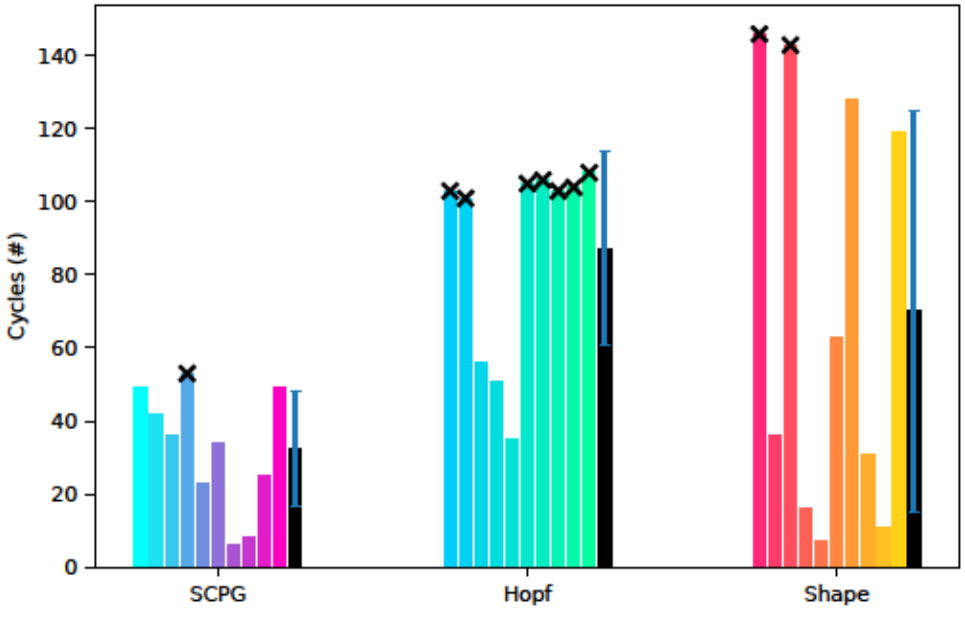


(b)

Figure 4.10: World 2 ($\rho = 80, \sigma_r = 0.5$) controller comparison. Trials with an "X" on top represent failed trials where 10,000 controller steps passed. Figure A.8 shows where the robot gets stuck.



(a)



(b)

Figure 4.11: World 3 ($\rho = 100, \sigma_r = 0.5$) controller comparison. The SCPG controller gets stuck in trial 3 (see A.9), the Hopf controller fails in every trial but 2, 3 and 4 (see A.11), and the shape controller in trials 0 and 2 (see A.10). The only statistically significant difference here is between the Hopf controller and the SCPG (for distance/peak, $p < .005$ and cycles/exit $p < 10^{-4}$

4.4 Comparison

Comparisons for each environment are calculated with a distance/period method, similar to the comparison used in [62], as well as the number cycles to exit the area. Overall, we see our SCPG implementation is on par with the state of the art controllers in our constructed worlds. In many cases, the SCPG outperforms the Hopf controller, but the advantage against the shape controller is a little more nuanced. We note that our controller failed less than the other two overall, having just one failure to exit in the last and most complex environment. Comparatively, the Hopf controller fails seven times in that environment, and the shape controller fails twice. We show a comparison of all the worlds in Figure 4.12. We note that the standard deviations of each controller is too large in many cases due to the failed trials, so we turn to survival analyses to gain a better understanding of the performance.

Our survival analysis is shown in Figure 4.13. It should be noted that the probability that the robot has exited at a given time-step is higher in our SCPG than the Hopf controller in World 2 ($p<0.02$) and World 3 ($p<0.001$), and the shape controller in World 2 ($p<0.005$). The same probability seems to be higher for our SCPG controller versus the shape controller in World 3, but it is not statistically significant ($p=0.17$).

Our CPG has similar patterns of performance (i.e a higher maneuverability and resistance to noise) to the Hopf controller. Both controllers suffer more in a denser environment. However, the Hopf controller has a very high variability in the time that it takes to exit and so the mean is biased towards its successful trials. A similar trend of variability was found for oscillator-based CPGs in Travers, Whitman, and Choset [61] and Travers et al. [62]. In best-case trials, the Hopf controller can compare and even outperform the shape controller, but it often gets stuck.

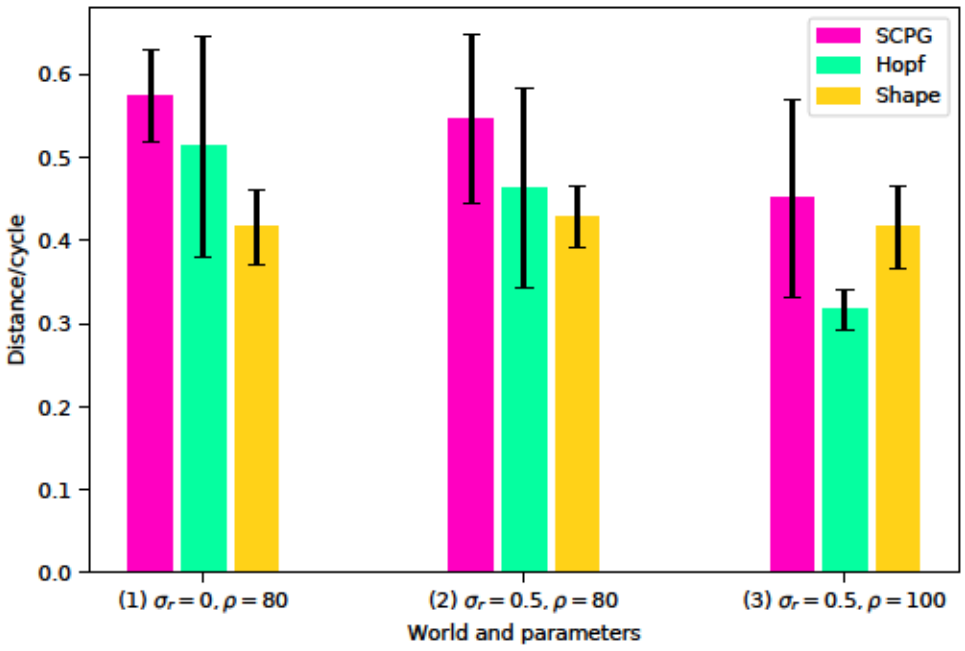
This could be from the integration of torque into the system. In both our implementation

of the oscillator-based CPG and in Travers, Whitman, and Choset [61], the torque modifies individual oscillator amplitudes. In the shape based controller and the oscillator CPG in Travers, Whitman, and Choset [61], this method works well since the amplitude of the windows of the shape based control are independent. However, in the Hopf based oscillator, the amplitudes of each oscillator are not independent. This means that a torque applied at the first joint could affect the last joint (although the effect will be small). This could result in undesired modification of the amplitudes at joints unrelated to the torque input.

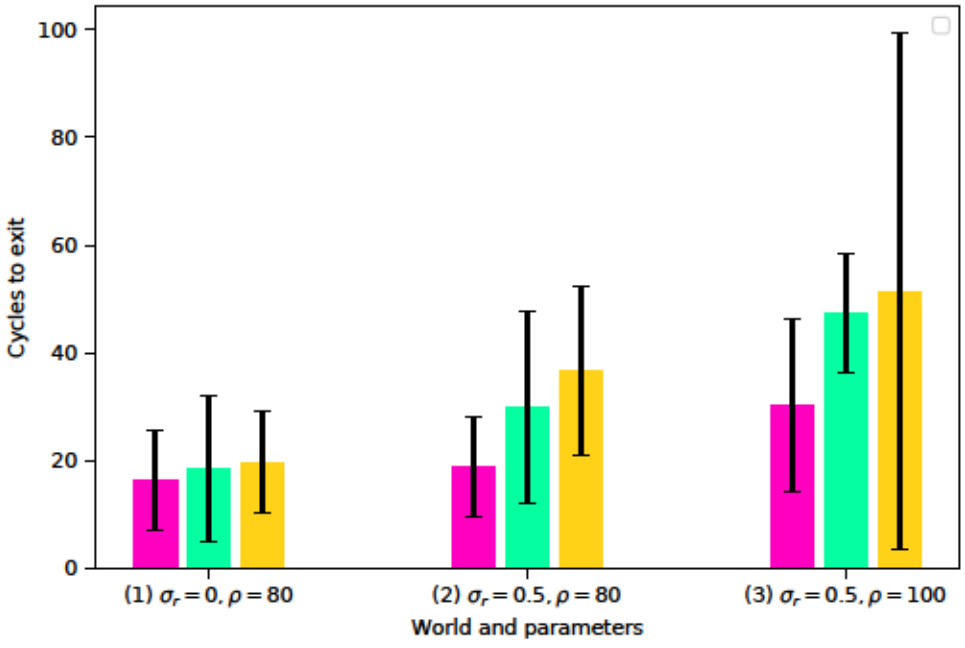
4.4.1 Emergent behaviors

In both the SCPG and Hopf controller, they form a sort of an emergent behavior, particularly in the two more "difficult" worlds. It first happens in Trial 1 for the first world in the Hopf controller, where the snake robot oscillates between two diagonal rows of pegs and moves backwards, exiting by its tail (see A.5). By the last environment, more than half the SCPG exits and two out of three successful Hopf exits use this method. However, this behavior comes at a cost, as the robot moves much slower towards the exit. This can be noted in Figure 4.11, where the two fastest trials of the SCPG in World 3 are the two that do not use this method. The one trial that does not use this method for the Hopf controller does not seem to move any faster.

There are several possible explanations for this behavior. It could be due to the slower response times of the torque input into the two CPG controllers, both of which are longer than the Shape controller. It could also be due to the relatively low amplitude of these controllers compared to the Shape controller.



(a)



(b)

Figure 4.12: Cross-world comparison of distance/cycle and cycles to exit.

