School of Computing
FACULTY OF ENGINEERING AND
PHYSICAL SCIENCES

Final Report

A Neuromechanical Approach to Exploring the Steering Behaviours of *Caenorhabditis elegans*

Saul Cooperman

Submitted in accordance with the requirements for the degree of BSc Computer Science

2023/2024

COMP3931 Individual Project

The candidate confirms that the following have been submitted.

Items	Format	Recipient(s) and Date
Final Report	PDF file	Uploaded to Minerva (02/05/24)
Link to online code repository	URL	Sent to supervisor and assessor (02/05/24)

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of Student) 

Summary

Problem

This study aimed to address the challenge of understanding and replicating the complex steering behaviours of the nematode *Caenorhabditis elegans*, specifically focusing on its chemotaxis in various environments. Despite the extensive body of research on *C. elegans*, integrating neuromechanical models to simulate its natural steering responses under controlled conditions remains a popular area of research. The goal was to enhance our understanding of the biological and mechanical principles underlying the nematode's movement. A neurologically correct steering circuit was implemented alongside a previously implemented locomotion circuit in the `simple-worm` framework.

Achievements

- `Integration of Sensory and Neural Mechanisms`: Combined multiple existing computational models into a cohesive neuromechanical model that successfully simulates the steering behaviour of C. elegans using a steering circuit integrated into the head of the model. Enhanced the model's realism by incorporating detailed sensory input processing through ASE neurons, enabling the model to respond to simulated changes in NaCl concentration gradients effectively.
- `Evolutionary Algorithm`: Applied evolutionary algorithms to optimize the parameters of the steering circuit, ensuring that the model's performance is robust across various simulated conditions.
- `Simulation in Varied Environments`: Evaluated the performance of the worm model by simulating it in diverse environments and quantitatively analysing the results.
- `Technical`: The development involved thorough testing with well-designed and documented classes and methods. A rigorous review of the model's space and time complexity was conducted.

This work sets a foundation for future research to explore more intricate behaviours of *C. elegans*, potentially leading to breakthroughs in understanding neural circuits and their applications in computational biology.

Acknowledgements

First and foremost, I am extremely grateful to the University of Leeds, particularly the School of Computing, for the education they have provided me over the past three years. Much of what I learned there was directly applied throughout this project.

I would like to extend my deepest gratitude to my supervisor, Professor Neta Cohen, for her unwavering support and invaluable insights throughout this project, especially during challenging times. Additionally, I am grateful to Dr. Rafael Kuffner dos Anjos for his sound advice on the writing and structuring of this dissertation.

Furthermore, I would like to thank my family and friends for their support throughout this project. Their encouragement and backing were invaluable to me. Finally, I would like to express

my gratitude to Larry Page and Sergey Brin for developing Google Search. This tool has been instrumental in facilitating the research for my project.

Contents

Chapter 1

Introduction and Background Research

1.1 Introduction

For an organism with only 959 cells and 302 neurons, Caenorhabditis elegans - commonly referred to as *C. elegans*, has garnered huge amounts of scientific attention. The organism is approximately 1mm in length (Wood, 1988), is transparent, and can be found in soil environments. The population consists of male and hermaphrodite nematodes, with a large male majority (Herman, 2005). It was first discovered in 1900 by Emile Maupas (Nigon and Félix, 2005) although it wasn't until the late 20th century with Sydney Brenner (Brenner, 1974) that the worm became a quintessential model organism in biological research.

So, what makes this organism so suited for scientific research? Its short lifecycle of 2-3 weeks, with a generation time of 3 days (Wikipedia, 2024), allows for the continuous and reliable testing of the worm. The worm is also genetically tractable, meaning that it's genetics can be easily altered, and the subsequent effects can be observed. Additionally, the small number of cells means that the organism has been mapped out in its entirety on a cellular (Sulston et al., 1983) and neurological level (Cook et al., 2019). However, despite the worm's simplicity, significant complexity arises in the behaviours of the worm that has yet to be understood, even today. *C. elegans* has demonstrated its rich behavioural repertoire, including crawling, swimming, social feeding, egg-laying, habituation, and associative conditioning (Izquierdo and Beer, 2015). Its simplicity, combined with its ability to exhibit complex behaviour, places it in a unique position to provide deep insights into the fundamental mechanisms of life.

Notably, the *C. elegans* worm displays a variety of taxes, or notable behavioural changes in response to external stimuli (applied for both attraction and repulsion) (Online, 2022). For example, in general, the *C. elegans* moves in a sinusoidal shape using undulations in its head, neck and body. However, despite its small size, it remarkably possesses the ability to change its gait when adapting to its surroundings. In aquatic environments, it swims by using longer wavelengths with lower frequencies. However, in more viscous environments, such as agar, it crawls with shorter wavelengths of higher frequency. This key characteristic is known as the swim-crawl transition.

Another taxis[1] which the worm demonstrates which is of particular interest is chemotaxis, the ability to navigate according to the concentration of a certain substance in its surrounding environment. In the case of the *C. elegans* worm it is used for a variety of tasks, included searching for food and avoiding harsh conditions. For example, Tomioka et al. (2006) placed starved worms in an area of low NaCl concentration. They found that, through associative learning, the *C. elegans* became attracted to higher concentrations of NaCl, showing a strong bias towards areas of higher NaCl concentration.

Over decades of research, many full-body computational models of the *C. elegans* worm have been developed. The aim of these models is to accurately represent the behaviour of the

[1] Singular form of taxes.

real *C. elegans* worm, giving researchers the ability to run significantly more tests in the time it would take to use the biological equivalent, particularly if laboratory conditions are hard to replicate. They often have the potential to be more cost effective in the case where laboratory equipment can be expensive. Ultimately, by developing a complete model, we will have insights into how the worm works on a fundamental level, which could ideally map onto other animals and, finally, humans.

Klinotaxis is a form of chemotaxis that involves orientation changes through alternating side-to-side movements of part or the entire body. This mechanism involves comparing the intensity of stimuli from one position to another, leading to a decision-making process between the two.

This paper aims to combine several different models into a unified neuromechanical model of the worm that exhibits additional chemotaxing behaviour. By integrating a steering circuit into the head of a *C. elegans* model, we successfully attempted to replicate the key behaviour of klinotaxis in *C. elegans*, evaluating the performance of the worm model by placing it in a variety of simulated environments with varying materials and chemical gradients.

1.2 Applications

The simplicity of *C. elegans* makes it an ideal model for advancing our understanding of more complex organisms. Through analysis of its behaviours, we can further understand neuroplasticity, learning, and memory in humans. Additionally, the *C. elegans*'s ability to adapt to different environments gives insight into how organisms in general adapt to their surroundings. Being able to model *C. elegans*'s behaviour in a finely tuned environment is extremely advantageous over real experimentation which may require specialised equipment and resources.

More specifically, *C. elegans* experimentation could unlock the key to ageing in humans. It has been used to discover that mutations in the insulin encoding daf-2 gene doubles the lifespan of the worm (Kimura et al., 1997). The same gene, with strong links to ageing, can be found in humans. *C. elegans* also plays a instrumental role in drug discovery (Carretero et al., 2017) and treatment of neurological diseases such as Alzheimer's and Parkinson's (Hughes et al., 2022).

1.3 Steering in *C. elegans*

Despite *C. elegans* only having 95 muscles[2], they demonstrate the ability to efficiently manoeuvre themselves to more favourable environments. The research into the genetic basis of chemotaxis began in the 1970's when Lewis and Hodgkin (1977) observed the correlation between the worm's ability to move towards chemoattractants[3] and the functionality of sensory neurons at the tip of the head. Only in the late 20th and early 21st centuries were extensive neural maps of the *C. elegans* worm made, improving the modelling and analysis of the worm's movement though understanding of the underlying circuitry. This paper discusses a particular circuit used for chemotaxis of NaCl which the worm uses to navigate towards food.

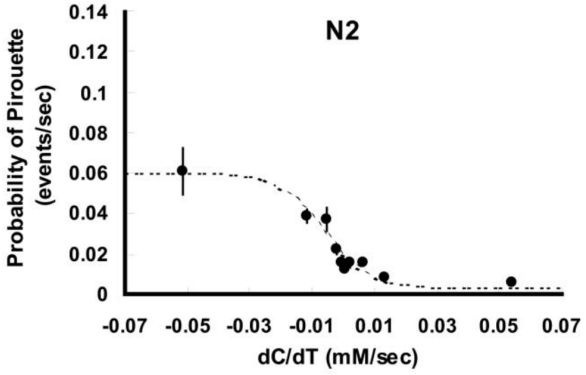
[2]Muscles that run along opposite sides of the worm are known as the dorsal and ventral muscles.
[3]Chemicals that attract *C. elegans*.

1.3.1 Pierce-Shimomura et al.: pirouette strategy

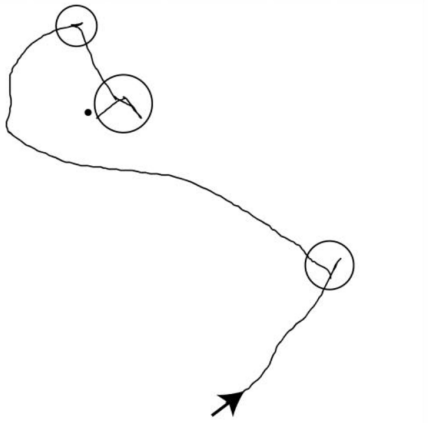
Initially, the mechanism by which *C. elegans* was observed to perform chemotaxis was through pirouettes (Pierce-Shimomura et al., 1999), whereby the worm exhibits a series of sharp turns and omega-shaped reversals. When moving though areas of decreasing concentrations of NaCl, the worm would sharply change direction, aiming to move towards the area of higher NaCl concentration (Figure 1.1). This method can also be referred to as a random bias walk with eventually ending up in an area of higher concentration.

Pierce-Shimomura et al. (1999) used a combination of computer simulations and real worms to analyse the pirouette strategy. Real worms were placed on an agar plate with a NaCl gradient, and their paths were observed with tracking software. They discovered that the worm's locomotion could be divided into periods of smooth motion and periods of sharp turns. The frequency of the sharp turns increased as the worm moved away from the NaCl source and was correlated with the size of the gradient, as opposed to absolute values (Figure 1.1a).

Using the movement statistics of real *C. elegans* worms, Pierce-Shimomura et al. (1999) developed a computer model that deduced the pirouette model was sufficient and general for chemotaxis. However, when attempting to replicate the full chemotaxis driven by the solely pirouette approach, simulated worms did not perform as well as their real counterparts.



(a) Probability of a pirouette occurring given a NaCl gradient. Source: Pierce-Shimomura et al. (1999).



(b) Trace of a worm path. Arrow represents start points and direction. Dot represents concentration source. Source: Pierce-Shimomura et al. (1999).

Figure 1.1: An analysis of chemotaxis in *C. elegans* in a non-uniform NaCl concentration.

1.3.2 Iino and Yoshida: weathervane strategy

Building further upon the work of Pierce-Shimomura et al. (1999), Iino and Yoshida proposed an alternative chemotaxis method known as the weathervane mechanism. This involved smooth[4] forward locomotion towards an area of higher concentration. When simulated alone, the weathervane approach was shown to deliver consistent successful chemotaxis, but not as efficiently when compared to the real worms. As Pierce-Shimomura et al. (1999) did, Iino and Yoshida grew worms in a NaCl gradient and their paths were tracked[5]. Results showed that the

[4]Smooth unlike sharp piroette turns

[5]Tracked using a combination of a microscopes, cameras, and a motorized stage.

worms also used smooth curves to move towards areas of higher concentration thereby proving this to be another successful method of chemotaxis.

Additionally, using cell abolition [6], they confirmed AIZ and AIY neurons play a pivotal role in chemotaxis.

Overall, these two chemotaxis methods were demonstrated to be compatible, both playing a key role in the *C. elegans*'s ability to navigate complex environments. Contextual examples of the weathervane and pirouette mechanism are shown in Figure 1.2.

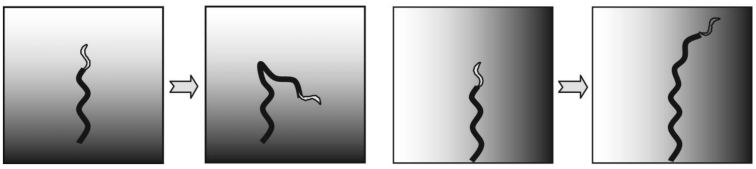


Figure 1.2: Left: pirouette technique to turn around. Right: weathervane technique to veer right. Source: Pierce-Shimomura et al. (1999).

1.3.3 Sensory input

The sensory neurons responsible for chemotaxis are the ASEL-ASER pair located near the head of the worm, which can detect a wide range of soluble chemicals (Altun et al., 2024). This was determined by genetically inhibiting the ASE neurons of *C. elegans* (Suzuki et al., 2008), modified worms demonstrated a significant lack of ability to steer. Both neurons react to changes in concentration of the surroundings as opposed to absolute values. Due to their anterior location on the worm, the required sensory information is gathered through side-to-side motion of the head, up steps and down steps in concentration changes, perpendicular to the head, are detected and processed (Figure 1.3a).

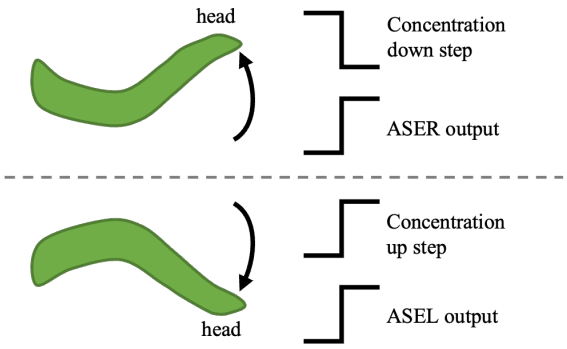
Despite the overall bilateral symmetry of the *C. elegans* worm (Schulze et al., 2012), ASEL and ASER exhibit asymmetric properties. ASEL is an ON cell, activated by increases in NaCl concentration. Conversely, ASER is an OFF cell, activated by decreases in NaCl concentration. The individual behaviours of each ASE neuron are slightly more complex; ASER not only increases in activation with decreases in NaCl concentration, but also decreases in activation with increases in NaCl concentration. ASEL does not display the same contrasting effect with decreases in NaCl. Figure 1.3b shows the response of each neuron to increases and decreases in NaCl concentration. Additionally, the cells exhibit other asymmetries such as different sensitivities to Na^+ and Cl^- ions (Bargmann, 2006)s.

