NTNU
Innovation and Creativity

Self-organized Synaptic Learning of Gaits in Virtual Creatures

A neural simulation study within Connectology

Vebjørn Wærsted Axelsen

Master of Science in Computer Science
Submission date: June 2007
Supervisor: Jørn Hokland, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

A simulation study on artificial neural networks (ANNs) based on the theory of Connectology, proposed by J. Hokland (2006). Modeling and simulation of animal motion behavior through ANN-controlled virtual creatures with connectological synaptic learning (Skinner, Pavlov, and Hume mechanisms), operating in a physically realistic outer environment. Evaluation of ANN performance by means of visualization of creature behavior.

Assignment given: 18. January 2007
Supervisor: Jørn Hokland, IDI

Abstract

The theory of Connectology sets forth three psychologically founded synaptic learning mechanisms that may describe all aspects of animal learning. Of particular interest to this thesis is the learning of animal motion behavior, or, more specifically, the development of synchronized and repetitive movement patterns - gaits.

Computer simulations are performed according to the methodology of computational neuroethology: Artificial neural networks are simulated operating in a tight feedback loop with a structurally simple but mechanically realistic body and a physically realistic environment. Neural network learning is purely synaptical and is performed solely within the lifetime of one such ANN-controlled system. Additionally, the configuration parameter space is searched by means of genetic algorithms.

Simulation results show examples of synchronized and repetitive movement patterns developing when neuronal and mechanical model parameters are appropriately specified. These simulations thereby provide the first examples known to us of a fully unsupervised and self-organized artificial neural system that synaptically learns synchronized and repetitive motor control. In spite of limited mechanical model complexity, the most efficient movement patterns to some degree resemble the gaits seen in nature.

Preface

This Master's Thesis has been written as the final part of my Master's degree studies in computer science at the Norwegian University of Science and Technology (NTNU). The project owner has been associate professor J. Hokland at the Department of Computer and Information Science, Faculty of Information Technology, Mathematics and Electrical Engineering. J. Hokland has also been my main supervisor.

This document has been written using the LaTeX document processing system. All simulation software has been written in C++ using Microsoft Visual Studio 2005. For dynamics simulation, Open Dynamics Engine (ODE) v0.8 has been used. Further, Open Graphics Layer (OpenGL) has been used for 3D visualization, with interactive user input by Simple DirectMedia Layer (SDL) v1.2.11.

I would like to thank professors Gertjan Ettema and Beatrix Vereijken at the Programme for Human Movement Science, Faculty of Social Sciences and technology Management for sharing their expert knowledge. The advice provided at the subject of mechanical modeling has been highly valuable, and has allowed me to focus the better part of my attention at the main project task: neural network modeling and simulation. Finally, I would like to thank my supervisor J. Hokland for valuable advice, inspirational ideas, contagious enthusiasm, and above all for providing access to and insight into the theory of Connectology.

Contents

1 Introduction

This Master's Thesis is a simulation study on artificial neural networks (ANNs) incorporating synaptic learning based on the principles and mechanisms described by J. Hokland in *Connectology: Research Programme for Brain-Psychology* [Hok06]. Connectology is a complete neuronal theory containing a potentially complete set of synaptic learning mechanisms that may describe all aspects of synaptic learning in biological systems. In a previous simulation study [Axe06], I examined ANNs based on only one of the synaptic learning mechanisms of Connectology, mainly operating in simple outer environments described by multi-dimensional mathematical functions. Although incorporating a highly different ANN-environmental context, this thesis builds on the results and experience obtained there, and can in that respect be regarded as a continuation of [Axe06].

The neuronal theory of Connectology aims at being biologically realistic. This implies that the networks used herein in some important respects differ from most of the neural network types examined in the literature concerning ANN simulations. Many network types are based on *supervised* learning meaning that, during simulation, there is an external entity evaluating ANN performance. The network is thereby heavily assisted in learning to solve the problem at hand. The most familiar example is, perhaps, networks based on *backpropagation* learning; there, the output of the network is compared directly to some prespecified correct output, and the error is propagated back through the network. Such types of learning are, of course, not biologically realistic, and the networks examined herein are therefore *unsupervised*: Rather than having an external entity providing "error correction", the networks must learn to internally evaluate the sensory feedback they receive from the environment, and thereby adjust their output, i.e. behavior.

The latter is closely related to the principle of *self-organization*. A self-organizing system is a system where a collection of units coordinate with each other to form a system that adapts to achieve a goal more efficiently [CT03]. In connection with neural networks, this means that networks are capable of adaptively organizing their internal structures to solve the task at hand in a best possible way. Adaption is a key principle. In the same way that all biological nervous systems are thought to be self-organizing, the neural networks of Connectology are aimed at being self-organizing. The ability for self-organization is believed to be an essential property that networks must possess for unsupervised learning of systematic and efficient behavioral patterns to take place.

The models defining the simulations of this thesis are founded on the assumption that it is pointless to study and simulate models of biological neuronal systems in isolation; only when such models are coupled with, and operate in tight connection with, an outer environment incorporating a physical, or physically simulated, body, may the capabilities of the neuronal models at hand be evaluated properly.

The methodology used for the simulations of this thesis, then, is that of *computational neuroethology* [Bee90, CB97]. Hillel J. Chiel and Randall D. Beer define the term as follows:

> *Computational neuroethology involves creating joint models of the relevant parts of an animal's nervous system, body and environment. [...] Using these models, one can study the contributions of the components to adaptive behavior, and the new phenomena that may emerge from their interactions. ([CB97] p. 556)*

Based on the above, this thesis concerns ANNs operating in tight connection with outer environments described by physically and mechanically realistic bodies. The two model compo-

nents, the neuronal model (ANN) and the mechanical model (body), operate in a tight feedback loop where the neuronal model provides muscular activation signals to control the limbs of the mechanical model, and where the mechanical model provides the neuronal model with sensory feedback upon which behavior can be adjusted. The goal of the thesis is to examine the ability for sensible ANN-controlled movement behavior to emerge in such systems when challenged with different system level goals, such as rising up or moving forward. The latter - forward movement - is the main experimental area of focus herein. More specifically, thereof, this thesis investigates the possibility for (partly) synchronized motional control patterns to develop in ANNs based on unsupervised and hopefully self-organized synaptic learning.

Of this it becomes evident that animal motion behavior is relevant to this thesis. Animal motion behavior is characterized by repetitive and partly synchronized movement patterns such as crawling, walking, jumping, galloping, and so on. Collectively, such systematic movement patterns are termed *gaits*. A central task in the evaluation and inspection of the simulation experiments of this thesis, then, is to look for such gait-like movement patterns, or, perhaps more realistically, to look for tendencies indicating that such synchronized movement patterns may develop.

I personally believe the modeling and simulation of motion behavior is the right place to start in evaluating and developing a biologically realistic neuronal model of animal learning. As Roger Sperry, Nobel Prize winner 1981, said in 1952 [Kol06]:

> *The brain is the organ that moves the muscles. It does many other things, but all of them are secondary to making our bodies move. (p. 280)*

To me, therefore, when considering a reasonable time horizon, the modeling and simulation of animal motion behavior is *the* most exciting goal of Connectology, and the possibility for "intelligent" motion behavior to emerge in a virtual creature operating in a (simulated) physical environment summarizes the essence of my motivation for working with the theory.

1.1 Final problem description

The theory of Connectology (J. Hokland, 2006) sets forth three psychologically founded synaptic learning mechanisms that may describe all aspects of animal learning. The task is to perform a simulation study on artificial neural networks (ANNs) based on this theory, of which the main goal is to investigate the possibility for synchronized and repetitive movement patterns to develop in ANN-controlled mechanically realistic virtual creatures operating in a physically realistic outer environment. Synaptic learning should be by means of the three synaptic learning mechanisms of Connectology: the Skinner, Pavlov and Hume mechanisms. The configuration parameter space should be searched by means of genetic algorithms (GAs). Performance evaluation should be by means of visualization of creature behavior.

The main project tasks can be summarized as follows:

- Development of program system for ANN simulations, incorporating neuronal and mechanical model calculations, GA parameter search and 3D visualization of creature behavior.
- Investigation of the possibility for synchronized and repetitive movement patterns to develop in ANN-controlled virtual creatures by means of computer simulations.

1.2 Outline

Section 2 gives basic background information on biological and artificial neural networks, including a short historical overview of ANNs. Additionally, a short introduction to the theoretical context of this thesis, the Connectology research programme, is given. Finally, a short review of previous work is provided.

Section 3 presents the model constituting the theoretical and executional basis for the computer simulations of this thesis. The model is comprised of two main parts: A neuronal model specifying the structure and semantics of ANN calculations, and a mechanical model describing the environment in which the ANNs operate. The section introductorily gives a structural overview of the entire model, whereafter the neuronal and mechanical models are described in greater detail. Finally, the use of GAs to search the parameter space of neuronal and mechanical model parameters is described and justified.

Section 4 presents the simulations constituting the experimental work of this thesis. Simulation results are measured by means of the level of goal achievement, and goals vary among different experiments. The section is organized such that experiments are ordered according to increasing system goal difficulty.

Section 5 provides a discussion on the findings and results obtained throughout the work with this thesis. Theoretical contributions are listed, and concluding remarks are drawn. Finally, it marks out the course for further experimental work on motion behavior utilizing ANNs with synaptic learning based on the mechanisms of Connectology.

2 Background

This section gives basic background information on biological and artificial neural networks, including a short historical overview of ANNs. Additionally, a short introduction to the theoretical context of this thesis, the Connectology research programme, is given. Finally, a short review of previous work is provided.

2.1 Neural networks

A neuron is a cell that is capable of generating electromagnetic nervous impulses (termed action potentials or spikes). When a neuron generates an impulse, it is said to *fire*. The level of activity of a neuron is measured by how frequently it fires. More specifically, a neuron's level of activity is measured by an instantaneous firing rate, here termed *drive*. In principle, neurons are functionally simple; they give output (i.e. fire with some frequency) as a function of total input from other neurons. Neurons are the main actors in central nervous systems (CNSs) and spinal cords of animals; every nervous system consists of large amounts of neurons.

Neurons are interconnected by *synapses*. Synapses are connection points between axons of *presynaptic* neurons and dendrites of *postsynaptic* neurons, which are capable of transferring neural signals by means of electromagnetic impulses. Neurons can have thousands of such synaptic connections to other neurons, and thereby have a means of affecting and *driving* each other. A synapse is a one-to-one directed connection, i.e. it transmits impulses from the presynaptic neuron to the postsynaptic neuron. There are two kinds of synapses: excitatory and inhibitory. Action potentials transmitted through an excitatory synapse contribute to increasing the activity of the postsynaptic neuron. An inhibitory synapse is converse, in that the action potentials it transmits contribute to decreasing the activity of the postsynaptic neuron.

Synapses are characterized by their synaptic efficacy. This property represents the strength of synaptic connections, i.e. how good they are at transmitting action potentials from the presynaptic neuron to the postsynaptic neuron. Synapses are plastic in that their efficacy can vary over time.

A neural network is a complex system of synaptically interconnected neurons. Examples of neural networks include the central nervous system (CNS) of animals. As an example, the human brain is a large neural network that contains roughly 100 billion neurons, each of which has 1000-10000 synaptic connections to other neurons [CS92, Kan00].

2.2 Artificial Neural Networks

Neural networks are imitated by means of artificial neural networks. Artificial neural networks are comprised of artificial neurons and artificial synapses. These units are usually modeled to mimic the functional behavior of their biological counterparts; artificial neurons are simple functional units that give output as a function of total input, and artificial synapses are connections between neurons that propagate neuronal signals. ANNs are generally realized through computer modeling and simulation.

Due to the inherent complexity of biological neural networks, some simplifications are usually made when ANNs are modeled for computer simulation. Most importantly, neuronal activity (i.e. the firing of electromagnetic impulses, or action potentials) is simply represented by the firing rate, or drive for short. In other words, no explicit concept of an impulse or spike is implemented; artificial neurons and synapses do not explicitly generate and transmit impulses.

Rather, drive solely represents neuronal activity, and all calculations are done using drive values. Drive is usually represented by a floating-point number between 0 and 1, where a drive value of 0 corresponds to the minimum firing rate (no activity) and a drive value of 1 corresponds to the maximum firing rate (full activity).[1]

The efficacy of a synapse is represented by a single floating-point number. Combining this with the floating-point representation of neuronal drive, the effective drive contribution from the presynaptic neuron to the postsynaptic neuron over a synapse can be calculated as the product of the presynaptic drive and the synaptic efficacy. This yields a simple and computationally efficient model of drive (i.e. action potential) propagation through synaptic connections.

2.3 Short historical overview

In 1895, Sigmund Freud wrote his *Project for a Scientific Psychology* [Fre95], where he proposed several psychological principles that have become widespread today. Firstly, Freud proposed the principle of neuronal inertia, suggesting that nervous systems function so as to minimize stimuli from the environment. As will be seen later on, this principle is fundamental to the theory investigated in this thesis. Secondly, Freud proposed what has erroneously, one must say, become known as the Hebb rule, suggesting that if two synaptically connected neurons have simultaneous high activity, the synapse between them will grow so as to make one neuron (the presynaptic) more easily excite the other neuron (the postsynaptic).[2]

In 1949, Donald O. Hebb wrote *The Organization of Behaviour* [Heb49], where he proposed the Hebb rule (which, as mentioned, is very similar to the above principle of synaptic growth proposed by Freud in 1895). Hebbian learning has become extensively popular, and variants of the Hebb rule as synaptic learning mechanisms substantiate the greater part of connectionist research on neural networks claiming biological plausibility.

In 1982, John J. Hopfield showed that recurrent artificial neural networks with binary threshold neurons and Hebbian-based synaptic learning could serve as a content-addressable memory [Hop82]. Hopfield's model renewed general interest in neural networks considerably. The networks are often termed *associative* neural networks because of their abilities for associative memory. In 1984, Hopfield extended the results to neurons with graded response [Hop84]. A year later, Ackley, Hinton and Sejnowski showed that a similar type of network based on stochastic units, the Boltzmann machine, when paired with simulated annealing procedures could develop efficient internal representations of the environment [AHS85].

In 1986, McClelland, Rumelhart and the PDP Research Group presented a new learning algorithm for multi-layered feedforward networks called backpropagation [MRtPRG86], and once again renewed general interest in neural networks. A neural network trained with backpropagation could solve problems that are not linearly separable, thereby overcoming earlier problems related to nonlinearity. Feedforward networks with backpropagation learning have been shown to solve several complex, often engineering-related, tasks. Backpropagation learning is, however, not considered biologically plausible, and is therefore of little relevance in the context of this thesis.

In 1988, A. Harry Klopf presented a neuronal model for classical conditioning called the drive-reinforcement model [Klo88]. The model included a synaptic learning mechanism based on local changes in presynaptic and postsynaptic cells, as did the Freud-Hebb rule mentioned

[1] In biological neurons, the maximum firing rate is approximately 300 Hz [Kan00].

[2] Freud used the concept "contact barrier" which is highly reminiscent to what is known today as a synapse.

above, but that differed in some other respects (discussed later). The model was shown to predict several empirically established properties of classical conditioning. This work by Klopf lays the structural foundation on which the synaptic learning mechanisms of Connectology are based, and is therefore of great importance.

2.4 Theoretical context - the Connectology research programme

This thesis is a project within the Connectology research programme for brain-psychology proposed by J. Hokland [Hok06]. Connectology is a complete psychological theory, based on the principles of connectionism and aimed at being biologically plausible. As expressed in [Hok06]:

> *Connectology is my will to, and fantasy of a beautifully connected* science *of psychology, linking all major principles of neural science to every peak insight in the history of psychology. (p. 8)*

Or in the words of Freud, in his *Project for a Scientific Psychology* [Fre95], to which Connectology is strongly related:

> *The intention is to furnish a psychology that shall be a natural science: that is, to represent psychical processes as quantitatively determinate states of specifiable material particles, thus making those processes perspicuous and free from contradiction. (p. 265)*

Connectology sets forth three different synaptic learning mechanisms in the form of three basic principles: Hedonism, Anticipation and Reason. The following brief explanation of these principles and their respective synaptic learning mechanisms is largely adopted from [Hok06].

Hedonism is the first principle, based on Freud's principle of neuronal inertia [Fre95]:

> *The endeavour of the nervous system, maintained through every modification, is to avoid being burdened by Qή or to keep the burden as small as possible. (p. 301)*

In current psychological or physiological terms, this translates into the hypothesis that nervous systems try to keep themselves free from stimulus, or to keep stimulus as low as possible. Consequently, low signal levels are assumed favorable ("pleasure"), while high signal levels are assumed unfavorable ("pain"). The above gives rise to the **Skinner mechanism**,[3] which takes *operant conditioning* to the neuronal level.[4]

Anticipation is the second principle, based on the work of A. Harry Klopf [Klo88]:

> *The efficacy of a synapse changes in a direction such that the neuron comes to anticipate the unconditioned response; that is, the conditioned stimulus comes to produce the conditioned response prior to the occurrence of the unconditioned stimulus and the unconditioned response. (p. 88)*

This gives rise to the **Pavlov mechanism**, which takes *classical conditioning* to the neuronal level.[5] This learning mechanism is due to Klopf; the definition of the Pavlov mechanism given in [Hok97, Hok06] is basically identical to the learning mechanism proposed in [Klo88].

[3] Although not identical to [Hok06], the mechanism was first proposed in [Hok97].

[4] [All07] defines operant conditioning as: *"Learning that occurs due to the manipulation of the possible consequences".*

[5] [All07] defines classical conditioning as: *"The behavioral technique of pairing a naturally occurring stimulus and response chain with a different stimulus in order to produce a response which is not naturally occurring".*

Reason is the third principle, based on the work of Piaget [Pia71] and Hume [Hum88]. The corresponding mechanism is called the **Hume mechanism**,[6] which takes the *assimilation and accommodation of concepts* to the neuronal level.

The three principles introduced above (Hedonism, Anticipation and Reason) and their corresponding synaptic learning mechanisms (Skinner, Pavlov, Hume) are described in detail with a view to modeling and simulation in Sections 3.2.3 and 3.2.5. For psychological foundations for the theory of Connectology, the reader is referred to [Hok06], where each principle and synaptic mechanism is explained in psychological terms and reasonings.

As regards neural components, and in addition to neurons and synapses discussed above, Connectology introduces the concept of *clusters*. A cluster is a collection of adjacent neurons with similar properties and connectivity [Hok06];[7] the cluster concept is thus quite resemblant to the more frequently used concept of neuronal layers. Connectology states that clusters are the basic building block of large neural systems, and that neuronal connectivity, i.e. which neurons connect synaptically onto each other, is to be specified at the level of clusters rather than at the level of single neurons.[8] When a cluster A is said to connect onto a cluster B, then, what is meant is that neurons inside cluster A connect onto neurons inside cluster B.

2.5 Previous work

In a previous simulation study [Axe06] the Hedonism principle of Connectology and the corresponding Skinner synaptic learning mechanism was examined in detail. The study dealt with small ANNs used to detect minima in simple outer environments modeled as multi-dimensional mathematical functions. Learning in these ANNs was based on Skinner type synaptic learning only, meaning that the other two learning mechanisms of Connectology, Pavlov and Hume, were not included.

The study resulted in some interesting findings, the most important being:

- **Conflict cases**, where several synapses (with their presynaptic neurons) compete in driving and inhibiting a common postsynaptic neuron, elucidated problems with diverging synaptic efficacies which are inherent in the synaptic learning mechanisms of Connectology. Two modifications to the synaptic learning mechanisms were suggested, and simulations were performed that demonstrated their abilities with respect to solving divergence related problems. These modifications and the background for why they are needed are revisited in Section 3.2.3 of this thesis.

- **Multi-layer learning** proved to be difficult in ANNs with synaptic learning based solely on the Skinner mechanism. Multi-layer networks as simple as two-layer chains gave highly unstable ANN behavior, even for simple one-dimensional unimodal minimization problems which were easily and efficiently solved in corresponding single-layer networks.

- An important property of the Skinner mechanism of Connectology is that reward (i.e. decrease of presynaptic drive) induces learning, while **punishment** (i.e. increase of presy-

[6] Although not identical to [Hok06], the mechanism was first proposed in [Hok98].

[7] According to [Hok06], intra-cluster synapses always learn by the Hume mechanism. This thesis further examines the possibility for intra-cluster learning by the Pavlov mechanism, see Section 3.2.6. For both cases, however, all intra-cluster synapses are of the same type, maintaining the similar connectivity property within clusters.

[8] Connectology uses the term *Cluster Diamond Matrix* when referring to the matrix that specifies which clusters connect onto each other.

naptic drive) does not. This assumption was challenged through a comparison of simulations differing only in the inclusion (i.e. modified Skinner mechanism) or exclusion (i.e. original Skinner mechanism) of punishment learning. For simple non-conflict simulation cases, punishment learning had no significant effect. However, for conflict cases which were solved sensibly without punishment learning, the inclusion of punishment learning gave divergence on synaptic efficacies and no sensible results.

Conflict cases are seemingly inevitable in connectological simulations with ANNs of non-trivial size; for instance, the simulations presented herein may include systems with potentially conflicting needs. The findings in [Axe06] on conflict cases, and the problems they pose, are therefore of high relevance to this thesis. The fact that the ANNs of this thesis have all three types of connectological synapses (Skinner, Pavlov and Hume) should not change this; on the contrary, more complex network structures and combinations of different learning mechanisms should only make the matter even more important.

The findings in [Axe06] on multi-layer networks are important when designing network topologies, because series of Skinner-synapses cannot be expected to behave in the way intended. When dealing with multi-layer networks in this thesis, Skinner synapses should probably be used only in connecting needs to the internal network, i.e. as connections between need inputs and the first layer.[9] Pavlov and Hume synapses can then be used when building multi-layer extensions of this basic structure.

The findings in [Axe06] on punishment learning provide significant indications in favor of the hypothesis that is fundamental to the Skinner mechanism, that reward induces learning, while punishment does not. As for the relevance of this thesis, this simply confirms that this property of the Skinner mechanism most probably is "correct", or at least propitious with respect to the simulations of this thesis, and that one can settle for the exclusion of punishment learning for the simulations performed herein.

[9] The exception is when affect systems (Section 3.2.6.1) are used, where Skinner synapses connect the affect cluster to the internal network.

3 Model

The model used for the simulations of this thesis is comprised of two main parts: A neuronal model specifying the structure and semantics of ANN calculations, and a mechanical model describing the environment in which the ANNs operate. This section introductorily gives a structural overview of the entire model, whereafter the neuronal and mechanical models are described in greater detail. Finally, the use of GAs to search the parameter space of neuronal and mechanical model parameters is described and justified.

3.1 Model overview

The modeling approach taken for the simulations of this thesis can be summarized in the words of Chiel and Beer:

> *[A]daptive behavior can best be understood within the context of the biomechanics of the body, the structure of an organism's environment, and the continuous feedback between the nervous system, the body and the environment. ([CB97] p. 553)*

The two main system components are the ANN (as in *neuronal model*) and the creature body and environment (as in *mechanical model*). These are shown in Figure 1, which further depicts the tight feedback loop in which they operate. Communication between the neuronal model and the mechanical model is defined by means of *needs*, *senses* and *motor activation*:

- **Needs:** The mechanical model provides the ANN with need signals. Needs represent unwanted neural values ("pain") which the ANN will try to minimize.
- **Senses:** The mechanical model further provides the ANN with sense signals. Senses represent neutralized neural values which are not subjected to minimization (or any other value preference).
- **Motor activation:** The ANN provides the mechanical model with motor activation signals. Motors determine the force developed in the muscles of the virtual creature, and thus provide the ANN with the ability to control and steer the mechanical model.