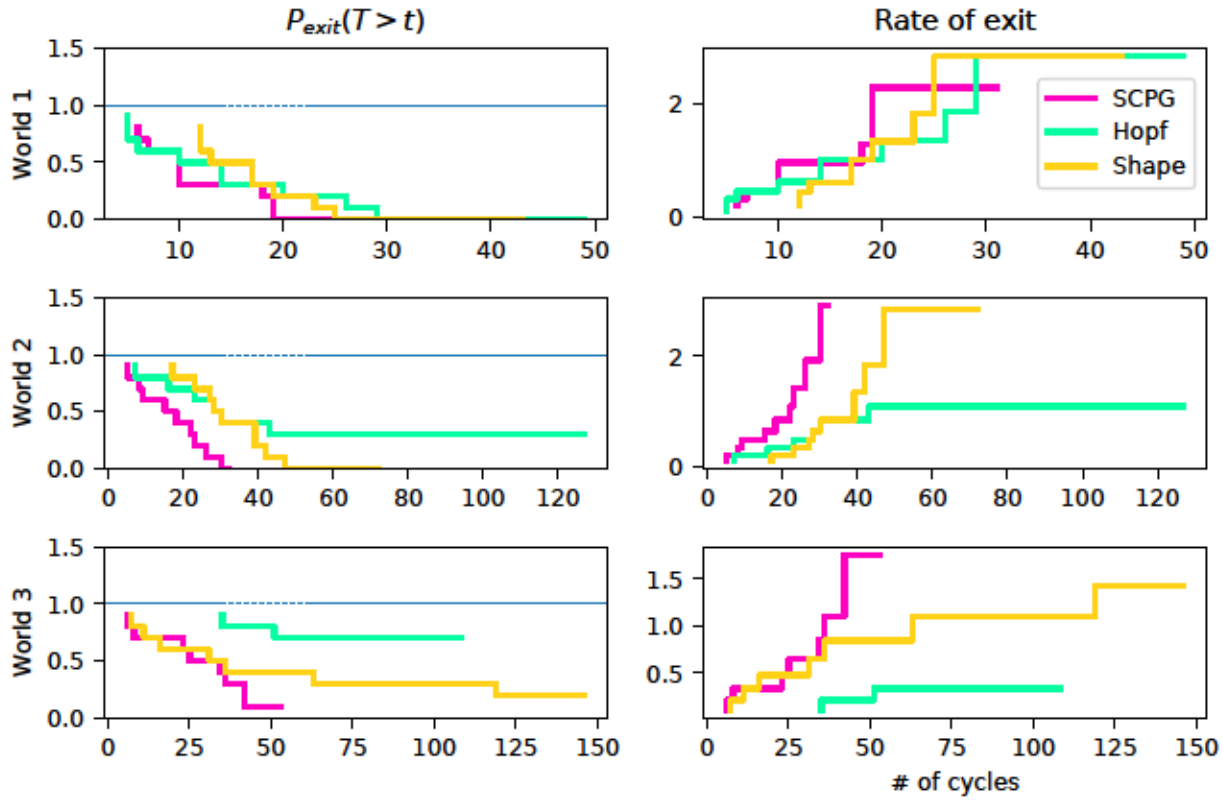


Figure 4.13: Cross-world comparison of survival function. The first column represents the estimated probability that the robot using that controller will exit the arena after t cycles. The right column estimates the approximate (unbounded) probability that the robot has already left after t cycles.

Chapter 5

Discussion

5.1 Interpretation of results

5.1.1 Exploration of the motif space

Most of the motifs that performed decently have a coupled two neuron motif (a "two-oscillator") in them. In particular, it seems in some situations that the extra neurons within a single motif may not contribute to the overall oscillatory behavior of a single oscillator. This is consistent with Curto and Morrison [11], which describes the properties of the dynamics of embedded motifs within a larger graph. In this context, the embedded motif is the two-oscillator, while the larger graph is the single motor module. Alternatively, one can use a larger scope where the single motor module is the embedded motif within the entire CPG. We can consider the oscillatory behavior to be a dynamic fixed point within the overall system, and that fixed point is maintained despite having extra connectivity. From Burtscher et al. [7], it is shown that an embedded motif will maintain its attractor state if external synapses have a symmetry to all neurons in the motif. Due to our assumptions on the inter-module and intra-module connectivity, this will hold true no matter the generated connectivity matrices.

Despite this, it is important to show that between adjacent modules within the CPG,

there is a notable phase difference, which is crucial for snake robot movement. If we had found that the top performer was an immutable motif within a larger graph, our motif search would not be useful. However, we also show that the top performer is unique in the fact that removing an edge or adding a new neuron *does* decrease the performance and change the phase between two adjacent MCs, which is evidence that our search was useful.

On important difference between this work and the works on embedded motifs [11][7] is the choice of neuron non-linearity. In the above works, the authors use a threshold linear firing rate model, and here, we are using an adaptive leaky integrate and fire model.

5.1.2 Simulations

In our survival analysis, we see that the SCPG performance degrades less than the other two controllers. We also see that on average, the number of trials it takes for the SCPG increases less than the other two when transitioning the controller to a more complex environment. We can regard this as increased robustness across different environments.

Our SCPG's consistency is another strength of the controller. Typically, spiking neural networks have an increased level of noise resiliency, but this comes at the cost of a background level of noise [52] and uncertainty in the dynamics. Here, we find that is not the case, and that the spiking CPG is more consistent than even the shape controller, whose output should be neatly deterministic.

We also note several trends across environments for each controller, that might inform which controllers are useful for particular environments. As noted previously, these environments are an extremely narrow subset of possible environments. We observe that the CPG controller and Hopf controller perform better in an environment where they have been tuned for the specific density. On the other hand, the shape controller tends

to perform as well in an environment and handle randomness when the environment is denser than its tuned environment, but does not handle randomness well when the environment is as dense, but randomized.

5.2 Implications

We find that our controller can compare to state of the art performance in several environments and by three metrics. We demonstrate that our compliant controller naturally adapts in frequency and amplitude without an assumption on the model of the robot.

We also present methods for compliance and neural architecture search such that central pattern generators can be found for other robot morphologies and problems.

However, we do not claim that our controller is "better" than the two compared controllers. We acknowledge that there are numerous limitations to our approach and results. On the other hand, because we approach the problem in a novel way, there is a lot of room for expansion in this space. We highlight a few of these below.

5.3 Areas for future research

5.3.1 On the exploration of the neural circuit space

In our search through feasible motifs, we make several assumptions in order to restrict the search space. Because the space is highly non-linear and non-convex, our results are not guaranteed to be optimal, and our chosen CPGs may just represent a local minima within with high-dimensional circuit space. Furthermore, in the search-and-reduce method that we employ, we could potentially leave out circuits that would perform better in the next search space; i.e there could be a circuit that does not perform well in the first step of searching that could perform better when it is chained.

Lastly, our choice to fit the outputs to a sine wave might be an incorrect assumption. Perhaps, in order to robustly control a snake robot, we do not want clean oscillatory behavior, and a more optimal motion might fit some other function. There is an assumption at large that due to the way biological snakes form shapes and move on flat surfaces that all robot snake gaits must be sine-wave based. It has been observed that most of the time, snakes in any environment employ many more gaits and movements [26]. In particular, the generalization of a sine-wave based gait across a set of cluttered environments is demonstrably low. However, the exploration of alternative locomotion paradigms is outside the scope of this work.

5.3.2 On compliance

Our compliance analysis resulted in a usable introduction of modulatory current into our CPG model. However, we limit our analysis to a step response, which is far from the type of torque encountered in robotic locomotion. One could apply traditional controls analyses to the compliance system, which might provide better insight and results in better locomotion. Furthermore, there might be an opportunity to take inspiration from biology[58] and utilize diffuse neurotransmitter approximations.

5.3.3 On bio-inspiration and bio-fidelity

We do not make the claim that our CPG is bio-fidelic nor that it is biologically feasible. Due to the assumptions we make in the motif search, we cannot say with certainty that our CPG is close to anything in biology. However, it is not within the scope of this paper to make a bio-fidelic or biologically feasible CPG, but rather to take inspiration from biology to help guide the development of a robotic controller. Despite that, it could be a future direction of research to see if a CPG designed to more closely mimic biology has similar performance. Looking at the biological CPGs that were identified in Chapter 2 could be

a good place to start. We had previously attempted to use a chained version of the leech heart, as discussed in Chapter 4 but the parameter space was too large to tune well, which was partly the motivation about doing the motif search.

5.3.4 On simulations

Our first limitation is our number of trials, which is set at 10 due to time constraints and the speed of the simulations. With more trials, we could sample the performance of each controller better, and perhaps get more distinct results.

Furthermore, we limit ourselves in our choice of environment. It is entirely possible that our CPG controller performs well in a peg environment, but not, for example, in an environment that is full of complex objects or having a 3D terrain surface. We also do not attempt non-homogenous environments, a gap noted in Travers, Whitman, and Choset [61]. Future work could use our developed framework to expand on the worlds used and the number of trials, which could future demonstrate the uses of the controller.

Our tuned parameters could also be sub-optimal for all three controllers, and it may be that with better tuned parameters, the other two controllers far outperform our CPG controller. However, it was not the goal nor was it in scope of this paper to find optimal parameters for a given environment, but rather to see how one set of parameters affects results in multiple environments.

We made many assumptions on what the frictional model should look like, and decided on making the friction for each link in the Y direction (perpendicular to the "forward" axis of the robot) to be greater than the friction along the X (parallel to the forward axis) plane. However, the actual frictional coefficients used in the Gazebo simulation are hard to derive given their abstraction within the simulation platform, and were estimated based on frictional coefficients between ABS plastic (from which the physical robot's shell is printed)

and concrete. This change of parameter had a negligible effect on the locomotion of the robot, and did not permit locomotion on free terrain. In this sense, it is both a strength and weakness of our simulation framework. On one hand, we were not hindered by the frictional model. At the same time, the robot also did not benefit from being able to use the floor to locomote when getting stuck if an obstacle was too distant to push off of. This led to several cases where the tail of the robot was in a void and the front of the robot could not pull the robot forward. This is why we chose to let the robot exit the area via its tail, since when this was not allowed, the tail could leave the area, leaving the head inside without a chance to escape. This means that our results are skewed by this allowance, and by removing this assumption, we might see our performance degrade significantly compared to the shape controller, which never exited via its tail.

Applying some of the numerous publications on the effect of frictional models[38] might provide some more insight on how to design SCPG circuits to better traverse an environment.