1.4 Computational models

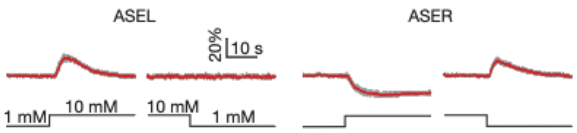
Computer modelling has been deeply ingrained into the history of biological research since Wiener (1948) pioneered modelling of biological processes in his book Cybernetics. As Moore's law continues to hold true (James, 2022), using computer models to investigate the behaviour of an organism is increasingly popular[7]. Modelling uses complex equations to simulate an organism

[6] Removal of a particular cell from an organism.

[7] Modelling played an instrumental role in "The Human Genome Project" (Lander et al., 2001) as well as hundreds of publications over the last 70 years.



(a) ASEL and ASER output with changes in concentration.



(b) ASEL and ASER output with changes in concentration. Source: Suzuki et al. (2008).

Figure 1.3: The analysis of the *C. elegans* connectome to derive a functional steering circuit.

in its environment, observing its behaviour. The advantages of a modelling approach are evident: it allows for the reliable and accurate reproduction of results. Moreover, parameters can be adjusted effortlessly, with faster analysis of the effects.

However, these programs are typically computationally demanding due to the complexity of the calculations involved. Extensive simulations can be time-consuming, even on high-end computer hardware and with optimized software. Thus, there is a trade-off between accuracy and time when conducting these simulations. Numerical errors are prone in longer simulations due to the differential equations involved. For smaller simulation lengths e.g. 10s, these errors can be ignored but for longer simulations e.g. 50+s, they amalgamate chaotically. The *C. elegans* models are no different in their approach to simulate the worm. Resolution of the model can take the form of many different variables, common ones include the time step in the program (dt), and the resolution[8] of the worm's body (N).

1.4.1 Boyle et al.: Proprioceptive control model

Prompted by studies indicating that the transition from swimming to crawling may reflect a consistent fundamental behaviour across varying environments, Boyle et al. (2012) developed a model of the worm in C++[9] which emulated the forward locomotion without a signal generator. It used proprioception to achieve this, whereby the neurons received sensory feedback from nearby muscle stretch receptors. Ji et al. (2023) showed that these proprioceptive signals are essential for locomotion in *C. elegans*.

The model consisted of a physical construction controlled by a neural circuit. The physical model comprised of springs, dampers and fixed-length rods and operated using a basic physics engine (Figure 1.4 a,b). The neural circuit was comprised of 12 repeating units (Figure 1.4 c), each unit controlling 4 virtual muscles on both the ventral and dorsal side.

The model proved that proprioception alone was sufficient to drive the worm, successfully creating the sinusoidal forward locomotion patterns naturally. When placed in a simulated water/agar environment, it demonstrated its ability to capture the swim-crawl transition. Additionally, parameters were chosen to reflect the neurobiology and physics of the worm, by

[8]Number of points/entities used in the modelling.
[9]SUNDIALS was used as the equation solver.

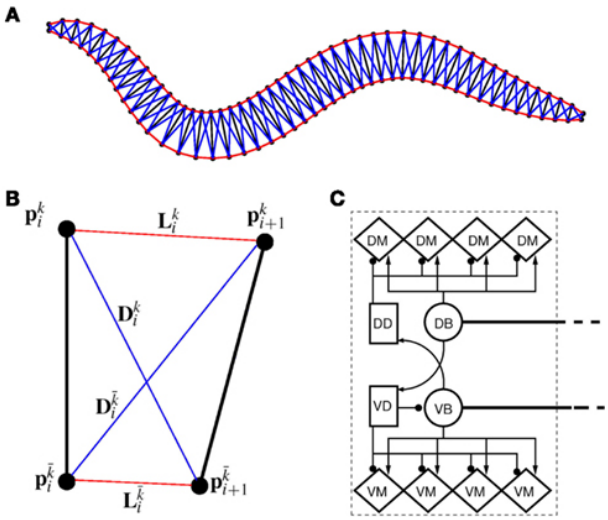


Figure 1.4: (A) Physical model. (B) Individual segment of physical model. (C) One of 12 repeating neural units. Source: Boyle et al. (2012).

comparing it with real experimental data.

Despite its success, there are drawbacks to the model. The physical model is not representative of the real *C. elegans* worm's internal structure, although it does offer an approximation sufficient to prove the initial hypothesis. Moreover, due to the number of entities being modelled, the computations required for the physical model are especially intense.

1.4.2 Denham et al.: Continuous Viscoelastic Model

To further extend upon the work of Boyle et al. (2012), a new model was developed by Denham et al., using C++[10] to discover the extent of the control that proprioceptive feedback has in forward locomotion. Denham et al.'s model differed from Boyle et al.'s model by modelling the worm as a continuous incompressible viscoelastic shell which tapered at both ends.

The worm's midline was segmented into N segments which would pivot at either end. Dorsal and ventral muscles of the worm can be interpreted as curvatures at points along the worm as muscles act antagonistically; one muscle relaxes as the other contracts. Muscle activations were modelled as a continuous function along the worm's body. Despite a simplified model compared to Boyle et al. (2012),it nonetheless demonstrated the ability to replicate the swim-crawl transition.

Denham et al.'s work was built on the findings of Boyle et al. and was a large success, further deepening our understanding of proprioceptive control in *C. elegans*. It found that that, during undulatory locomotion, this proprioceptive feedback has the ability to suppress neuromechanical phase lags[11]. It further detailed how the worm adjusts its gait based on changes in its surroundings.

The width of the worm at any given point was assumed to be fixed, which allowed for all internal and external forces to be collapsed to the midline and solved. Thus, this model has significant computational benefits over the Boyle et al. (2012) spring-damper model.

More recently, Ranner (2020) reimplemented the model using a similar approach with

[10]BLAS and Suiteparse used for equation solver.
[11]Time or phase difference between neural signals and mechanical changes

Python[12]. It extended the work to 3 dimensions and utilised viscoelastic rods[13]. Although Python is slower than C++, it offers a customisable interface as well as easy extensibility. The implementation took the form of a framework known as `simple-worm`.

1.4.3 Smyth: Neuromechanical model

Smyth (2023) took a step towards a complete neuromechanical model of *C. elegans* by attempting to unify the Ranner (2020) and Boyle et al. (2012) models. The complete[14] implementation and parameters of the neural circuit developed by Boyle et al. (2012) were injected into `simple-worm`. Smyth constructed a way of converting the muscle lengths into preferred curvature. Proprioceptive input was used to drive the forward locomotion in the worm, rather than a oscillatory input, and the muscles were modelled using equations from Denham et al. (2018).

The model succeeded in replicating the swim-crawl transition, however, the paper addressed, but didn't fix, several issues. By copying parameters into a different model without adjustment, subtle bugs may have been introduced into the model by Smyth, due to the underlying differences in the respective implementations of each model. Notably, Smyth's model inherited issues from `simple-worm`, whereby the worm exhibited asymmetrical forward locomotion. This error is further amplified with changes in the environment variable K which is discussed in Chapter 3, although, it may be argued that worms rarely move in an exclusively straight line and so this behaviour may be ignored in some situations.

Running `simple-worm`

The `simple-worm` framework offers a high customisable modelling interface. After importing the `simple_worm.worm` module the user can create a `Worm` object, it contains all the physical information for a singular simulated worm. The number of segments in the model (N) and timestep (dt) are the only required parameters for its construction. Additional optional parameters may be specified, including a `neural_control` flag which specifies if the worm should use the neural circuit.

Several methods are available in the `Worm` class. Executing the simulation is done with the `solve()` method. The only required parameter, T, defines the length of the simulation in seconds. This is also where the material parameters are provided in the form of a `MaterialParameters` object and where log output filenames are defined. After the simulation ends, `solve()` returns the simulation results in the form of a `FrameSequenceFenics`, a data structure which holds all the historical physical information of the worm throughout the run. The `to_numpy()` class method is used to convert it to a more usable NumPy format. Once in this format, the data can be distributed to a wide range of available visualisation functions in `plot2d.py` and `plot3d.py`. The development in this project was done in a forked copy of Smyth's version of `simple-worm`.

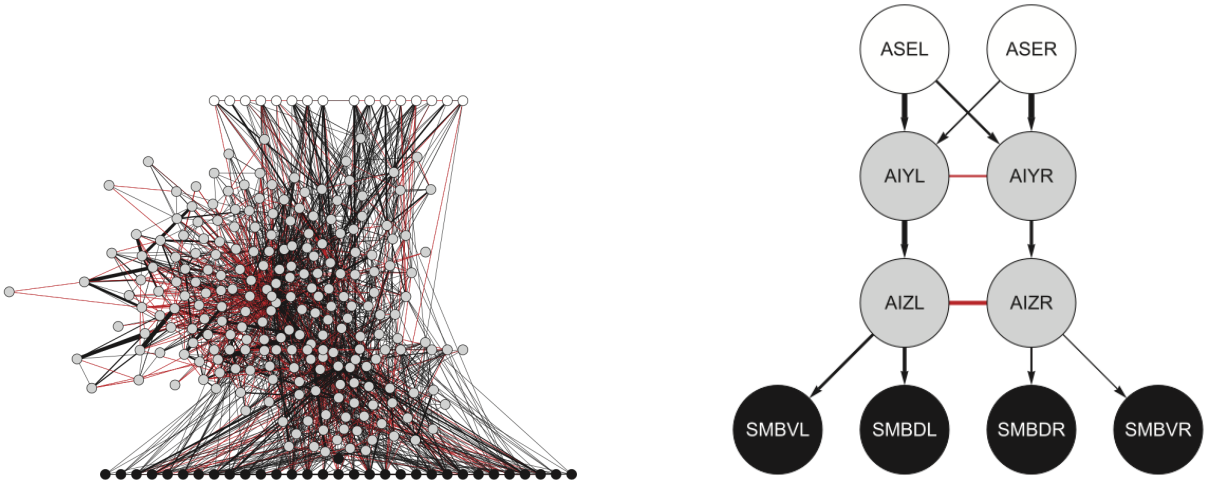
1.4.4 Izquierdo and Beer: Neuromechanical steering model

Izquierdo and Beer (2013) discovered a head and neck circuit which could be used for chemotaxis

[12]FEniCS used as equation solver.
[13]Uses a branch of mathematics known as Kirchoff rods.
[14]A few minor adaptations were made to the code.

in the *C. elegans* worm, using a largely complete[15] neural map of the worm known as a connectome, isolating the key neurons responsible for the chemotaxis. Using data-mining[16], the connectome was analysed and neurons responsible for chemotaxis were isolated. Figure 1.5a and 1.5b show the initial connectome and the post-data-mined network. The resulting neurons connected the head and neck muscles and ASE sensory neurons, using intermediary neurons. Unlike previously discussed models which were used proprioceptive input, the worm was powered with a continuous sinusoidal input.

AIZ and AIY were used for interneurons, which aligns with Luo et al.'s 2014 further discovery that they play a key role in *C. elegans* NaCl chemotaxis. Similar to Pierce-Shimomura et al. (1999), Luo et al. used cell abolition and AIZ and AIY neurons were identified as crucial to both positive and negative chemotaxis[17].



(a) The initial connectome with 300+ neurons. Source: Izquierdo and Beer (2013).

(b) The final steering circuit after data mining. Source: Izquierdo and Beer (2013).

Figure 1.5: The analysis of the *C. elegans* connectome to derive a functional steering circuit.

The approach, although not explicitly mentioned in the paper, used the weathervane strategy for the chemotaxis, aligning with the theory. Pirouette reversals were not considered in the model due to the model's circuit simplicity. This finding demonstrates that a simple circuit with few components can successfully enable the worm to perform chemotaxis. Additionally, it reveals that a steering circuit confined solely to the head and neck regions is adequate for chemotaxis. Figure 1.2 demonstrates how the worm was able to successfully navigate to a concentration source with 100% accuracy.

Expanding further on their 2013 work, in 2015 Izquierdo and Beer iterated upon their previous steering model. The head circuit used was almost identical to the one in 2013. However, this time there was no signal-generated oscillatory input and the worm's locomotion was orchestrated by self-connections in the motor neurons. Furthermore, the parameters were symmetrical along the dorsal-ventral line in the 2015 model, whereas the 2013 model did not impose this symmetry.

[15]The data, compiled by White et al. (1986) is missing connectivity data for 39 of the 302 neurons, including the most posterior 21 of the 75 motor neurons. As of 2019, a full connectome is available.

[16]Data-mining involved finding pathways between ASE neurons and head and neck muscles. Path length was capped at three

[17]Moving towards and away from a stimulus.

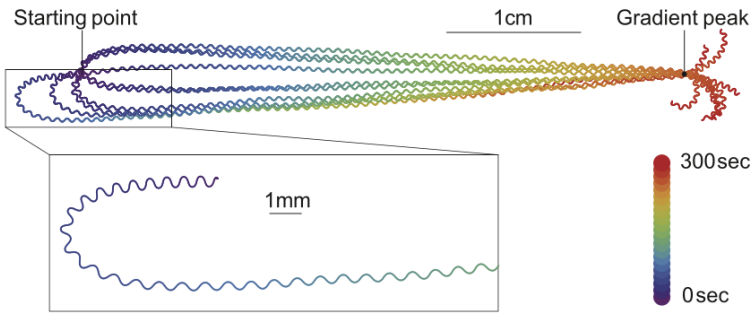


Figure 1.6: Nine worms initialised at random angles, navigating towards a concentration source. Source: Izquierdo and Beer (2015)

For both models the worms were trained using an evolutionary algorithm to optimise the steering circuit's parameters. Both models used 100 initial networks and evolved them over 300[18] (2013) and 200 (2015) generations. 17 of the 100 networks in the 2013 model failed to demonstrate chemotaxis. The 2015 model selectively chose the best performing models for further testing, disregarding the remaining majority.

Using an evolutionary algorithm creates two issues however, the first being the disconnect between the real *C. elegans*'s biological parameters and those of the model. The second is that the circuit is subject to overfitting, losing its flexibility, so it cannot be easily integrated into other models without significant adjustment.