With the above definitions of needs, senses and motor activation, the system goal can be specified as follows:

> The ANN should steer motor output values so as to minimize need input values utilizing neutralized sense input values as appropriate.

This further implies that the behavioral goal specification for the system as a whole is directly and wholly determined by the set of needs provided by the mechanical model as inputs to the ANN. As an example, a need provided as inverse body height (such that greater height yields a lower need value) translates into a system level goal of getting the creature body as high up as possible. Correspondingly, a need provided as inverse body forward velocity represents a system level goal of achieving and maintaining a high creature body forward velocity.

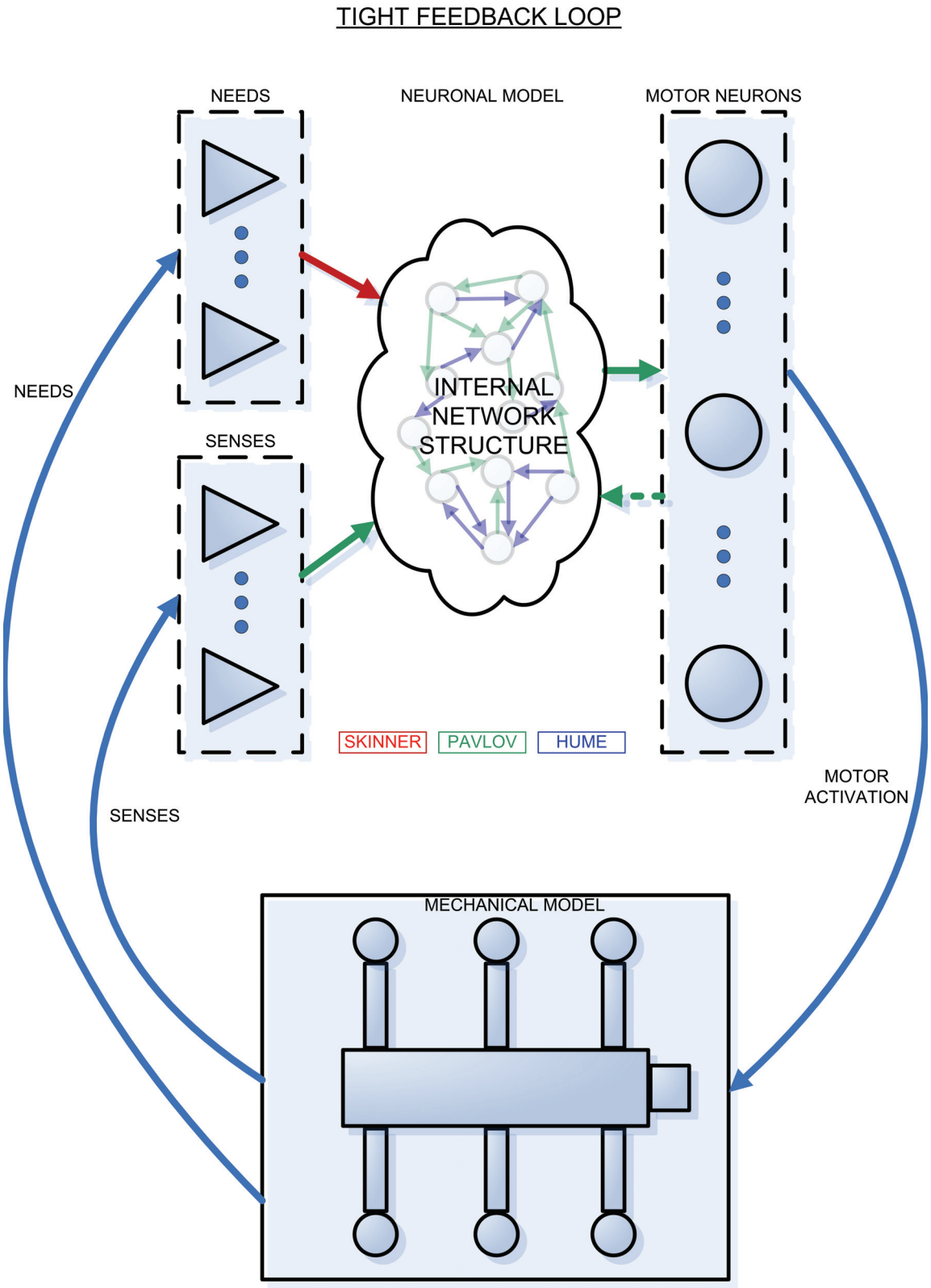
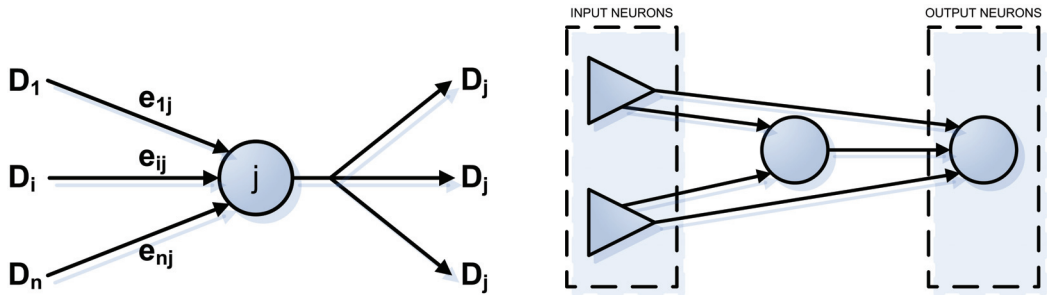


Figure 1: Model overview: neuronal model synapse types as depicted in color (dotted arrows are optional)



(a) Artificial neuron **(b)** A small ANN

Figure 2: Structural Overview

3.2 Neuronal model

This section describes the neuronal model in detail, including some empirical findings that are relevant to modeling. The model is made precise through a thorough mathematical specification. The description is partly adopted from [Axe06], with some modifications based on the findings therein (see Section 2.5). Further, the description has been extended to cover Pavlov and Hume type synaptic learning in addition to the Skinner type synaptic learning dealt with in [Axe06].

3.2.1 Structural overview

The main structural element of the ANN is the artificial neuron, as shown schematically in Figure 2(a). Artificial neurons are interconnected by means of artificial synapses. A small network of such artificial neurons and synapses is shown schematically in Figure 2(b). The diagram structures used there are standardized throughout this thesis: Inputs are represented by filled triangles, neurons are represented by filled circles, and synapses are represented by arrows. Also, as mentioned earlier, neurons with similar properties and connectivity are grouped into neuronal clusters.

The neurons of the ANN are divided into three sets: input, internal, and output. Input neurons get their drives from the outer environment, modeled thorough environment functions. Output neurons give their drives as input to environment functions. Finally, internal neurons exist on paths between input neurons and output neurons.

3.2.2 Neuronal activation function

The neuronal activation function describes the output (drive, i.e. firing rate) behavior of neurons. Much research has been carried out with respect to modeling the activation function for biological neurons mathematically, and as biologically plausible as possible. Several models exist, but variants of mathematical *sigmoid* functions are most frequently used, particularly for ANN computer simulations. However, some modifications and extensions to a basic sigmoid model of neuronal activation are seemingly propitious with respect to biological plausibility.

Lanthorn, Storm and Andersen [LSA84] demonstrated that when net input to a neuron is high, and remains high for some time interval, its output firing rate declines rapidly, in spite of constant net input. This is the concept of *firing rate adaption*. With respect to neuronal activation these empirical findings suggest that a purely static activation function is too simple

a model; it is apparently necessary to make the model more dynamic with respect to letting time and previous activity history affect neuronal output behavior.

Random bursting is another well-known concept of neurobiology, describing the tendency that the probability of a neuron exhibiting random firing behavior increases when neuronal activity has been low for some time [CS92]. In other words, one should expect that the probability of a neuron firing increases with sustained low activity. Pairing this with the above-mentioned empirical findings on decline of firing frequency for sustained high input, it is reasonable to assume that neuronal activity at average should decrease for sustained high activity and increase for sustained low activity. Mathematically, these two properties can be represented jointly by keeping a trace of neuronal activity.

Another matter is whether some kind of stochastic element should be included in the model for neuronal activity, and how large it should be compared to the deterministic elements discussed above. In spite of the dynamic character of the outer environments that the ANNs of this thesis operate in, some kind of stochastic element proves necessary to get the ANN-dynamics going.

3.2.3 Synaptic plasticity

Learning in neural networks is achieved through changes of synaptic efficacies. Modeling these changes is thus one the most important tasks in neural network theory, and there exist several different approaches to explaining how synaptic learning mechanisms operate, a few of which were mentioned in Section 2.3. As explained in Section 2.4, this thesis is based on the three learning mechanisms of Connectology, the Skinner, Pavlov and Hume synaptic learning mechanisms, as proposed by Hokland [Hok97, Hok98], and later in his Connectology research program [Hok06].

An important aspect of these learning mechanism is their temporal characteristics; changes in synaptic efficacies are based on particular sequences of changes in the drives of pre- and postsynaptic neurons [Hok06].

3.2.3.1 Skinner mechanism

The semantics of the Skinner mechanism are as follows:

- Postsynaptically increased drive followed by presynaptically decreased drive yields increased efficacy for excitatory synapses and decreased efficacy for inhibitory synapses.
- Postsynaptically decreased drive followed by presynaptically decreased drive yields decreased efficacy for excitatory synapses and increased efficacy for inhibitory synapses.

Figure 3 gives a schematic overview for excitatory synapses. The mechanism is built on the assumption that neurons try to keep themselves free from stimulus, a principle proposed by Freud [Fre95], and later Hokland [Hok97, Hok06]. According to this, increased stimuli (i.e. drive) corresponds to punishment, while decreased stimuli corresponds to reward.

By analyzing the two rules stated above, one can quite easily see that they adhere to this minimization principle, in that the rules always alter the synaptic efficacy so as to reduce the drive of the presynaptic neuron:

- If increased drive in the postsynaptic neuron resulted in decreased drive in the presynaptic neuron, the presynaptic neuron will want the drive of the postsynaptic neuron to be higher

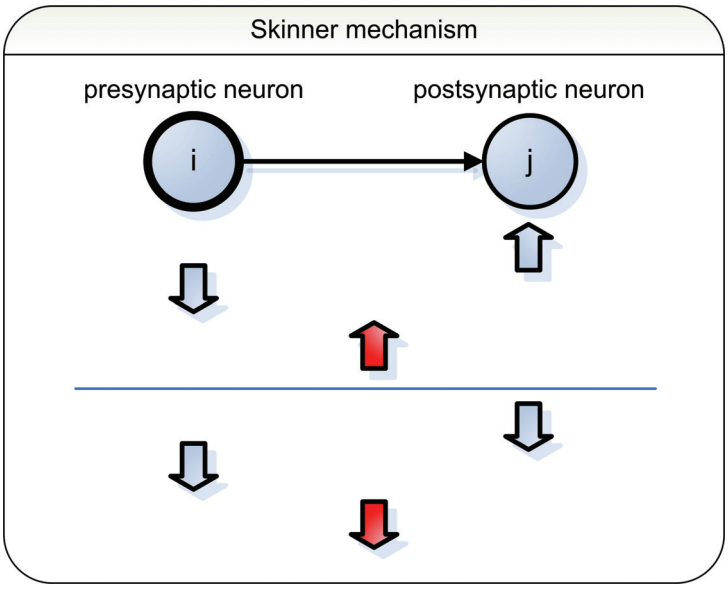


Figure 3: Schematic overview of the Skinner mechanism

(this tendency seems favorable). Therefore, the synaptic efficacy should increase if the synapse is excitatory (presynaptic drive should contribute more to driving the postsynaptic neuron), or decrease if it is inhibitory (presynaptic drive should contribute less to inhibiting the drive of the postsynaptic neuron).

- Similarly, if decreased drive in the postsynaptic neuron resulted in decreased drive in the presynaptic neuron, the presynaptic neuron will want the drive of the postsynaptic neuron to be lower (again, a favorable tendency). Therefore, the synaptic efficacy should decrease if the synapse is excitatory (presynaptic drive should contribute less to driving the postsynaptic neuron), or increase if it is inhibitory (presynaptic drive should contribute more to inhibiting the drive of the postsynaptic neuron).

An important property of the Skinner mechanism is that only decreases in presynaptic drive induce learning (i.e. changes of synaptic efficacies). This agrees with the empirical findings of such psychologists as B. F. Skinner and E. L. Thorndike, that reward leads to learning, while punishment does not [Hok06]. ANN synapses learning by the Skinner mechanism are termed *Skinner synapses*. These synapses exist as connections between needs and internal clusters (or, alternatively, as direct connections between needs and motors).

3.2.3.2 Pavlov mechanism

The semantics of the Pavlov mechanism are as follows:

- Presynaptically increased drive followed by postsynaptically increased drive yields increased efficacy for excitatory synapses and decreased efficacy for inhibitory synapses.

- Presynaptically increased drive followed by postsynaptically decreased drive yields decreased efficacy for excitatory synapses and increased efficacy for inhibitory synapses.

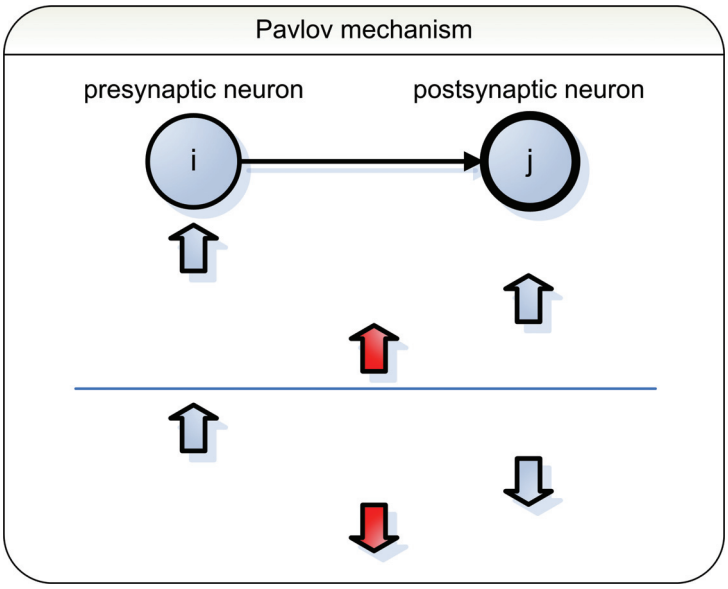


Figure 4: Schematic overview of the Pavlov mechanism

Figure 4 gives a schematic overview for excitatory synapses. The mechanism is built on the principles of classical conditioning, in that the efficacy of a synapse is changed in a direction such that the (postsynaptic) neuron becomes anticipative, i.e. it comes to anticipate the unconditioned response. These principles were suggested by Pavlov [Pav28] at the system level, and later taken to the neuronal level by Klopf [Klo88] and Hokland [Hok97, Hok06].

By analyzing the two rules stated above, one can quite easily see that they adhere to this anticipation principle, in that the rules always alter the synaptic efficacy so as to anticipate the drive of the postsynaptic neuron:

- If increased drive in the presynaptic neuron is followed by increased drive in the postsynaptic neuron, to make the postsynaptic neuron anticipative, the synaptic efficacy should increase if the synapse is excitatory (presynaptic drive should contribute more to driving the postsynaptic neuron) and decrease if it is inhibitory (presynaptic drive should contribute less to inhibiting the postsynaptic neuron). Later, then, if this tendency is repetitive, increases of presynaptic drive will induce increases of postsynaptic drive, and the postsynaptic neuron has become anticipative.

- If increased drive in the presynaptic neuron is followed by decreased drive in the postsynaptic neuron, to make the postsynaptic neuron anticipative, the synaptic efficacy should decrease if the synapse is excitatory (presynaptic drive should contribute less to driving the postsynaptic neuron) and increase if it is inhibitory (presynaptic drive should contribute more to inhibiting the postsynaptic neuron). Once again, if this tendency is repetitive, subsequent increases of presynaptic drive will induce decreases of postsynaptic drive, and the postsynaptic neuron has become anticipative.

An important property of the Pavlov mechanism is that only increases of presynaptic drive induce learning (i.e. changes of synaptic efficacies).[10] ANN synapses learning by the Pavlov

[10]Klopf argues why this positivity requirement is appropriate: "A negative change in presynaptic signal level means that the presynaptic signal is falling away - that it is headed toward zero. If such a negative change in

mechanism are termed *Pavlov synapses*. These synapses exist as connections between senses and internal clusters (or, alternatively, as direct connections between senses and motors). Additionally, Pavlov synapses exist as connections between internal clusters, both directed from inputs to outputs (forming habit paths) and from outputs to inputs (forming attention paths).

3.2.3.3 Hume mechanism

The semantics of the Hume mechanism are as follows:[11]

- Presynaptically increased drive followed by postsynaptically decreased drive yields increased efficacy for excitatory synapses and decreased efficacy for inhibitory synapses.
- Presynaptically increased drive followed by postsynaptically increased drive yields decreased efficacy for excitatory synapses and increased efficacy for inhibitory synapses.

Figure 5 gives a schematic overview for excitatory synapses. The mechanism is built on the assumption that living creatures know the objects of the external world through their *contours*.[12] The neural representation of concrete objects and abstract phenomena are collectively termed *concepts*, and the two rules stated above are termed *concept assimilation* and *concept accommodation*, respectively. Concepts arise within neural clusters. The workings of the concept assimilation mechanism are to extract contours of concrete objects or, correspondingly, borders of abstract phenomena. Concept assimilation, on the other hand, separates different (kinds of) concepts by making them inhibit each other.

By analyzing the two rules stated above, one can see that the first (concept assimilation) adheres to the contour extraction principle and that the second (concept accommodation) adheres to the concept differentiation principle, as follows:

- Concept assimilation: Consider two synaptically connected neurons spaced to cover opposite sides of the border of some concept. If this concept is moving across the neuronal surface in the direction from the postsynaptic to the presynaptic neuron, there will at some time be an increase of drive in the presynaptic neuron (the concept just reached the presynaptic neuron) shortly followed by a decrease of drive in the postsynaptic neuron (the concept just left the postsynaptic neuron). To assimilate the concept, the presynaptic neuron should learn to drive the postsynaptic neuron. Consequently, the synaptic efficacy should increase if the synapse is excitatory (presynaptic drive should contribute more to driving the postsynaptic neuron) or decrease if it is inhibitory (presynaptic drive should contribute less to inhibiting the postsynaptic neuron). Over time, one can expect that the concept will move across the neuronal surface in all directions, thus allowing for a full concept border representation to emerge.
- Concept accommodation: When two neurons inside a cluster represent different (potentially competing) concepts, they should be expected to inhibit each other [Hok06]. Hokland further proposes that neurons inside clusters compete for earliest warning [Hok06],

presynaptic signal level were to trigger the neuronal learning mechanism and possible cause a synaptic weight change, then a synaptic weight would have changed for a synapse that had just ceased to carry the signal that caused the change. That is to say, the relevant part of the signal on which the synaptic weight should operate would no longer be present." ([Klo88], p. 89)

[11] For ease of recollection, notice that the Hume mechanism is identical to Pavlov mechanism except that the direction of efficacy changes are inverted.

[12] Quoting [Kan00], on the primary visual cortex: "In fact, contour information may be sufficient to recognize an object. Monotonous interior or background surfaces contain no critical visual information!" (p. 537)

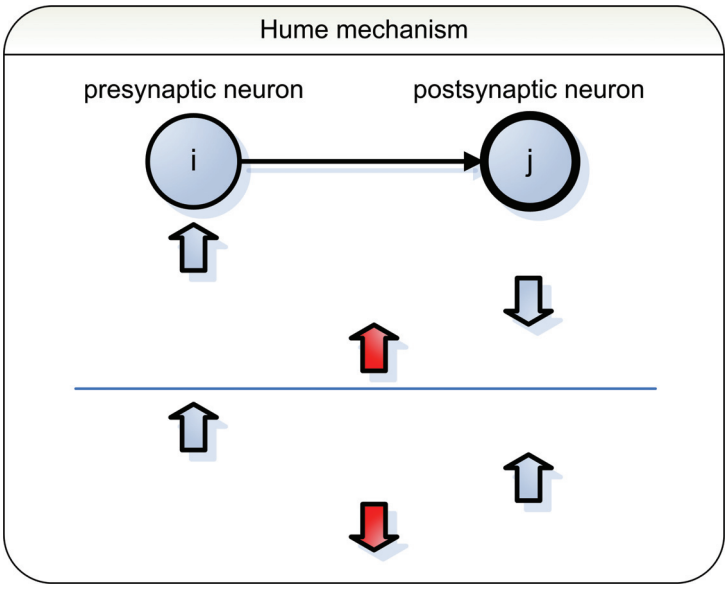


Figure 5: Schematic overview of the Hume mechanism

i.e. they compete to be first to represent a concept. Now, consider a presynaptic and a postsynaptic neuron within the same cluster. If increased drive in the presynaptic neuron is followed by increased drive in the postsynaptic neuron, this can be regarded as the two neurons competing to represent the same concept. By letting this sequence of drive changes induce synaptic change such that the efficacy is decreased for excitatory synapses (presynaptic drive contributes less to driving the postsynaptic neuron) and increased for inhibitory synapses (presynaptic drive contributes more to inhibiting the postsynaptic neuron), this hopefully should encourage the passified postsynaptic neuron to take on new purposes and functions [Hok06].

Note that, as for the Pavlov mechanism, only increases of presynaptic drive induce learning. ANN synapses learning by the Hume mechanism are termed *Hume synapses*. These synapses only exist as connections inside clusters, i.e. between neurons belonging to the same cluster.

3.2.3.4 Summary

The connectological mechanisms described above contrast the still extensively popular Hebbian rule for animal learning, proposed by Hebb [Heb49]. The Hebbian rule states that synaptic efficacies change as a result of *simultaneous activity* in presynaptic and postsynaptic neurons.[13] To summarize the above, it can be noted that Hokland adopts two fundamental modifications to the Hebbian model, based on the work of Klopf [Klo88]: Firstly, *changes* in levels of activity induce learning, not levels of activity. Secondly, rather than simultaneity, the *temporal order* of presynaptic and postsynaptic activity changes is what induces learning. For ease of comparison and recollection, Figure 6 provides an overview of the three synaptic learning mechanisms discussed above.