5.3.5 On robustness

We considered the change in performance of a controller from a lower complexity environment to a higher complexity environment to be a measure of robustness. Since we only varied two parameters, this does not necessarily mean that a controller that we consider to be "robust" here will be robust to all environments and situations. Particularly, as we discuss in other sections within this chapter, there are many other environmental parameters and situations where our SCPG controller might not be robust. We also did not examine the effects of sensor noise into the controller, which is a good next step to explore.

5.3.6 On robot models

The robot model with alternating motor axes and discrete links is a common robot model in the literature [10][56][62][60][53], as it is simple to construct, easy to simulate and the kinematics are trivial to derive (although complex in form). However, it is not the same as a typical snake, and as such, mapping the outputs of a neural circuit found in biology to a 12 in-plane motor approximation is a non-trivial task. Overall, the objective of any snake robot controller is to mimic the shape of a biological snake over time. Our SCPG implementation generally does that, but another choice of robot model might make more sense in terms of neuron-motor mappings. For example,a robot constructed similarly to a continuum manipulator [63] might provide a simpler transform between neurons and actuators. Another approach could be finding a method to generally map between shape spaces of two different dimensions (e.g a CPG with 6 outputs and a robot with 13 motors), such as was developed in Aljalbout et al. [3]. A shape-based method might be well applied here, where generalized shapes are generated instead of sine waves. Our approach to optimizing structure might be able to produce such waves.

5.3.7 On neuromorphic hardware implementation

Due to the complexities of the software implementation, the CPG controller executes much slower than both of the comparison controllers, even on a relatively powerful computer. It follows that an onboard computer would likely not be able to simulate the full software stack. Therefore, it is necessary to implement the network on neuromorphic hardware. While several publications have focused on in-silico implementations of SCPGs [33] [51][4], it is outside the scope of this work. Small implementations of these chips could be built into the robot itself, such that the complexities and power consumption of running a neural simulator on-board could be mitigated. The advantage of our network in Nengo is

that there is a simplified process from code to silicon by utilizing the Intel Loihi chip [12]. However, only certain neuron types are supported directly, and the ALIF neuron model that we used is not one of them. Polykretis, Tang, and Michmizos [48] used a custom implementation of a bursting neuron with similar dynamics to put their network on a Loihi, so something similar might work for our model.

As discussed previously, a neuromorphic power analysis for a snake robot might not be particularly useful, as the actuators themselves consume orders of magnitude more power than any controller. However, it might be relevant for microrobots, such as the insect robot developed in Goldberg et al. [20]. For comparison, their robot controller runs on a Atmel Atmega1284RFR2, which consumes 4.7mW [1], while a 60 neuron network might consume $20\mu W$ [52] when implemented on neuromorphic hardware. Polykretis, Tang, and Michmizos [48] implemented an SCPG on neuromorphic hardware using an Intel Loihi, and estimated their power consumption by comparing to their network size to a 15k compartment neuromorphic SLAM implementation [59]. Their network size was less than 0.5% of the the size of the network in Tang, Shah, and Michmizos [59], which consumed 9mW. Although the power consumption does not scale linearly with the number of neurons, that puts the consumption of that CPG network on the order of $50\mu W$. Our network is a similar size.

5.3.8 On robotic hardware

While the simulation was designed around the physical robot, torque sensors were added in simulation that are not available in the Dynamixel motors on the physical robot. Therefore, in implementing this work onto the physical robot, the motors will need to be swapped with a different model or additional sensors need to be attached.

Furthermore, a physical instantiation of this network was not implemented on hardware. An easy extension of this work would be to take our existing design and verify, by putting it on a physical robot, that the performance of the SCPG is not dependant on some simulation parameter. On this note, since the collision and inertial parameters were assumed to be a cylinder in simulation, there might be a noticeable difference in the performance of all controllers. This could be solved with a different design of the shell such that the edges could not get caught on objects. A thin skin for the robot would be a good solution, following some of the work in Whitman et al. [65].

Chapter 6

Conclusion

In this work, we showcase a snake robot controller inspired by a class of neural circuits: central pattern generators. In order to create the controller, we devise a way to search over the possible ways that the neurons are connected, instead of modifying the parameters of a hand-made static structure. We also form a way to introduce external torque input from the robot's sensors into the central pattern generator to result in changes to the patterned output, much like sensory neurons do in biology. This results in compliance in the robot motors, such that the robot body forms around obstacles. We then validate the controller's performance versus two other controllers from recent publications. The first, which we modify to be compliant, is a central pattern generator controller where dynamics inspired by central pattern generators are used instead of a simulated network of neurons. The second is a compliant shape based controller, where shapes can be propagated down the snake robot body. To compare the controllers, we develop a simulated model based off our designed snake robot and three simulated environments of increasing complexity in which to test the controllers. We tune the parameters of each controller on the first environment, and leave them the same for the more complex ones. We show that our controller performs well in these environments, in some cases outperforming the others. Beyond the snake-robot application we show here, our methods and approach can be applied towards other robots of all shapes and sizes, as well as towards the designs

of systems inspired by neuroscience. Our work represents a possible path forward for neuro-inspired systems, which often cannot measure up to traditional methods in terms of performance, but have an advantage in power usage. We show that we can not only design a neuro-inspired system that can compare to state of the art, but that has increased robustness to environmental parameters.

Appendix A

Supplementary figures and metrics

Included in this appendix is the other attempts from the compliance filter. Also included is the visualized data for each trial, including the each path the robot took and the commanded motor position for the first motor. The colors here are matched to the trial colors in the summary plots in Figures 4.9, 4.10, and 4.11.

Lastly is the structure and output of the leech heartbeat CPG with a forward connection. This shows the leech heart not performing well when chained together, which demonstrates that our structure search was fruitful.

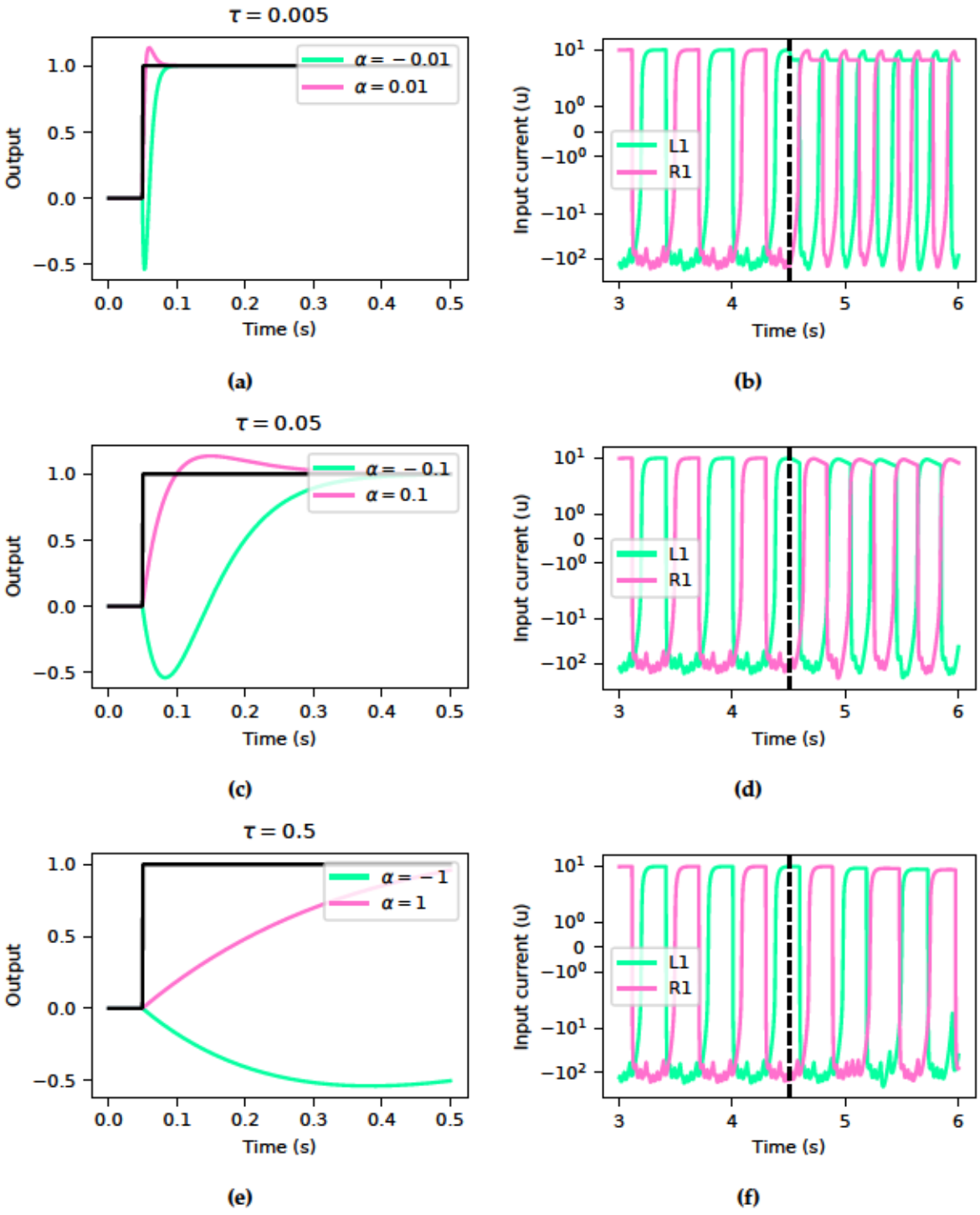


Figure A.1: Step responses of the modulatory synaptic filter and resulting current inputs into M1R1 and M1L1. The slower filters result in smoother current peaks. However, since the neurons start firing at 0.2 units, the overall firing rate is similar. This results in faster frequency and lower amplitude waves.