1.5 Objectives and aims

This project aims to explore the steering circuit proposed by Izquierdo and Beer when implemented alongside Smyth's neural adaptation of `simple-worm`. We will implement the steering circuit within the framework. To ensure ease of use for the framework's clients, all design and development plans will be thoroughly discussed in accordance with best practice.

Our primary aim is to develop a model which demonstrates correct chemotaxing behaviour (positive or negative) when placed in a chemical environment with a simulated NaCl gradient. The movement and path of the worm will be analysed using the pre-existing visualisation tools, as well as other tools developed by us.

An evolutionary algorithm will be used to explore the solution space for the parameters of the steering circuit in a similar way to Izquierdo and Beer. The performance of a set of parameters will be examined quantitatively to uphold impartiality. Once the best performing steering circuit is isolated, we will test it in a wide range of chemical and physical environments to examine its performance and behaviours, as well as attempting to dissect the underlying neural behaviours.

The overarching idea in this research field is to create a neuromechanically complete model of the *C. elegans* worm to aid with hypothesis testing and general discovery. This paper presents itself as a steppingstone towards this ultimate goal.

[18] No dorsal-ventral symmetry so more parameters required to optimise.

Chapter 2

Methods

2.1 Design

The primary aim of this project was to implement a steering circuit within Smyth's integrated neuromechanical model, granting the worm the ability of chemotaxis. As with any biological modelling exercise, abstractions are necessary to simplify the process. However, with this implementation, many of the design decisions were based on the real *C. elegans* worm so as to maximise the preservation of its biological characteristics.

The efficiency of a model also plays an important role in its measure of success there during implementation unecessary expensive calculations were minimised. Additionally, as the `simple-worm` framework is used for a wide range of research purposes, the implementation was designed to be extensively customisable but also intuitive to use with the help of easy-to-use functions and classes; all whilst ensuring robustness. To help analyse and understand the behaviour of the worm in different environments, the development of the project also included appending to the already existing visualisation functions available in the framework. To optimise the parameters of the circuit, an evolutionary algorithm was used, the variables of the algorithm had to be specifically designed to deliver results with a reasonable computation time. Finally, as we were building upon a pre-existing framework, it was imperative that all changes were backwards compatible with all new features being clearly documented.

2.1.1 Circuit

The steering circuit was implemented on top of Smyth's neural model. Implementation of the circuit was heavily inspired by the work of Izquierdo and Beer's 2015 model; adaptations were made to fit the circuit into Smyth's model without extensive modification of the original work.

The two ASE sensory neurons were designed to mimic the key behaviour of their real counterparts, activating during concentration changes as opposed to absolute concentration values. Neurons were modelled using the following instantaneous function of a derivative operator which was applied to the recent history of the NaCl concentration detected by the sensors.

$$y_{\mathrm{ON}}(t) = \frac{\sum_{t-N}^{t} c(t)}{N} - \frac{\sum_{t-(N+M)}^{t-N} c(t)}{M} \tag{2.1}$$

as defined by Izquierdo and Beer (2013), where $c(t)$ is the concentration at time t, and N and M represent the two intervals in which the concentration is being averaged. In the case of an OFF cell (ASER) the signs were inverted so that decreases in concentration would yield positive activations. Negative activations were set to 0 for both ON and OFF cells. The interneurons were modelled in the following way:

$$\tau_i \frac{dy_i}{dt} = -y_i + \underbrace{\sum_{k=0}^{N} w_{ji}\sigma(y_j + \theta_j)}_{\text{Sum of chemical synapses}} + \underbrace{\sum_{k=1}^{N} g_{ki}(y_k - u_i)}_{\text{Sum of electrical synapses}} + I_i \tag{2.2}$$

as defined by Izquierdo and Beer (2013), where y represents the membrane potential also known as neuron activation. It is relative to its resting point so it can take both positive and negative values. θ represents the bias term that shifts the sensitivity of the neuron. w_{ij} represents the strength of the chemical synapse between neurons i and j. Term g represents the electrical synapse (gap junction). The final term I represents other external input. The gap junctions (also known as electrical synapses) are shown to play an important role in locomotion and touch withdrawals behaviour in *C. elegans* (reference in ensemble) and hence are included.

The sigmoid function $\sigma(x) = 1/(1 + e^{-x})$ was used to represent the neuron activation as it replicates the activation threshold behaviour of biological neurons (Rumelhart et al., 1986), only firing when above a specific value. Emulating the non-instantaneous nature of real neurons is done using the time constant variable, it limits how fast the potential of a neuron can change and is explained later in more depth. This is unlike Smyth's model where the connections are instantaneous. The circuit is implemented within the `SteeringCircuit` class which accepts an optional `SteeringParameters` object with the steering parameters.

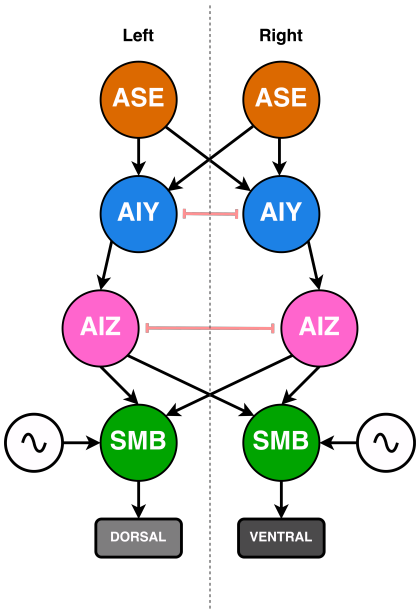


Figure 2.1: The steering circuit adapted from Izquierdo and Beer (2015). Arrows represent chemical junctions between neurons. Red lines represent gap junctions (electrical synapses). ASE neurons are sensory. AIZ and AUY are interneurons. SMB are motor neurons connected to dorsal and ventral muscles. Signal symbol represents Smyth's original circuit motor input.

Rather than redefining new motor neurons for the head and neck, the pre-existing motor neurons provided by Smyth were used. We used the formula 2.3 below to adapt these motor neurons, including the steering circuit output from the AIZ neurons.

$$O_{New} = O_{Old} + \underbrace{\sum_{k=0}^{2} w_{ij}\sigma(y_j + \theta_j)}_{\text{Summing the 2 AIZ neuron outputs.}} \tag{2.3}$$

The proprioceptive TRPC channels used by Smyth to generate the undulations are also present in the head of *C. elegans* (Yeon et al., 2018), thereby ensuring the models validity. Figure 2.1 shows the slightly modified circuit implemented; the oscillatory input represents the initial oscillatory movement generated by the proprioceptive feedback[1].

2.1.2 Evolutionary algorithm

An evolutionary algorithm was used to discover a set of parameters which succeeded in meeting our criteria by steering the worm in the direction of increasing gradient. An evolutionary approach was used over a mechanics based hard coding of parameters, as it allowed us to define a more accurate biological model of the worm similar to how evolution has evolved it. It also may lead to unforeseen new chemotaxis behaviours which couldn't be predicted. Environment variables and fitness functions were identical to the ones used by Izquierdo and Beer (2015) however several changes were made to the variables of the algorithm to facilitate computational limitations. The worm was placed in an environment with a concentration source emitting a linearly decreasing radial signal at 4.5cm from the worm. The concentration at any point $(x(t), y(t))$ was $c(t) = -\alpha\sqrt{x(t)^2 + y(t)^2}$. Each worm was run in 3 simulations where the worm was placed at angles of $\pi/2, 0$ and $-\pi/2$ to the worm (up/forward/right). A single-points concentration source is representative of a food source which may be found in the natural habitat of *C. elegans*.

These 3 points were selected because they each represent key behaviours which the worm must exhibit. Sources at $-\frac{\pi}{2}$ and $\frac{\pi}{2}$ were chosen to assess the ability of the worm to change direction left and right. The concentration source ahead of the worm was chosen to remove false positives, establishing that the worm was actively making the left-right decision.

The performance of a worm ought to be captured in a quantitative way. A fitness function was chosen, which captures if the worm is traveling towards the goal (Figure 2.2). The fitness function is the average bearing between the line worm-head⇒start-position and start-position⇒concentration-source ((2.4)) over the simulation, with the overall fitness value equal to lowest of the 3 runs.

$$f \;=\; 1 - \frac{1}{T}\int_0^T \frac{a(t)}{\pi}\,dt \tag{2.4}$$

as defined by Izquierdo and Beer (2015).

Izquierdo and Beer (2015) ran each simulation for 50 seconds[2] per environment, totalling 150 seconds. This was computationally unfeasible without resources, hence a population size of 60 over 100 generations was used, where each run was only 15 seconds per environment, totalling 45 seconds. This dramatic decrease in simulation time is justified as Izquierdo and Beer (2015) had different aims for their model than we did. They attempted to recreate a full chemotaxis

[1]Not using a signal generator.
[2]Time in simulation, independent of execution time.

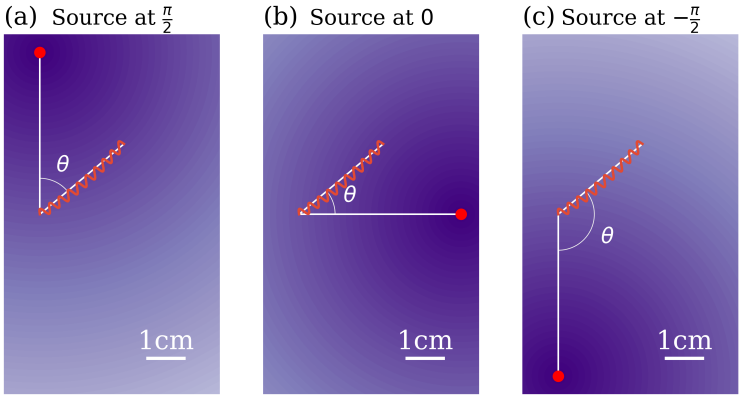


Figure 2.2: Visualisation of fitness function. Red dot denotes concentration source. Red line represents path of worm. Angle is the angle calculated for the fitness function for each given source. Gradient in background represents NaCl concentration, colour saturation proportional to concentration.

behaviour. We are concerned with the replication of certain key mechanisms that form a part of the chemotaxis, hence less simulation time is required. The variables of the evolutionary algorithm (survival, mutation, population size) were researched, balancing execution time and solution quality.

At the end of each generation, the current worm population was saved to a CSV. As the training may take a long time[3] backups were essential to mitigate the risk of losing data due to system crashes and unhandled errors. The CSV can be supplied to the algorithm at the start, so it can pick up the evolution where it previously finished. If no file is provided, the algorithm starts from scratch and a new population is randomly initialised.

2.2 Development

The development of this project was done over a timeline of 6 months. Incremental progress was made on the project with an emphasis on consistent and thorough testing throughout. Although the model uses a 3D environment, the steering phenomena being replicated are entirely 2D. Additionally, most experiments done with *C. elegans* regarding chemotaxis are only concerned with the planar dimensions. Hence, we only test and visualise the worm in 2D but develop in 3D to ensure future-proofed code. All code changes to the framework can be found in Appendix F.

The direction of the project shifted in several directions over 6 months. Hence, deliverables were not adhered to as strictly as possible. A large proportion of time was dedicated to research before writing, testing, and running the code.

2.2.1 Environment

All the development of this project was done on a MacBook Air M1 2020 (Appendix C). FEniCS, a popular open-source computing platform for solving partial differential equations, is used by `simple-worm` to solve the necessary differential equations and is not easily compatible with the Apple M1 chip. Running the code on our system required using a specifically tuned x86 Conda

[3]Longer runs took place over days.

environment running Python 3.9 as well as other libraries with specific versions. To get the functionality of `simple-worm` that was required, local support for fenics-adjoint was removed. This prevents some parts of `simple-worm` from being able to run on the system although it can be fixed by reinstalling fenics-adjoint on a system that supports it. We verified core calculations and simulation of the worm remained unaffected.

2.2.2 Visualisation

Worm

Throughout development, visualisation tools were a key method of interpreting positional data which plotted the midline of the worm. Visualisation was favoured over numerical analysis when assessing the movement of the worm because interpreting the patterns numerically doesn't yield as much insight. Smyth implemented the `plot2d.py` file which offers basic plotting functionality such as drawing a singular worm frame, or generating an MP4 video of the worm in the simulation. Each function offers a high degree of customisation to suit the users' requirements but are simple in nature.

New functionality was appended alongside Smyth's work by implementing additional visualisation functions. With the development of each new feature, emphasis was placed on scalability, usability, and customisability. The static function `multiple_FS_to_clip()` accepts an array of tuples, containing the FEniCS frame data (FrameSequenceNumpy) and the name of each simulated worm. It also accepts arguments which specify the axis bounds and output filename for the video. The function generates an MP4 of the worms travelling, all overlayed on top of each other on one grid. This is incredibly useful when comparing the effect that changing of parameters can have on different worms.

In dynamic data, such as MP4 videos, patterns can be easily overlooked. Hence, a static visualisation function `plot_worm_paths()` was developed. It accepts the same parameters as `multiple_FS_to_clip()` but generates a singular plot containing the path each worm took and saves it as a PNG file. The path is traced using the anterior most coordinate of the worm, as it is assumed the rest of the worm follows in its path.

Neurons

To aid with the analysis of the neural circuit, we created a GUI[4] application which would allow for neural activation data of the steering circuit to be exported and imported for live visualisation. This proved useful when visualising when neurons were activated in the real environment of the worm. The user can change the circuit parameters using sliders and observe example output in real time with almost instantaneous results.

The application was developed using Matplotlib's rich set of visualisation tools. We chose Matplotlib over Tkinter for the application as it offers powerful built-in plotting capabilities such as panning, zooming, and saving. Due to the ever-evolving nature of the development, the application had to be easy to prototype with less overhead than Tkinter, only requiring several lines of code to create a sufficiently functional application. Less time was spent focusing on

[4]Graphical User Interface

developing the user interface of the application as it was designed for research purposes with a small clientele.

The application consists of 6 graphs[5], 14 sliders and 2 buttons (Figure 2.3). Widgets can be easily added and integrated into the application with their `on_click()` and `on_update()` functions. The parameters of the steering circuit can be supplied as a command line argument with a path to the .ini file. Once the parameters are loaded, they can be adjusted with the sliders and changes in the steering circuit are displayed in real time on the graphs. The graphical data can be exported to a CSV file for further analysis. The newly changed parameters can be saved to the loaded in .ini file. If none exists it creates a new parameters.ini file in the current path.