[13] Hebb proposed that "When an axion of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased." ([Heb49], p. 50)

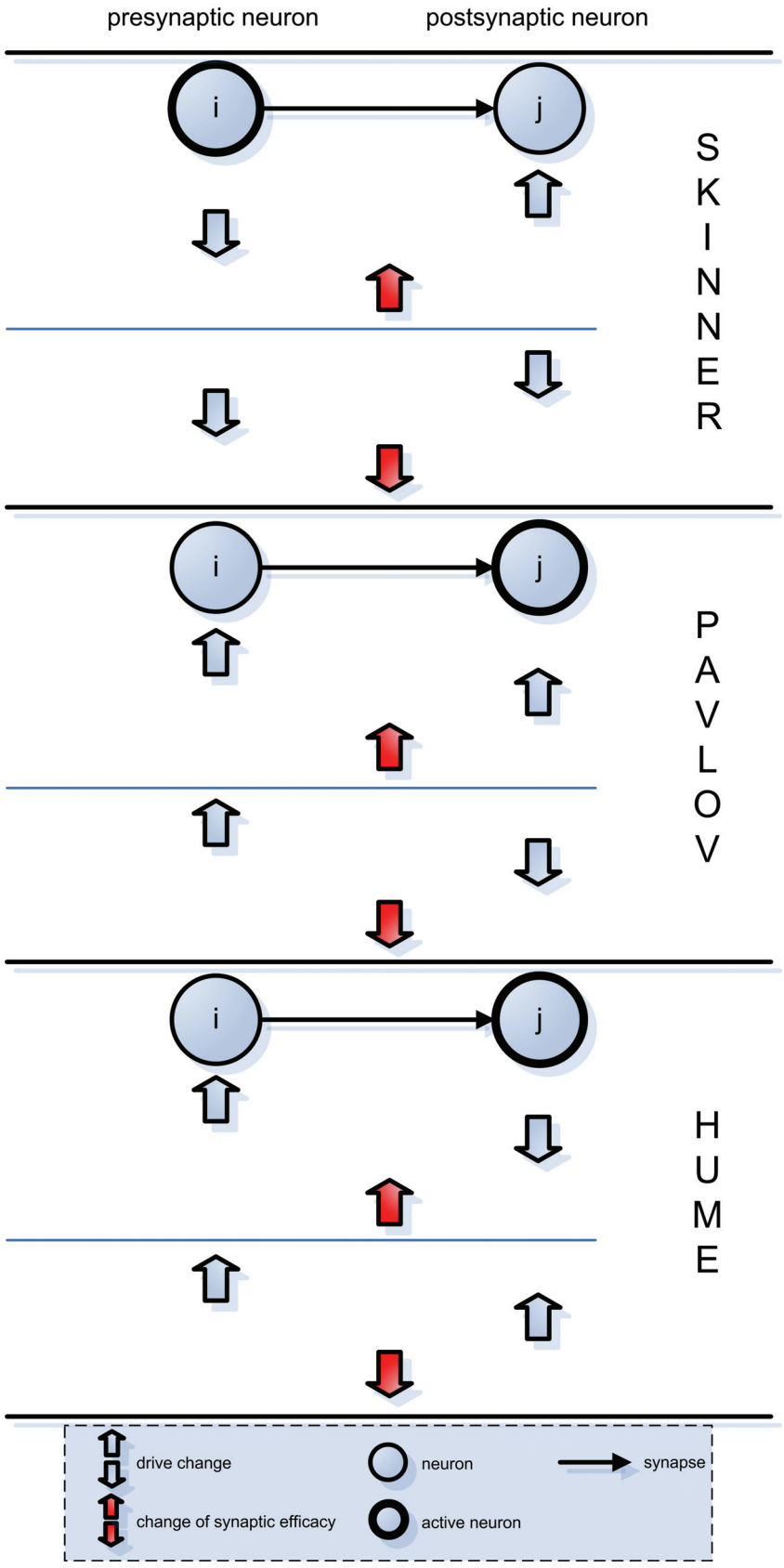


Figure 6: Schematic overview of the synaptic learning mechanisms of Connectology

3.2.4 Neuronal classification

For the neural simulations of this thesis, where ANNs control and receive feedback from a biologically inspired mechanical model operating in a physically realistic outer environment, it is natural to relate the different classes of neurons to their biological counterparts. This section discusses three well-defined classes of neurons which are highly relevant for connectological modeling and simulation: *needs*, *senses* and *motors*. In addition to these three classes which are either inputs or outputs, there are general purpose internal neurons.

3.2.4.1 Needs

Quoting [Hok06]:

> *A need is a type of receptor system on which a push is instinctively, i.e. innately, transformed along drive paths into behavioral pressure to remove that push. (p. 103)*

Needs represent unpleasant or unwanted signals, and biological systems behave so as to remove them (or make them as low as possible). For the ANNs simulated in this thesis, needs are represented by input neurons. A few examples of needs in biological systems are [Kan00]: *glucoreceptors* pushed by low glucose levels (hunger), *osmoreceptors and baroreceptors* pushed by low levels of liquids (thirst), *nocireceptors* pushed by painful objects against the skin (tissue damage) and *thermoreceptors* pushed by non-ideal skin and blood temperatures (cold, heat).

For the simulation purposes of this thesis, needs have a strong and direct connection to the Skinner synapse: A need is by current definition the presynaptic neuron of a Skinner synapse continuously receiving its drive value directly from the mechanical model.[14] As explained in Section 3.2.3.1, Skinner synapses function so as to minimize the drive of their presynaptic neuron. Consequently, in our simulations, needs are subjected to minimization. Further, as explained in Section 3.1, the set of need sources provided by the mechanical model as need inputs to the ANN fully and wholly determines the behavioral goal specification for the simulated system.

3.2.4.2 Senses

Senses are similar to needs, in that they are represented by input neurons in the ANNs. Further, as for need inputs, sense inputs continuously receive their drive values directly from the mechanical model. There is, however, one significant distinction between needs and senses: senses represent neutralized environmental signals and are therefore not subjected to minimization. As a consequence of this, instead of Skinner synapses, Pavlov synapses are used to connect sense inputs to the first network layer, providing the ANN with potential capabilities at *anticipating* postsynaptic drives based on presynaptic sense values (for details on Pavlov synapses, see Section 3.2.3.2). The connection between senses and Pavlov synapses is not as strong as the connection between needs and Skinner synapses; Pavlov synapses also exist on internal paths in the ANN, linking the first network layer to other layers including the output layer.

[14] Recall from Section 2.5 that Skinner synapses are normally only used for connecting need input values (i.e. one type of environmental feedback) to the first layer of the ANN. An exception is, however, made herein in connection with affect systems (Section 3.2.6.1), allowing the presynaptic neuron of a Skinner synapse to be an affect neuron.

3.2.4.3 Motors

Motors are inherently different from both needs and senses because they are represented by the output neurons of the ANN; all output neurons are motors. Motor neurons control behavior; the behavior of a neural network at any instant is defined by the momentaneous drive values of motor neurons, and behavior during some time interval is thereby determined by the course of motor drive values over the interval. For the simulations of this thesis, motors are tightly connected to the mechanical model (virtual creature): they dictate the force generated at creature muscles. Hence, motors control the creature body, and the time-dimensional patterns generated at motors thus constitute what ultimately becomes the measurement of system performance.

Summarizing the above, and incorporating the intuition provided on synaptic learning, the overall goal of the neural system specified in Section 3.1 can be refined further as follows:

> The ANN should steer motor output values so as to minimize need input values utilizing neutralized sense input values for anticipation and concept learning.

3.2.5 Mathematical specification

The strategies that dictate how the calculations in the computer simulations are to be performed are of highest importance. This section makes them formal by defining them in a precise mathematical language.

3.2.5.1 Notational remarks

The drive of a neuron j is represented symbolically as D_j, and synaptic efficacy for a synapse from a (presynaptic) neuron i to a (postsynaptic) neuron j is represented as e_{ij}. Additionally, the trace (history) of neuronal drive for a neuron j will be termed acc_j, alluding the accumulating character of mathematical traces.

Each neuron is connected to a number of other neurons through incoming and outgoing synapses. For a neuron j, the set of presynaptic neurons on incoming synapses is termed $\mathcal{I}_j^*$, and the set of postsynaptic neurons on outgoing synapses is termed $\mathcal{O}_j^*$.

The mathematical specification will be based on discrete time steps, such that D_j^t and e_{ij}^t represent drive and synaptic efficacy at time t, respectively. More, the changes in drive and efficacy between two consecutive time steps $t-1$ and t are represented as

$$\Delta D_j^t = D_j^t - D_j^{t-1}$$

and

$$\Delta e_{ij}^t = e_{ij}^t - e_{ij}^{t-1},$$

respectively. Yet another value is important in connection with changes in synaptic efficacy, namely the synaptic trace of presynaptic or postsynaptic drive differentials. For a synapse between neuron i and neuron j, this trace value is represented by T_{ij}. For the Skinner mechanism this trace is postsynaptic, whereas for the Pavlov and Hume mechanisms the traces are presynaptic. The same symbol T_{ij} will be used to represent this synaptic trace of drive differentials for all learning mechanisms. Hence, the exact meaning of T_{ij} is context sensitive.

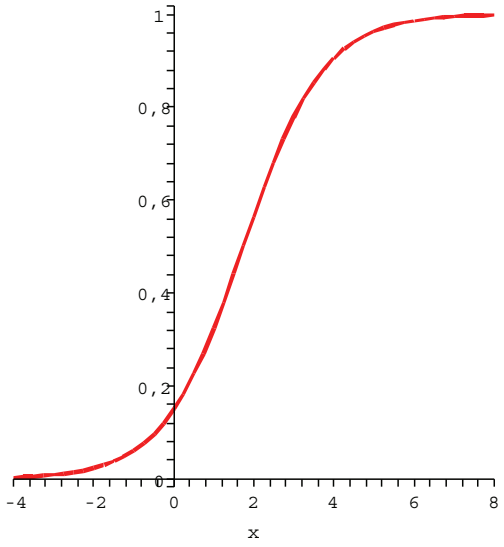


Figure 7: Sigmoid neuronal activation function for $x_0 = 0.15$

3.2.5.2 Neuronal drive

Neuronal drive (i.e. the output of a neuron) is calculated using the following *sigmoid* function (Figure 7):

$$g(x) = \frac{1}{1 + e^{-(x - \ln((1-x_0)/x_0))}}, \tag{1}$$

where x_0 is the y-axis crossing point. For the simulations of this thesis, $x_0 = 0.15$.

Activation function Three elements affect the neuronal activation function: net drive input *net*, neuronal drive trace *acc* and stochastic perturbation *stoc*. The complete neuronal activation function can then be specified as follows (for neuron j at time t):

$$D_j^t = g\left(net_j^t - acc_j^t + stoc_j^t\right) = \frac{1}{1 + e^{-\left(net_j^t - acc_j^t + stoc_j^t - \ln((1-x_0)/x_0)\right)}} \tag{2}$$

The signs of the elements in the sum of the exponential (i.e. the input argument to the sigmoid function) are important. High values of net drive input *net* should give high output, and vice versa, and *net* must therefore have a positive sign. The neuronal trace element *acc*, on the other hand, is converse, in that a high value should lower the output, and vice versa; it must therefore be preceded by a minus sign. For the stochastic element *stoc*, the sign is of no importance, as the distribution of the samples will be symmetric with respect to the vertical zero-axis.

The following describes the three elements of the neuronal activation function in detail.

Net drive input The net drive input to a neuron is calculated as a weighted sum over all incoming synapses and their presynaptic neurons. Considering neuron j at time t,

$$net_j^t = \sum_{i \in \mathcal{I}_j^*} e_{ij}^t D_i^t. \tag{3}$$

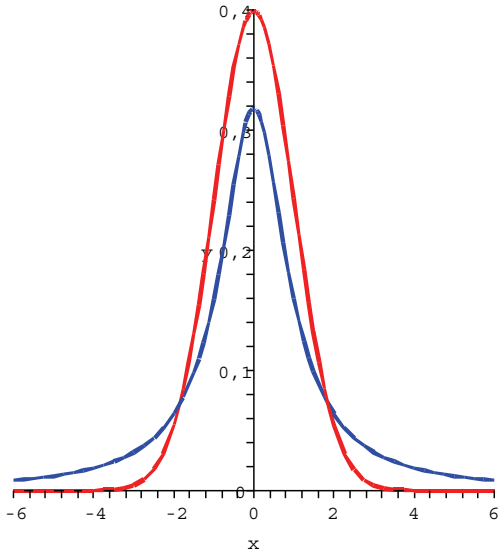


Figure 8: Gaussian probability distribution with $\sigma = 1$ compared to Cauchy probability distribution with $\gamma = 1$

Neuronal trace As discussed in Section 3.2.2, the neuronal drive should be dynamic with respect to time and previous drive history (sustained high input should lower the probability of firing, and vice versa). This can be accomplished by incorporating the neuronal trace *acc* into the activation function. The trace for neuron j at time t is defined as

$$acc_j^t = (1 - \alpha_j)acc_j^{t-1} + \alpha_j(D_j^t - 0.5), \tag{4}$$

where α_j is a number between 0 and 1 that controls the shape of the trace function; the larger α_j is, the more sensitive the trace is to recent changes, and vice versa. By subtracting 0.5 from the drive value D, *acc* is made symmetric with respect to zero, such that sustained low drive values give a negative neuronal trace, while sustained high drive values give a positive trace. The neuronal trace specified above will range between -0.5 and 0.5. When used in the neuronal activation function, as specified in Eq. (2), the neuronal trace acts as a contrast term that contributes to letting neurons utilize the range of drive values more uniformly.

Stochastic perturbations Section 3.2.2 also mentions the necessity of stochastic perturbation in the activation function. A few approaches to sampling stochastic perturbations have been considered in this thesis. The first, and simplest, is sampling from the Gaussian probability distribution $G(\mu, \sigma^2)$ (Figure 8):

$$G(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{5}$$

where μ is the mean, and σ^2 is the variance. For direct sampling purposes, $\mu = 0$, which yields a symmetric distribution around zero, as desired. Sampling from the Gaussian distribution is performed using the Box Muller algorithm [Rip87] (Algorithm 1).

Algorithm 1 Box Muller

Let U_1, U_2 be uniformly distributed random variables on $(0, 1)$
Let $X \leftarrow \sqrt{-2 \ln U_1}$
Let $Y \leftarrow 2\pi U_2$

Then
$N1 \leftarrow \mu + \sigma X \cos(Y)$ and
$N2 \leftarrow \mu + \sigma X \sin(Y)$ and
are independent Gaussian variables with mean μ and variance σ^2

Another alternative is based on sampling from the Cauchy probability distribution (Figure 8):

$$f(x; x_0, \gamma) = \frac{1}{\pi\gamma \left[1 + \left(\frac{x - x_0}{\gamma}\right)^2\right]} \tag{6}$$

Here, x_0 is the location parameter that specifies the location of the peak of the distribution. For the purpose of sampling stochastic perturbations, we use $x_0 = 0$ (which is analogous to using $\mu = 0$ for the Gaussian). The parameter γ is the scale parameter, specifying the half-width at half-maximum (similar to the standard deviation parameter σ of the Gaussian). A reason for using the Cauchy probability distribution instead of, say, a Gaussian probability distribution, might be the assumed favorable (and easily controllable) large-tail property of the Cauchy, allowing for occasional large stochastic perturbations, while still retaining the stability achieved from most of the samples being from within a small interval (chosen by setting the γ parameter). See Figure 8 for a visual comparison of the two.

Instead of sampling values directly from either the Cauchy distribution or the Gaussian distribution, Metropolis sampling [Tan06] can be used (Algorithm 2).

Algorithm 2 Metropolis sampling

Let $X_1, X_2, ...$ be a chain of random variables
Let π denote a base density function
Let f denote a symmetric transition probability function

If the chain is currently at $X_n = x$, generate candidate value y^* for next location X_{n+1} from $f(x)$
With probability $\alpha(x, y^*) = \min\left\{\frac{\pi(y^*)}{\pi(x)}, 1\right\}$ accept the candidate value and move the chain by letting $X_{n+1} \leftarrow y^*$
Otherwise, reject and let $X_{n+1} \leftarrow x$

Relating our sampling to the algorithmic specification, π is the Cauchy or the Gaussian, and f is some symmetric transition function, such as the uniform distribution or the Gaussian distribution. This sampling strategy introduces inertia to the stochastic perturbation, in that a sample normally will be in the vicinity of the previous sample. In other words, the distance between successive samples is reduced at average. Still, the Metropolis sampling algorithm guarantees that the distribution of the samples will approach the distribution of the base density function π as more and more samples are generated (i.e. when the number of samples tends to

infinity). The inertia property of the Metropolis sampling algorithm might be favorable with respect to simulation stability, because the average distance between samples is reduced.

To summarize, sampling stochastic perturbations can be done by sampling directly from the Gaussian or the Cauchy,[15] or Metropolis sampling can be used with a Cauchy or Gaussian as the base density and a symmetric transition function. For the latter, in accordance with Algorithm 2, the mean of the symmetric transition function is always the previous sample. Thus, programmatically, each neuron must keep track of its stochastic perturbation, so that the previous value can be used to generate the next value.

3.2.5.3 Synaptic plasticity

The three synaptic learning mechanisms presented in 3.2.3 are all based on the same basic structure of temporally ordered drive changes in the presynaptic (ΔD_i) and postsynaptic (ΔD_j) neurons. The general equation describing changes of synaptic efficacies is as follows:

$$\Delta e_{ij}^t = \psi \beta_{ij} f_i(\Delta D_i^t) f_j(\Delta D_j^t), \tag{7}$$

where ψ is the sign (+ or −). f_i and f_j represent functions (discussed below) on presynaptic and postsynaptic delta drives, respectively. The requirements on temporal ordering do not appear from the equation itself.

As discussed earlier, synapses can be excitatory or inhibitory. We let negative synaptic efficacies define inhibitory synapses.[16] Recall that for all three learning mechanisms presented in Section 3.2.3 inhibitory and excitatory synapses always change in an opposite manner. Thus, by letting negative synaptic efficacies define inhibitory synapses, the two cases (excitatory and inhibitory) of a synaptic mechanism can be expressed using one equation only. Note also that the equations of the form in Eq. (7) allow for synapses to change between being excitatory and inhibitory, there is no restriction on zero-crossings. By allowing efficacies to vary freely, there is no need to prespecify synapses as excitatory or inhibitory; hopefully, synapses will self-organize into excitatory and inhibitory, as needed. The latter is supported by the findings in [Axe06], where it is shown that for simple networks of limited size, synapses appropriately stabilize as excitatory or inhibitory.

Going back to f_i and f_j, these will either be the identity function or some mathematical trace function representing the history of presynaptic or postsynaptic drive differentials, both possibly excluding negative or positive delta drive contributions (not shown below). For the presynaptic neuron i:

$$f_i(\Delta D_i^t) = D_i^t \quad \text{or} \tag{8}$$

$$f_i(\Delta D_i^t) = T_{ij}^t = (1 - \alpha_{ij}) T_{ij}^{t-1} + \alpha_{ij} \Delta D_i^t \tag{9}$$

The parameter α_{ij} of Eq. (9) is important, as it determines the shape of the trace function. The larger α_{ij}, the more sensitive the trace is to recent changes, and vice versa.

Inspecting Eq. (7), changes in synaptic efficacies are proportional to the change of both presynaptic drive and postsynaptic drive (or their corresponding differential traces). The parameter β_{ij} is a learning rate parameter, allowing for adjustments with respect to the trade-off between rate of learning on one hand, and accuracy and stability on the other. A small β_{ij}

[15] Direct Cauchy sampling, not yet discussed, can easily be performed using e.g. inversion sampling [Rip87].

[16] Mathematically, that is. The biological separation between inhibitory and excitatory synapses is more intricate. For computer simulation purposes, however, this definition is the most common.

will give slow learning with high stability and accuracy, and increasing the β_{ij} parameter gives increased rate of learning at the cost of decreased stability and accuracy.

The ij-subscripts on α and β denote that the parameters potentially are specific to a single synapse, allowing for varying values of α and β for different synapses.

Skinner mechanism The Skinner synaptic learning mechanism was first presented in [Hok97] and later in the Connectology research programme [Hok06]. The two rules describing Skinner synapses, as specified in Section 3.2.3.1, can be composed into the following equations (at time step t):

$$\Delta e_{ij}^t = -\beta_{ij} \min\big(\Delta D_i^t, 0\big) T_{ij}^t \tag{10}$$

$$T_{ij}^t = (1 - \alpha_{ij}) T_{ij}^{t-1} + \alpha_{ij} \Delta D_j^t, \tag{11}$$

where the Skinner trace of postsynaptic drive differentials represents recent delta drive history at the postsynaptic neuron.

Pavlov mechanism The Pavlov synaptic learning mechanism is basically identical to the learning mechanism introduced by A. Harry Klopf in [Klo88]. Hokland adopted this classically conditioned mechanism in [Hok97] and later in the Connectology research programme [Hok06]. The two rules describing Pavlov synapses, as specified in Section 3.2.3.2, can be composed into the following equations (at time step t):

$$\Delta e_{ij}^t = \beta_{ij} T_{ij}^t \Delta D_j^t \tag{12}$$

$$T_{ij}^t = (1 - \alpha_{ij}) T_{ij}^{t-1} + \alpha_{ij} \max\big(\Delta D_i^t, 0\big), \tag{13}$$

where the Pavlov trace of presynaptic drive differentials represents recent delta drive history at the presynaptic neuron. Note that in the above specification of the Pavlov mechanism (which is equivalent to the one given in [Klo88]) the positivity check for the presynaptic delta drive is provided in the trace function. Alternatively, and in a manner more similar to the above Skinner mechanism, the positivity check on presynaptic delta drive can be provided directly in the learning mechanism equation:

$$\Delta e_{ij}^t = \beta_{ij} \max\big(T_{ij}^t, 0\big) \Delta D_j^t \tag{14}$$

$$T_{ij}^t = (1 - \alpha_{ij}) T_{ij}^{t-1} + \alpha_{ij} \Delta D_i^t \tag{15}$$

Both of these alternative specifications of the Pavlov mechanism will be examined experimentally in this thesis.