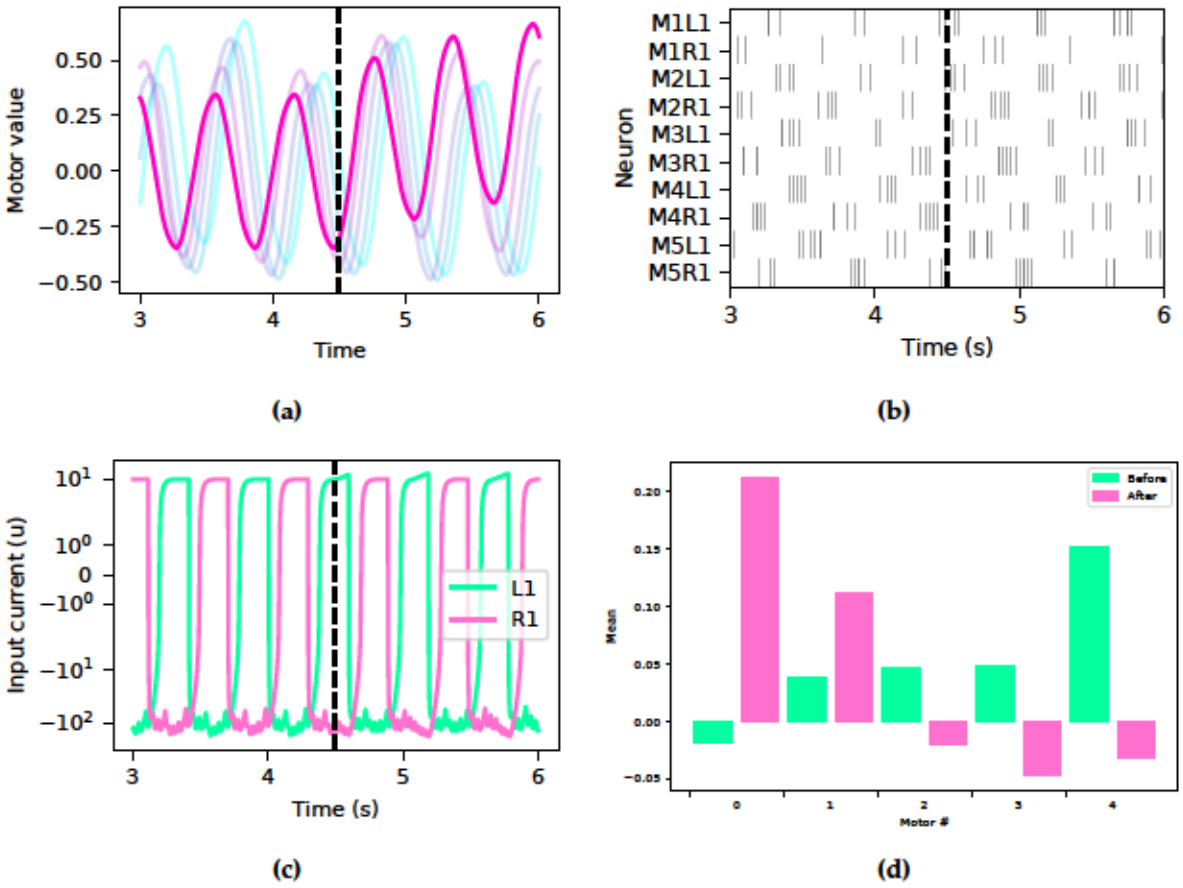
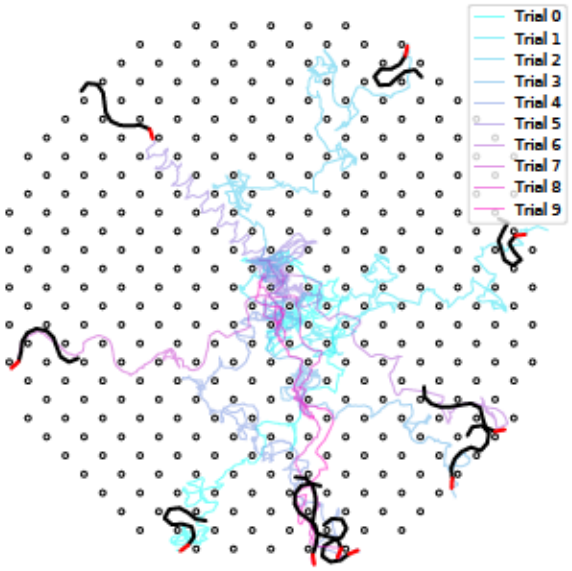
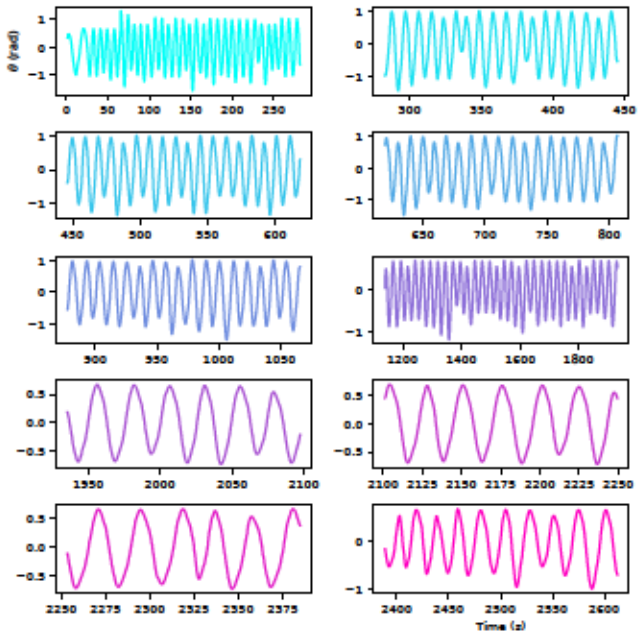


Figure A.2: Injecting current into just the L1 neuron causes turning. (a) the output of the first five MC. (b) Spikes of the first five MC output neurons. Note that MLL1 fires twice as often as M1R1. (c) The input current for each neuron in MC 1. (d) The effect is that the mean of the oscillation moves upward.

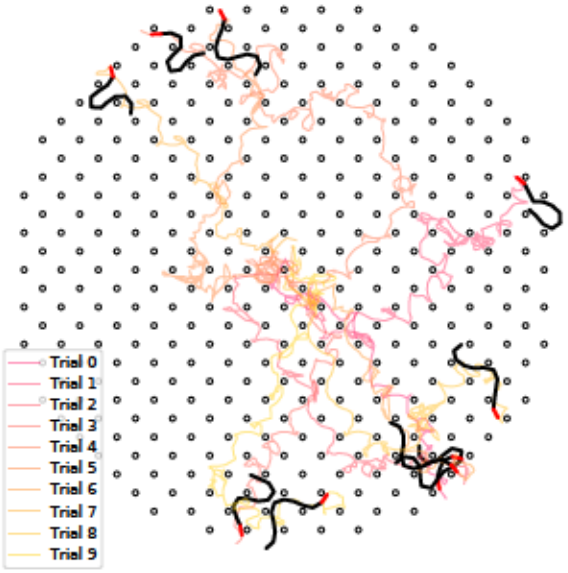


(a)

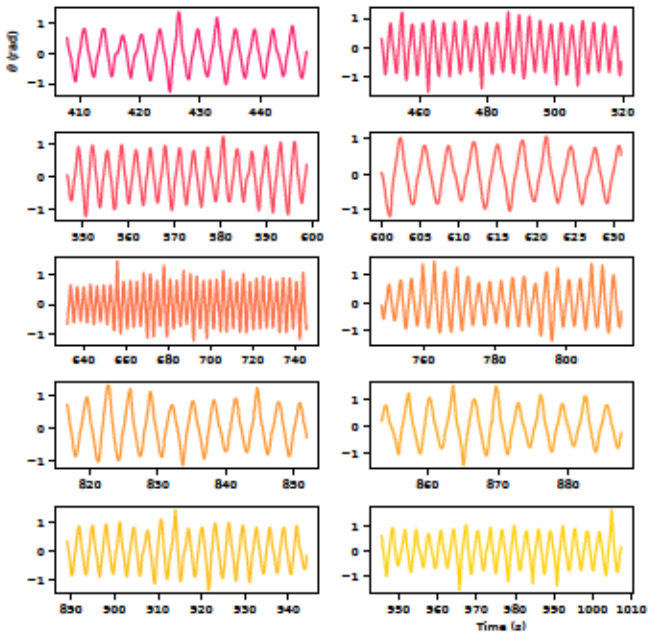


(b)

Figure A.3: Paths and oscillations of the first motor for the CPG controller in World 1. Note the startup time in the first trial.

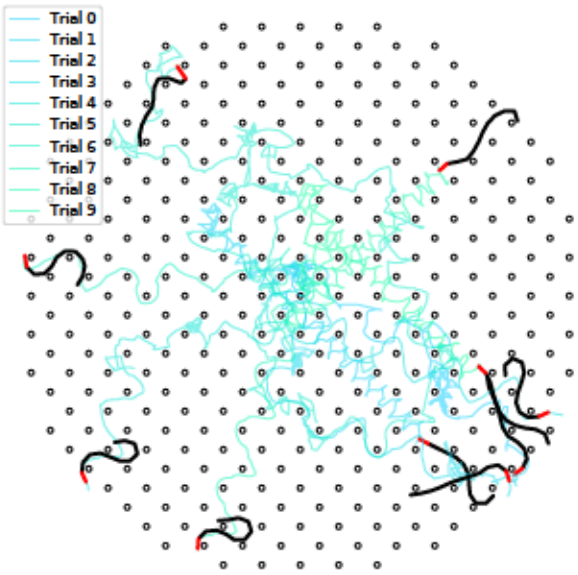


(a)

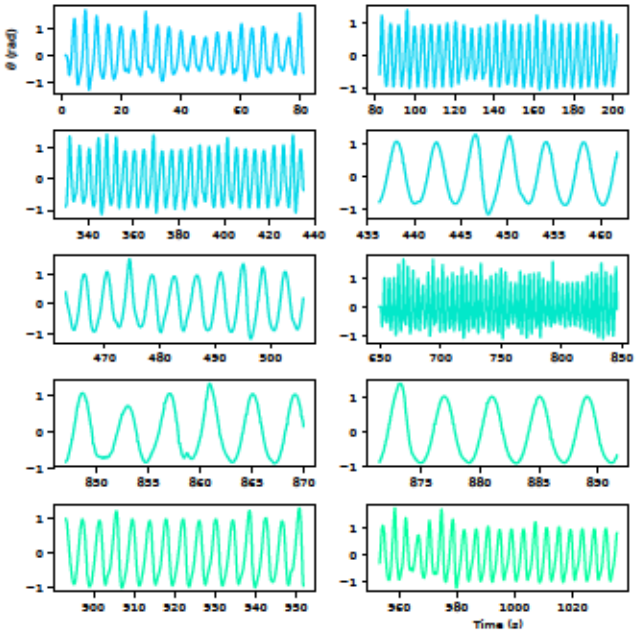


(b)