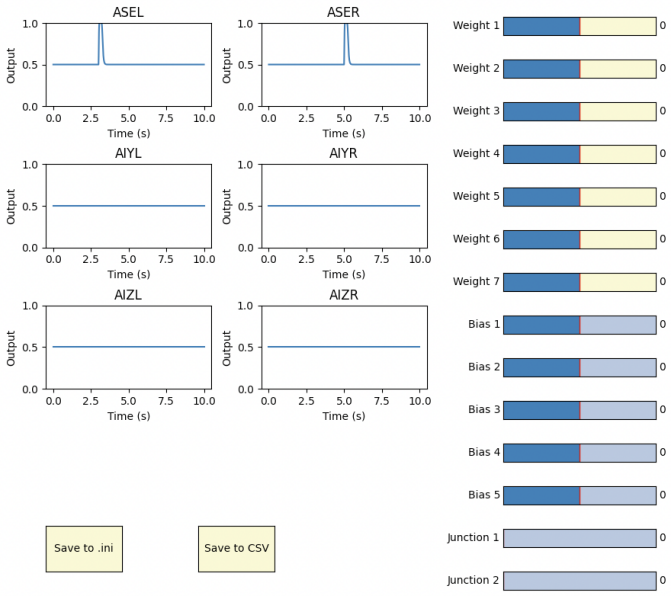


Figure 2.3: A screenshot of the neuron output visualisation Matplotlib application.

Additionally, at the end of every iteration, the output data from each neuron is appended to a CSV. This CSV can be visualised using the `plot_neurons()` function.

2.2.3 Modelling

In any modelling exercise, trade-offs are essential to balance simulation accuracy with execution time. Two main variables which can be changed to adjust the accuracy of the `Worm` model in `simple-worm` are dt and N. Throughout most of development, variables dt=0.01 and N=48 were used. The value of dt was chosen as it accommodated reasonable runtime[6] but still captured the smooth movement of the worm.

An examination was conducted on Smyth's circuit to determine the numerical limits of the implementation. For values of dt < 0.005, the worm displayed incorrect behaviours such as curling into a ball and teleporting sporadically. For final simulations run in Results, values for dt $= 0.005$ and N $= 128$ were used to generate accurate data (Denham et al., 2018) for analysis.

The neural model implemented by Smyth contains many numerical variables copied from Boyle et al.'s code. These parameters used by Boyle et al. are not necessarily compatible with `simple-worm`, hence changes may need to be made. To improve flexibilty and extensiblity of the

[5]3 for left neurons, 3 for right.

[6]Average simulation time of 10 seconds took 30 seconds to execute

code, we refactored the `NeuralCircuit` class to accept a newly created `NeuralParameters` class. It allowed the user to redefine the forward locomotion parameters during the creation of the `Worm` by suppling a `NeuralParameters` object as a parameter. Its design followed closely with the previously created `MaterialParameters` class, ensuring consistency across the framework. `NeuralParameters` significantly improved the reproducibility and adaptability of experimentation done throughout this project.

Materials

During the creation of the `Worm` object in `simple-worm`, a `MaterialParameters` object can be optionally supplied to the `solve()` function as a parameter. Variables that affect both the physics of the worm and its environment can be adjusted[7]. A key parameter is K, defined as the ratio of drag coefficients K_v (normal) and K_t (tangential) (Denham et al., 2018). An increasing K corresponds to an increase in strength of lateral resistive forces due to the viscoelasticity of the containing fluid. The K value for water (K_{water}) is 1.5 and for agar (K_{agar}) is 40.

Drift

Smyth's 2023 model inherited a right-side movement drift. This presented a large issue as we were assessing the ability of the worm to chemotax. Ideally, the worm would move in a straight path so we could analyse its turning behaviour without noise. Significant time was spent attempting to mitigate the drift. We experimented on several symmetrical variables[8], which were chosen so the symmetry of the model was retained. Relative success was found by adjusting the value for AVB from 0.675 to 0.405. Average deviation from the horizontal line was the chosen metric for success. Figure 2.4 shows the path of each worm for different values of AVB. Table 2.1 numerically shows the average deviation distance $\frac{1}{T}\sum_{t=0}^{T}|y(t)|$. With AVB values of 0.1, 0.2 and 0.9 the worm did not locomote using coherent undulations similar to *C. elegans* and hence were disqualified, despite their low scores.

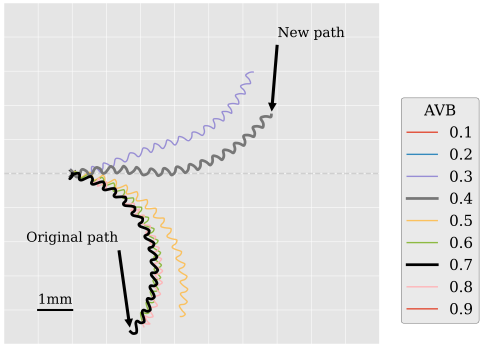


Figure 2.4: The path of each worm for different AVB values. Bold line denotes original extremely curved path.

Table 2.1: Mean deviation for AVB values

AVB value	Mean Deviation
~~0.1~~	0.00000
~~0.2~~	0.00000
0.3	1.08136
0.4	0.40727
0.5	1.56546
0.6	1.72410
0.7	1.97303
0.8	1.84471
~~0.9~~	0.01148

[7]Parameters include: bending modulus and twisting viscosity and modulus
[8]Variables that affect ventral and dorsal side

Parallelisation

Utilising multiple computational cores effectively played a key role in the speedup of execution time. With the framework is attached an `environment.yml` which details the specific software requirements for `simple-worm`. Our development took place on an Apple M1 ARM64 system; meaning processing time is increased when translating the x86 instructions to run on local hardware. Hence, significant simulation time was parallelised with the help of Python's concurrent.futures library.

As our task is CPU-bound (as opposed to I/O bound) a `ProcessPoolExecutor` was used to parallelise the computation. Each task, consisting of the 3 independent runs, was submitted to the executor with the results stored in an array for an evaluation of the entire population. Using `argparses`, `evolution.py` accepts the number of processors to use as a parameter (`-n N` where N is the number of cores). During training, 95% (5% system processes) of the CPU was dedicated to the program by closing all other programs.

2.3 Testing

We used a test-driven approach throughout the project's development. Unit and integration tests were already defined in the `/tests` folder of the framework for code already written so we did the same for consistency. All tests were written with accuracy, precision, and reproducibility in mind. The tests aim to verify that the implementation is correct and robust. We introduced four new test files.

- `steering_circuit.py`: Tests the steering circuit. Includes the Neuron class, sigmoid function and ASE derivative processing.
- `steering_parameters.py`: Tests the creation of `SteeringParameter` object with/without filename and tests the bounds of each parameter.
- `worm_environment.py`: Tests for defining the `Environment` for the worm and adding variables.
- `chemotaxis_plot2d.py`: Integration test for a full worm simulation in a chemical environment with all plot2d visualisations.

Just like Smyth, tests were not written for plot2d functions due to their logical simplicity. Additionally, their correctness was evident in the images and videos generated.

We utilised pytest's powerful testing features so that our code was as watertight as possible. Parameterised testing was heavily employed for function unit tests, ensuring the input domain was sufficiently covered and handled correctly with assertions. The data for the Parameterised testing was calculated with other software[9] using the implemented equations. When testing for numerical accuracy, the pytest `approx()` function was used with an accuracy of 0.0001. Less scrutiny was imposed on the results due to the wide range of environments the code could run on.

The testing of the steering parameters and visualisation function required file creation. We used the `tmpdir` functionality from pytest which allowed us to clean up temporary files with ease after running tests.

[9] Online scientific calculator.

Chapter 3

Results

3.1 Initialisation

3.1.1 Steering circuit

Parameters

Parameters of the circuit were stored in the class SteeringParameters using class variables. There were two methods of non-default initialisation for the class. The first allowed for circuit parameters to be manually defined and supplied with construction parameters. Although this method is functional, it is not recommended due to its tedious nature. The second method is through an ini parameter file. This uses the Python `config` module to generate configuration files. The `SteeringParameters` has `read()` and `save()` as class methods which can be used to read and write to and from the ini files. These configuration files allow for parameters to be changed and swapped around during testing; making the code very adaptable. Just like (Izquierdo and Beer, 2015) the circuit parameters are symmetrical along the dorsal-ventral line. In reality, the circuit parameters are not idential but a symmetrical parameter set was found sufficient for chemotaxis. Additionally, it simplifies the evolutionary algorithm by reducing dimensionality with a smaller solution space, decreasing computation time.

The neural logic for the steering circuit is found in the class `SteeringCircuit`. It contains a constructor which optionally accepts a `SteeringParameters` object as a parameter. If no parameters are supplied, it uses predefined default values. The `SteeringCircuit` is supplied as an optional argument to the NeuralParameters class during the creation of the `Worm`. The steering will only be used if the `STEERING` flag is set to `True`[1].

Network

Code for the steering circuit is contained within the `SteeringCircuit` class. A `SteeringCircuit` object is created when the `NeuralCircuit` object is created at the start of each simulation. The `SteeringCircuit` stores the current states of each sensory and inter neurons, handling the flow of information from concentration inputs to AIZ outputs.

The neurons are respresented by the `Neuron` class which stores its bias and current potential. The ASE neurons and interneurons (AIY, AIZ) each have their own class which inherits from `Neuron`, named `SensorNeuron` and `InterNeuron` respectively. To fetch the output of a neuron the `get_output()` method is used, calculating the output for both neuron types using the formula $\sigma(y_i + \omega_i)$. This can be overidden if necessary when creating other neuron classes.

Each neuron has a time constant which defines how long a neuron takes to reach a certain value. A high time constant means a longer time taken for the potential in a neuron to rise and fall; whereby a shorter time constant leads to more rapid rates of change. The time constant

[1] `STEERING` flag defaults to `False`

used for the neurons is 0.01. This is the approximate value found in real biological neurons as discussed by Gerstner et al. (2014, Sec. 1.3).

To ensure that the calculations for the ASE neurons align with the timestep used, dt is supplied as as paramter during the creation of the `SteeringCircuit` object. At the end of each `NeuralCircuit` iteration, the `update_state()` class method is called on the `SteeringCircuit` with the current concentration as a parameter. The current environment variables are calculated in the `solve()` function and passed to the `NeuralCircuit` when calling its `update_all()` method.

The `update_state()` contains the signal propogation logic for the circuit. Firstly the ASE concentraion input data is processed using the derivative operator, calculating the average differences between the two timeframes. Using a Python collections deque, the concentration values history can be stored for this calculation. A maximum queue size is specified at its creation so that old items are automatically dequed for memory efficiency. This computation is handled by `get_differential()` with the current concentration. Using index slicing of the queue, the values for ASEL are calculated and returned; applying the negation operator gives us the value for ASER.

Once the output of the ASE neurons is established, the information flows though the circuit's AIY and AIZ neurons (Eq 2.2). The AIZ values are appended to the current input used for Smyth's neck and head motor neurons. Izquierdo and Beer attach the circuits to the most anterior 1, 2 and 3 segments to evaluate the effect different muscles had on chemotaxis performance. We struck a balance and attached the circuit to the 2 segments closest to the head.

3.1.2 Worm environment

When simulating the worm, it was necessary to emulate a environment with a non-constant concentration. We implemented a general solution to this with the developement of the `Environment` class[2]. An `Environment` object is created using a no-parameter constructor and can be provided as an optional parameter during `Worm` construction.

An environment variable[3] can be created using the `add_parameter()` function, which accepts the name of your variable and a continous function defining the distribution. The function can be defined using a regular Python function or a lambda expression. The function parameters can optionally accept any combination (including none) of the variables x, y, z and t, referring to the value at positional point (x, y, z) at time t. For our purposes, only the x and y variables were used to represent the concentration at a point, the short timeframes used in the experimentation did not exceed 20s, which meant conditions did not change significantly over time. Variables z and t were included to future-proof the implementation; for longer simulations in 3 dimensions they may be required.

The `get_data_at()` function accepts coordinates and time as parameters, it returns a dictionary containing a list of the variables and their corresponding values for that coordinate and time. To fetch the function for a specific variable, the `get_variable_func()` is used. This function can be provided as an optional variable for several of the visualisation functions created. A NumPy meshgrid is created, using the function to generate an image of the gradient to overlay

[2]Can be found in `worm_environment.py`.

[3]Examples include temperature, concentration or pressure.

onto the graph. For this to work however, the variable function must be capable of accepting both NumPy arrays and individual values.

The visualisation functions developed for this project optionally accept an 'environment function' which is used to assist visualisation of environment variables as colored gradients.

To assist with developmety, the `Environment` class also provides several utility functions which can be used to define environments from a set of pre-defined patterns. The create_linear_gradient function accepts a direction (x or y) and a gradient, with an optional bounding box coordinate parameter[4]. It simulates a gradient moving across an axis decreasing/increasing at a constant rate. For radial gradients, create_linear_concentration_source simulates a single concentration source with a linear or gaussian diffusion pattern. It accepts the coordinate of the location and gradient as parameters. Both function additionally require the environment variable name as a parameter.

At each iteration of the simulation, the current environment variables are calculated using the anterior most point in the worm. This offers a good approximation for the location of real ASE sensory neurons in *C. elegans*. The dictionary is provided to the `update_all()` method of the `NeuralCircuit` which in turn provides the concentration value to the `SteeringCircuit`.

3.2 Evolution

The file containing the evolution algorithm code is in `evolution.py`. The first stage in the algorithm is initialising a randomly generated[5] population using the `create_population()` method. The returned population is stored in an array of `SteeringParameter` objects. Writing the population to a CSV[6] is done using the `write_population` function which accepts a filename for saving as a parameter. Similarly, the `read_population()` method accepts a filename and parses all data into an array of `SteeringParameter` objects for testing.

When evolving the population, the `crossover()` method is used. It accepts two parents and randomly pivots each data type[7], generating two children for the new population. Two parents are randomly chosen repeatedly until the new population is sufficiently large. Due to Python's copy by reference of objects, deepcopies of each parent were made during this process using the `SteeringCircuit copy()` method. This may have presented memory concerns, however, we deemed it insignificant due to the large availability of memory and small size of the objects being copied. Mutations occur in the children at a predefined rate, calling the `mutate()` method with a child as a parameter will randomly choose a parameter to mutate, ensuring the correct ranges are used.