Hume mechanism The Hume synaptic learning mechanism was first presented in [Hok98] and later in the Connectology research programme [Hok06]. The two rules describing Hume synapses, as specified in Section 3.2.3.3, can be composed into the following equations (at time step t):

$$\Delta e_{ij}^t = -\beta_{ij} T_{ij}^t \Delta D_j^t \tag{16}$$

$$T_{ij}^t = (1 - \alpha_{ij}) T_{ij}^{t-1} + \alpha_{ij} \max\big(\Delta D_i^t, 0\big), \tag{17}$$

where the Hume trace of presynaptic drive differentials represents recent delta drive history at the presynaptic neuron. Alternatively, as for the Pavlov mechanism described above, the

positivity check for the presynaptic delta drive can be done directly in the learning mechanism equation:

$$\Delta e_{ij}^t = -\beta_{ij} \max\left(T_{ij}^t, 0\right) \Delta D_j^t \tag{18}$$

$$T_{ij}^t = (1 - \alpha_{ij}) T_{ij}^{t-1} + \alpha_{ij} \Delta D_i^t \tag{19}$$

3.2.5.4 Divergence preventing modifications

Considering first the Skinner mechanism, it is necessary to make one modification to the basic specification of its trace of postsynaptic drive differentials T_{ij}. The trace function stated in Eq. (11) could lead to problems with self-reinforcement: If the change in postsynaptic drive ΔD_j was caused (partly or exclusively) by a change in presynaptic drive ΔD_i, the resulting efficacy change (Eq. (10)) could potentially lead to a similar change in ΔD_i, causing a similar change in ΔD_j, and so on. The potential for divergence of synaptic efficacies is evident. Excluding the contribution from the presynaptic drive change ΔD_i to the postsynaptic drive change ΔD_j solves this problem, by removing the premises for self-reinforcement. This problem is also pointed out in [Hok06]:

> *[T]o avoid i from keep reducing e_{ij} we may speculate that a change in $[T_{ij}^t]$ depends only on that contribution of change to $[\Delta D_j^t]$ that is driven, not by neuron i, but by all presynaptic neurons other than neuron i. Thus, to the extent that the change in $[\Delta D_j^t]$ is due to a drive change in $[\Delta D_i^t]$, this leaves no trace in $[T_{ij}^t]$ at that synapse, to prevent i through j and behavior from keep reinforcing itself. (p. 110, footnote 207)*

Considering Eq. (11), the ΔD_j factor must be altered to exclude the contribution from ΔD_i to ΔD_j. Mathematically, this can be specified as (for time step t):

$$\Delta_i^* D_j^t = g\left(\sum_{\substack{k \in I_j^* \\ k \neq i}} e_{kj}^t D_k^t + e_{ij}^{t-1} D_i^{t-1} - acc_j^t + stoc_j^t\right) - g\left(net_j^{t-1} - acc_j^{t-1} + stoc_j^{t-1}\right) \tag{20}$$

where g denotes the neuronal activation function from Eq. (2). The second term of Eq. (20) is simply D_j^{t-1}. It can be seen that the only difference from the normal computation for drive change ($\Delta D_j^t = D_j^t - D_j^{t-1}$) is that the term $e_{ij}^t D_i^t$ in the net input to neuron j at time t has been replaced with $e_{ij}^{t-1} D_i^{t-1}$, i.e. the same term from the previous time step. Hence, the most recent presynaptic contribution to postsynaptic change is given zero influence.

Using Eq. (20), the modified mathematical specification for the Skinner trace of postsynaptic drive differentials is then:

$$T_{ij}^t = (1 - \alpha_{ij}) T_{ij}^{t-1} + \alpha_{ij} \Delta_i^* D_j^t. \tag{21}$$

For later reference, the divergence preventing learning mechanism described above is termed **MOD1**.

Another modification, which is independent of the underlying learning mechanism, proves to be necessary to prevent divergence on synaptic efficacies. Consider a synapse with its presynaptic and postsynaptic neurons, and where keeping the postsynaptic drive at a given level leads to drive reduction in presynaptic drive (e.g. when satisfying a need). Then, the lower the presynaptic drive gets, the higher the synaptic efficacy must be in order for the presynaptic neuron

to drive the postsynaptic neuron to its optimal drive value.[17] Hence, this situation will lead to diverging synaptic efficacies. Divergence can be prevented by introducing an upper limit on the sum of efficacies for the incoming synapses of a neuron. Based on discussions at supervisor meetings concerning simulation cases showing diverging behavior of the type outlined above, Hokland has suggested the following restriction on synaptic weights:

$$\sum_{i \in \mathcal{I}_j^*} |e_{ij}| \leq \left|\mathcal{I}_j^*\right| e_{max}, \tag{22}$$

where $|\mathcal{I}_j^*|$ is the number of neurons on incoming synapses for neuron j, and e_{max} is a constant specifying how large the efficacy of a synapse may become, at average.

Experiments carried out in [Axe06] indicate that the above limit, which is linear in $|\mathcal{I}_j^*|$, is too simple a model when networks become larger, and the average number of incoming synapses grows significantly.[18] As an alternative, therefore, I suggest that the right side of Eq. (22) is replaced by a function which is nonlinear in $|\mathcal{I}_j^*|$. Therefore, throughout this thesis, the following limit function is used:

$$h(\left|\mathcal{I}_j^*\right|) = \mathrm{A}\upsilon \left(1 - \exp\left[\frac{1 - \left|\mathcal{I}_j^*\right|}{\upsilon} \right] \right) + \upsilon, \tag{23}$$

where A controls the steepness of h (i.e. how fast the limit grows with $|\mathcal{I}_j^*|$), and υ is the limit when $|\mathcal{I}_j^*| = 1$, i.e. when a neuron has only one incoming synapse. For the experiments of this thesis, $\mathrm{A} = 10$ is used, and υ is a free parameter that may vary among different simulations. The effect of Eq. (22) is, of course, that the average maximum absolute efficacy a synapse may obtain gets smaller as the number of incoming synapses to a neuron gets larger. Figure 9 shows $h(\left|\mathcal{I}_j^*\right|)$ plotted for two different values of υ.

Formally, then, the restriction on synaptic weights can be expressed as:

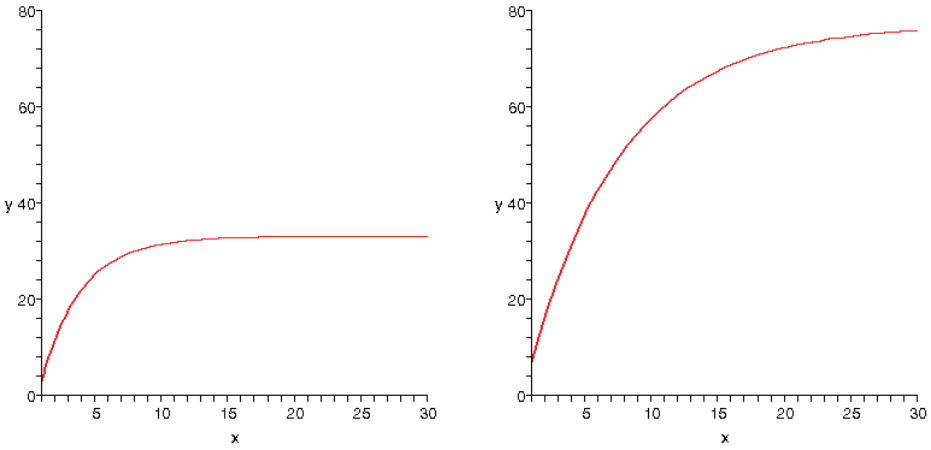
$$\sum_{i \in \mathcal{I}_j^*} |e_{ij}| \leq h(\left|\mathcal{I}_j^*\right|), \tag{24}$$

Algorithmically, this can be enforced as specified in Algorithm 3. If the maximum limit is exceeded for some postsynaptic neuron j, all incoming synapses ij are scaled geometrically such that the new sum of efficacies equals the exact limit. Scaling is done once per iteration, meaning that all synaptic efficacies are updated before limits are checked.[19] Geometric scaling also guarantees that the operation never causes the efficacy to cross zero (i.e. it will not go from excitatory to inhibitory, or vice versa). For later reference, the divergence preventing learning mechanism described above is termed **MOD2**.

[17] Recall from Eq. (3) that input from one (presynaptic) neuron to another (postsynaptic) neuron is given by the product of the presynaptic drive and the efficacy of the synapse connecting them.

[18] The problem, as pointed out in [Axe06], is that when $|\mathcal{I}_j^*|$ grows large and the limit is linear in $|\mathcal{I}_j^*|$, synapses with high effective learning rates ("fast learners") typically obtain inordinately large efficacies.

[19] An alternative approach would be to perform the limit checks after each efficacy update. The latter has somewhat different semantics, in that if two synapses compete to drive/inhibit the same postsynaptic neuron, the one that is updated first has a greater chance of growing at the expense of the other. The chosen strategy, where synaptic update order is irrelevant, is in a sense fairer, in that all synapses have equal growth opportunities. Also, by doing the limit checking and efficacy scaling once per iteration, the asymptotic running time is reduced by a factor of $|E|$ (see Section 3.2.7.1 for a detailed ANN run time analysis).



(a) A = 10, $\upsilon = 3$ **(b)** A = 10, $\upsilon = 7$

Figure 9: Efficacy limit function $h(|\mathcal{I}_j^*|)$ for MOD2 divergence preventing mechanism (Eq. (23))

Algorithm 3 Maximum summed synaptic efficacies (MOD2)

{Consider neuron j}
Let $\Sigma \leftarrow 0$
for each $i \in \mathcal{I}_j^*$ **do**
  $\Sigma \leftarrow \Sigma + |e_{ij}|$
end for
if $\Sigma > h(|\mathcal{I}_j^*|)$ **then**
  for each $i \in \mathcal{I}_j^*$ **do**
    $e_{ij} = e_{ij} \frac{h(|\mathcal{I}_j^*|)}{\Sigma}$
  end for
end if

3.2.6 Neural network topology

As pointed out earlier, ANN topology is to be defined at the clustral level; network connectivity is not specified directly as synaptic connections between presynaptic and postsynaptic neurons, but as cluster-to-cluster connections indicating that neurons of some presynaptic cluster connect onto neurons of some postsynaptic cluster according to predefined rules, and by a given synaptic learning mechanism.

The cluster connection strategies used for the simulations of this thesis are simple and based on the all-to-all principle. In later simulations more complex connection strategies may be needed, but at this point all-to-all connections between and within clusters are believed to be adequate. Two profound arguments support this choice: Firstly, a more fine-grained control over neuron-to-neuron synaptic connections requires more complex programmatic connection semantics, and as a result the initial manual setting up of topology specifications would be much more time consuming.[20] Secondly, and perhaps most importantly, at the present point

[20]This argument also applies when ANN topologies are searched by means of genetic algorithms (Section 3.4); a more fine-grained control over synaptic connections vastly increases the size of the search space, making the

of time one usually does not know which specific synaptic connections are needed for some wanted behavior to emerge. By including all possible connections between the neurons of two clusters, no possibilities are ruled out. Hopefully, connections that are in effect not needed should self-organize to cancel out (with close to zero synaptic efficacy).[21]

The rule for inter-cluster connections used for the simulations of this thesis is simple: Every neuron of the presynaptic cluster connects onto every neuron of the postsynaptic cluster. All inter-cluster synapses are Skinner or Pavlov synapses, and within the set of synapses constituting an inter-cluster connection, all synapses are of the same type.

In addition to inter-cluster connections, there may be intra-cluster connections, i.e. synaptic connections between neurons inside the same cluster. The rule for intra-cluster connections is similar to the above: Every neuron connects onto every other neuron inside the same cluster (i.e. not itself). All intra-cluster synapses are Pavlov or Hume synapses.[22] Further, as for inter-cluster connections, all synapses within one cluster are of the same type.

3.2.6.1 Affect systems

One extension to the topological building blocks discussed thus far is the use of *affect systems* in connection with need inputs. Affect systems are closely related to need inputs; one can say that an affect cluster is an ANN-internal representation of the corresponding need input cluster to which other clusters relate and connect. The topology of an affect system is depicted in Figure 10.

Figure 10 introduces a new type of connections termed *constant*. These are not synaptic connections, but merely constant and unalterable transmitters of neuronal signals from need inputs to affect clusters. The affect cluster has exactly as many neurons as there are need inputs, and there is a one-to-one connection mapping between need inputs and affect neurons. Each neuron of the affect cluster can be regarded as a neuronal representation of one specific need input. Because of this, affect neurons are not subjected to stochastic perturbations in the calculation of drive values.

When not considering other synaptic inputs, need values are mapped directly onto drive values of corresponding affect neurons. Other neurons may, however, connect onto affect neurons, and thus influence drive values. Need input contributions to affect neuron drives are calculated by inversion of the need values. This value is included in the standard summed synaptic input, and the total *net* input dictates the final drive value, as calculated by the standard sigmoid

GA process correspondingly more time consuming.

[21] There is, of course, a possibility that superfluous synapses can disturb the ANN in its workings toward some goal; indications of this were indeed seen in the simulations of [Axe06]. These potential disadvantages of using all-to-all connections are, however, considered to be of considerably less importance than the disadvantages of possibly excluding connections that are essential to the goal at hand.

[22] According to [Hok06] intra-cluster synapses are always Hume synapses. As a result of discussions at supervisor meetings, however, the use of Pavlov type intra-cluster synaptic connections is also included as an option for the simulations herein. For instance, when considering an intraconnected motor cluster, one can imagine that the ability for some motor neurons to anticipate their drive based on the activity in other motor neurons could be useful in developing efficient motion behavior. When Pavlov synapses are used to intra-connect motor clusters, however, the same learning rate as the one used for Hume synapses is used (Pavlov learning rates are typically an order of magnitude higher than Hume learning rates, and are hence inappropriately high for intra-cluster learning). For completeness, recall from Section 2.5 that Skinner synapses don't seem to be suitable for multi-layer learning; they are thus not included as an option for intra-cluster synaptic connections.

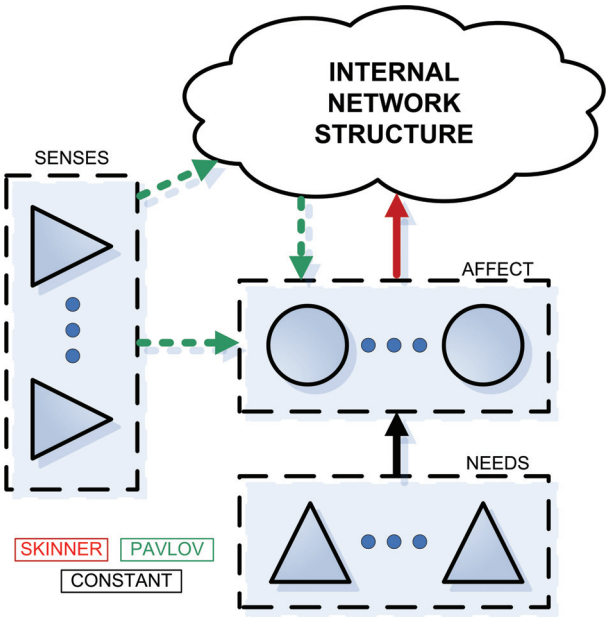


Figure 10: Affect system topology overview

activation function (Eq. (2)). Inverting the activation function gives:

$$x = g^{-1}(y) = \ln\left(\frac{1-x_0}{x_0}\right) - \ln\left(\frac{1-y}{y}\right), \tag{25}$$

where x is the synaptic input value, y is the need value and x_0 is the y-axis crossing point for the sigmoid function. The x calculated from Eq. (25) is used in place of the corresponding $e^t_{ij}D^t_i$ element of the net input calculations (Eq. (3)). If the affect neuron has only incoming connection - the constant connection from the need input - the affect neuron drive value will, of course, be the exact same as the need value. For equation (25) to be valid, however, we must require $y \in (0, 1)$. If y occasionally is below or above this range, the drive value contribution from the need input is set to 0 or 1, respectively.

An alternative approach to the above affect neuron drive calculations, which also was the original approach for this thesis, is to consider constant connections as synapses with constant efficacies, thus allowing net synaptic input and drive values to be calculated in a normal fashion. The efficacies would be set such that, when not considering drive contributions from other neurons, need values map roughly equally onto drive values at corresponding affect cluster neurons. An optimal efficacy value could e.g. be calculated based on the sigmoid y-axis zero point x_0 from Eq. (2); for our simulations, this value would be approximately 8.0. The problem with this approach, however, is the nonlinearity of the sigmoid function, making the effective learning rate for Skinner synapses highly dependent on the current need value range; because need values are transformed through the activation function, learning would be very slow when needs are close to 1.[23] Simulations have, indeed, shown that the latter approach can make early learning difficult, because, initially, need values are high (often > 0.9). The reason for the latter

[23]Recall that efficacy changes in the Skinner learning mechanism (Eq. (10)) are proportional to presynaptic drive changes ΔD_i.

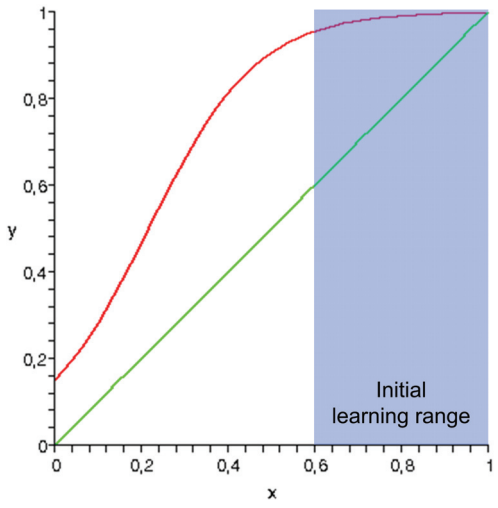


Figure 11: Comparison of sigmoid and linear need transforms, scaled to range within $[0, 1]$

becomes evident by studying Figure 11, where a comparison of the sigmoid and linear need transforms is shown.[24]

As suggested by the topology depicted in Figure 10, affect systems allow need signals (represented as neuronal drives in the affect cluster) to be anticipated by Pavlovian synaptic learning based on input signals from neutralized senses. More, affect systems are not expected to restrict the capabilities of the ANNs in any way; when using the above inversion based mapping of need values to affect neuron drives, no disadvantages have been seen nor are expected to be seen from the inclusion of affect systems. In fact, by appropriate connectivity specifications, topologies incorporating an affect cluster can be made completely equivalent to similar topologies excluding the affect cluster. Thus, in the search for optimal ANN topologies (discussed next), affect systems can be bypassed. Based on discussions at supervisor meetings, affect systems are used in connection with need inputs for all simulations performed throughout this thesis.

3.2.6.2 Topology specification

Connectology depicts how neural network topology can be specified using a cluster connection matrix called the *Cluster Diamond Matrix*. The matrix based specification is adopted for the topology specifications herein; ANN topology is defined at the clustral level by means of a cluster connection matrix $\mathbf{M}$ with elements m_{ij} where the row index i represents the presynaptic cluster and the column index j represents the postsynaptic cluster. Matrix elements m_{ij} are given values based on the connection status between clusters i and j, of which there are several possibilities:

[24] The effective learning rate will be proportional to the derivative of these transforms, given by the current need value range.

	N	S	A	I1	I2	$\cdots$	In	M
N	X	X	C	X	X	$\cdots$	X	X
S	X	X	(P)	(P)	(P)	$\cdots$	(P)	(P)
A	X	X	X	(S)	(S)	$\cdots$	(S)	(S)
I1	X	X	(P)	(H)	(P)	$\cdots$	(P)	(P)
I2	X	X	(P)	(P)	(H)	$\cdots$	(P)	(P)
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
In	X	X	(P)	(P)	(P)	$\cdots$	(H)	(P)
M	X	X	(P)	(P)	(P)	$\cdots$	(P)	(P/H)

Table 1: Generalized ANN topology specification

Entry	Description
X	Connection not possible (blocked) (-1)
0	Not connected
S	Connected by Skinner synapses (1)
P	Connected by Pavlov synapses (2)
H	Connected by Hume synapses (3)
C	Connected by Constant connections (4)

The topology of the ANN, including the type and learning mechanism of all connections, is thus completely specified by the integral matrix $\mathbf{M}$ (integral values shown in parentheses). For instance, the entry $m_{1,3} = P$ specifies that every neuron of the cluster represented by index 1 connects onto every neuron of the cluster represented by index 3 by means of Pavlov synapses. Similarly, the entry $m_{3,3} = H$ specifies that the cluster represented by index 3 is intraconnected by means of Hume synapses, i.e. every neuron of cluster 3 connects onto every other neuron of the same cluster by means of Hume synapses.

Table 1 shows a generalized matrix of which the set of specific instances covers the entire space of allowed ANN topologies. Matrix entries in parentheses are optional, and we let E_{ij} define the set of entry options for element m_{ij}.[25] The clusters are indexed by N (need inputs), S (sense inputs), A (affect cluster), Ii (internal cluster i) and M (motor cluster). Two additional requirements for a topology to be valid do not appear from Table 1: Firstly, we require that there exists a path N$\rightarrow \cdots \rightarrow$M from needs to motors and at the same time a path S$\rightarrow \cdots \rightarrow$M from senses to motors. Secondly, we require that each cluster has at least one incoming connection (except N and S) and one outgoing connection (except M).[26] An example of one specific valid ANN topology specification is given in Table 2, and the corresponding structural ANN topology diagram is given in Figure 12. The topology has need inputs, sense inputs, an affect cluster, two internal clusters and a motor cluster.

[25] For instance, m_{ij} = (P) means that the element may be P or 0, i.e. $E_{ij} = \{P, 0\}$. Similarly, m_{ij} = (P/H) means that the element may be P, H or 0, i.e. $E_{ij} = \{P, H, 0\}$

[26] In the context of genetic algorithms (Section 3.4), this validity checking is merely a matter of running time/performance. ANN topologies that are invalid according to the two requirements stated above are considered meaningless, and are not expected to produce any good or interesting results. By refraining from performing simulations on ANNs incorporating topologies that are known to be incapable of producing good results, instead of letting the GA process prune such topologies, considerable amounts of running time can be spared.

	N	S	A	I1	I2	M
N	X	X	C	X	X	X
S	X	X	P	P	0	0
A	X	X	X	0	S	S
I1	X	X	0	H	P	P
I2	X	X	0	P	H	P
M	X	X	0	P	P	0

Table 2: Example ANN topology specification

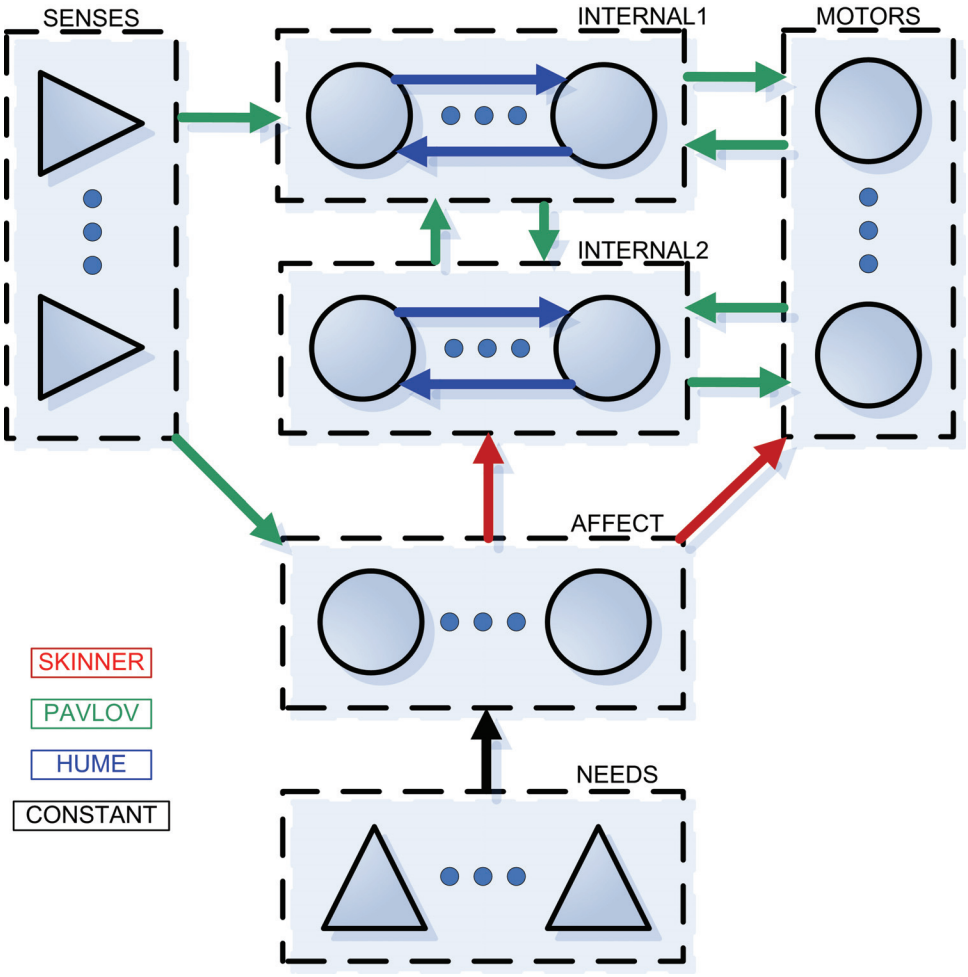


Figure 12: Example ANN topology structure

3.2.7 Simulating the neural network

As described earlier, the simulations are based on discrete time steps. At one time step, the drive value for each neuron j is calculated as a function of weighted input net_j, neuronal trace acc_j, and stochastic perturbation $stoc_j$. A change of neuronal drive gives rise to changes in the synapses adjacent to neuron j, including traces of postsynaptic drive differentials T_{ij}, and synaptic efficacies e_{ij}.