Figure A.4: Paths and oscillations of the first motor for the shape controller in World 1

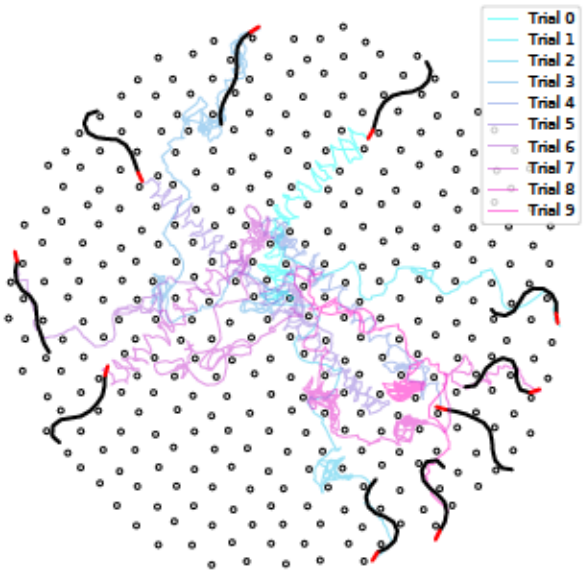


(a)

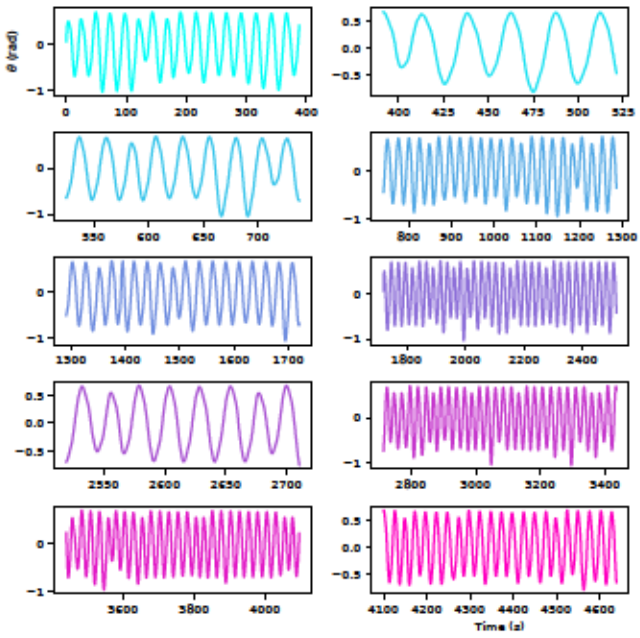


(b)

Figure A.5: Paths and oscillations of the first motor for the Hopf controller in World 1

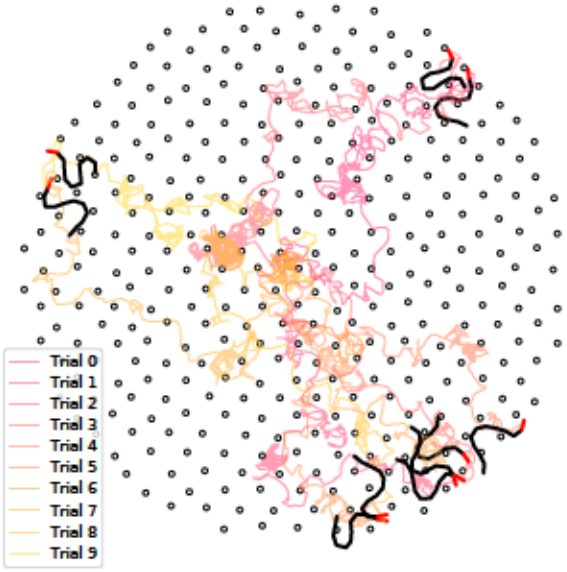


(a)

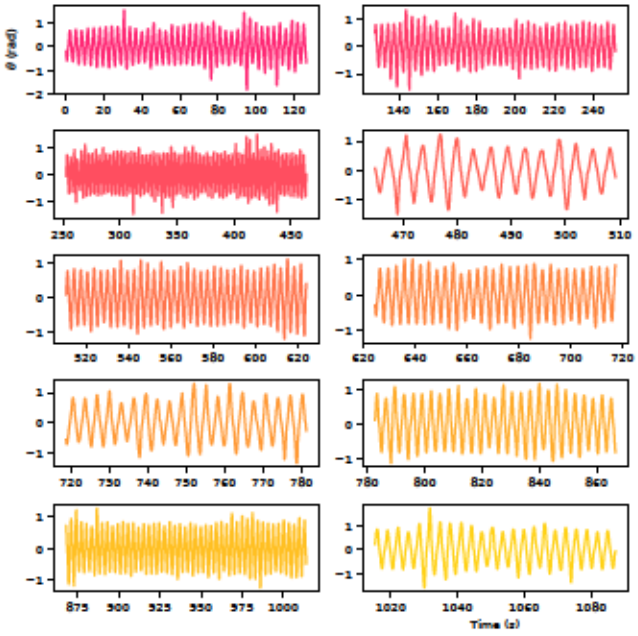


(b)

Figure A.6: Paths and oscillations of the first motor for the CPG controller in World 2

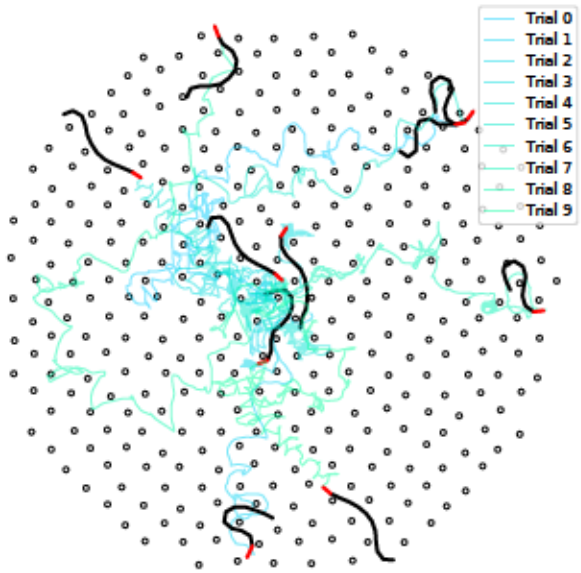


(a)

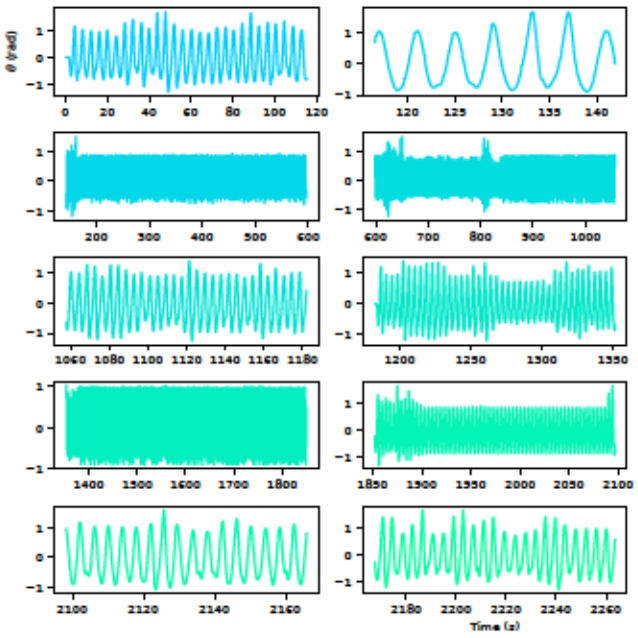


(b)

Figure A.7: Paths and oscillations of the first motor for the shape controller in World 2

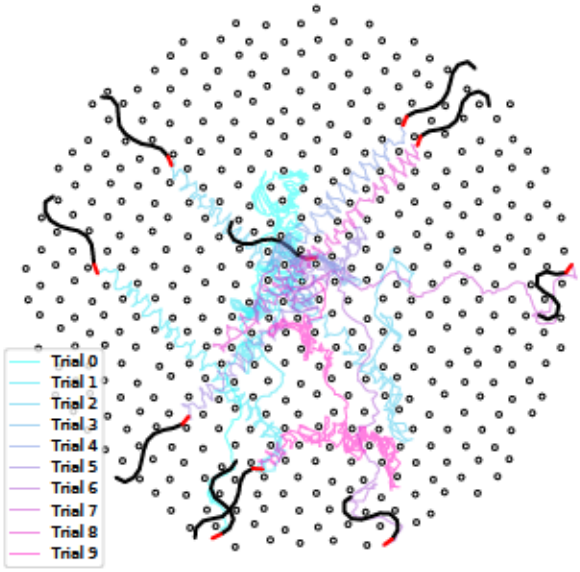


(a)

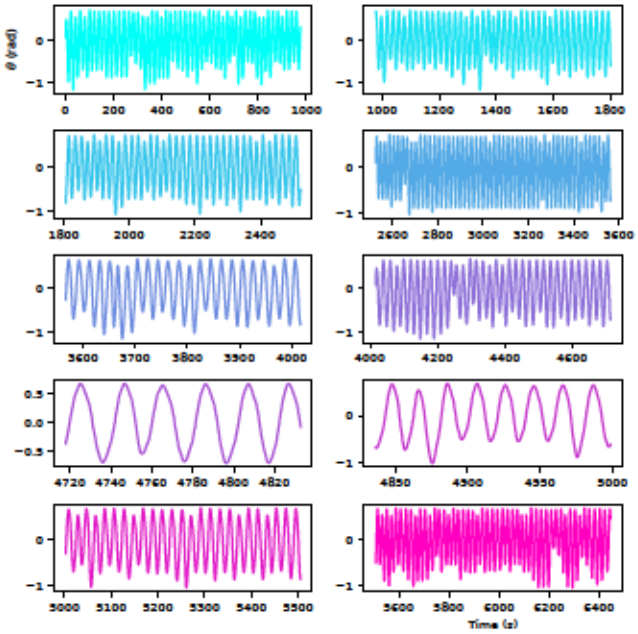


(b)

Figure A.8: Paths and oscillations of the first motor for the Hopf controller in World 2