At the start of a population evolution, a CSV file containing the current population's networks was loaded using `read_population()`. The entire file was loaded into memory as opposed to a line by line approach with the loaded population data only using 8kB of memory. However, for larger file sizes a different approach should be used. Iterating over the generated array of `SteeringParameter` objects the networks were evaluated in the three different environment for 15 seconds each. The fitness value of each worm was calculated post-simulation

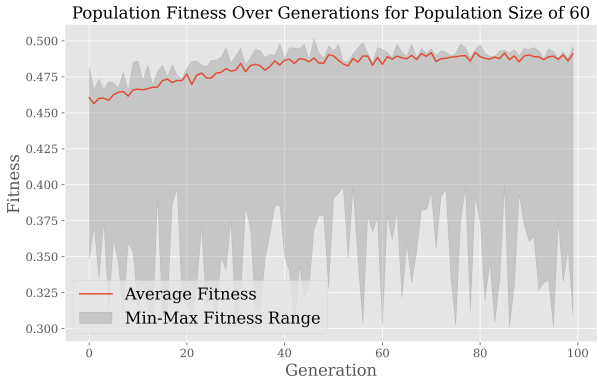
[4]Defines what section of the graph is displayed in the visualisation.
[5]Uniformly generated parameters.
[6]Each row of the CSV corresponds to a different set of parameters, columns are each parameter.
[7]Synapse, bias, junction, time.

using the NumPy frame data. The fitness data for each generation over a run was stored in a CSV for analysis.

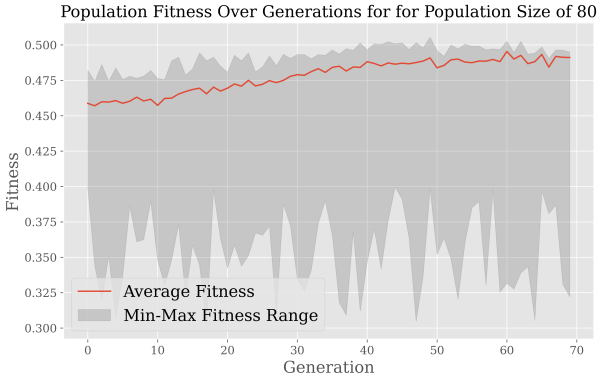
The conflict between computation power and valuable results made itself particularly apparent when executing the algorithm. Several adjustments to the algorithm's variables were made to balance the two. Initially, a population size of 60 was evolved over 100 generations. At the end of each generation, 40% were selected for potential crossover at a 10% mutation rate. Figure 3.1a shows the fitness data of each generation over the training period.

The graph displays the data converging at an average fitness of 0.47577. This is close to the fitness value of the worm without using a steering circuit. For the steering circuit to have any notable effect on the locomotion of the worm, the parameters must be tuned in a certain way. We believe the M and N ASE time variables play a key role in the detection of concentration changes. The data shows several similar solutions dominated over the species which led to a premature convergence and a solution which did not meet our criteria.

In a second version of the algorithm the population size and selection proportion were increased to 80 and 50% respectively, boosting the variance in the population over generations. However, it ran for only 70 generations due to the added computation time required with a larger population. Figure 3.1b displays the fitness data for this run. Worms had a significantly larger variance and converged slower than the previous attempt. Additionally, it yielded several parameter solutions with fitness scores > 0.5, meaning the worm was turning in the correct direction.



(a) 60 generations with popoulation size of 100.



(b) 100 generations with popoulation size of 80.

Figure 3.1: Fitness data for the training of different evolutionary algorithms. Grey area shows range of fitness values for a generation (max/min).

Figure 3.2 shows the different worms in the testing environment superimposed upon one another. Interestingly, a high fitness score does not necessarily indicate a good performance as shown in Figure 3.2a where the fitness score is 0.57, yet when plotted the worm seemingly has no coordination. This is due to an oversight in the fitness function used, thereby allowing the worm to cheat. An adaptation of the formula should have been considered when altering other aspects of Izquierdo and Beer's work. A newer version of the function could use the direction of the worm's head relative to its tail. Despite this oversight, Figure 3.2c-d all display the worm succeeding in moving towards the area of highest concentration with different degrees of success.

Once we had taken the top candidates from the evolutionary algorithm we needed to decide how to evalute their performance. The worm with the highest fitness score (which visually

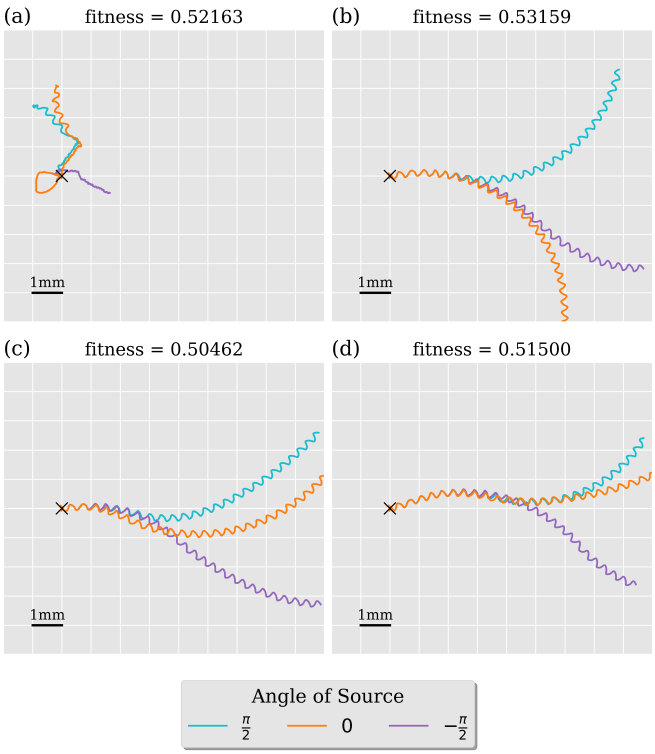


Figure 3.2: The path of different worms when tested in the evolutionary network. The fitness value denotes the performance of each network. The three paths show the path taken by the worm for the three food locations.

appeared to behave correctly) was chosen for further testing. Hence the parameters associated with Figure 3.2c were chosen for further testing.

3.3 ASE neurons

The model used for the sensory ASEL and ASER neurons represents the key characteristics of each neuron. ASEL neurons are activated by up steps in concentration, whereas ASER neurons are activated by down steps in concentration.

Rather than observing the behaviour of the steering circuit within the complexity of worm's environment, an isolated[8] instance of a `SteeringCircuit` was created to determine the effect of specific inputs on the steering circuit. The `update_state()` method was called with pre-defined concentration values to assess the performance of each ASE neuron. Data generated was visualised using the neural visualiser.

Figure 3.3 shows the ASER and ASEL output during changes in concentration. The ASEL neuron demonstrates an increase in activation with a increase in concentration. The simulated ASER does not change unlike in the real ASER neuron which decreases when there is an increase in concentration, displaying this additional behaviour would be a possibility for further work which is discussed in Chapter 4. Additionally, the ASEL neuron increases and then decreases, showing that it is reacting to the change in concentration as opposed to an absolute value of concentration. During concentration decreases, ASER output increases, with ASEL remaining the same, and accurately mimics the real behaviour of the neurons.

[8]Isolated from a `Worm` object

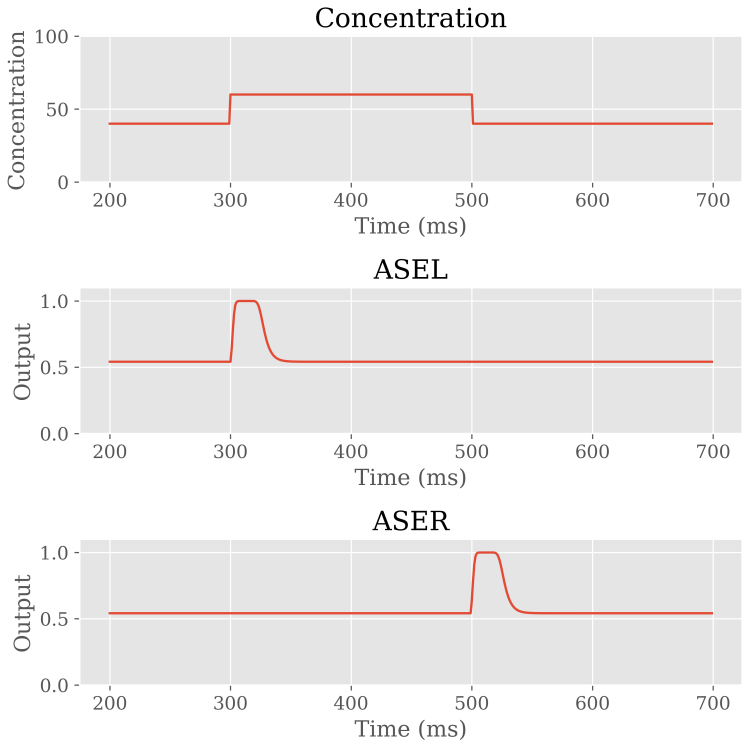


Figure 3.3: The output of simulated ASE neurons with a simulated NaCl concentration input.

For the purposes of this task, the behaviour of the neurons was sufficient to generate chemotaxis in the worm and hence the decrease in accuracy of the model was justified. It becomes difficult to accurately model the real behaviour of the ASE neurons without using a data-driven approach.

M and N

Sensitivity of the steering circuit predominantly takes place in the M and N time parameters used to calculate ASEL and ASER output. Figure 3.4 shows the different effects of small and large M and N values on the neural output, with an oscillatory concentration input. The sinusoidal input represents the up and down undulations of the head in the NaCl environment.

Smaller values of M and N show higher sensitivity to the changes in concentration as the history buffer is smaller which is ideal for shorter simulations where the worm has to make a quicker turning decision. Larger values are more affected by long term changes in concentration due to their larger history buffer. Figure 3.4 also shows how it takes some time for the buffer to saturate. This may offer an explanation as to why the worm takes some time to stabilise and begin turning. Longer simulations would reduce the proportionate effect of this 'loading'[9] stage on the forward locomotion. This logic error which produces the 'loading' effect is in our ASE differential calculation with our assumption that $t - N \geq 0 \wedge t - N - M \geq 0$. An updated version should consider if this clause is false. Retrospectively, we could have saturated the buffer with the initial concentration at the start but this oversight was discovered too late and simulations could not be rerun.

We do not choose the M and N values but rather evolve them. Future versions of the

[9]The time taken for the buffer to fully saturate, hence loading.

evolutionary algorithm may be tuned to increase diversity in other parameters and not M or N.

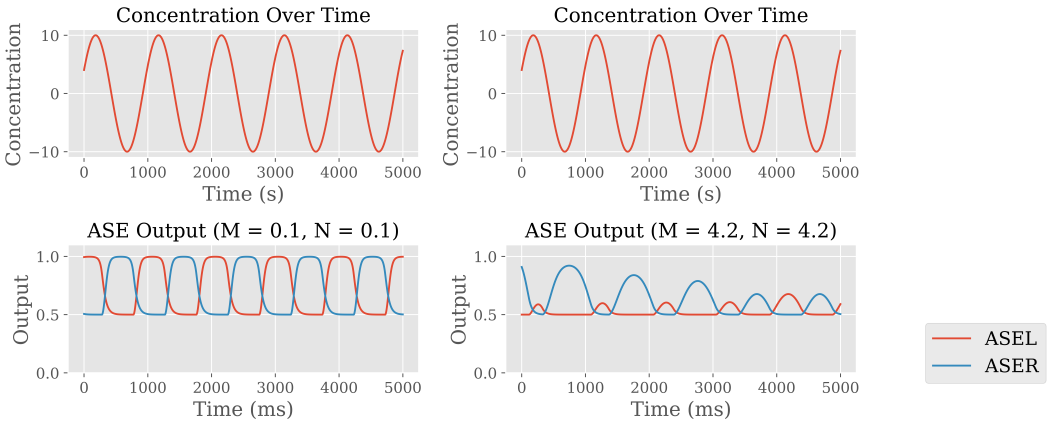


Figure 3.4: The output of the ASE neurons with different values for parameters, M and N. A sinusoidal concentration input simulates the movement of the worms head.

3.4 Steering

3.4.1 Weathervaning

A worm with our chosen steering parameters was placed in two environments, each with a linear gradient perpendicular to the initial direction of the worm. The gradients were in opposite directions in the two environments. Figure 3.5a and 3.5b show the worm in gradients of ±5/mm. in both cases the worm had a movement bias towards the direction of increasing gradient.

This compares to experimental evidence with real `C. elegans` worms when they are placed in a chemical gradient (citation). Asymmetry in the movements of the worm can be attributed to asymmetrical biases in the worm's initial locomotion. Despite attempting to remove these biases, they still remain in the worm to some degree. The curving of the worm is not instant, we hypothise this is due to two reasons. Firstly, the worm requires time to orient from its initial horizontal starting position. Secondly, that effect of the steering circuit on the dorsal and ventral muscles is subtle and hence it takes some time for the bias effect to become apparent in movement.