Synchronous vs. asynchronous updates Neural networks are inherently highly parallel, while most computers are purely sequential. Hence, when doing computer simulations of ANNs, parallelism (and continuous time) must be simulated, and the sequence of operations must be considered. With a synchronous update strategy, updates are equivalent to the entire vector of ANN values (neuronal drives and synaptic efficacies) being updated simultaneously. More specifically, when updating ANN values for one time step, the values of elements that constitute parts of the calculations are always taken from the previous times step, and newly calculated values are not put to use until all values are ready and the next time step can be commenced. With an asynchronous update strategy, however, when an ANN value is updated, the newest available values are always used, and the network-level notion of a time-step thus becomes vague. Further, using a deterministic sequential order of neuron updates throughout the simulation could influence simulation results, while, ideally, the update order should have no influence whatsoever on the simulation results. Therefore, in order to simulate parallelism in the best possible way using asynchronous updates, a random update strategy should be used, implying that a new neuron update order is generated randomly for each iteration.

There are both advantages and disadvantages with both synchronous and asynchronous update strategies. For the simulations in this thesis, synchronous updates are used. This choice of update strategy is mainly based on the findings in [Axe06]; there, an asynchronous update strategy was chosen, but both asynchronous and synchronous strategies were investigated. Through the experimental work performed therein, it became clear that for ANNs with synaptic learning of the type considered, where the temporal order of events (drive changes) is critical, asynchronous updates pose a few challenges that become considerable for larger networks; semantics are easily cluttered when updates are asynchronous, in that the aspect of a time step becomes vague. When neurons in a large network are updated asynchronously and in random order, it becomes difficult to analyze the course and correctness of events inducing synaptic learning. A synchronous update strategy is far cleaner in this respect, with well defined temporal properties through step (i.e. iteration) based time discretization.

Algorithm 4 gives an algorithmic overview of how an ANN iteration, i.e. one complete update of all neurons and synapses, is carried out.

3.2.7.1 Run time analysis*

The run time of an ANN iteration is dependent on two important quantities: The number of neurons $|V|$ and the number of synapses $|E|$. The loop of lines 1-3 is bounded by the number of input neurons, which is bounded by $|V|$, and is thus executed $O(|V|)$ times. Line 2 is trivially $O(1)$, and the loop of lines 1-3 is therefore $O(|V|)$. Equivalently, the loop of lines 4-6 is $O(|V|)$.

The loop of lines 7-11 is also bounded by $|V|$ and is thus executed $O(|V|)$ times. Referring to Eq. (3), we see that net_j is calculated as a sum over all incoming synapses to neuron j. Line 8 by itself is therefore $O(|E|)$. Line 9 implies an execution of Algorithm 1 and Algorithm 2,

Algorithm 4 Simulating the ANN (one complete update, i.e. one time step)

1: **for** each need input neuron i **do**
2: set D_i from associated need value
3: **end for**
4: **for** each sense input neuron i **do**
5: set D_i from associated sense value
6: **end for**
7: **for** each neuron j, excluding inputs **do**
8: calculate weighted input net_j (Eq. (3))
9: generate stochastic perturbation $stoc_j$ (Alg. 2)
10: calculate new drive value $D_j^{'}$ (Eq. (2))
11: **end for**
12: **for** each neuron j, excluding inputs **do**
13: let $D_j \leftarrow D_j^{'}$
14: update acc_j (Eq. (4))
15: **end for**
16: **for** each synapse ij **do**
17: update e_{ij} (Eq. (10), (12)/(14) or (16)/(18))
18: update T_{ij} (Eq. (11)/(21), (13)/(15) or (17)/(19))
19: **end for**
20: **for** each neuron j, excluding inputs **do**
21: scale incoming synaptic efficacies e_{ij} (Alg. 3 (MOD2))
22: **end for**

both of which can easily be seen to be $O(1)$, and line 9 is therefore $O(1)$. Line 10 is also trivially $O(1)$. Therefore, lines 8-10 are $O(|E|)$. This gives an upper bound for the loop of lines 7-11 of $O(|V|\,|E|)$. This bound is, however, not tight. This can be seen by amortized analysis, as follows: Every synapse is connected to exactly one presynaptic neuron and exactly one postsynaptic neuron. Line 8, where the implicit loop ranges over the incoming synapses of a neuron, will therefore contribute with $O(|E|)$ in total. With that, an asymptotically tight upper bound for the loop of lines 7-11 can be established as $O(|V| + |E|)$.

The loop of lines 12-15 is bounded by $|V|$. Lines 13 and 14 are trivially $O(1)$, and the entire loop is therefore $O(|V|)$.

Moving to the loop of lines 16-19, we see that it iterates each synapse in the ANN and is thus executed exactly $|E|$ times. Line 17 only involves basic arithmetic operations, and is $O(1)$. The running time of line 18, however, is somewhat more intricate: For Skinner synapses using the MOD1 divergence preventing modification the running time is not immediately obvious. As can be seen from Eq. (20), the mathematics involved contain computing the weighted input over all incoming synapses at two different time steps. Normally, these calculations would be $O(|E|)$. The second activation function term is simply the previous drive of the postsynaptic neuron, which has already been calculated. It can therefore be stored, and recalled in $O(1)$ time. Similarly, most of the input to the first activation function in Eq. (20) is already computed; the only modifications that must be made are subtracting the $e_{ij}^t D_i^t$ term from the net_j^t and adding $e_{ij}^{t-1} D_i^{t-1}$. All of this can be done in $O(1)$ by storing previous drive and previous efficacy for all neurons and synapses, respectively. Now the call to the activation function can be made, which

is also $O(1)$, and with that, the entire calculation of $\Delta_i^* D_j^t$ can be done in $O(1)$ time. $\Delta_i^* D_j^t$ is specific to single synapses, so calculating these values is $O(|E|)$ in total. However, by including the $\Delta_i^* D_j^t$ calculations in the loop of lines 7-11, which is already $O(|V| + |E|)$, this causes no increase in the asymptotic running time of the algorithm, and line 18 can be regarded as $O(1)$. From this, it can be seen from Eq. (21) that the calculations of T_{ij} can be done in $O(1)$ time. With that, the body of the loop of lines 16-19 is $O(1)$, and the loop is therefore $O(|E|)$.

The loop of lines 20-22 is bounded by $|V|$. Line 21 is calculated over all incoming synapses to neuron j, and is therefore $O(|E|)$. However, by amortized analysis, realizing that each synapse will be visited only once, the entire loop only contributes with $O(|E|)$ in total.

Based on the above arguments, the total running time for one complete ANN update can be established as $O(|V| + |E|)$. Note that any ANN update algorithm must visit every neuron and every synapse in the network at least once (for drive and efficacy updates, respectively), such that the running time must be $\Omega(|V| + |E|)$. With that, Algorithm 4 is $\Theta(|V| + |E|)$.

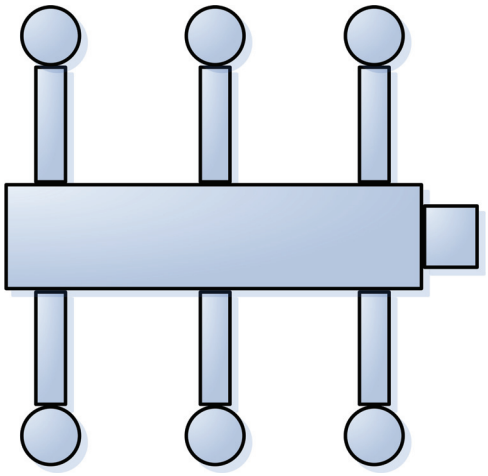


Figure 13: Structural overview of mechanical model

3.3 Mechanical model

The mechanical model represents the body of the virtual creature and therein the environment that the ANN operates in. The structural design of the creature's body is based on generic structural properties found in biological insects, examples including ants and spiders. Such biological creatures are, however, way too complex for (close to) exact mechanical modeling, and several simplifications have therefore been made in the design of the mechanical model. The resulting initial simplified insect-like mechanical structure has further been adjusted based on the recommendations received at meetings with scientific experts Gertjan Ettema and Beatrix Vereijken [EV07].

The mechanical model is based on rigid-body dynamics. The use of rigid-body dynamics imply that the shape and size of each component in the mechanical system is unchangeable and strictly constant; each limb of the simulated virtual creature is rigid and by all means non-deformable. This, of course, is strictly not optimal with respect to biological plausibility and physical realism, but it significantly simplifies the mechanical model and the mathematics and numerical calculations needed to obtain physically realistic system behavior. The choice of using rigid-body dynamics is also heavily influenced by the functionality offered by the range of commercially and freely available physics engines; most of these simulate rigid-body dynamics and have limited or no support for non-rigid bodies.

Open Dynamics Engine (ODE) is the physics engine of choice for this thesis. ODE is purely rigid-body based, and is well suited for simulating virtual creatures of the type investigated here. The reasons for choosing ODE are numerous; it is quite widely used, richly featured, seemingly very stable, mature, and platform independent with an easy to use C/C++ API. Furthermore, ODE is open source, allowing for modifications and extensions, as needed.[27] The use of ODE for the simulations of this thesis is detailed in Appendix B.

[27] Another advantage of using an open source library such as ODE compared to other free but closed source libraries is the lower probability for commercialization in the near future.

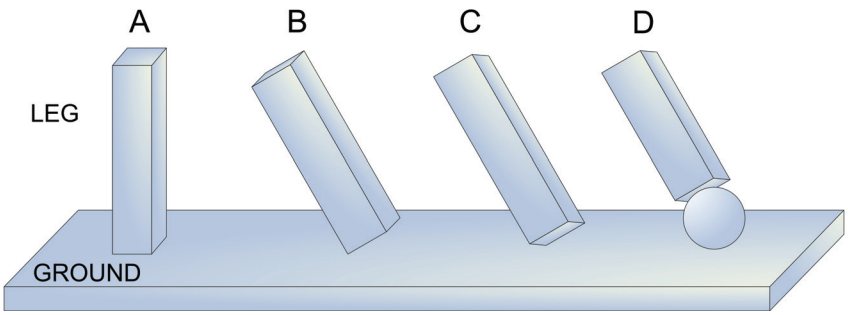


Figure 14: Leg ground contact: A) square contact surface B) line contact surface C) single point contact surface D) orientation independent single point contact surface.

3.3.1 Limbs and joints

The mechanical model is built using two simple primitives: cuboids (boxes) and spheres (balls). A virtual creature consists of the following parts:

- Head: 1 cube
- Torso: 1 cuboid
- Legs: 1 cuboid (thigh) and 1 sphere (foot) per leg

Figure 13 gives a structural overview of a virtual creature with six legs. The number of legs is variable, and is specified by the number of leg pairs L. The length of the torso grows linearly with L, implying that a creature's torso approaches the shape of a stick as the number of leg pairs grows. All body parts have equal and uniform mass density.

The use of spheres to model the feet of the creature is a compromise solution introduced in order to overcome the greatest disadvantages of using as simple body part primitives as cuboids: Purely box shaped legs makes the contact surface between a creature's leg and the ground heavily dependent on the exact orientation of the leg, see Figure 14. With sphere shaped feet the contact surface between a leg and the ground becomes independent of leg orientation, which is closer to the relative orientation independence obtained in diverse biological systems having flexible and non-rigid feet.

Joints are used to connect different body parts, and to constrain their movement relative to each other. The mechanical model used herein implements two different joint types: fixed and universal.

Fixed joints maintain a fixed relative position and orientation between the two parts they connect, making these physically equivalent to one composite body part. Relative position and orientation is specified at model initialization, and these two properties thus remain constant throughout an entire simulation. The neck-joint between the torso and the head and the ankle joints between the thighs and the feet are of fixed type, implying that these connected body parts will never move or rotate relative to each other.

The other joint type used is the universal joint, as depicted in Figure 15. Universal joints have two degrees of rotational freedom: rotation about a horizontal axis (Axis 1) and about a vertical axis (Axis 2). The third degree of rotational freedom is constrained such that the rotation of the two body parts about the direction perpendicular to the two axes will be equal; the effect of this is that if you grab one body part and rotate it, the other will twist as well.

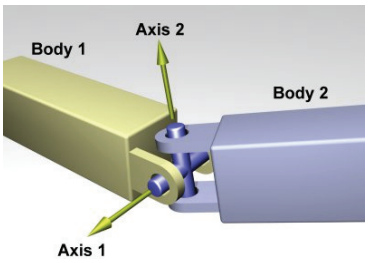


Figure 15: Universal joint (image courtesy of ODE Manual/Russel Smith)

The crossing point between the two rotational axes is the anchor point which constrains the relative position between the two body parts; the two body parts are hinged together at the anchor point. The hip-joints between the torso and the thighs (i.e. legs) are of universal type, implying that legs can be raised/sunken and moved forth/back, but not rotated about their own axis. Further, the anchor point ensures that the legs are always tightly connected to the torso.

Figure 16 provides an overview of the joint type structure of the mechanical model, showing joint positioning and type specification.

For simulation purposes, joint angles for universal (i.e. hip) joints must be constrained to some minimum and maximum values. The model used herein utilizes the approach of specifying joint angle range by two parameters for each dimension: base (equilibrium) angle and maximum angle of deflection, as depicted in Figure 17. These parameters are global and apply equally to all legs, giving rise to a total of four joint angle parameters (two for each dimension) for the entire mechanical systems.

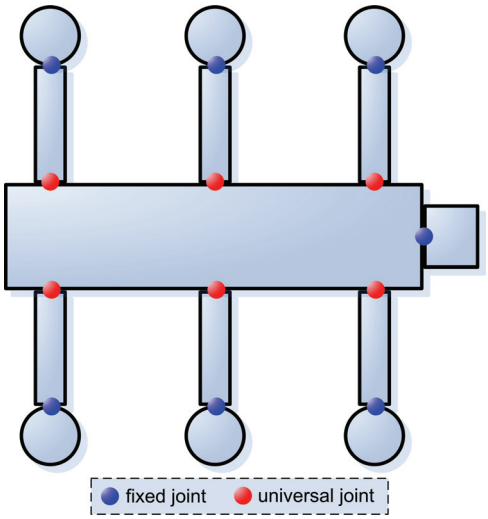


Figure 16: Joint type overview

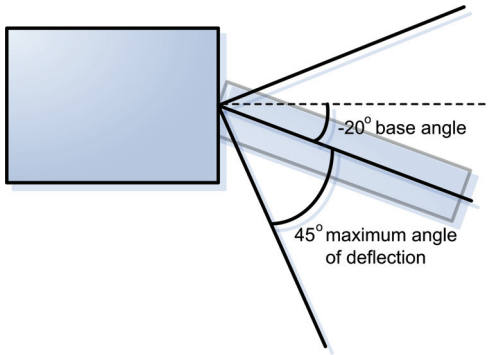


Figure 17: Joint angle specification: base and deflection

3.3.2 Motors and muscles

As discussed in Section 3.1, and more thoroughly in Section 3.2.4, the ANN controls the mechanical model by means of motor activation signals, which are simply neuronal drive values at motor output neurons. Motor activation signals, in turn, control mechanical model *muscles*. Muscles act upon limbs by generating torques at non-fixed joints. As specified in the previous section, the only applicable joints are the universal hip-joints connecting the thighs (i.e. legs) to the torso. These universal joints have two rotational degrees of freedom which translate into moving the legs up/down and forth/back.

From this, each leg must be controlled by four muscles: two vertical for moving the leg up and down, and two horizontal for moving the leg forth and back, as depicted in Figure 18.[28] Of considerable importance at this point is that the muscles of the mechanical model are purely logical constructs; they do not constitute any concrete part of the physically simulated creature, they are simply mathematical functors through which instantaneous joint torque is calculated based on motor activation history, maximum muscle force etc. (details discussed below). In-

[28]The two respective muscle pairs correspond closely to what is termed antagonistic muscle pairs in biological systems; two muscles working over the same joint and pulling in opposite directions [EV07].

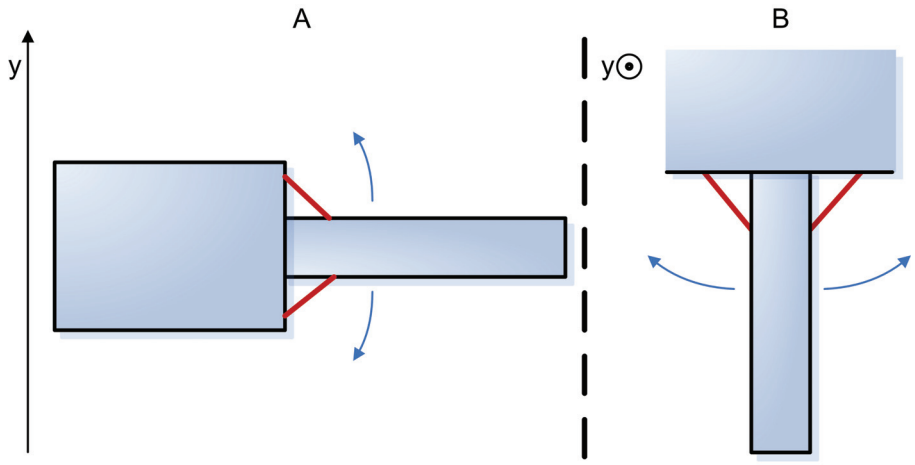


Figure 18: Creature muscles: A) Vertical moving leg up/down B) Horizontal moving leg forth/back.

evitably, this also implies that our muscle concept has no spatial attributes such as base length or limb anchoring points; the spatial properties depicted in Figure 18 have no purpose other than specifying in what direction the different muscles pull.

Some comments are appropriate as regards the adequacy of this approach; a more biologically realistic model would have incorporated spatially dependent and physically simulated muscles that stretch and contract and thereby interact mechanically with the limbs of the creature's body by pulling jointed limbs together. Our model, as suggested earlier, makes one protrudent simplification in that an entire step in the sequence of actions constituting neurally controlled mechanical motion in biological systems is skipped: the logical muscles used here give joint torques directly instead of indirectly by pulling body parts together. This omission of spatially dependent muscles is equivalent with joint torque as a function of muscle force being represented by the identity function for all muscles.

However, the entire mechanical model, as it is described in the preceding, is already a crude simplification compared to the real biological systems it is meant to describe. Great simplifications, such as that of having very few geometrically simple and pure rigid body parts with few non-fixed joints, are necessary to make manageable the amount of work required to stably, consistently and realistically simulate the mechanical system. With this in mind, the muscle model simplification mentioned above is presumably not too restrictive.

3.3.2.1 Muscle model overview

The muscle model adapted herein is heavily inspired by the functioning of muscles in biological systems, as described in [Kan00]. Muscle force at any time is composed of two types of force with different basis:

- **Active muscle force** dependent on motor activation signals (i.e. neurally controlled)
- **Passive muscle force** not dependent on motor activation signals (i.e. purely mechanically originate)

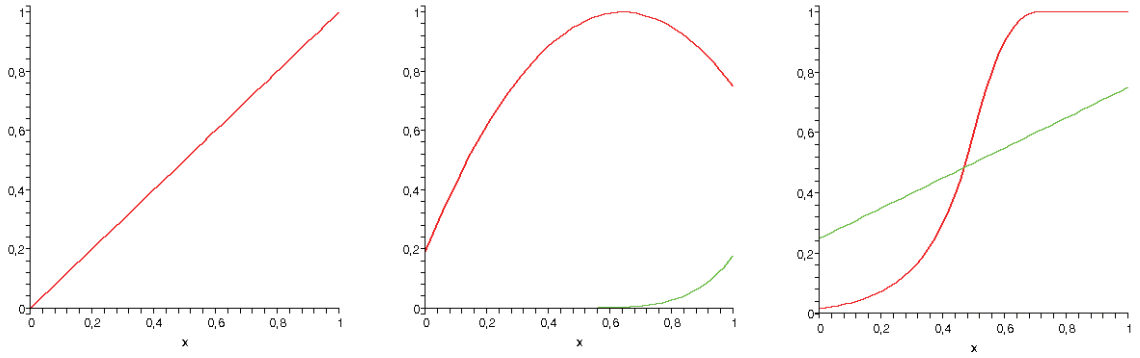
Active muscle force, controlled by neuronal activation, is dependent on three variables:[29]

- Level of neuronal motor activation
- Muscle length
- Muscle length rate

Passive muscle force depends only on the latter two of these; muscle length and muscle length rate. For the mechanical model described above, where muscles are purely logical constructs without mechanical or spatial properties, muscle length and muscle length rate are approximated by joint angle θ and joint angular velocity $\dot{\theta}$, respectively.

Active muscle force is described mathematically by the relative contributions induced by neuronal activation, muscle length and muscle length rate as $f_{a1}(D)$, $f_{a2}(\theta)$ and $f_{a3}(\dot{\theta})$, respectively. Each muscle is associated with one motor neuron giving neuronal activation level D. Correspondingly, passive muscle force is described by the relative contributions induced by muscle length and muscle length rate as $f_{p2}(\theta)$ and $f_{p3}(\dot{\theta})$, respectively.

[29] [Kan00]: "Contractile force depends on the level of activation of each muscle fiber and its length and velocity." (p. 680)



(a) X-axis: neuronal activation **(b)** X-axis: muscle length **(c)** X-axis: muscle length rate

Figure 19: Muscle force dependency functions (active, passive)

The shapes of these five relative dependency functions are adopted from the description of biological muscles given in [Kan00]. All five (three active and two passive) are plotted in Figure 19 (the mathematical specifications are given in the next section), normalized for input range $[0, 1]$. For f_{a2} and f_{p2}, this implies that minimum length equals 0 and maximum length equals 1. For f_{a3} and f_{p3}, muscle shortening is represented by values less than 0.5 and muscle lengthening is represented by values greater than 0.5.[30]

Active and passive muscle forces are additive. Furthermore, the functions describing the different dependencies within the active or passive type are multiplicative [Kan00]. Thus, if we denote by F_{max} the maximum muscle force for a specific muscle, current force contribution is given by:

$$F^{'} = f(D, \theta, \dot{\theta}) = F_{max} \left(f_{a1}(D) \cdot f_{a2}(\theta) \cdot f_{a3}(\dot{\theta}) + f_{p2}(\theta) \cdot f_{p3}(\dot{\theta}) \right) \tag{26}$$

Further, to take time dependence into account, effective muscle force should also be dependent on previous force history. A very simple and yet quite effective approach is the use of a simple mathematical trace function:

$$F = (1 - \alpha_m)F + \alpha_m F^{'}, \tag{27}$$

where $\alpha_m \in (0, 1)$. Such a trace will introduce inertia to the force generated at the muscle based on previous force history; the length of the trace and thus the impact of the inertia is controlled by the α_m parameter. An immediate consequence of the introduction of a muscular trace is that, with an appropriately sized α_m, very rapid muscle force fluctuations are eliminated, thus heavily suppressing the possibility for fast-paced vibratory legs in the mechanical model. In the words of Chiel and Beer:

> *Muscle acts as a low pass filter of motor neuronal outputs, that is, it filters out the high frequency components of the neural outputs. ([CB97] p. 553)*

[30] In addition, since muscle length rates have no real limits on minimum and maximum values, the source range is restricted to $[-6, 6]$ rad/s, which by trial and inspection is seen to cover the range of interest. Occasional values outside this range are truncated.