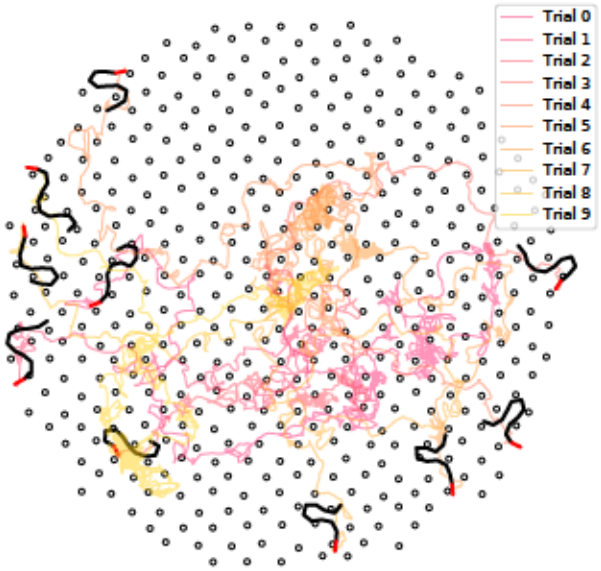


(a)

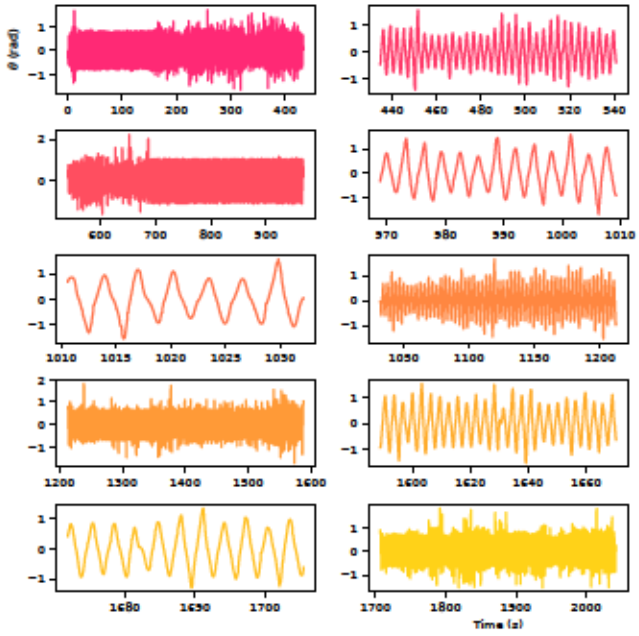


(b)

Figure A.9: Paths and oscillations of the first motor for the CPG controller in World 3

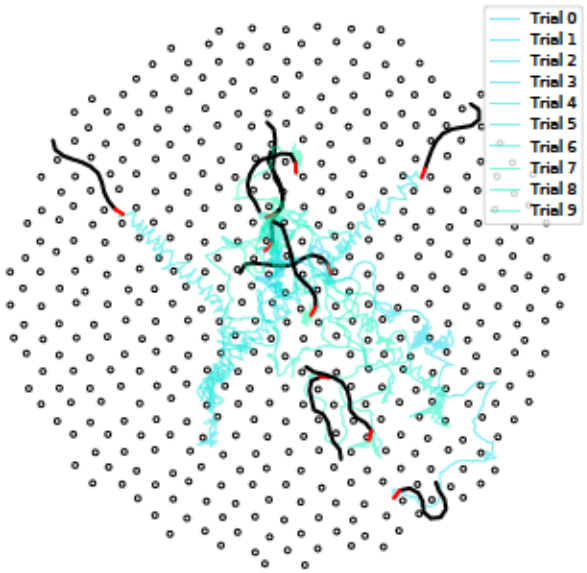


(a)

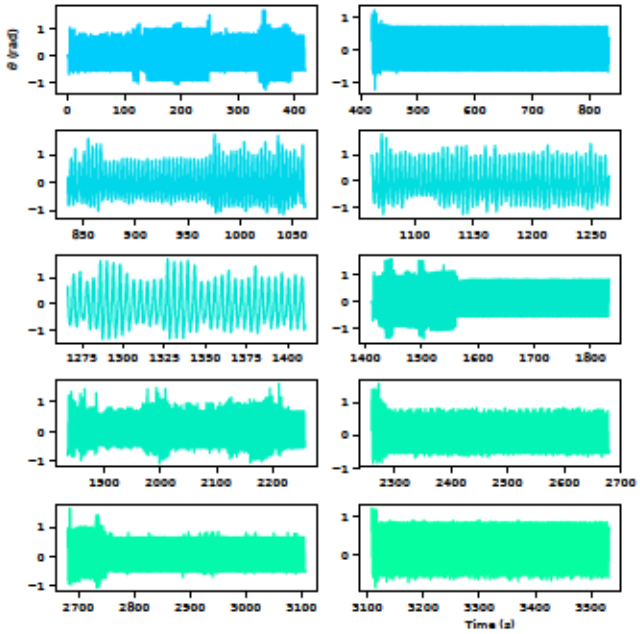


(b)

Figure A.10: Paths and oscillations of the first motor for the shape controller in World 3

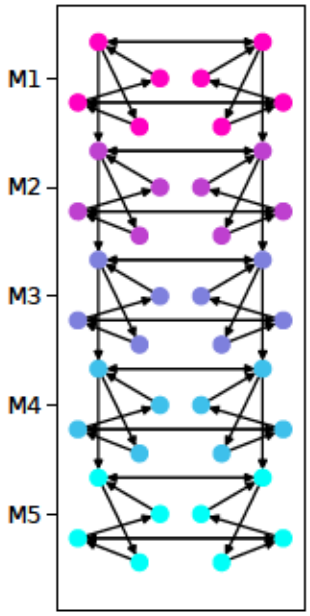


(a)

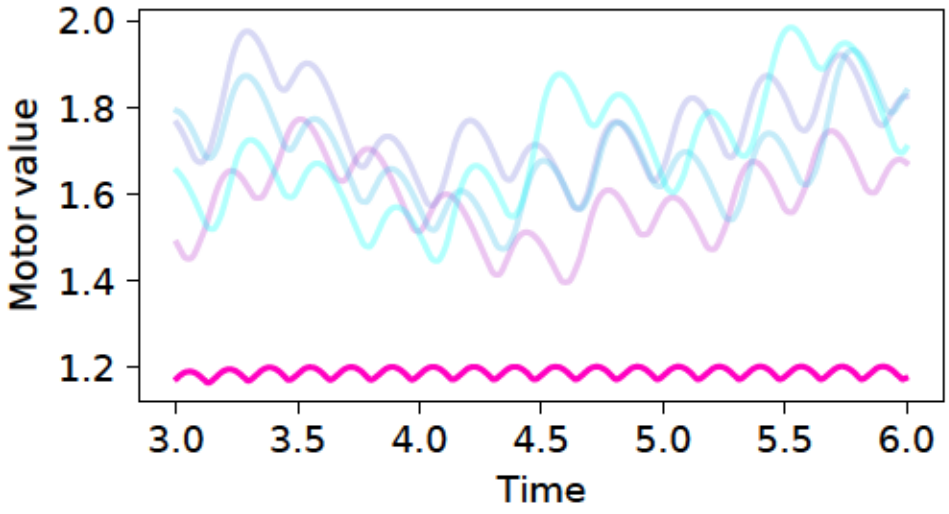


(b)

Figure A.11: Paths and oscillations of the first motor for the Hopf controller in World 3



(a)



(b)

Figure A.12: Structure and output of a chained leech heartbeat

Appendix B

Raw data

Here, the raw data from the trials is shown. This includes whether or not the trail was a failure ("True" denotes a failure), the number of peaks (i.e cycles) in the trial, the distance traveled (in meters), and the distance per peak. This data can be used to verify or run additional analyses if needed. Summarized and visualized versions of this data can be found in Chapter 4.

type	trial	failure	peaks	distance	distance/peak
CPG	0	False	31	18.2423	0.588463
	1	False	18	10.409	0.57828
	2	False	18	10.5829	0.587939
	3	False	19	10.7984	0.568338
	4	False	18	10.2423	0.569015
	5	False	31	15.2789	0.492867
	6	False	6	4.18436	0.697393
	7	False	7	3.68957	0.527082
	8	False	6	3.59502	0.59917
	9	False	10	5.27952	0.527952
Hopf	0	False	20	8.95425	0.447713
	1	False	29	10.6153	0.366044
	2	False	26	9.82831	0.378012
	3	False	6	4.40951	0.734918
	4	False	10	5.76933	0.576933
	5	False	49	25.6838	0.52416
	6	False	6	3.49817	0.583028
	7	False	5	3.4819	0.69638
	8	False	14	5.39334	0.385239
	9	False	20	8.79415	0.439708
Shape	0	False	13	5.48767	0.422129
	1	False	25	9.92964	0.397185
	2	False	19	7.2112	0.379537
	3	False	12	4.37457	0.364547
	4	False	43	16.8271	0.391329
	5	False	23	9.53461	0.414548
	6	False	13	5.40678	0.415907
	7	False	12	4.66244	0.388537
	8	False	17	8.55998	0.503528
	9	False	19	9.19905	0.484161

Table B.1: World 1 raw data

type	trial	failure	peaks	distance	distance/peak
CPG	0	False	15	7.36815	0.49121
	1	False	5	3.69091	0.738182
	2	False	9	6.02065	0.668961
	3	False	23	13.0584	0.567757
	4	False	18	7.97336	0.442965
	5	False	32	13.1449	0.410777
	6	False	8	4.76813	0.596017
	7	False	30	14.2387	0.474622
	8	False	26	13.9093	0.534975
	9	False	22	11.8825	0.540113
Hopf	0	False	28	13.0689	0.466746
	1	False	7	4.09971	0.585673
	2	True	115	27.4614	0.238795
	3	True	111	29.9528	0.269845
	4	False	30	11.232	0.3744
	5	False	43	15.2313	0.354216
	6	True	127	38.5896	0.303855
	7	False	61	21.1586	0.346863
	8	False	16	10.554	0.659626
	9	False	23	10.3769	0.45117
Shape	0	False	42	18.1644	0.432487
	1	False	42	18.5814	0.442415
	2	False	72	30.9179	0.429416
	3	False	17	6.92228	0.407193
	4	False	39	14.0894	0.361266
	5	False	30	13.3701	0.445671
	6	False	23	9.38612	0.408092
	7	False	28	13.1505	0.469661
	8	False	47	23.1487	0.492525
	9	False	27	10.6332	0.393821