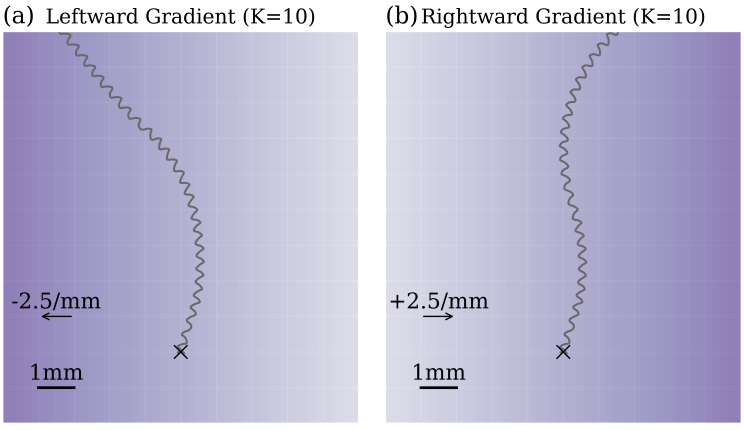
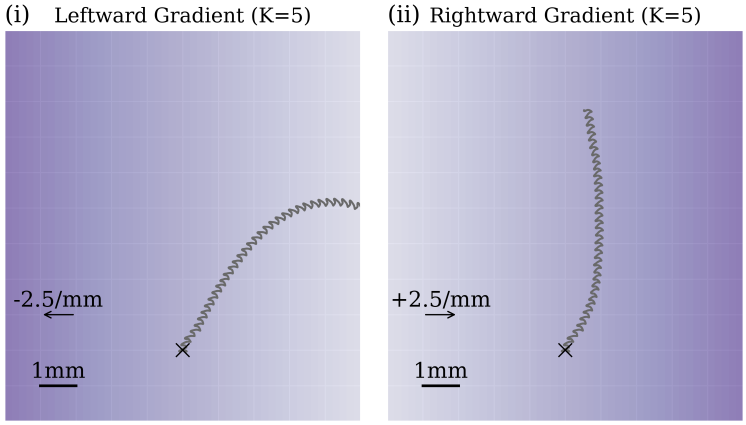


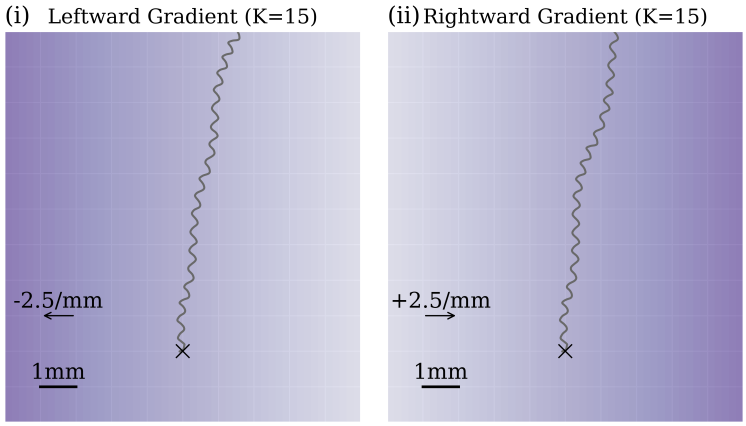
Figure 3.5: Weathervaning for K = 10. dC/dx = ±2.5.

3.4.2 Materials

An identical steering circuit was tested in environments with different material parameters. Notably changing the value of K which defines the force that drag exerts on the worm by the viscosity of the surroundings. Figure 3.6a and 3.6b shows the worm in different environments run in the same linearly increasing/decreasing concentration environment used previously. Values of 5 and 15 were chosen for K as they are similar to the value of 10 used in training, but different enough to guage different responses to the environment.



(a) Weathervaning for K = 5. dC/dx = ±2.5.



(b) Weathervaning for K = 15. dC/dx = ±2.5.

Figure 3.6: Worm placed in identical NaCl concentration gradients with different material parameters.

When tested in these different environments, the worm loses significant steering coordination and appears to move with extremely assymetry. We assume this is due to underfitting with the evolutionary algorithm as the low number of generations may have impeded its ability to learn. To ensure a more general solution is found, future versions of the algorithm should include testing in different material parameters, which would contribute to each network's fitness score. This would require a significantly larger population size to accommodate for the required initial diversity and sizable computational requirements.

The worm successfully performing weathervaning in an environment it wasn't trained in (Figure 3.5) suggests that it can learn the behaviour in a wider range of environments. It also implies that *C. elegans* can employ such behaviours in the wild with the correct biological

parameters in its steering circuit.

3.4.3 Concentration gradient

In addition to testing different material parameters, Figure 3.7 shows the worm with the steering parameters tested in chemical gradients with varying rates of change of concentration. In no chemical gradient (Figure 3.7a) the worm displays a right-side bias. This could be attributed to the steering parameters or the underlying bias discussed previously. Regardless, the movment of *C. elegans* is rarely exclusively straight, often exhibiting random movement. Having this bias in the model may not actually hinder any future chemotaxis-related research.

Figure 3.7c shows the worm veering towards the direction of increasing concentration, however, this could be attributed to a false positive. Figures 3.7d, 3.7e and 3.7f exhibit incorrect behaviour although it appears that the size of gradient is proportional to the time and angle of the turning. A steeper gradient leads to an earlier and slightly sharper turn. This lines up with the experimental findings of Iino and Yoshida (2009), where the coefficient of skewness (assymetry in the worm) increased proportionately with the size of the NaCl concentration gradient. By turning, even in the wrong direction, the worm demonstrates the ability of negative chemotaxis.

3.4.4 Motor neurons

There is a key inconsistency between the motor neurons used in Smyth and the ones used by Izquierdo and Beer. The motor neurons in Smyth's model receive a bistable[10] signal whereas Izquierdo and Beer's motor neurons receive a continuously uniform oscillatory output. In Smyth's model, the oscillations are produced by proprioceptive feedback, which might have disrupted the output of the steering circuit. We suspect interference caused the associated muscles to become desynchronized from the body's overall locomotion phase.

These discrepencies may be the cause of the strange behaviours exhibited by the worm and explains the worm's mechanical difficulties. Motor neuron output from Smyth's implementation are seen in Figure 3.8a, showing the output with and without the steering circuit. The continous nature of our steering circuit paired with the discrete motor neurons could potentially have introduced undesirable behaviour.

3.5 Computation

Time

During computational modelling, time and space complexity factor into many of the design decisions. Specifically in this paper, where execution time and result quality were carefully balanced. The addition of a steering circuit doesn't scale with the number of neural units of the worm, unlike Smyth's locomotion circuit which does. Appendix C contains the specifications for the system which the timings tests were run on. Figure 3.9 shows the linear relationship of simulation length and computation time.

[10]They have 2 states to be in, on or off.

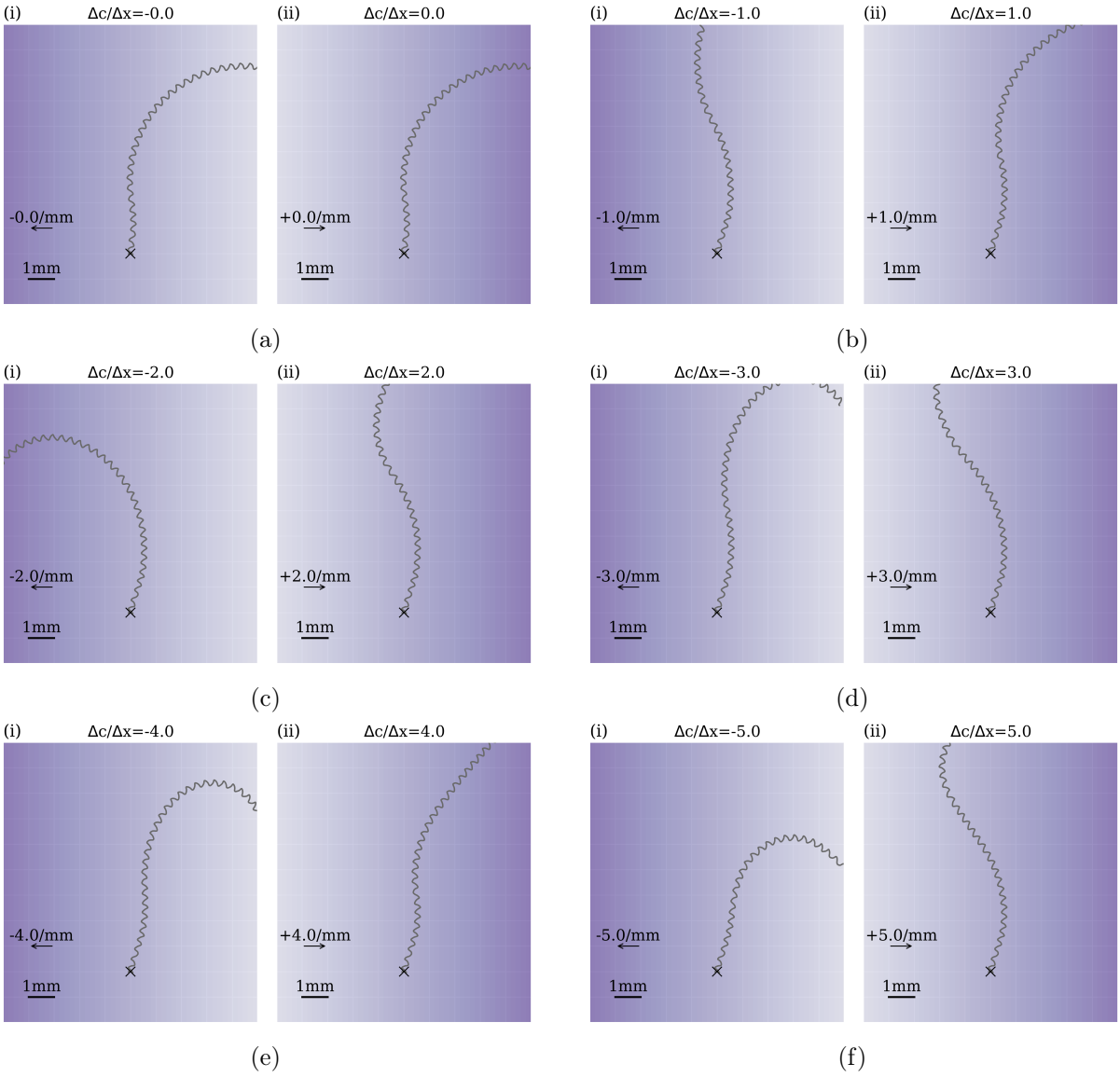


Figure 3.7: Worm placed in identical material parameters (K=10) with different NaCl gradients.

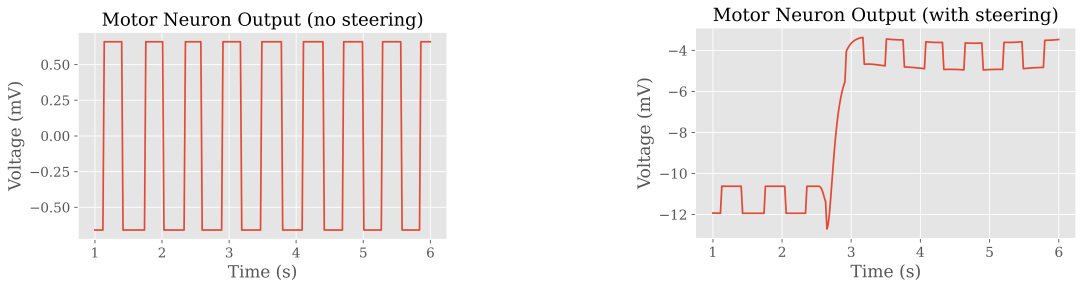
The values generated are highly machine-dependent. To run FEniCS on the system, an x86 conda environment was used which uses Rosetta 2[11], allowing it to run on the Apple M1 chip where it otherwise wouldn't. This approach to running the code on a non-native machine introduces significant time penalties.

A comprehensive list of timing data can be found in Appendix E. The timing data did not include the `Worm` construction. The addition of the steering circuit adds a constant time to the calculation of each model iteration. For dt=0.1, the average time increase for adding the circuit was approximately 3.18%, for dt=0.01 it was 1.39% and for dt=0.001 it was 0.76%. Therefore the steering circuit holds relatively small impact on the overall runtime.

Figure 3.9 visualises the timing data for simulating the worm with different parameters. The linear O(n) time complexity remains intact after adding the steering circuit.

Parallel execution time is imporatant to consider when running multiple simulations in an experiment, as it can greatly speed up the research. Although it is a trivial concurrency problem,

[11]Dynamic binary translator.



(a) Motor neuron output without steering circuit. (b) Motor neuron output with steering circuit.

Figure 3.8: Motor neuron output in `simple-worm` with and without steering circuit.

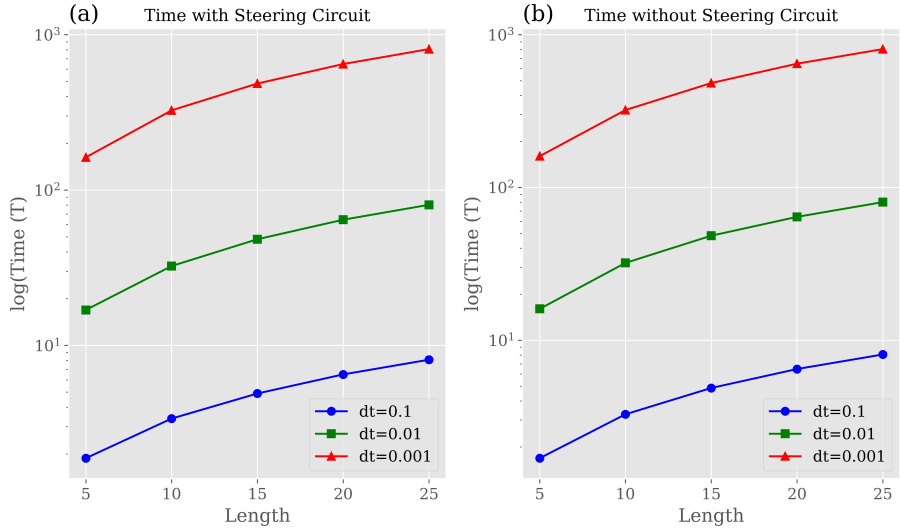


Figure 3.9: Execution time for simulations with different simulation length and timestep. Logarithmic time scale.

the undelying calculations behind `simple-worm` are already highly parallelised with its underlying libraries PyTorch, FEniCS and NumPy. The speedup is not equal to the number of cores, as our serial runs are not truly serial. Moreover, the 8 cores were not identical, 4 are designed for efficiency and 4 for performance.

For parallel runs on 8 cores, the average parellel speedup over all simulations was 3.15. Despite this suboptimal parallel result, it still greatly improved the overall computation time when running the evolutionary algorithm by cutting execution times from weeks to days.

Space

Memory usage of the `SteeringCircuit` and `SteeringParameter` objects was monitored using the Python module memory-profiler. The console output in Appendix G shows the total current memory usage after executing each line with each change. The objects and methods used little memory due to only a small number of components with numerical datatypes. Repeatedly calling `update_state()` did not use any additional memory.

Memory usage will increase slightly with higher M and N values, as well as smaller values for dt; these changes increase the maximum size of the ASE history buffer. This will have no noticable affect on performance, as it remains only in the realms of bytes and kilobytes.