3.3.2.2 Muscle model details

Biological muscles have several types of muscle fibers, and the functioning of an entire biological muscle is the result of the force generated at several different muscle fibers [Kan00]. The muscle model used herein, however, is based on only one type of muscle fiber, thus making the qualitative functioning of an entire muscle equivalent to that of a single fiber. An initial guess is that the muscle fiber modeled should be most resemblant to *type I slow-twitch* muscle fibers.[31] Anyhow, variable parameters such as the α in Eq. (27) introduces flexibility in allowing for the properties of the muscle fiber to be changed easily; for instance, a higher α gives a faster (as in more quickly responding) muscle fiber.

Active dependence on neuronal activation is given by:

$$f_{a1}(D) = D \tag{28}$$

The function is shown in Figure 19(a). Active muscle force thus increases linearly with neuronal activation.[32]

Active dependence on muscle length is given by:

$$f_{a2}(\theta) = -(1.4\theta - 0.9)^2 + 1 \tag{29}$$

The function is plotted in red in Figure 19(b). The shape indicates that the potential for active muscle force is greatest when the muscle is of medium length; active muscle force declines both when the muscle is longer (i.e. stretched) and when the muscle is shorter (i.e. compressed), indicating that the optimal muscular area of operation is around the middle of the joint angular deflection range.

Active dependence on muscle length rate is given by:

$$f_{a3}(\dot{\theta}) = \begin{cases} \exp(\dot{\theta} - 0.572975)^7, & \text{if} \quad \dot{\theta} < 0.5 \\ -10(\dot{\theta} - 0.7)^2 + 1, & \text{if} \quad 0.5 \leq \dot{\theta} < 0.7 \\ 1, & \text{if} \quad \dot{\theta} \geq 0.7 \end{cases} \tag{30}$$

The function is plotted in red in Figure 19(c). Its shape indicates that the potential for active muscle force is considerably greater when the muscle is being stretched than when it is being compressed. The functional consequence of this is that muscles can be said to possess implicit movement dampening characteristics; they are more probable of counteracting their current direction of movement than reinforcing it.[33] The S-shape of f_{a3} further intensifies this

[31] [Kan00]: "The red muscles of the legs are specialized for standing and walking[...] Red muscles are composed mostly of slow-twitch fibers, also called type I fibers. The force produced by type I fibers rises and falls relatively slowly in response to an action potential." (p. 684)

[32] The function is probably not biologically exact; the relationship between neuronal activation and muscle force is assumed to be nonlinear. Based on expert recommendations received at discussion meetings [EV07], however, indicating that the importance of the exact shape of f_{a1} is limited, the simple linear model of activation dependence given in Eq. (28) is used.

[33] The force generated by a muscle always contributes to shortening the muscle. When a muscle is being lengthened, the relative force is large, making the muscle highly capable of stopping the movement causing muscle lengthening, while when a muscle is shortened, the relative force is small, making the muscle less capable of reinforcing the movement. Now considering a joint with a corresponding antagonistic muscle pair. When there is limb movement over this joint, then, the muscle lengthening in one muscle is easily counteracted, while the muscle shortening in the other muscle is not as easily reinforced. Hence, the consequence of active muscle length rate dependence as given by Eq. (30) is dampening of movement.

tendency, increasing the relative difference between stretching and compression compared to a linear model.

Passive dependence on muscle length is given by:

$$f_{p2}(\theta) = (\theta - 0.25)^6 \tag{31}$$

The function is plotted in green in Figure 19(b). The shape of the function shows that muscles generate passive force when extensively stretched. The functional consequence is that muscles automatically counteract extensive stretching, making antagonistic muscle pairs probable of passively forcing joints away from of the outermost extreme parts of their range. Immediately evident, also, is that the absolute size of passive muscle force is considerably smaller than that of active muscle force. The latter should also be expected; the forces that biological muscles are capable of generating actively are of greater magnitude than the corresponding passive muscle forces [EV07].

Passive dependence on muscle length rate is given by:

$$f_{p3}(\dot{\theta}) = 0.5\dot{\theta} + 0.25 \tag{32}$$

The function is plotted in green in Figure 19(c). The functional consequence is similar to that of the corresponding active dependence on muscle length rate; f_{p3} contributes to dampening passive movement.

All these equations are based on rough approximations of corresponding functions plotted in [Kan00], which are approximations of what has been found empirically in biological muscles, and should therefore be of high relevance with respect to biological plausibility. The functions have further been adjusted to better fit the recommendations given by scientific experts Gertjan Ettema and Beatrix Vereijken [EV07].[34]

3.3.3 Needs

As discussed in Section 3.1, and further in Section 3.2.4, the mechanical model provides the ANN with need signals. These needs, which constitute the goal specification for the entire simulated system, are subjected to minimization by the ANN. The set of need values chosen to be fed back from the mechanical model to the ANN is therefore of great importance.

Many different types of needs are conceivable, and a sensible non-empty set must be chosen among these to reflect the wanted system behavior. The following sections discuss different needs examined throughout this thesis.

[34] Also important in adjusting the model were preliminary simulations that made evident the need to decrease the magnitude of passive forces in the Kandel [Kan00] model compared to active forces: For specific settings of mechanical model parameters, the original model allowed repetitive movement patterns to emerge based solely on passive muscle forces.

3.3.3.1 Torso height need

The torso height need is based on inverse torso height above ground, and the need thus translates into the goal of standing up. Several configurations are possible giving rise to one or several need values; different points on the torso could for instance represent different needs, possibly with unequal weights. For the purpose of getting up on all feet, however, a multiplicative weighting of the four upper corners of the torso cuboid giving one single need is most suitable, because such a configuration yields a minimized need value when the entire torso is lifted as high up as possible while simultaneously being stabilized horizontally.

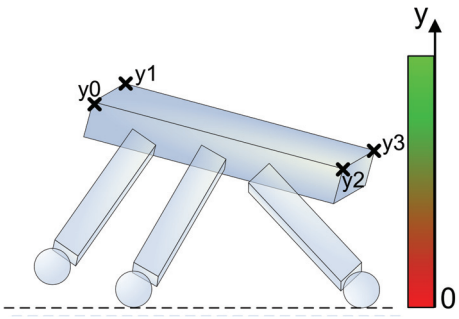


Figure 20: Multiplicative weighting of upper torso corner points.

Mathematically, this need can be implemented as:

$$need = 1 - y_0 \cdot y_1 \cdot y_2 \cdot y_3, \tag{33}$$

where y_0, y_1, y_2 and y_3 are the current heights of the four upper corners of the torso, as depicted in Figure 20, and scaled to range within $[0, 1]$.

3.3.3.2 Head height need

The head height need is based on inverse head height above ground, and is thus similar to the torso height need discussed above. As opposed to the latter, however, the head height need does not incorporate multiplicative averaging of several point heights favoring horizontally and vertically balanced head orientation. The mathematics are straightforward:

$$need = 1 - y, \tag{34}$$

where y is the height component of the head's current position, scaled to range within $[0, 1]$.

3.3.3.3 Belly pain sensor needs

The belly pain sensor needs are based on monitoring the position of the four lower corners of the torso cuboid, and seeing if they touch (are very close to) the ground, see Figure 21. Thus, these four needs translate into the goal of getting up or, equivalently, not lying on the ground. If one corner touches (is very close to) the ground, this almost instantaneously gives a very high need value. Thus, the semantics of the belly pain sensor needs are very similar to those of the torso height need described above, except that the height interval of interest is very much shorter. As for the corresponding goals, the torso height need requests getting the torso as high as possible, whereas the belly pain sensor needs merely request getting the torso above (as in not touching) the ground. The mathematics are equivalent to those presented above for the head height need.

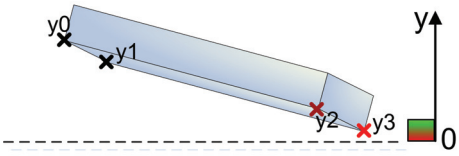


Figure 21: Four belly pain sensors (3 is painful, 2 is close, 1 and 0 are safe).

3.3.3.4 Velocity need

The velocity need is based on inverse torso speed in creature-relative forward direction. It translates into the goal of achieving as high forward velocity as possible or, equivalently, moving forward (crawling, walking etc.) at maximum speed; the higher the forward velocity, the lower the need.

The velocity need is based on a mathematical trace function, such that the value of the need is affected by previous need history. The mathematics are as follows:

$$need^t = (1 - \alpha_v) need^{t-1} + \alpha_v \frac{\max(A - \nu^t, 0)}{A}, \qquad (35)$$

where $\alpha_v \in (0, 1]$ controls the shape (i.e. length vs. responsiveness) of the trace, ν is the velocity in creature relative forward direction (Figure 22) and A represents some maximum and assumed unattainable velocity. For the simulations herein, $A = 6.0m/s$.

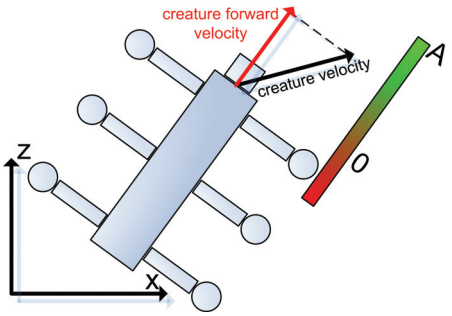


Figure 22: Creature relative forward velocity.

Considering the trace parameter α_v, note that a large α_v makes the need value responsive to rapid changes of velocity,[35] while a small α_v makes the need represent the general tendency of motion (is the creature doing the right thing?) in a better way.[36] Ideally, therefore, to utilize both these advantages, α_v should be both small and large, which is impossible. Initial assessments indicate that the advantages of having a large α_v overshadow those of a small α_v; the ability to quickly respond to and learn from rapid changes of velocity is assumed to be crucial in learning successful motion behavior. Thus, for the simulations herein, $\alpha_v = 1$.

3.3.3.5 Extreme joint angle needs

The extreme joint angle needs are based on monitoring the joint angles at the creature's hip-joints, and inducing pain (i.e. high need values) when the angles are at or close to their extremes. The biological foundation for these needs is that muscles and tendons become heavily stretched at extreme joint angles, or joints may be in awkward positions, and this may cause some degree of pain. Extreme angle needs do not map as directly as the others onto a specific goal, but they are connected to the above goal of moving forward through the hypothesis that such pain signals at joint extrema may be necessary for the creature to move its limbs in a pattern so as to develop walking-resemblant gaits. For each leg there are two rotational axes with minimum and maximum angles; thus, extreme joint angle needs give rise to four needs per leg.

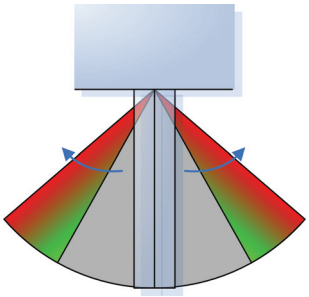
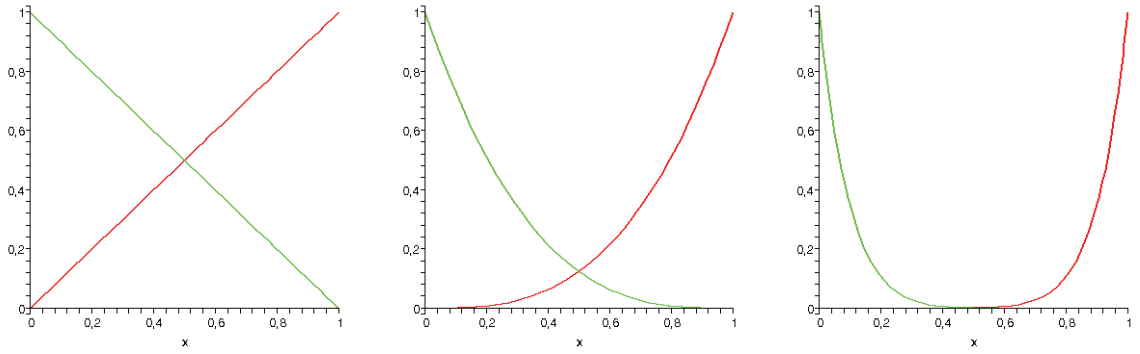


Figure 23: Extreme angle sensors.

[35] For large values of α_v, a rapid and perhaps short-lived positive change of forward directed movement will affect the need almost instantaneously, allowing the system to learn what motor behavior caused the reduction in need value.

[36] For relatively small values of α_v, seemingly random jiggling forth and back will cancel out and hence not induce any learning. This is, indeed, propitious because such behavior is unwanted and should not make the basis for synaptic learning.



(a) $a = 1$ **(b)** $a = 3$ **(c)** $a = 10$

Figure 24: Extreme joint angle need functions x^a and $(1-x)^a$.

Extreme joint angle needs are calculated from the following simple mathematical functions:

$$n_1(x) = x^a \tag{36}$$
$$n_2(x) = (1-x)^a, \tag{37}$$

where a is a variable parameter. The given equation pair represents the two needs associated with a leg's movement in either horizontal or vertical direction. Joint angles used as input x to these functions are normalized to range within the interval $[0, 1]$. The functions are plotted for three different values of a in Figure 24.

3.3.3.6 Muscle force needs

The muscle force needs are based on feeding back as pain signals the muscle force generated at each muscle relative to the corresponding maximum force, thus giving rise to one need per muscle or, equivalently, four needs per leg. Biologically, this need is related to the energy optimization found in many biological systems [EV07], and these needs therefore translate into the goal of behaving or moving in a manner which is optimal with respect to energy efficiency. In connection with this, however, it is of severe importance to be aware of the potential for need conflicts with the inclusion of muscle force needs; there is for instance an inherent conflict between the velocity need and the muscle force needs, in that achieving and maintaining forward velocity requires muscle activity. For this reason, muscle force needs should be used with great care, as they may contribute to suppressing otherwise promising motion behavior.[37]

Since the muscle force need values simply are scaled versions of effective muscle forces, as presented in Section 3.3.2, the mathematics are not repeated here. Important to note, however, is that need values are normalized with respect to the maximum muscle forces of particular muscles; stronger muscles do not in general induce higher need values than weaker muscles.

[37] Previous but simpler ANN simulations [Axe06] have showed that the inclusion of force based needs similar to those discussed here may lead to more energy efficient and still successful ANN behavior.

3.3.3.7 Foot friction needs

The foot friction needs are based on inducing pain signals when the creature's feet slide along the ground; the physical or biological analogy being the potentially painful heat production or tissue pressure caused by sliding friction. These needs do not translate into a specific end goal, but are thought to make the creature more probable of lifting its feet above ground instead of sliding them along the ground when moving. The latter is assumed to be favorable in connection with the above velocity need, in that movement patterns based on lifting and setting the feet down should be more efficient than those based on dragging the feet along the ground.

The semantics of the foot friction needs are straightforward: If a foot is on the ground,[38] and also was on the ground at the previous time step, calculate foot friction need value as follows:

$$need = \sqrt{x_m^2 + z_m^2}, \tag{38}$$

where x_m and z_m represent the distance between the previous position and the current position in the x and z direction, respectively. Obviously, from this definition, if the distance traveled between two consecutive time steps is zero, the foot friction need value is also zero, which is equal to the foot being above ground.

3.3.4 Neutralized senses

As discussed in Section 3.1, and further in Section 3.2.4, in addition to need signals, the mechanical model provides the ANN with neutralized sense signals. As opposed to needs, senses are not subjected to minimization by the ANN. Senses are anyhow assumed important in providing neutralized signals describing the current mechanical (bodily) and environmental state on which the ANN can steer behavior.

Several different types of senses are possible. Kandel et al. [Kan00] and Hokland [Hok06] discuss senses originating from muscle *spindle endings* that are assumed important in the development of motion behavior in biological systems:[39] Most importantly, the group Ia spindle endings are sensitive to muscle length, whereas the group II spindle endings are sensitive to muscle velocity. The senses included in the simulations of this thesis are therefore described by muscle length and muscle length rate, respectively.

3.3.4.1 Muscle length senses

Muscle length is, as mentioned earlier, approximated by joint angle. Hence, in the mechanical model of this thesis, the muscle length senses are given by a linear transformation of joint angles, such that values range within $[0, 1]$.

3.3.4.2 Muscle length rate senses

In correspondence with the above muscle length senses, muscle length rate is approximated by joint angular velocity. Hence, in the mechanical model of this thesis, muscle length rate senses are given by a linear transformation of joint angular velocities, such that values range within $[0, 1]$

[38] Meaning that the hight component of the position vector is lower than some prespecified low value very close to the ground

[39] [Kan00]: "Muscle spindles are small encapsulated sensory receptors that have a spindle-like or fusiform shape and are located within the fleshy part of the muscle." (p. 718)

Both joint angle and joint angular velocity senses are specified using the exact same equations as those used for extreme joint angle needs, i.e. Eqs. (36)-(37) (Figure 24), with possibly different values for the a parameter. Empirical findings do indeed indicate that the transformations described by Eqs. (36)-(37) are linear or close to linear [EV07], while the more provisional extreme joint angle needs seem more reasonable using considerably higher a-values for Eqs. (36)-(37). For muscle length senses and muscle length rate senses, therefore, $a \in [1, 2]$.

3.3.5 Reciprocal scaling of need and sense values

As mentioned earlier, the simulated system will normally implement several of the need and sense types described above. A readily occurring question is then how strong the different types of needs and senses should be compared to each other. This question is not easily answered, but it is obvious that some reciprocal scaling may be necessary. Thus, for each need and sense type included in the model, a corresponding variable scaling parameter is also included to describe its relative intensity.

3.4 Genetic algorithms

The preceding pages have made evident the large number of model parameters that the neuronal and mechanical computer simulations depend on. Setting these parameters manually is extremely time consuming, and, most importantly, it is very hard to discover the optimal set of parameter values with such an approach. Further, the dimensionality of the parameter space is so high that an exhaustive search is far out of reach, even with very coarse discretization of floating-point parameter ranges.

Furthermore, when considering the biological inspiration inherent in the systems simulated, a natural choice is to search the parameter space by means of genetic algorithms (GAs). The GA process is an imitation of the evolutionary processes found in nature, using terms like individual, generation, fitness, mutation and recombination. GAs can be used to find approximate solutions to optimization and search problems, and are commonly used when the problem size leaves non-heuristic approaches such as exhaustive search intractable.

Key aspects of the GA process are as follows:

- **Individuals** are represented by a specific complete setting of GA parameter values.
- A **generation** is a collection of individuals.
- Individuals of a generation are compared by means of **fitness evaluation**. A fitness value is obtained by monitoring the individual's performance throughout it's lifetime according to some prespecified criteria, termed the fitness function.
- The fittest individuals (i.e. those with highest fitness values) are chosen for **recombination**:
  - The offspring's parameter values (genes) are chosen from the mother or father with equal probability.
  - Offspring parameter values may **mutate**, i.e. change stochastically with a given probability.

Algorithmically, the GA semantics are as specified in Algorithm 5. In Line 6 of Algorithm 5,

Algorithm 5 GA

```
1: create the mother of all individuals from an initial (hopefully qualified) guess
2: generate a set of new individuals from the above by mutation
3: {the individuals from lines 1-2 constitute the first generation}
4: for each generation do
5:    evaluate the fitness of each individual
6:    let the fittest individuals survive (live on to the next generation)
7:    generate a set of new individuals from the above by mutation and recombination
8:    {the individuals from lines 6-7 constitute the next generation}
9: end for
```

the fittest individuals are allowed to live on into the next generation, assuring that the genetic combinations which made these individuals proficient are not lost during recombination and

mutation. This principle is termed **elitism**.[40]. It can further be noted that the group of individuals used as source for recombination (the parents) coincides exactly with the elitism group.

3.4.1 GA configuration details

When searching a parameter space using GAs, two things must be defined: 1) a genetic representation of the solution domain, and 2) a fitness function to evaluate the solution domain.

3.4.1.1 Genetic representation

As mentioned above, the genetic representation of an individual in the GA process is a specific complete setting of all GA parameters. There are three single value parameter types in the simulations of this thesis:

- **Integral parameters** take on integral values and are limited by a specified minimum and maximum integral value: $param_i \in \{min_i, \ldots, max_i\} \mid param_i, min_i, max_i \in \mathbb{N}$.
  - Example: number of leg pairs $L \in \{2, 3, 4\}$.
- **Floating-point parameters** take on any real value between a specified minimum and maximum value: $param_f \in [min_f, max_f] \mid param_f, min_f, max_f \in \mathbb{Z}$.
  - Example: synaptic learning rate $\beta_{ij} \in [1, 20]$.
- **Boolean parameters** take on the boolean values $true$ or $false$: $param_b \in \{true, false\}$.
  - Example: use of modified Pavlov synaptic learning mechanism (i.e. use of Eqs. (14)-(15) instead of Eqs. (12)-(13)).

In addition to the three single value parameter types listed above, the genetic representation also incorporates an **ANN topology specification**, which is implemented as a cluster connection matrix where each matrix entry specifies whether some cluster A connects synaptically onto some cluster B, and what type of connection, if any, is to be used.

In summary, a valid specification of an individual in the GA process is a complete mapping of legitimate values to all integral, floating-point and boolean parameters, such that every value is within the prespecified value range for the corresponding parameter, and a complete and valid setting of the cluster connection matrix (for details on ANN topology specifications see Section 3.2.6).

3.4.1.2 Fitness function

The fitness function, which is used to select the best individuals from a genetic simulation, must be tailored to reflect wanted system behavior and is therefore closely connected to the need specification of the simulation at hand.[41] Hence, because the set of needs, and with that

[40] For GA simulations of purely deterministic systems, the use of the elitism principle assures that maximum fitness as a function of generation number is a monotonically increasing function. The simulations herein are not deterministic, but the tendency is anyhow assumed favorable.

[41] Recall from Section 3.3.3 that the set of needs implemented in the mechanical model fully represents the goal specification for the system as a whole.

the definition of wanted system behavior, may vary, it is not possible to define a general fitness function for all simulation cases.

Although no universally suitable fitness function can be specified, the basic structure is general and applicable to all goal specifications. For the simulations herein, a fitness value is updated once every iteration, based on the following structure:

$$fitness^0 = 0 \tag{39}$$

$$fitness^t = fitness^{t-1} + \frac{t}{N} \cdot f(state), \tag{40}$$

where t is the current iteration number, N is the total number of iterations and $f(state)$ is some function of the current state giving instantaneous performance. Fitness is consequently accumulated for each iteration, and each contribution is weighted such that the significance of the creature's behavioral performance with respect to fitness evaluation increases linearly with simulation time passed. As opposed to an unweighted accumulative model, the above structure makes the initial period of trial and error (before any recognizable behavior has emerged) less influential.