Table B.2: World 2 raw data

type	trial	failure	peaks	distance	distance/peak
CPG	0	False	49	21.1558	0.431752
	1	False	42	18.0094	0.428796
	2	False	36	13.8275	0.384098
	3	True	53	7.6583	0.144496
	4	False	23	8.754	0.380609
	5	False	34	11.999	0.352911
	6	False	6	3.9363	0.656049
	7	False	8	5.19368	0.64921
	8	False	25	8.64055	0.345622
	9	False	49	20.8996	0.426522
Hopf	0	True	103	18.1481	0.176196
	1	True	101	12.1622	0.120418
	2	False	56	17.845	0.31866
	3	False	51	14.9096	0.292345
	4	False	35	11.9139	0.340398
	5	True	105	23.4887	0.223702
	6	True	106	32.516	0.306754
	7	True	103	11.4906	0.111559
	8	True	104	29.5474	0.284109
	9	True	108	28.7024	0.265763
Shape	0	True	146	46.7414	0.320147
	1	False	36	16.0872	0.446867
	2	True	143	28.7477	0.201033
	3	False	16	5.66319	0.35395
	4	False	7	3.54135	0.505907
	5	False	63	25.9262	0.411528
	6	False	128	50.1362	0.391689
	7	False	31	11.6048	0.374349
	8	False	11	5.01795	0.456177
	9	False	119	46.5427	0.391115

Table B.3: World 3 raw data

Bibliography

[1] "8-bit Microcontroller with Low Power 2.4GHz Transceiver for ZigBee and IEEE 802.15.4". 2014. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42073-MCU_Wireless-ATmega2564RFR2-ATmega1284RFR2-ATmega644RFR2_Summary_Datasheet.pdf.

[2] Kyoichi Akiyama et al. "Undulatory Swimming Locomotion Driven by CPG with Multimodal Local Sensory Feedback". In: *Biomimetic and Biohybrid Systems*. Ed. by Vouloutsi Vasiliki et al. Cham: Springer International Publishing, 2018, pp. 1–5. ISBN: 978-3-319-95972-6.

[3] Elie Aljalbout et al. "Task-Independent Spiking Central Pattern Generator: A Learning-Based Approach". In: *Neural Processing Letters* 51.3 (2020), pp. 2751–2764. ISSN: 1573773X. DOI: 10.1007/s11063-020-10224-9. URL: https://doi.org/10.1007/s11063-020-10224-9.

[4] Matthieu Ambroise et al. "Real-time biomimetic central pattern generators in an FPGA for hybrid experiments". In: *Frontiers in Neuroscience* 7.7 NOV (2013), pp. 1–11. ISSN: 16624548. DOI: 10.3389/fnins.2013.00215.

[5] Emmanouil Angelidis et al. "A Spiking Central Pattern Generator for the control of a simulated lamprey robot running on SpiNNaker and Loihi neuromorphic boards". In: (2021). URL: http://arxiv.org/abs/2101.07001.

[6] Trevor Bekolay et al. "Nengo: A Python tool for building large-scale functional brain models". In: *Frontiers in Neuroinformatics* 7.JAN (2014), pp. 1–13. ISSN: 16625196. DOI: 10.3389/fninf.2013.00048.

[7] Felicia Burtscher et al. "Nerve theorems for fixed points of neural networks". In: (2021), pp. 1–25. URL: http://arxiv.org/abs/2102.11437.

[8] Paolo Cignoni et al. "MeshLab: an Open-Source Mesh Processing Tool". In: *Eurographics Italian Chapter Conference*. Ed. by Vittorio Scarano, Rosario De Chiara, and Ugo Erra. The Eurographics Association, 2008. ISBN: 978-3-905673-68-5. DOI: 10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136.

[9] Alessandro Crespi, Auke Jan Aj Ijspeert, and Ecole Polytechnique F. "AmphiBot II : An Amphibious Snake Robot that Crawls and Swims using a Central Pattern Generator". In: *... Conference on Climbing and Walking Robots (...* September (2006), pp. 19–27. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.73.582&rep=rep1&type=pdf.

[10] Alessandro Crespi et al. "AmphiBot I: An amphibious snake-like robot". In: *Robotics and Autonomous Systems* 50.4 (2005), pp. 163–175. ISSN: 09218890. DOI: 10.1016/j.robot.2004.09.015.

[11] Carina Curto and Katherine Morrison. "Relating network connectivity to dynamics: opportunities and challenges for theoretical neuroscience". In: *Current Opinion in Neurobiology* 58 (2019), pp. 11–20. ISSN: 09594388. DOI: 10.1016/j.conb.2019.06.003. URL: https://linkinghub.elsevier.com/retrieve/pii/S0959438819300443.

[12] Mike Davies et al. "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning". In: *IEEE Micro* 38.1 (2018), pp. 82–99. ISSN: 02721732. DOI: 10.1109/MM.2018.112130359.

[13] Rui Ding et al. "Dynamic modelling of a CPG-controlled amphibious biomimetic swimming robot: Regular paper". In: *International Journal of Advanced Robotic Systems* 10 (2013), pp. 1–11. ISSN: 17298806. DOI: 10.5772/56059.

[14] Rui Ding et al. "Robust gait control in biomimetic amphibious robot using central pattern generator". In: *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings* (2010), pp. 3067–3072. DOI: 10.1109/IROS.2010.5651475.

[15] Chris Eliasmith and Charles H. Anderson. *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*. Cambridge, MA, USA: MIT Press, 2003. ISBN: 0262550601.

[16] A. Espinal et al. "Quadrupedal robot locomotion: A biologically inspired approach and its hardware implementation". In: *Computational Intelligence and Neuroscience* 2016 (2016). ISSN: 16875273. DOI: 10.1155/2016/5615618.

[17] Alexander E. Filippov and Stanislav N. Gorb. "Modelling of the frictional behaviour of the snake skin covered by anisotropic surface nanostructures". In: *Scientific Reports* 6.August 2015 (2016), pp. 1–6. ISSN: 20452322. DOI: 10.1038/srep23539.

[18] Thomas R Fleming and David P Harrington. "A class of hypothesis tests for one and two sample censored survival data". In: *Communications in Statistics - Theory and Methods* 10.8 (1981), pp. 763–794. DOI: 10.1080/03610928108828073. URL: https://doi.org/10.1080/03610928108828073.

[19] Qiyuan Fu and Chen Li. "Robotic modeling of snake traversing large, smooth obstacles reveals stability benefits of body compliance". In: *arXiv* (2020). ISSN: 23318422.

[20] Benjamin Goldberg et al. "Power and Control Autonomy for High-Speed Locomotion with an Insect-Scale Legged Robot". In: *IEEE Robotics and Automation Letters* 3.2 (2018), pp. 987–993. ISSN: 23773766. DOI: 10.1109/LRA.2018.2793355.

[21] Daniel Gutierrez-Galan et al. "NeuroPod: a real-time neuromorphic spiking CPG applied to robotics". In: April (2019). URL: http://arxiv.org/abs/1904.11243.

[22] Robert Haschke and Morgan Quigley. *xacro*. 2018. URL: http://wiki.ros.org/xacro.

[23] F. Herrero-Carrón, F. B. Rodríguez, and P. Varona. "Bio-inspired design strategies for central pattern generator control in modular robotics". In: *Bioinspiration and Biomimetics* 6.1 (2011). ISSN: 17483182. DOI: 10.1088/1748-3182/6/1/016006.

[24] Shigeo Hirose. *Biologically Inspired Robots: Snake-Like Locomotors and Manipulators*. Vol. 12. 3. Cambridge University Press, 1994, pp. 282–282. DOI: 10.1017/s0263574700017264. URL: https://www.cambridge.org/core/product/identifier/S0263574700017264/type/journal_article.

[25] Auke Jan Ijspeert and Alessandro Crespi. "Online trajectory generation in an amphibious snake robot using a lamprey-like central pattern generator model". In: *Proceedings - IEEE International Conference on Robotics and Automation* April (2007), pp. 262–268. ISSN: 10504729. DOI: 10.1109/ROBOT.2007.363797.

[26] Bruce C. Jayne. "What defines different modes of snake locomotion?" In: *Integrative and Comparative Biology* 60.1 (2020), pp. 156–170. ISSN: 15577023. DOI: 10.1093/icb/icaa017.

[27] Tetsushi Kamegawa et al. "Development of The Snake-like Rescue Robot " KOHGA "". In: April (2004), pp. 5081–5086.

[28] Tetsushi Kamegawa et al. "Three-Dimensional Reflexive Behavior by a Snake Robot with Full Circumference Pressure Sensors". In: *Proceedings of the 2020 IEEE/SICE International Symposium on System Integration, SII 2020* (2020), pp. 897–902. DOI: 10.1109/SII46433.2020.9026245.

[29] E L Kaplan and Paul Meier. "Nonparametric estimation from incomplete samples". In: *Journal of the American Statistical Association* 53.282 (1958), pp. 457–481. URL: http://www.jstor.org/stable/2281868.

[30] Paul S. Katz. "Evolution of central pattern generators and rhythmic behaviours". In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 371.1685 (2016). ISSN: 14712970. DOI: 10.1098/rstb.2015.0057.

[31] Robert Kwiatkowski and Hod Lipson. "Task-agnostic self-modeling machines". In: *Science Robotics* 4.26 (2019), eaau9354. ISSN: 2470-9476. DOI: 10.1126/scirobotics.aau9354. URL: http://arxiv.org/abs/1707.06347%20http://robotics.sciencemag.org/lookup/doi/10.1126/scirobotics.aau9354.

[32] M. Anthony Lewis, Francesco Tenore, and Ralph Etienne-Cummings. "CPG design using inhibitory networks". In: *Proceedings - IEEE International Conference on Robotics and Automation* 2005.April (2005), pp. 3682–3687. ISSN: 10504729. DOI: 10.1109/ROBOT.2005.1570681.

[33] M. Anthony Lewis et al. "An in silico central pattern generator: Silicon oscillator, coupling, entrainment, and physical computation". In: *Biological Cybernetics* 88.2 (2003), pp. 137–151. ISSN: 03401200. DOI: 10.1007/s00422-002-0365-7.