Chapter 4

Discussion

4.1 Conclusions

This project shows that an adapted version of Izquierdo and Beer's steering circuit can be successfully trained for chemotaxis in the `simple-worm` framework. In certain environments, the worm performed the weathervaning mechanism correctly and adjusted its orientation towards the direction of increasing gradient. Ultimately, we explored an implementation of the circuit and thoroughly investigated its capabilities in a wide range of environments. For this reason, we consider the project to be am overall success.

However, it must be noted that the circuit demonstrated a significant performance drop in different environments it was not trained on. The evolutionary algorithm used, created a circuit that was underfitted for general chemotaxis. To create a truly holistic neuromechanical model of steering, a significantly larger pool of candidates will have to be created and tested in a more diverse range of physical environments. The evolutionary algorithm used in this paper converged early, yet still yielded successful results. This phenomenon may be attributed to the stochastic nature of the initial population generation, which might have included a viable solution. Ideally with a larger solution space a more robust solution would be found.

The fitness function provided an accurate way of measuring the performance of the worms over long simulation runs. For shorter runs the worm could 'cheat' the function and achieve a high score even with a bad performance. A fitness function tuned to our requirements should have been used, one which encompasses the core requirements of the particular simulation and environment

Throughout research, a common theme was computational requirements. The nature of the physics and neural modelling is CPU intensive which severely restricted the pace in which the project could unfold. Reducing the solution space has the benefit of reduced computation time but risks finding a less optimal solution.

Design decisions played an instrumental role in the consistency of the project. When implementing more technical classes such as `Environment`, we implemented a user friendly interface in the form of easy-to-use utility function for creating gradients. Whilst implementing additional classes and methods, we adhered to the design style of the underlying framework, so that the codebase remained consistent. Above all, thorough testing was emphasised to ensure reliability and correctness during development, with emphasis on edge case scenarios. All previously written tests were also run to ensure there were no unintended consequences.

When modelling anything, the assumptions made are the foundations to the validity of the model. All decisions made throughout the project were made with the *C. elegans* biology taken heavily into consideration. Understandably however, a comparitavly simple model[1] such as the one presented in `simple-worm` will invariably be disconnected from the real worm and drawing conclusions from the models should be done with sufficient scrutiny.

[1] To the real worm.

Appendix F details all files changed and added in this project, as well as example `plot2d.py` visualisations.

4.2 Ideas for future work

Only in recent years has a full connectome for *C. elegans* been established (Cook et al., 2019). Using this, together with experimentally derived physics and neural parameters, a more anatomically accurate neuromechanical model of the worm can be created. Nonetheless, the computational demands will persistently overshadow the research landscape. Hence, simplifications will need to be made just as Boyle et al. (2012) and Denham et al. (2018) did.

All research in this project has been exclusively in two dimensions, as has all other research discussed in this paper. Recent research into 3D chemotaxis by Lee et al. showed that mutations in genes, which would normally be overlooked in 2D observations, can have a significant impact on *C. elegans*. Extending our model into the third dimension (with circuit modifications) may yield further insight into chemotaxis in *C. elegans*.

A reduction on runtime would have a considerable improvement on research. One solution could be migrating the code to C++, which offers high performance. However, this has the downside of being less memory efficient if not handled correctly, and harder to use for some inexperienced developers.

The ASE neurons were moderately simplified in comparison to their biological counterparts. Reimplementing the logic using a data-driven approach or hard-coding more behaviours into them may unveil more insight into how *C. elegans* interprets sensory information and converts it to forward locomotion changes. Their physical location in the model was simplified to the most anterior point of the worm; a future model may place them in a more biologically accurate location.

Regarding the framework as a whole, it can be improved in numerous ways. Using the latest features available in Python and the modules it uses, the code quality and efficiency can be improved. An issue found in the initial stages of development was getting the code to run, so adjusting the code for different systems by using run-time OS checks woulc significantly aid portability.

The primary method for evaluating the worms during the investigation was through visualization. Despite their effectiveness, these tools still have considerable potential for enhancement, particularly in conveying non-positional information. Currently, essential details related to simulations are embedded solely within filenames. Enhancing the output files with annotations, such as indicating the steepness of gradients, could provide more comprehensive information to users with these generated files. For more detailed graphical representations, we utilized a Jupyter Notebook instead of relying on the `plot2d.py` functions. For longer simulations the worm can go out-of-bounds, adding a dynamically changing bounding box for videos would improve their effectiveness.

Finally, before any further development is done, all current issues need to be fixed and improved. The code Smyth used was from 2012 and significant progress has been made into understanding the forward locomotion of *C. elegans* from a neural perspective. In light of this new research, a more biologically accurate neural circuit should be developed.

References

Z.F. Altun, L.A. Herndon, C.A. Wolkow, C. Crocker, R. Lints, and D.H. Hall. Wormatlas, 2024. URL `http://www.wormatlas.org`. Accessed: 2024-03-25.

CI Bargmann. Chemosensation in C. elegans. `https://www.ncbi.nlm.nih.gov/books/NBK19746/`, Oct 25 2006. Available from: `https://www.ncbi.nlm.nih.gov/books/NBK19746/`.

Jordan H. Boyle, Stefano Berri, and Netta Cohen. Gait modulation in c. elegans: an integrated neuromechanical model. *Frontiers in Computational Neuroscience*, 6:Article 10, 2012. doi: 10.3389/fncom.2012.00010. Edited by Misha Tsodyks, Reviewed by Alexander G. Dimitrov.

S Brenner. The genetics of caenorhabditis elegans. *Genetics*, 77(1):71–94, May 1974. doi: 10.1093/genetics/77.1.71.

M. Carretero, G. M. Solis, and M. Petrascheck. C. elegans as model for drug discovery. *Curr Top Med Chem*, 17(18):2067–2076, 2017. doi: 10.2174/1568026617666170131114401.

S. J. Cook, T. A. Jarrell, C. A. Brittin, et al. Whole-animal connectomes of both caenorhabditis elegans sexes. *Nature*, 571:63–71, 2019. doi: 10.1038/s41586-019-1352-7.

Jack E. Denham, Thomas Ranner, and Netta Cohen. Signatures of proprioceptive control in caenorhabditis elegans locomotion. *Philosophical Transactions of the Royal Society B*, 373 (20180208), 2018. doi: 10.1098/rstb.2018.0208. URL `http://rstb.royalsocietypublishing.org/content/373/20180208`.

Wulfram Gerstner, Werner M. Kistler, Richard Naud, and Liam Paninski. *Neuronal Dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

Robert K. Herman. Introduction to sex determination, December 2005. URL `http://www.wormbook.org`. Published December 24, 2005. Available under a Creative Commons Attribution License.

S. Hughes, M. van Dop, N. Kolsters, D. van de Klashorst, A. Pogosova, and A. M. Rijs. Using a caenorhabditis elegans parkinson's disease model to assess disease progression and therapy efficiency. *Pharmaceuticals (Basel)*, 15(5):512, April 2022. doi: 10.3390/ph15050512.

Yuichi Iino and Kazushi Yoshida. Parallel use of two behavioral mechanisms for chemotaxis in caenorhabditis elegans. *Journal of Neuroscience*, 29(17):5370–5380, 2009. doi: 10.1523/JNEUROSCI.3633-08.2009.

Eduardo J. Izquierdo and Randall D. Beer. Connecting a connectome to behavior: An ensemble of neuroanatomical models of c. elegans klinotaxis. *PLoS Computational Biology*, 9(2): e1002890, 2013. doi: 10.1371/journal.pcbi.1002890.

Eduardo J. Izquierdo and Randall D. Beer. An integrated neuromechanical model of steering in c. elegans. In *Proceedings of the European Conference on Artificial Life 2015*, pages 199–206, 2015. doi: 10.7551/978-0-262-33027-5-ch040.

Luke James. Moore's law in 2022: What's the status quo?, Feb 2 2022. URL `https://www.power-and-beyond.com/moores-law-in-2022-whats-the-status-quo-a-dc63a87e669b554d4d33d2a5ba73692a/`.

Hongfei Ji, Anthony D. Fouad, Zihao Li, Andrew Ruba, and Christopher Fang-Yen. A proprioceptive feedback circuit drives c. elegans locomotor adaptation through dopamine signaling. *bioRxiv*, March 2023. doi: 10.1101/2022.10.14.512295. Preprint not peer reviewed.

K. D. Kimura, H. A. Tissenbaum, Y. Liu, and G. Ruvkun. daf-2, an insulin receptor-like gene that regulates longevity and diapause in caenorhabditis elegans. *Science*, 277(5328):942–946, August 1997. doi: 10.1126/science.277.5328.942.

Eric S. Lander, Lauren M. Linton, Bruce Birren, Chad Nusbaum, Michael C. Zody, Jennifer Baldwin, Keri Devon, Ken Dewar, Michael Doyle, William FitzHugh, Roel Funke, Diane Gage, Katrina Harris, Andrew Heaford, John Howland, Lisa Kann, Jessica Lehoczky, Rosie LeVine, Paul McEwan, Kevin McKernan, James Meldrim, Jill P. Mesirov, Catherine Miranda, William Morris, John Naylor, Christina Raymond, Mark Rosetti, Raffi Santorini, Glenn Sheridan, Carrie Sougnez, Yucheng Zia, Matthew Friedlander, Tracie Nicol, Cynthia Carter, Steven Feinberg, Michael G. Naylor, Jane Stange-Thomann, Charles E. Barrett, Shawn Allen, Michelle Seaman, Richard Bulyk, J. Farrell, Kenneth Gordon, Michelle LeProust, Barbara A. Tatusova, Susan M. Cawley, Mark A. Clark, Kristen A. Weinstock, Kim C. Weinstock, Jennifer Smith, Marco deJong, Linda Yandell, Chris Andrews, Peter Nguyen, Chunshu Tang, Joseph Willey, Yongzhu Tsuji, Janice White, Geetha Kannan, Anibal Craven, William D. Crawford, Mark Gardner, Peter Garnick, William Heil, Michelle L. Mason, Tenny M. Hanna, Anthony P. Abbey, Robert V. Tinkle, Eric S. Glenn, Chris Jensen, Arthur T. Cosentino, Elia V. Arnold, Nicholas M. Perna, Natalie H. Ouellette, Kevin J. Hogan, Alan S. Montmayeur, James G. Skupski, Richard L. Kent, Charles J. Giovanoni, Todd Kuiper, Michael A. Santorini, M. Y. Smith, Gerard V. Basilio, Keith E. Wheeler, Sarah P. Gibson, Christophe B. Willson, Patrick S. Yoder, M. R. Bergan, Steve G. Kemp, Charle W. Vidal, Paul T. Mardis, Evan Mauceli, Wayne Fleischmann, Jane Peterson, Andrew M. Adams, Jay Shendure, Janet Ding, Daniel G. Gibson, Mary L. Wheeler, Iain Dunwell, Paul Flicek, Donna Muzny, David R. Bentley, Jeffrey Reid, Mark A. Batzer, Lynn Jorde, Eric S. Lander, Joel Hirschhorn, Richard Durbin, Jane Rogers, and Robert Waterston. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, 2001. doi: 10.1038/35057062.

H.K. Lee, T.Y. Lee, J.I. Lee, K.S. Park, and K.H. Yoon. Precise sensorimotor control impacts reproductive fitness of C. elegans in 3d environments. *Neuroreport*, 35(2):123–128, 2024. doi: 10.1097/WNR.0000000000001986. Epub 2023 Dec 12.

J. A. Lewis and J. A. Hodgkin. Specific neuroanatomical changes in chemosensory mutants of the nematode caenorhabditis elegans. *Journal of Comparative Neurology*, 172:489–510, 1977.

Linjiao Luo, Quan Wen, Jing Ren, Michael Hendricks, Marc Gershow, Yuqi Qin, Joel Greenwood, Ed Soucy, Mason Klein, Heidi K. Smith-Parker, Ana C. Calvo, Daniel A. Colón-Ramos, Aravinthan Samuel, and Yun Zhang. Dynamic encoding of perception, memory, and movement in a c. elegans chemotaxis circuit. *Neuron*, 82(5):1115–1128, 2014. doi: 10.1016/j.neuron.2014.05.010. URL `https://www.cell.com/neuron/fulltext/S0896-6273(14)00418-5`.

Victor M. Nigon and Marie-Anne Félix. History of research on c. elegans and other free-living nematodes as model organisms, 2005. URL `https://www.ncbi.nlm.nih.gov/books/NBK453431/`.

Jonathan T. Pierce-Shimomura, Thomas M. Morse, and Shawn R. Lockery. The fundamental role of pirouettes in caenorhabditis elegans chemotaxis. *Journal of Neuroscience*, 19(21): 9557–9569, 1999. doi: 10.1523/JNEUROSCI.19-21-09557.1999.

Thomas Ranner. A stable finite element method for low inertia undulatory locomotion in three dimensions. *arXiv preprint arXiv:1904.01325v3*, 2020.

David E. Rumelhart, James L. McClelland, and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press, Cambridge, MA, 1986.

Jens Schulze, Wouter Houthoofd, Jana Uenk, Sandra Vangestel, and Einhard Schierenberg. Plectus - a stepping stone in embryonic cell lineage evolution of nematodes. *EvoDevo*, 3(1):13, July 2012. doi: 10.1186/2041-9139-3-13. URL `https://doi.org/10.1186/2041-9139-3-13`.

Amelia Jade Smyth. An improved neuromechanical model of movement in C. Elegans. Final report submitted in accordance with the requirements for the degree of BSc Computer Science, 2023.

J. E. Sulston, E. Schierenberg, J. G. White, and J. N. Thomson. The embryonic cell lineage of the nematode caenorhabditis elegans. *Developmental Biology*, 100(1):64–119, Nov 1983. doi: 10.1016/0012-1606(83)90201-4.

Hiroshi Suzuki et al. Functional asymmetry in c. elegans taste neurons and its computational role in chemotaxis. *Nature*, 454:114–117, 2008. doi: 10.1038/nature06927.

M. Tomioka, T. Adachi, H. Suzuki, H. Kunitomo, W.R. Schafer, and Y. Iino. The insulin/pi 3-kinase pathway regulates salt chemotaxis learning in caenorhabditis elegans. *Neuron*, 51(5): 613–625, 2006. doi: 10.1016/j.neuron.2006.07.024.