When fitness is measured by the torso height above ground, $f(state)$ could simply be the inverse height value, equivalent to the mathematical specification given in Section 3.3.3.1. Similarly, when fitness is measured by the creature forward velocity, $f(state)$ could be the inverse forward velocity value, equivalent to the mathematical specification given in Section 3.3.3.4.

Some quantitative properties of the GA process have not yet been discussed, namely population size, size of the elitism group, mutation probability and mutation amount. Also, the precise semantics of the mutation process have not been specified.

3.4.1.3 Population size and size of elitism group

There exists a multitude of opinions as regards the optimal size of populations when using genetic algorithms to search parameter spaces. By optimal is normally meant the population size that requires the least number of individuals (which translates into minimum running time) before genetic convergence (best available solution has been found to the problem); if running time is not considered, the larger the population size the better [Ala92], but that is of theoretical interest only. In practice, the number of generations needed to obtain genetic convergence is normally inversely proportional to the size of populations. The matter of finding the optimal population size then comes down to finding the crossing point that minimizes the total number of individuals simulated before genetic convergence is obtained.

Much of the work that has been done on the subject seems to be concentrated around GA configurations where the genetic representations are simple bit strings, i.e. strings of 0's and 1's (e.g. [Ala92, GR00]).[42] For such representations, the *problem size* is the number of bits, which is equivalent to the number of parameters in our representation. A clear tendency leveraged by many is that the optimal population size grows sub-linearly with the size of the solution space; [Ala92] states that the optimal population size as a function of the solution space size is logarithmic:

> *It seems that for moderate problem complexity the optimal population size for problems coded as bitstrings is approximately the length of the string in bits[...] (p. 1)*

[42] Relating to the GA configuration described above, this would correspond to all parameters being of boolean type.

In mathematical terms, this can be expressed as (adopted from [Ala92]):

$$\log(N) \leq S_{opt}(N) \leq 2\log(N) \tag{41}$$

where $N = \prod_{i=1}^{n} n_i$ is the size of the solution space. Correspondingly, as a function of the number of parameters n, the optimal population size can be expressed as:

$$n \leq S_{opt}(n) \leq 2n \tag{42}$$

At this point it is important to recall that in addition to the boolean type parameters on which the above sizing model is based, our genetic representation is comprised of integral and floating-point parameters, as well. Thus, the problem size cannot be described combinatorially by the number of possible combinations. The logarithmic (or sub-linear) tendency is, however, still relevant, and the above model, together with other similar findings [GR00], is still regarded as providing important indications on population size for more general parameter spaces.

The GA simulations performed herein search a parameter space with approximately 30 variables.[43] To take the increased complexity of using integral and floating-point parameters in addition to boolean parameters into account, a population size of 50 is used. No analysis of population size optimality has been performed, as the focus of this thesis lies elsewhere. The choice made is, however, supported by the fact that the GA process does indeed seem to find good solutions to the problems presented.[44]

The size of the combined parent and elitism group is another matter that must be considered. For the simulations of this thesis, the parent/elitism rate is 0.2, meaning that the members of the fittest fifth of each generation both are used as sources for recombination and survive onto the next generation.

3.4.1.4 Recombination

Recombination is a process where a pair of parent individuals are combined to create an offspring, such that the offspring's genes typically share many characteristics with the genes of its predecessors.

Single value parameters

For the simulations herein, the semantics for recombination of single value GA parameters are simple. Consider an individual C whose single value parameter set is to be created by recombination of two parent individuals A and B. Then, for each of C's parameters, the parameter value is adopted as the value of the corresponding parameter from either A or B, with equal probability. After recombination is complete, then, C represents a random combination of A and B, where, at average, A and B have had equally large influence on the result.

[43] The number of free parameters varies among experiments according to which needs are set as *optional*, i.e. whose inclusion or exclusion is determined by the GA process.

[44] One can, of course, speculate that another population size would have been better, but it is believed that the disadvantage brought about by the potentially non-optimal population size of 50 is merely a matter of increased convergence time, and not of worsened actual simulation end results.

ANN topology specifications

The semantics for recombination of ANN topology specifications are largely equivalent to the straightforward semantics presented above, only adjusted for matrix representations. Consider an individual C whose topology specification is to be created by recombination of two parent individuals A and B. Then, for each entry m_{ij} of C's cluster connection matrix $\mathbf{M}$, the matrix entry is adopted as the value of the corresponding matrix entry from either A or B, with equal probability. A's and B's cluster connection matrices may, however, be of unequal size. For such cases, the size of C's matrix is chosen as the size of either A's or B's matrix, with equal probability. Further, if the larger of these is chosen as the size of C's matrix, the latter will contain some matrix elements having only one valid parental source; for these, the single valid source entry is adopted.

3.4.1.5 Mutation

Mutation implies stochastic perturbation of simulation parameters, the extent of which depends on the mutation probability P and mutation amount A. For the simulations of this thesis, $P = 0.1$ and $A = 0.2$ is used. The former implies that, at average, in the making of a generation every tenth parameter is mutated. The latter implies that, at average, the size of the mutation is one fifth of the entire value range.[45] The choice of $P = 0.1$ may, perhaps, seem high compared to the general recommendations on mutation rates of 1-5% presented elsewhere.[46] There is, however, one important argument support having such a high mutation rate: The initial generation of the simulations herein is based on seeding an individual and mutating it to generate the rest of the individuals, whereas most other GA simulations create the first generation by random selection.[47] The latter approach will, naturally, span a greater range of the entire solution domain than the approach taken here. Therefore, a higher mutation rate is needed to make the search explore areas of the parameter space that are distant from the initial seed.

Single value parameters

To simulate randomness in the size of the mutation for single value parameters, Box Muller Gaussian sampling (Algorithm 1) with $\mu = 0$ and variable σ^2 is used. With that, the semantics of the mutation process can be specified as in Algorithm 6, where $val(p_i)$, $min(p_i)$ and $max(p_i)$ represent the value, minimum and maximum of parameter p_i, respectively.

ANN topology specifications

ANN topology specifications are mutated according to a few simple rules. Let $|C|$ represent the number of clusters, $\mathbf{M}$ represent the cluster connection matrix, and m_{ij} represent the matrix element at row i and column j. The mutation process is simple:

[45] For integral and floating-point parameters, that is; boolean parameters and ANN topology specifications are not affected by the value of A.

[46] A mutation rate of $1/l$ is often suggested when individuals are represented by bit strings, where l denotes the length of the bit string [Bäc93]. For the simulations performed herein, this would roughly translate into a mutation rate of 2-3%.

[47] This choice was made based on discussions at supervisor meetings: The potential advantage of having a highly diverse first generation (random selection) was considered to be outweighed by the advantage of allowing known capable individuals to be seeded. The latter allows parts of the search space to be explored manually, based on trial and error. The experience gained thereof, combined with whatever intuition one might have on the variables in use, can then aid the GA search process in that the starting point is known to be favorable according to some specified criteria.

Algorithm 6 Mutation of single value parameters

{a new generation has been generated by recombination}
for each individual i in the generation **do**
  {every individual is subjected to mutation}
  for each parameter p_i of i **do**
    with probability P mutate p_i:
    if p_i is integral **then**
      sample U_0 from $N(0, (max(p_i) - min(p_i)) \cdot A)$
      calculate $\nu = val(p_i) + round(U_0)$
      let $val(p_i) \leftarrow \min(\max(\nu, min(p_i)), max(p_i))$ (truncate)
    end if
    if p_i is floating-point **then**
      sample U_0 from $N(0, (max(p_i) - min(p_i)) \cdot A)$
      calculate $\nu = val(p_i) + U_0$
      let $val(p_i) \leftarrow \min(\max(\nu, min(p_i)), max(p_i))$ (truncate)
    end if
    if p_i is boolean **then**
      if $val(p_i) = false$ **then**
        let $val(p_i) \leftarrow true$
      else
        let $val(p_i) \leftarrow false$
      end if
    end if
  end for
end for

- With probability P, the number of clusters is changed by one:
  - With equal probability, $|C| \leftarrow |C| + 1$
  - or $|C| \leftarrow |C| - 1$
- With probability P, each matrix element m_{ij} is mutated:
  - Let E_{ij} represent the set of m_{ij} entry options, as specified by Table 1 in Section 3.2.6
  - With equal probability, m_{ij} gets one of the values in E_{ij}

As described in Section 3.2.6.2, there are some criteria that must be fulfilled for a topology specification to be valid, and the above mutation (and recombination) semantics may very well violate these criteria. As a part of the mutation process, therefore, ANN topology specifications are mutated until they are valid according to the criteria of Section 3.2.6.2.

3.4.2 GA parameters overview

This section provides a complete list of all parameters used throughout the neuronal and mechanical model simulations, including constant single value parameters (CONST), GA single value parameters (GA) and ANN topology specifications (TOP). In addition, for some parameters the status varies among different experiments between begin CONST and GA; these are

termed VAR. The list gives an overview of the simulation parameter space, and thereby provides important indications as regards the complexity of the GA search process. The list further acts as a reference to the use of simulation parameters in the source code; all single value parameters have names by means of strings of seven characters, and these are the exact same names that are used in the code to look up simulation parameter values.[48] Actual parameter properties are provided in the specification of each experiment of the results part of this thesis (Section 4); for constant single value parameters the actual chosen value is stated (Section 4.1), whereas for GA single value parameters the minimum and maximum values, which define the range within which the parameter is allowed to vary, are provided.[49]

3.4.2.1 Neuronal model parameters

Table 3 lists all parameters used within the neuronal model simulations of this thesis.

Type	**Value type**	**Name**	**Description**	**Model element**
CONST	FLOAT	`zero_sg`	Y-axis zero point for the sigmoid neuronal activation function	x_0 in Eq. (1)
GA	FLOAT	`std_stc`	Standard deviation parameter for Gaussian or Cauchy base probability distribution used in stochastic perturbations	γ in Eq. (6) or σ in Eq. (5)
CONST	BOOL	`stc_dec`	Linear decrease with lifetime in size of stochastic perturbations	N/A
GA	BOOL	`in_ntrc`	Inclusion of neuronal trace element in neuronal activation function	acc_j in Eq. (2)
GA	FLOAT	`alph_ne`	Trace controlling parameter for neuronal trace function	α_j in Eq. (4)
GA	BOOL	`in_dtrc`	Inclusion of trace at neuronal drive values	N/A
CONST	BOOL	`metropo`	Use of Metropolis sampling for neuronal stochastic perturbations	Alg. 2
GA	FLOAT	`lr_skin`	Learning rate for Skinner synapses	β_{ij} in Eq. (10)
GA	FLOAT	`lr_pavl`	Learning rate for Pavlov synapses	β_{ij} in Eq. (12)/(14)
GA	FLOAT	`lr_hume`	Learning rate for Hume synapses	β_{ij} in Eq. (16)/(18)
GA	FLOAT	`alph_sk`	Trace controlling parameter for Skinner synapses	α_{ij} in Eq. (11)
GA	FLOAT	`alph_pa`	Trace controlling parameter for Pavlov synapses	α_{ij} in Eq. (13)/(15)
GA	FLOAT	`alph_hu`	Trace controlling parameter for Hume synapses	α_{ij} in Eq. (17)/(19)

Continued on next page

[48] The reason for using strings of exactly seven characters as parameter names is presented in Appendix C.

[49] Note that the size of the GA parameter space is dictated by this setting of value ranges; each parameter represents a dimension in parameter space, and the value range limits the extension of the space along this dimension.

Table 3 :: Continued				
Type	**Value type**	**Name**	**Description**	**Model element**
GA	BOOL	`newpavl`	Use of modified Pavlov synaptic learning mechanism	Eq. (14)-(15)
GA	BOOL	`newhume`	Use of modified Hume synaptic learning mechanism	Eq. (18)-(19)
CONST	BOOL	`in_mod1`	Inclusion of MOD1 synaptic efficacy divergence preventing modification	Eq. (20)-(21)
CONST	BOOL	`in_mod2`	Inclusion of MOD2 synaptic efficacy divergence preventing modification	Alg. 3
GA	FLOAT	`mod2val`	Parameter controlling the function describing MOD2 efficacy limit as a function of the number of synapses	υ in Eq. (23)
GA	INT	`num_cns`	Number of neurons in internal clusters	N/A
GA	MAT	N/A	ANN topology specification; cluster connection matrix	**M**, Table 1

Table 3: Neuronal model parameters

Comments

Considering the number of parameters used in these simulations, the importance of limiting the number of free (GA) parameters becomes evident. Recall that the size of the GA parameter space grows exponentially with the number of parameters (a.k.a. dimensions); the expected time for GA convergence is therefore heavily dependent on the number of free parameters.

The parameter `zero_sg`, which controls the positioning of the sigmoid neuronal activation function zero-point, is set constant. The chosen value 0.15 is based on what is known of biological neurons; there is some low (possibly random) activity in neurons when the synaptic input is zero, and 0.15 is believed to be an appropriate level [EV07].

The parameter `stc_dec`, not yet discussed, denotes whether or not the standard deviation of stochastic drive perturbations (i.e. the average perturbation size) decreases linearly with time.[50] Preliminary simulations have indicated that such a decrease in randomness is, indeed, favorable. Intuitively, also, this property seems appealing; after initial learning, the creature is hoped to exercise continually increasing determinism in its behavior. Therefore, `stc_dec` is constant, and the value is `true`.

The parameter `in_dtrc`, not yet discussed, denotes whether or not neuronal drives are transformed through a trace function.[51] The idea of using trace functions for neuronal drives is to calm down rapid muscular oscillations, thus hopefully arranging for the slower oscillating movements that are needed for gaits to emerge. It is not known whether the use of trace functions for neuronal drives is propitious or not, and the parameter is therefore free.

[50] The mathematics are straightforward: $\gamma' = \frac{N-t}{N} \cdot \gamma$, where γ is the standard deviation parameter, t is the current iteration number and N is the total number of iterations. The strategy was suggested by J. Hokland, based on discussions at supervisor meetings.

[51] The mathematics are as follows: $D' = (1 - \alpha_D) \cdot D' + \alpha_D \cdot D$ where D' is the traced drive value, D is the instantaneous drive value calculated by the activation function, and α_D is a trace-controlling parameter. For the simulations herein, when drive tracing is included, the value for α_D is five times that of the corresponding muscular alpha (α_m of Section 3.3.2).

The parameter `metropo` is set constant and the value is `true`, denoting that Metropolis sampling is used as the basis for all neuronal drive stochastic perturbations. As discussed in Section 3.2.5.2, Metropolis sampling introduces inertia to the sampling procedure. Thereby, occasional trails of large perturbations, which may be needed for propitious behavior to be discovered, are allowed while still retaining simulation stability and a sufficiently large degree of neuronal determinism. For the symmetric transition function, a Gaussian with standard deviation one tenth of `std_stc` is used.

The parameters `in_mod1` and `in_mod2` are both constant and `true`, denoting that both of the divergence preventing learning mechanism modifications of Section 3.2.5.4 are included. The decision of including these modifications is based on previous experience: In a previous simulation study [Axe06], for some cases (conflicts) both MOD1 and MOD2 were found to be necessary for sensible synaptic learning to take place.

The parameter `num_cns`, not yet discussed, denotes how many neurons each internal ANN cluster has. The importance of this number is not known, and the parameter is therefore free. The range, which is adjusted based on the average number of needs and senses, has been set based on discussions at supervisor meetings.

3.4.2.2 Mechanical model parameters

Table 4 lists all parameters used within the mechanical model simulations of this thesis. Denominations are given in description parentheses.

Type	**Value type**	**Name**	**Description**	**Section**
GA	INT	`num_lps`	Number of leg pairs	3.3.1
CONST	FLOAT	`len_lgs`	Leg length, excluding diameter of spherical feet (m)	3.3.1
GA	FLOAT	`b_l_ver`	Leg vertical base angle (°)	3.3.1
GA	FLOAT	`b_l_hor`	Leg horizontal base angle (°)	3.3.1
GA	FLOAT	`maxdefv`	Maximum vertical angle of deflection (°)	3.3.1
GA	FLOAT	`maxdefh`	Maximum horizontal angle of deflection (°)	3.3.1
CONST	FLOAT	`mx_l_up`	Absolute maximum upward leg angle (°)	3.3.1
CONST	FLOAT	`mx_l_dw`	Absolute maximum downward leg angle (°)	3.3.1
CONST	FLOAT	`mx_l_fo`	Absolute maximum forward leg angle (°)	3.3.1
CONST	FLOAT	`mx_l_ba`	Absolute maximum backward leg angle (°)	3.3.1
CONST	FLOAT	`maxf_mu`	Maximum muscle force (N)	3.3.2
CONST	FLOAT	`m_up_cf`	Maximum force scaling coefficient for muscle pulling leg up	3.3.2
CONST	FLOAT	`m_dw_cf`	Maximum force scaling coefficient for muscle pulling leg down	3.3.2
CONST	FLOAT	`m_fo_cf`	Maximum force scaling coefficient for muscle pulling leg forth	3.3.2
CONST	FLOAT	`m_ba_cf`	Maximum force scaling coefficient for muscle pulling leg back	3.3.2
GA	FLOAT	`alph_mu`	Trace controlling parameter for muscle force trace function	3.3.2

Continued on next page

Table 4 :: Continued				
Type	**Value type**	**Name**	**Description**	**Section**
CONST	FLOAT	`frc_tor`	Friction coefficient for torso against ground	N/A
CONST	FLOAT	`frc_leg`	Friction coefficient for feet against ground	N/A
GA	FLOAT	`ex_angn`	Exponent in functions describing extreme angle needs	3.3.3
VAR	BOOL	`in_hein`	Inclusion of torso height need	3.3.3
VAR	BOOL	`in_hedn`	Inclusion of head height need	3.3.3
VAR	BOOL	`in_beln`	Inclusion of belly pain sensor needs	3.3.3
VAR	BOOL	`in_veln`	Inclusion of velocity need	3.3.3
VAR	BOOL	`in_angn`	Inclusion of extreme joint angle needs	3.3.3
VAR	BOOL	`in_mufn`	Inclusion of muscle force needs	3.3.3
VAR	BOOL	`in_lfrn`	Inclusion of foot friction need	3.3.3
GA	BOOL	`f_hei_n`	Scaling factor for torso height need	3.3.3
GA	BOOL	`f_hed_n`	Scaling factor for head height need	3.3.3
GA	BOOL	`f_bel_n`	Scaling factor for belly pain sensor needs	3.3.3
GA	BOOL	`f_vel_n`	Scaling factor for velocity need	3.3.3
GA	BOOL	`f_ang_n`	Scaling factor for extreme joint angle needs	3.3.3
GA	BOOL	`f_muf_n`	Scaling factor for muscle force needs	3.3.3
GA	BOOL	`f_lfr_n`	Scaling factor for foot friction need	3.3.3
GA	FLOAT	`ex_angs`	Exponent in functions describing muscle length senses	3.3.4
GA	FLOAT	`ex_aves`	Exponent in functions describing muscle length rate senses	3.3.4
GA	BOOL	`in_angs`	Inclusion of muscle length senses	3.3.4
GA	BOOL	`in_aves`	Inclusion of muscle length rate senses	3.3.4
GA	BOOL	`f_ang_s`	Scaling factor for muscle length senses	3.3.4
GA	BOOL	`f_ave_s`	Scaling factor for muscle length rate senses	3.3.4
CONST	INT	`fitfunc`	Integral code (enumeration) specifying which GA fitness function to use	3.4.1.2
CONST	FLOAT	`timstep`	ODE simulation step size (s)	B.2

Table 4: Mechanical model parameters

Comments

The parameter `num_lps`, which denotes how many leg pairs a creature has, is a free parameter that may take on the values 2, 3 and 4. The decision of using such a limited range of values is based on recommendations received at meetings with experts in the field [EV07]; with very few exceptions, the biological systems we wish to imitate (animals/insects/spiders) have four, six or eight legs.

The parameter `len_lgs`, which controls the length of the creature's legs, is set constant. Preliminary simulations have been performed with variable leg length, and the inspection and analysis of these led to the conclusion that the potential advantage of letting the GA process search for an optimal length is by far outweighed by the disadvantages introduced: Firstly, the

length of the legs is crucial in determining how large muscle forces are appropriate for developing sensible behavior, and this would complicate some aspects of the muscle model. Secondly, the length of the legs may affect the creature's innate fitness potential in an unwanted fashion; for instance, when a creature's fitness is measured by how high it manages to elevate its torso, increasing the leg length would be an easy way for the GA process to discover fit individuals. To comment on the latter, it is important to keep in mind that the purpose of the GA search process is to arrange for self-organized ANN learning, not to solve the problem at hand by tuning parameters.

The absolute maximum upward, downward, forward and backward leg angles (`mx_l_up`, `mx_l_dw`, `mx_l_fo`, and `mx_l_ba`, respectively) are included as constants to limit the allowed joint angles within sensible ranges. For instance, the absolute limit on maximum downward angle ensures that creatures never will be able to cross their legs beneath the torso.

The maximum muscle force parameter `maxf_mu` is set constant; the value is based on a combination of trial and error (inspection of real-time simulations) and expert recommendations [EV07]. The reason for keeping it constant is simple: Allowing muscle force to vary freely is not compatible with the above-mentioned principle that it should be the ANN that solves the task, not the GA search process. As an example, this became evident in a preliminary simulation with fitness measured by a creature's average forward velocity: The GA process would vastly increase the creatures' maximum muscle force, resulting in the behavior of fit creatures being characterized by huge erratic and seemingly random jumps around the scene. Such behavior is, of course, detrimental to the development of gaits.

The four scaling coefficients `m_up_cf`, `m_dw_cf`, `m_fo_cf` and `m_ba_cf` denote the size of the maximum muscle force of the four joint muscles relative to the value given by `maxf_mu`. These are constant, and their values are set based on intuition (e.g. that greater force is needed in the downward than in the upward direction) and expert recommendations [EV07].

The parameters `frc_tor` and `frc_leg` denote the friction coefficients between the torso and the ground and between the feet and the ground, respectively. These are set constant for the same reasons discussed earlier; an exceedingly low value for `frc_tor` would, for instance, make the goal of achieving forward velocity much more easily attainable. Preliminary simulations have, indeed, confirmed the above: When `frc_tor` is a free parameter, the GA search process would set this value very low, resulting in creatures sliding on their torso as if the ground were made of ice. An important principle we come upon here is that the GA process should be limited to selected changes in the creature itself (neuronal or mechanical), and not modify the environment in which the creature operates.

The parameter `timstep` which denotes the time step size of an iteration in the mechanical simulation is set constant. Implicitly, the value of this parameter defines the interval with which the ANN is provided with feedback from the mechanical model, and with which the mechanical model is provided with motor activation signals from the ANN. It is far from certain that the chosen value of 10ms, which means that the ANN and mechanical model state is sampled 100 times per second, is optimal.[52] The problem is, however, that a large portion of the other simulation parameters depend on this value. For instance, to obtain similar results, a larger value for `timstep` would require synaptic learning rates to be lower, standard deviations for stochastic perturbations to be larger, trace-controlling parameters to be higher, and so on. Consequently, the parameter is set constant, and a believed appropriate value of 10ms is chosen.