[34] M.A. Lewis and D.M. Zehnpfennig. "R7: a snake-like robot for 3-d visual inspection". In: (2002), pp. 1310–1317. DOI: 10.1109/iros.1994.407513.

[35] Pål Liljeback et al. "Snake Robot Locomotion in Environments With Obstacles". In: *IEEE/ASME Transactions on Mechatronics* 17.6 (2012), pp. 1158–1169. ISSN: 1083-4435. DOI: 10.1109/TMECH.2011.2159863. URL: http://ieeexplore.ieee.org/document/5959211/.

[36] Pål Liljebäck et al. "Hybrid modelling and control of obstacle-aided snake robot locomotion". In: *IEEE Transactions on Robotics* 26.5 (2010), pp. 781–799. ISSN: 15523098. DOI: 10.1109/TRO.2010.2056211.

[37] P. Liljebck et al. "A review on modelling, implementation, and control of snake robots". In: *Robotics and Autonomous Systems* 60.1 (2012), pp. 29–40. ISSN: 09218890. DOI: 10.1016/j.robot.2011.08.010. URL: http://dx.doi.org/10.1016/j.robot.2011.08.010.

[38] Jindong Liu, Yuchuang Tong, and Jinguo Liu. "Review of snake robots in constrained environments". In: *Robotics and Autonomous Systems* 141 (2021), p. 103785. ISSN: 09218890. DOI: 10.1016/j.robot.2021.103785. URL: https://doi.org/10.1016/j.robot.2021.103785.

[39] David Machin et al. *Survival analysis: a practical approach*. 2nd ed. Chichester, England ; Hoboken, NJ: Wiley, 2006. ISBN: 9780470870402.

[40] Akihiro Maruyama, Tomoyasu Ichimura, and Yoshinobu Maeda. "Hard-wired Central Pattern Generator Hardware Network for Quadrupedal Locomotion Based on Neuron and Synapse Models". In: *Advanced Biomedical Engineering* 4.0 (2015), pp. 48–54. ISSN: 2187-5219. DOI: 10.14326/abe.4.48.

[41] Christophe Maufroy, Hiroshi Kimura, and Kunikatsu Takase. "Towards a general neural controller for quadrupedal locomotion". In: *Neural Networks* 21.4 (2008), pp. 667–681. ISSN: 08936080. DOI: 10.1016/j.neunet.2008.03.010.

[42] Shunsuke Nansai, Mohan Rajesh Elara, and Masami Iwase. "Dynamic Hybrid Position Force Control using Virtual Internal Model to realize a cutting task by a snake-like robot". In: *Proceedings of the IEEE RAS and EMBS International Conference on Biomedical Robotics and Biomechatronics* 2016-July (2016), pp. 151–156. ISSN: 21551774. DOI: 10.1109/BIOROB.2016.7523614.

[43] Wayne Nelson. "Theory and Applications of Hazard Plotting for Censored Failure Data". In: *Technometrics* 14.4 (1972), pp. 945–966. ISSN: 00401706. DOI: 10.2307/1267144. URL: http://www.jstor.org/stable/1267144.

[44] Marko Nonhoff et al. "Economic model predictive control for snake robot locomotion". In: *Proceedings of the IEEE Conference on Decision and Control* 2019-Decem.1 (2019), pp. 8329–8334. ISSN: 07431546. DOI: 10.1109/CDC40024.2019.9029627.

[45] Erick O Olivares, Eduardo J Izquierdo, and Randall D Beer. "Potential role of a ventral nerve cord central pattern generator in forward and backward locomotion in Caenorhabditis elegans." In: *Network neuroscience (Cambridge, Mass.)* 2.3 (2018), pp. 323–343. ISSN: 2472-1751. DOI: 10.1162/netn{_}a{_}00036. URL: http://www.ncbi.nlm.nih.gov/pubmed/30294702%20http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC6145852.

[46] Wenjuan Ouyang et al. "Adaptive Locomotion Control of a Hexapod Robot via Bio-Inspired Learning". In: *Frontiers in Neurorobotics* 15.January (2021). ISSN: 1662-5218. DOI: 10.3389/fnbot.2021.627157.

[47] Sebastian Pölsterl. "scikit-survival: A Library for Time-to-Event Analysis Built on Top of scikit-learn". In: *Journal of Machine Learning Research* 21.212 (2020), pp. 1–6. URL: http://jmlr.org/papers/v21/20-729.html.

[48] Ioannis Polykretis, Guangzhi Tang, and Konstantinos P. Michmizos. "An Astrocyte-Modulated Neuromorphic Central Pattern Generator for Hexapod Robot Locomotion on Intel's Loihi". In: *arXiv* (2020). ISSN: 23318422.

[49] Filip Ponulak. "ReSuMe-new supervised learning method for Spiking Neural Networks". In: *Inst. Control Information Engineering, Poznan Univ.* 22.2 (2005), pp. 467–510. ISSN: 1530-888X. URL: http://www.ncbi.nlm.nih.gov/pubmed/19842989%5Cnhttp://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.60.6325&rep=rep1&type=pdf.

[50] Gill A. Pratt and Matthew M. Williamson. "Series elastic actuators". In: *IEEE International Conference on Intelligent Robots and Systems* 1 (1995), pp. 399–406. DOI: 10.1109/iros.1995.525827.

[51] Etienne-cummings Ralph and Lewis M Anthony. *Biomorphic Rhythmic Movement Controller*. URL: https://lens.org/108-498-335-043-756.

[52] Brian S Robinson et al. "Online learning for orientation estimation during translation in an insect ring attractor network". In: *bioRxiv* (2021), p. 2021.01.07.425323. DOI: 10.1101/2021.01.07.425323. URL: http://biorxiv.org/content/early/2021/01/07/2021.01.07.425323.abstract.

[53] David Rollinson et al. "Design and architecture of a series elastic snake robot". In: *IEEE International Conference on Intelligent Robots and Systems* Iros (2014), pp. 4630–4636. ISSN: 21530866. DOI: 10.1109/IROS.2014.6943219.

[54] Alex Russell, Garrick Orchard, and Ralph Etienne-Cummings. "Configuring of spiking central pattern generator networks for bipedal walking using genetic algorithms". In: *Proceedings - IEEE International Symposium on Circuits and Systems* (2007), pp. 1525–1528. ISSN: 02714310. DOI: 10.1109/iscas.2007.378701.

[55] Hansol X. Ryu and Arthur D. Kuo. "An optimality principle for locomotor central pattern generators". In: *bioRxiv* (2019). DOI: 10.1101/2019.12.30.890152.

[56] Shugen Ma, Hiroaki Araya, and Li Li. "Development of a creeping snake-robot". In: *Proceedings 2001 IEEE International Symposium on Computational Intelligence in Robotics and Automation (Cat. No.01EX515)*. 10751081. IEEE, 2001, pp. 77–82. ISBN: 0-7803-7203-4. DOI: 10.1109/CIRA.2001.1013176. URL: http://ieeexplore.ieee.org/document/1013176/.

[57] Alex Spaeth et al. "Neuromorphic Closed-Loop Control of a Flexible Modular Robot by a Simulated Spiking Central Pattern Generator". In: *2020 3rd IEEE International Conference on Soft Robotics, RoboSoft 2020* (2020), pp. 46–51. DOI: 10.1109/RoboSoft48309.2020.9116007.

[58] Yoichiro Sugiyama, Shinji Fuse, and Yasuo Hisa. "Central pattern generators". In: *Neuroanatomy and Neurophysiology of the Larynx* (2016), pp. 109–123. DOI: 10.1007/978-4-431-55750-0{_}14.

[59] Guangzhi Tang, Arpit Shah, and Konstantinos P Michmizos. "Spiking Neural Network on Neuromorphic Hardware for Energy-Efficient Unidimensional SLAM". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2019), pp. 4176–4181. DOI: 10.1109/IROS40897.2019.8967864. URL: https://ieeexplore.ieee.org/document/8967864/.

[60] Matthew Tesch et al. "Parameterized and scripted gaits for modular snake robots". In: *Advanced Robotics* 23.9 (2009), pp. 1131–1158. ISSN: 01691864. DOI: 10.1163/156855309X452566.

[61] Matthew Travers, Julian Whitman, and Howie Choset. "Shape-based coordination in locomotion control". In: *International Journal of Robotics Research* 37.10 (2018), pp. 1253–1268. ISSN: 17413176. DOI: 10.1177/0278364918761569.

[62] Matt Travers et al. "Shape-based compliance in locomotion". In: *Robotics: Science and Systems* 12 (2016). ISSN: 2330765X. DOI: 10.15607/rss.2016.xii.020.

[63] Ian D Walker, Howie Choset, and Gregory S Chirikjian. "Snake-Like and Continuum Robots". In: *Springer Handbook of Robotics*. Ed. by Bruno Siciliano and Oussama Khatib. Cham: Springer International Publishing, 2016, pp. 481–498. ISBN: 978-3-319-32552-1. DOI: 10.1007/978-3-319-32552-1{_}20. URL: https://doi.org/10.1007/978-3-319-32552-1_20.

[64] Zhelong Wang, Qin Gao, and Hongyu Zhao. "CPG-Inspired Locomotion Control for a Snake Robot Basing on Nonlinear Oscillators". In: *Journal of Intelligent and Robotic Systems: Theory and Applications* 85.2 (2017), pp. 209–227. ISSN: 15730409. DOI: 10.1007/s10846-016-0373-9. URL: http://dx.doi.org/10.1007/s10846-016-0373-9.

[65] Julian Whitman et al. "Snake Robot Urban Search after the 2017 Mexico City Earthquake". In: *2018 IEEE International Symposium on Safety, Security, and Rescue Robotics, SSRR 2018* (2018). DOI: 10.1109/SSRR.2018.8468633.

[66] R. Worst and R. Linnemann. "Construction and operation of a snake-like robot". In: (2002), pp. 164–169. DOI: 10.1109/ijsis.1996.565065.

[67] Ke Yang et al. "Simulation platform for the underwater snake-like robot swimming based on Kane's dynamic model and central pattern generator". In: *Journal of Shanghai Jiaotong University (Science)* 19.3 (2014), pp. 294–301. ISSN: 19958188. DOI: 10.1007/s12204-014-1502-x.