J. G. White, E. Southgate, J. N. Thomson, and S. Brenner. The structure of the nervous system of the nematode caenorhabditis elegans. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, 314(1165):1–340, 1986. doi: 10.1098/rstb.1986.0056.

Norbert Wiener. *Cybernetics*. M.I.T. Press, New York,, 1948.

Wikipedia. Caenorhabditis elegans, 2024. URL `https://en.wikipedia.org/wiki/Caenorhabditis_elegans`. [Online; accessed 23-April-2024].

WB Wood. *The Nematode Caenorhabditis elegans*. Cold Spring Harbor Laboratory Press, 1988. ISBN 978-0-87969-433-3.

Jihye Yeon, Jinmahn Kim, Do-Young Kim, Hyunmin Kim, Jungha Kim, Eun Jo Du, KyeongJin Kang, Hyun-Ho Lim, Daewon Moon, and Kyuhyung Kim. A sensory-motor neuron type mediates proprioceptive coordination of steering in c. elegans via two trpc channels. *PLoS Biology*, 16(6):e2004929, 2018. doi: 10.1371/journal.pbio.2004929. URL `https://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.2004929`.

Appendix A

Self-appraisal

A.1 Critical self-evaluation

Overall, the project was successful. I successfully replicated chemotaxis behaviour within a chemical gradient by integrating a steering circuit into the head and neck, guided by sensory input from ASE neurons. The design and implementation of the steering circuit were based on extensive biological research on real *C. elegans* worms, aiming to mimic the worm's physiological mechanisms as accurately as possible.

However, when placed in alternate environments the worm did not perform as well, sometimes behaving in an incorrect and unpredictable manner. Nonetheless, patterns were established in its behaviour relating to the the steepness of gradient it was in, which alligned with the findings of other researchers.

The code was integrated into the existing `simple-worm` framework, which is actively used by researchers. Therefore, a significant emphasis was placed on ensuring that all updates were backward compatible with previous versions. I made all new function parameters optional to maintain compatibility. The structure of the code was designed to align with the existing architecture for enhanced sustainability. The code is robust, designed to handle errors internally, and remains efficient and user-friendly. While no formal client requirement analysis was conducted, the enhancements were tailored to meet user needs for customization and adaptability, reflecting a deep understanding of the user base.

An in depth discussion on future work on this project is included in 4.2.

A.2 Personal reflection and lessons learned

Despite the overall project being a success, it was not all plain sailing. The initial stages of the project were difficult with the codebase being larger than anything I had encountered before. I slowly and inefficiently worked my way through it. Ideally, I would've methodically dissected the codebase to speedup progress, rather than reading code sporadically. My ability to interpret code across multiple files has been refined over the course of the project as well as writing professional code for a pre-existing codebase.

This is first time I've attempted a project of this size. My approach to academic research has improved significantly over the course of the project. Reading academic journals and papers has broadened my academic vocabulary and improved my style of writing. The writing of this dissertation was in LaTeX which is widely used in academia, an unfamiliar technology which I have vastly improved at. My background in biology is non-extensive, so this project was a big step out of my comfort zone. The research involved in this project was heavily oriented around biology and neuroscience, as well as computer modelling. I have learnt important skills for studying biology and modelling methods, which are applicable to many other fields. Additionally, I found the production of the code heavily reliant on the biological research done prior to writing

it. This meant I spent lots of time learning without starting development, narrowing the time for discovering bugs in the code. For future projects, I will map out the code being written with UML diagrams, and the necessary academic material needed to write it. This way, I can ensure everything will be written timely and correctly.

Due to this large disruption to development, issues surrounding the equation definitions of the ASE differential calculation and fitness function could not be rectified in time. In future projects I will carefully judge all equations by the situation they are used, so that adjustments can be made if necessary.

A module in Fall Semester 2022 (COMP2421 Numerical Analysis) covered the basics of computer modelling which helped in understanding the modelling and methods behind `simple-worm`. I still found myself delving significantly deeper into the inner workings of `simple-worm` which has furthered my knowledge of physics. Modelling will be a highly applicable skill I am eager to use in future.

The technology stack used in the project (Python, Pytorch, FEniCS, NumPy, Matplotlib, Conda) was largely unfamiliar to me as I preferer C and C++ for most programming tasks. Learning these industry standard tools was enlightening, particularly the power of NumPy was intriguing to me and something I will definitely consider using in future projects.

With regards to version control, the project's wide initial scope led to many different initial directions and using version control significantly helped organise and sort the code during development. It allowed me to switch to a different branch easily without significant overhead. I used a personal repository at the start and switched all code over to the university assigned one near the end. This discontinuity may seem confusing and the project should have used the assigned repository from the start to prevent this. The best practices was attempted but I still fell short on numerous occasions; practice makes perfect and version control is no exception.

At the start of the project, I drew up a roadmap outlining the key stages of the project. However, I did not foresee the extent of the computation times. When I came to running the evolutionary algorithm, it was incredibly slow, so its parameters had to be changed so it finished in a reasonable time. The code still ended up running over the course of several days. The computations were unavoidable, so being able to predict this earlier meant I could've attempted to organise a high-performance computing cluster for the worm training. Cloud computing was considered but quickly disqualified due to financial concerns.

A.3 Legal, social, ethical and professional issues

A.3.1 Legal issues

Many different resources were used during the development of this project, so, it was imperative to abide by the law. Intellectual property rights play a large role and all development was made in accordance with the open source rules and regulations to prevent legal issues.

When adapting the work of Izquierdo and Beer (2015), all methodologies used, and adaptations, were referenced accordingly and given proper credit. Although Izquierdo and Beer were not approached personally, material used and adapted falls under the UK Fair Dealing Law's regarding academia. All work used was cited and referenced properly.

There was no sensitive or user data used during the research so there was little

consideration towards data privacy laws and restrictions. All data was generated locally for the purpose of this project.

A.3.2 Social issues

C. elegans research is often used to extrapolate onto other animals, notably humans. They offer deeper understanding into evolution, biological processes, and neurology. Any discoveries that come about from *C. elegans* research can theoretically be used in adverse ways against individual humans and society. However, there is no reason to believe the work in this project can be used in any unfavourable way.

A.3.3 Ethical issues

There are numerous ethical issues regarding to the project and all future work with `simple-model`. Models which are CPU intensive have a high energy consumption which may cause environmental concerns. The code for `simple-worm` can be run on a standard lapop and hence does not have the same effect as large GPU farms. However, work was still done to ensure the code remained efficient and optimised.

Use of all open resources was heavily considered to ensure the correct licenses were used whilst giving credit and referencing all external code and materials used.

In academic writing, it is crucial to present data honestly and not to manipulate it to fit a hypothesis, as this can lead to misleading conclusions. All data in the paper were presented transparently, regardless of the outcomes. Careful consideration was given to the presentation and wording used in the paper to ensure that the data was not misrepresented and could be interpreted accurately.

There can be strong ethical concenrns when handling real biological worms regarding their quality of life (treatment, living conditons). This project did not require any experimentation outide of the vitualised *C. elegans* worm of `simple-worm`. Thus, this was of no concern. Any future research with real worms will have to face these ethical concerns.

A.3.4 Professional issues

If the work is to be used in any professional or non-professional setting, it is essential that all authors and developers are properly credited. To facilitate future users' understanding of the new features, it is crucial to ensure that all classes and methods are thoroughly documented. To this end, code comments were extensively employed during development to provide clear explanations and guidance.

Appendix B

External Material

Codebases

- The majority of the work was based on an adaptation of `simple-worm` by Smyth (2023).
- Original source: `https://gitlab.com/tom-ranner/simple-worm`
- Smyth's adaptation: `https://gitlab.com/sc20jds/ameliasmyth-fyp`

Development Tools

- **Programming Language:** *Python 3.9* with extensive use of the *NumPy* library.
- **Environment Management:** *Conda* was used to manage custom environments and handle package dependencies.
- **Data Visualization:** *Matplotlib* was utilized for generating visual representations of data. A *Jupyter Notebook* was used to generate graphs for the dissertation.
- **Documentation:** *LaTeX* was employed for the development and formatting of this paper.

Integrated Development Environment (IDE)

- Development was carried out using *Visual Studio Code*.

Auxiliary Tools

- **Figure Creation:** Custom figures were created using *Microsoft PowerPoint* and *diagrams.net*.
- **Data Analysis:** *Microsoft Excel* was employed for the analysis of some numerical data.
- **Time and Space Complexity Analysis:** Python modules *memory-profiler* and *time* were used.
- **Version Control and Hosting:** *GitHub* and *GitLab* were used for version control and hosting the project repository.

Appendix C

System Information

Table C.1: System Information Overview

Category	Details
Manufacturer	Apple
Model	MacBookAir 2020
Type	Laptop
Processor Model	Apple M1
Processor Cores	8 (4 high-performance & 4 high-efficiency)
Processor Threads	8
Memory Total	16 GB (17179869184 bytes)
Operating System	Darwin
OS Version	Darwin Kernel Version 23.1.0
OS Architecture	arm64

Appendix D

Steering Parameters

Table D.1: Neural Network Parameters

Parameter	Value
Synapses	
synapse_0	3.022298785387626
synapse_1	11.911473466208346
synapse_2	10.653198061886417
synapse_3	-10.911147612161198
synapse_4	-13.079986275756253
Thresholds	
bias_0	0.16618595627934596
bias_1	-13.50581900677038
bias_2	-1.7446090404245442
Junctions	
junction_0	0.6773266453542823
junction_1	0.8776291121709379
Times	
M	4.144022203904119
N	2.5405739640409295

Appendix E

Simulation Times

Each entry was timed three times, and the mean of these timings is displayed. All calculations were conducted under consistent conditions with regard to CPU and memory availability to ensure comparability. The timings are presented in seconds. Detailed specifications of the machine used for these tests can be found in Appendix C.

Table E.1: Simulation Times. N = 48.

With Steering Circuit

Length	**dt=0.1**	**dt=0.01**	**dt=0.001**
5	1.8777	16.9019	162.4135
10	3.3779	32.4118	325.8331
15	4.8990	48.2131	484.8069
20	6.4999	64.3853	647.1083
25	8.0840	80.3227	807.7478

Without Steering Circuit

Length	**dt=0.1**	**dt=0.01**	**dt=0.001**
5	1.6880	16.0324	160.2753
10	3.2725	32.0520	321.2584
15	4.8638	48.3022	482.3703
20	6.4730	64.1600	646.0250
25	8.0603	80.1303	804.8305

Table E.2: Simulation Times with Steering Circuit. N = 48.

Length	**Serial**			**Parallel**		
	dt=0.1	**dt=0.01**	**dt=0.001**	**dt=0.1**	**dt=0.01**	**dt=0.001**
5	13.8297	129.3996	1294.2555	7.0947	36.0309	352.4005
10	26.3349	257.5735	2580.9311	11.6803	75.8717	729.3233
15	39.1769	385.5878	3870.9876	15.4793	112.4950	1111.6127
20	51.9849	514.2203	5168.0048	18.9000	151.2634	1490.3244
25	65.1548	644.0530	6460.5310	23.0922	189.4954	1869.9305

Table E.3: Simulation Times without Steering Circuit. N = 48.

Length	**Serial**			**Parallel**		
	dt=0.1	**dt=0.01**	**dt=0.001**	**dt=0.1**	**dt=0.01**	**dt=0.001**
5	13.80	129.08	1285.32	7.03	35.75	349.40
10	26.21	253.88	2547.60	11.60	74.51	715.48
15	39.08	379.99	3867.47	15.24	110.59	1108.96
20	51.34	504.35	5160.47	18.58	148.64	1461.97
25	64.56	635.76	6405.70	22.82	188.22	1865.64

Appendix F

Code modifications, implementations and additional files

Edited Files

- `simple_worm/plot2d.py`:Additional visualisation functions.
- `simple_worm/worm.py`: Support for neural parameters, steering parameters and circuit.
- `simple_worm/neural_circuit.py`: Support for steering circuit and neural parameters.

Newly added files

- `simple_worm/steering_circuit.py`: Steering circuit and all internal logic.
- `simple_worm/steering_parameters.py`: Steering parameter class.
- `simple_worm/neural_parameters.py`: Neural parameters class.
- `simple_worm/worm_environment.py`: Worm environment class.
- `tests/steering_circuit.py`: Steering circuit class tests.
- `tests/steering_parameters.py`: Steering parameters class tests.
- `tests/chemotaxis.py`: Worm and environment integration tests.
- `tests/worm_environment.py`: Worm environment tests.

Additional files

- `evolution.py`: Evolutionary algorithm logic.
- `gradient_results.py`: Gradient analysis for results.
- `material_results.py`: Material analysis for results.
- `memory_results.py`: Memory analysis for results.
- `neuron_visualisation.py`: Neuron output visualization tool.
- `time_results.py`: Time complexity analysis.
- `timestep_results.py`: Timestep analysis for results.
- `environment_fpy.yml`: Environment configuration file.
- `parameters.ini`: Example parameter file for steering circuit.
- `images/*.png`: Example emages generated from visualisation functions.
- `videos/*.mp4`: Example videos generated from visualisation functions.

Appendix G

Console output

```
Line #  Mem usage          Increment          Line Contents
================================================================
     1  117.37890625 MiB   117.37890625 MiB   @profile(precision=8)
     2                                         def measure_performance():
     3  117.38671875 MiB     0.00781250 MiB       params = SteeringParameters()
     4  117.38671875 MiB     0.00000000 MiB       circuit = SteeringCircuit(dt=0.001, parameters=params)
     5  117.41015625 MiB     0.02343750 MiB       circuit.update_state(concentration=1.0)
     6  117.41015625 MiB     0.00000000 MiB       circuit.update_state(concentration=1.0)
     7  117.41015625 MiB     0.00000000 MiB       circuit.update_state(concentration=1.0)
     8  117.41015625 MiB     0.00000000 MiB       circuit.update_state(concentration=1.0)
     9  117.41015625 MiB     0.00000000 MiB       circuit.update_state(concentration=1.0)
    ...
    97 117.41015625 MiB     0.00000000 MiB       circuit.update_state(concentration=1.0)
    98 117.41015625 MiB     0.00000000 MiB       circuit.update_state(concentration=1.0)
    99 117.41015625 MiB     0.00000000 MiB       circuit.update_state(concentration=1.0)
```