[52] Although theoretical and empirical findings indicate that the value is, at least, sensible [Hok97, Hok98].

3.5 Simulation loop: Executional semantics and run time analysis

With the model components presented in preceding sections in place, the details as regards simulating the entire model on a computer can be summarized. Algorithm 7 provides an algorithmic overview of the simulation loop, depicting the main executional semantics.

Algorithm 7 Main simulation loop

```
1: provide an initial creature (complete parameter specification, seed)
2: generate the first GA generation by mutation of the seed (Section 3.4)
3: while true do
4:    for each creature 1 → G in the current generation do
5:       initialize mechanical model with current creature parameters
6:       initialize neuronal model with current creature parameters
7:       for iteration 1 → N do
8:          perform update of mechanical model (Appendix B)
9:          perform update of neuronal model (Alg. 4, Section 3.2.7)
10:         perform accumulative fitness evaluation (Section 3.4.1.2)
11:      end for
12:   end for
13:   output lifetime visualization data for the fittest creature of this generation to file
14:   output best, average, and worst fitness statistics for this generation to file
15:   generate next generation by elitism, recombination and mutation (Section 3.4)
16: end while
```

As regards total running time, important quantities are, of course, the generation size G and the number of iterations (creature lifetime steps) N. Section 3.2.7 argued that the running time of one neuronal model update is $O(|V| + |E|)$,[53] where $|V|$ and $|E|$ are determined by the current ANN topology. Further, the ODE manual [Smi06] states that the running time of one mechanical model update is $O(m^3)$,[54] where m is the total number of constraint rows. For the simulations herein, m is linear in the number of limbs, which is linear in the number of leg pairs L. Thus, an approximate upper bound on the running time of a mechanical model update is $O(L^3)$. Neuronal and mechanical model updates are performed a total of N times per creature simulation, and with G creatures per generation, the upper bound on the running time for one complete generation can be established as $O(GN(|V| + |E| + L^3))$. Total running time is linear in the number of generations simulated. The number of generations is not specified at startup; rather, the simulations are stopped manually based on inspection of GA fitness statistics.

[53] The bound $\Theta(|V| + |E|)$, of course, implies $O(|V| + |E|)$.

[54] This is the running time of the `dWorldStep()` update function. The `dWorldQuickStep()` function provided by ODE is considerably faster, but not accurate enough for our purposes.

4 Results

This section describes the simulations constituting the experimental work of this thesis.

4.1 Preliminaries

The description of a specific experiment consist of two main parts: a complete **system configuration** defining all startup conditions, and the **experimental results**.

A system configuration, which specifies the executional properties of an experiment and all its parameters, includes the following:

- An overview of which needs are included for the experiment
- A specification of the function used for GA fitness evaluation
- Initial values and value ranges for all GA parameters
- Initial ANN topology specification
  - The initial topology (Table 5) is equal for the first three experiments,[55] and is therefore not described separately for each.
  - Topologies change throughout the simulation according to the semantics of the GA recombination and mutation processes (Section 3.4)

	N	S	A	M
N	X	X	C	X
S	X	X	P	0
A	X	X	X	S
M	X	X	0	0

Table 5: Default ANN topology specification

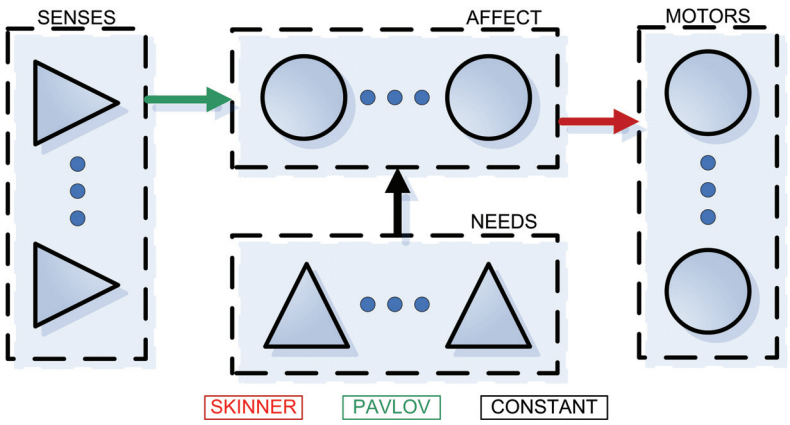


Figure 25: Default topology structure

[55] The remaining two experiments utilize different initial topologies which are described later.

The experimental results are twofold. Firstly, genetic algorithms are used to search a large parameter space and, hopefully, some propitious (according to some prespecified criteria) solutions are found. These resulting **GA parameter settings** constitute an important part of the results, as they indicate what type of configurations are likely to produce good or interesting results.

Secondly, and most importantly, the simulations produce **creature lifetime visualizations** that allow for visual inspection of the creature behavior that emerged as a combination of GA parameter settings and neural network learning.

When considering these two aspects of the experimental results, it is important to keep the following in mind: The use of genetic algorithms to search for propitious parameter configurations is simply a matter of supporting the principal objective of this thesis: performing and analyzing neural network simulations.

Constant simulation parameters

Section 3.4.2 presented all constant (CONST) and variable (GA) parameters used in the simulations of this thesis. For constants, the exact same value is used for every experiment. As a collective reference, Table 6 provides a complete overview specifying the values of all neuronal and mechanical model constant parameters.

Value type	**Name**	**Description**	**Value**
Neuronal model constants			
FLOAT	`zero_sg`	Y-axis zero point for the sigmoid neuronal activation function	0.15
BOOL	`stc_dec`	Linear decrease with lifetime in size of stochastic perturbations	true
BOOL	`metropo`	Use of Metropolis sampling for neuronal stochastic perturbations	true
BOOL	`in_mod1`	Inclusion of MOD1 synaptic efficacy divergence preventing modification	true
BOOL	`in_mod2`	Inclusion of MOD2 synaptic efficacy divergence preventing modification	true
Mechanical model constants			
FLOAT	`len_lgs`	Leg length, excluding diameter of spherical feet (m)	1.2
FLOAT	`mx_l_up`	Absolute maximum upward leg angle (°)	50
FLOAT	`mx_l_dw`	Absolute maximum downward leg angle (°)	-90
FLOAT	`mx_l_fo`	Absolute maximum forward leg angle (°)	80
FLOAT	`mx_l_ba`	Absolute maximum backward leg angle (°)	-80
FLOAT	`maxf_mu`	Maximum muscle force (N)	5
FLOAT	`m_up_cf`	Maximum force scaling coefficient for muscle pulling leg up	1/3
FLOAT	`m_dw_cf`	Maximum force scaling coefficient for muscle pulling leg down	1

Continued on next page

Table 6 :: Continued			
Value type	**Name**	**Description**	**Value**
FLOAT	`m_fo_cf`	Maximum force scaling coefficient for muscle pulling leg forth	1/3
FLOAT	`m_ba_cf`	Maximum force scaling coefficient for muscle pulling leg back	1
FLOAT	`frc_tor`	Friction coefficient for torso against ground	0.3
FLOAT	`frc_leg`	Friction coefficient for feet against ground	1
FLOAT	`timstep`	ODE simulation step size (s)	0.01

Table 6: Constant simulation parameters

The initial setting of variable (GA) parameters, i.e. initial (seed) values and value ranges, varies among simulation experiments. Consequently, all such settings are specified separately for each specific experiment.

Neural network initialization

At simulation startup, ANN values are initialized as follows:

- Need input and affect neuron drives are set from the mechanical model startup state.
- Sense inputs are set from the mechanical model startup state.
- Neuronal drives $D_j = 0.15$ for all neurons except needs, senses and affects (because the activation function is centered at `zero_sg` $= 0.15$).
- Neuronal traces $acc_j = -0.35$ for all neurons except needs, senses and affects (in accordance with the above drive value).
- Synaptic efficacies $e_{ij} = 0$ for all synapses (nothing has been learnt).
- Synaptic trace of postsynaptic drive differentials $T_{ij} = 0$ for all synapses (no changes of synaptic drive differentials have taken place).

Mechanical model initialization

At simulation startup, the mechanical model is initialized such that the torso is lying flat on the ground, with all legs pointing straight out from the torso and resting on the ground.

4.1.1 Outline

Actual simulation results are organized as follows: Sections 4.2-4.3 (Experiments 1 and 2) present the results obtained from early simulations where the system goal was based on torso or head height. Sections 4.4-4.7 (Experiments 3, 4 and 5) present the results obtained from simulations investigating the development of movement patterns for forward velocity.

Experiments 1 and 2 principally establish the validity and stability of the model and simulation system as a whole; compared to later experiments, the actual simulation results obtained

are of limited importance. In the thoroughness of their description, however, these experiments contain a good introduction to the simulation system and experimental setup.

Experiments 3, 4 and 5 constitute the main empirical value of the experimental work of this thesis. These experiments focus on the main experimental goal of this thesis: the development of synchronized and repetitive movement patterns for forward velocity.

The time-pressured reader may skip Sections 4.2-4.3 and concentrate on the last experiments. To get a thorough overview of the simulation system and experimental setup, and for a full understanding of the progress in simulation results, however, it is recommended that all sections be read.

4.2 Experiment 1: Torso height

This section describes the system configuration and simulation results constituting Experiment 1: Torso height. This experiment is considered a natural starting point for the neural network simulations of this thesis; raising and balancing the torso requires the ANN to perform simple yet determined and partly synchronized muscular control, and the experiment can thereby be used to verify the system's basic abilities.

4.2.1 Goal specification, needs and fitness function

For this experiment, the system goal is based on torso height: The creature is rewarded, both in terms of need values and in terms of fitness evaluation, for keeping its torso as high above ground as possible. The actual value used in these need and fitness calculations is based on a multiplicative average of the four upper corners of the creature's torso, as follows:[56]

$$heightvalue = y_0 \cdot y_1 \cdot y_2 \cdot y_3,$$

where y_0, y_1, y_2, and y_3 are the world coordinate vertical height components of the four upper torso corners, scaled to range within $[0, 1]$. Due to the multiplicative height averaging, the optimal torso orientation is when the torso is flat and fully balanced horizontally, and as high as possible. More specifically, this means that the creature is more heavily rewarded when its torso is held relatively high and balanced than when the torso is correspondingly high and unbalanced. An example of the latter is when the front of the torso is elevated very high at the expense of the back of the torso.

For this experiment, the configuration of needs is as follows:

Need	Configuration	Included
Torso height need	CONST	Yes
Head height need	CONST	No
Belly pain sensor needs	GA	Optional
Velocity need	CONST	No
Extreme joint angle needs	CONST	No
Muscle force needs	CONST	No
Foot friction needs	CONST	No

As indicated, the most important need for this experiment is the torso height need (as partially stated above and described in detail in Section 3.3.3.1), which wholly and fully incorporates the system goal of achieving maximum balanced torso height. The other needs are less important, or conflict the goal specification. The head height need conflicts the goal of keeping the torso horizontally balanced, and is thus excluded. The belly pain sensor needs may be helpful in the initial phase of getting the torso up from the ground, and the inclusion of this need type is therefore a free parameter in the GA process. The velocity need conflicts the goal, because if the creature is to be moving forward, its torso must be lower than it can be when standing still and fully stretched upward; it is therefore excluded. The same goes for the 1) extreme joint angle needs, 2) muscle force needs and 3) foot friction needs, all of these are excluded; 1) maximum torso height is most likely achieved when vertical joint angles are near their an extreme, 2) considerable muscle force is needed to lift and hold the torso highly elevated, and

[56] The torso height need, as described by Eq. (33) of Section 3.3.3.1, is simply the additive inverse of this value.

3) since the creature's legs are not articulated, the only way to lift the torso is by sliding the feet toward the center beneath the torso.

With the above inverted height value, the fitness function can be specified as follows:

$$fitness^0 = 0 \tag{43}$$

$$fitness^t = fitness^{t-1} + \frac{t}{N} \cdot heightvalue, \tag{44}$$

This fitness evaluation function should ensure that, in the selection of a small number of individuals for survival and recombination from a large and possibly diverse generation, those individuals whose parameter settings are propitious for achieving and retaining maximum torso height are chosen. To summarize, the desired simulation result is that the creature synaptically learns the behavior of getting up on its feet, lifting its torso high above ground, and stably and consistently maintaining a highly elevated and horizontally balanced torso orientation.

4.2.2 Parameters and settings

This section specifies the initial setting of all variable (GA) parameters for Experiment 1. The complete configuration of these parameters, which represents the *input* to the GA process, defines both the size of the GA parameter space and the seed, i.e. starting point, of the GA search. It is therefore of considerable importance, both with respect to performance (GA convergence time) and as regards what configurations are at all possible and discoverable. Table 7 lists the parameters together with their initial (seed) values and value ranges.

Value type	**Name**	**Description**	**Seed value**	**Range**
Neuronal model GA parameters				
FLOAT	`std_stc`	Standard deviation parameter for Cauchy probability distribution used in stochastic perturbations	1	[0.1, 2]
BOOL	`in_ntrc`	Inclusion of neuronal trace element in neuronal activation function	true	
FLOAT	`alph_ne`	Trace controlling parameter for neuronal trace function	0.5	[0.01, 1]
BOOL	`in_dtrc`	Inclusion of trace at neuronal drive values	false	
FLOAT	`lr_skin`	Learning rate for Skinner synapses	2000	[100, 10000]
FLOAT	`lr_pavl`	Learning rate for Pavlov synapses	500	[25, 2500]
FLOAT	`lr_hume`	Learning rate for Hume synapses	50	[1, 200]
FLOAT	`alph_sk`	Trace controlling parameter for Skinner synapses	0.1	[0.01, 1]
FLOAT	`alph_pa`	Trace controlling parameter for Pavlov synapses	0.1	[0.01, 1]
FLOAT	`alph_hu`	Trace controlling parameter for Hume synapses	0.1	[0.01, 1]

Continued on next page

Table 7 :: Continued				
Value type	**Name**	**Description**	**Seed value**	**Range**
BOOL	`newpavl`	Use of modified Pavlov synaptic learning mechanism	false	
BOOL	`newhume`	Use of modified Hume synaptic learning mechanism	false	
FLOAT	`mod2val`	Parameter controlling the function describing MOD2 efficacy limit as a function of the number of synapses	7	[5, 100]
INT	`num_cns`	Number of neurons in internal clusters	20	[10, 50]
Mechanical model GA parameters				
INT	`num_lps`	Number of leg pairs	3	{2, 3, 4}
FLOAT	`b_l_ver`	Leg vertical base angle (°)	-20	[-70, 0]
FLOAT	`b_l_hor`	Leg horizontal base angle (°)	0	[-20, 20]
FLOAT	`maxdefv`	Maximum vertical angle of deflection (°)	45	[20, 70]
FLOAT	`maxdefh`	Maximum horizontal angle of deflection (°)	45	[20, 70]
FLOAT	`alph_mu`	Trace controlling parameter for muscle force trace function	0.01	[0.001, 0.2]
BOOL	`in_beln`	Inclusion of belly pain sensor needs	false	
BOOL	`f_hei_n`	Scaling factor for torso height need	1	[0.1, 1]
BOOL	`f_bel_n`	Scaling factor for belly pain sensor needs	1	[0.1, 1]
FLOAT	`ex_angs`	Exponent in functions describing muscle length senses	1	[1, 2]
FLOAT	`ex_aves`	Exponent in functions describing muscle length rate senses	1	[1, 2]
BOOL	`in_angs`	Inclusion of muscle length senses	false	
BOOL	`in_aves`	Inclusion of muscle length rate senses	false	
BOOL	`f_ang_s`	Scaling factor for muscle length senses	1	[0.1, 1]
BOOL	`f_ave_s`	Scaling factor for muscle length rate senses	0.1	[0.01, 1]

Table 7: Experiment 1 - GA parameter settings

4.2.3 Experiment results

This section presents a representative selection of the results obtained from Experiment 1: Torso height. The GA process was run with the standard settings described in Section 3.4: mutation rate $P = 10\%$, mutation amount $A = 20\%$, and 50 individuals per generation. Each creature was simulated for 50000 iterations, which translates into a real-time lifetime of 8.33 minutes. The simulations were stopped after 41 generations, implying that a total of 2050 virtual creatures were simulated for this experiment.

4.2.3.1 Results overview

Figure 26 shows the progress in fitness values over the first 35 generations (numbered 0-34), depicting the best, average, and worst fitness values for each generation. There was no further

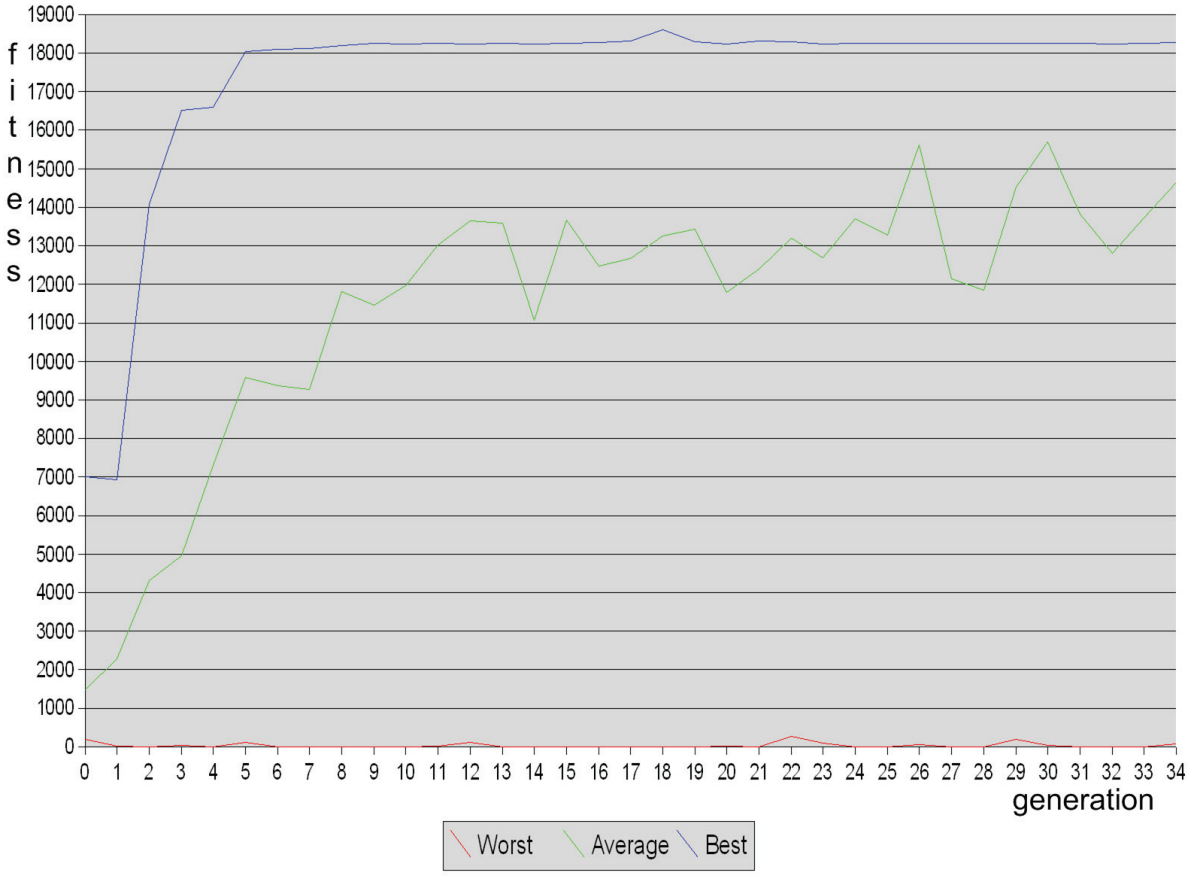


Figure 26: Fitness progress for Experiment 1: Torso height

development in fitness or behavior for the last six generations, and the corresponding fitness data are consequently not shown in the chart.

For a reference as regards torso height fitness values, to get a grip of what values to expect, the following should be noted: If the creature lies completely still on the ground without lifting its torso throughout its entire lifetime (as if it were dead), the lifetime fitness will be close to zero. At the opposite, if the creature stands perfectly still and balanced with its feet straight down, the instantaneous fitness value is approximately 0.732. Thus, over a lifetime of 50000 iterations, and with an average time-weighting of 0.5 (see Section 3.4.1.2), the lifetime fitness would be approximately 18300.

It is evident from the chart that the best fitness grows remarkably over the first six generations; the growth is from 7013 at generation 0 to 18040 at generation 5. Over the next four generations, the fitness continues to grow slowly toward the theoretical optimum of 18300; at generation 9 the fitness is 18265. From this point on, the best fitness values stabilize and never fall below 18200. For some generations, the best fitness even exceeds the theoretical optimum of 18300. With a view to the above reference on torso height fitness values, this is considered to be a very good result; despite of the fact that the creature uses some time to learn how to lift its torso, the value is very close to, and sometimes even above, the calculated theoretical

optimum. To summarize, this progress in best fitness tells us that the task at hand, i.e. lifting the torso as high and stably above ground as possible, is solved efficiently and successfully.

A very similar tendency is seen for average fitness; the value grows from 1483 to 9576 over the first six generations. Over the next six generations, the trend continues, and the average fitness for generation 11 is 13013. The value never stabilizes to the extent that the value for best fitness does - the graph is prominently jagged - but it is clearly concentrated around 13000, with most values ranging within $[12000, 14000]$.

There is not much progress in the values representing worst fitness per generation; with a few exceptions, this value remains quite stable at close to zero. This result should be seen in relation to the GA process at hand, and the way it is configured; both the mutation probability and the mutation amount are relatively large (10% and 20%, respectively), and the probability that some individual in a generation ends up having one or a few highly unfavorable parameter settings is rather large. Thus, it is not unexpected that the worst fitness per generation remains very low throughout the entire simulation; this is the price to pay for having a GA process where diversity within generations is favored.

4.2.3.2 Analysis of specific creatures - GA output and qualitative descriptions

This section presents the lifetime visualizations (i.e. observable behavior) of the best creatures from a few selected generations, together with the corresponding GA *output*, i.e. GA parameter values. The selection has been made with the purpose of demonstrating the main developmental steps seen in the progress toward successful goal achievement. Corresponding lifetime fitness values are shown in the headings below.

Generation 0: 7013

The best creature from generation 0 achieved a lifetime fitness of 7013. The ANN topology of this creature, shown in Table 8, is identical to the default topology. The remaining GA output is shown in Table 9, where only those parameters whose values have changed compared to the initial GA input specification (Table 7) are listed.

Value type	Name	Description	Seed value	Value	Range
BOOL	`in_ntrc`	Inclusion of neuronal trace element in neuronal activation function	true	false	
BOOL	`in_dtrc`	Inclusion of trace at neuronal drive values	false	true	
FLOAT	`maxdefh`	Maximum horizontal angle of deflection (°)	45	36.5	[20, 70]
FLOAT	`alph_mu`	Trace controlling parameter for muscle force trace function	0.01	0.04	[0.001, 0.2]

Table 9: GA output GEN0

Because the first generation is generated merely by mutation of the initial seed, the chances for parametric changes are limited. As regards the specific parameters changed, a reduction in `maxdefh` is not unexpected as it makes the creature less probable of falling back or forth. Also,

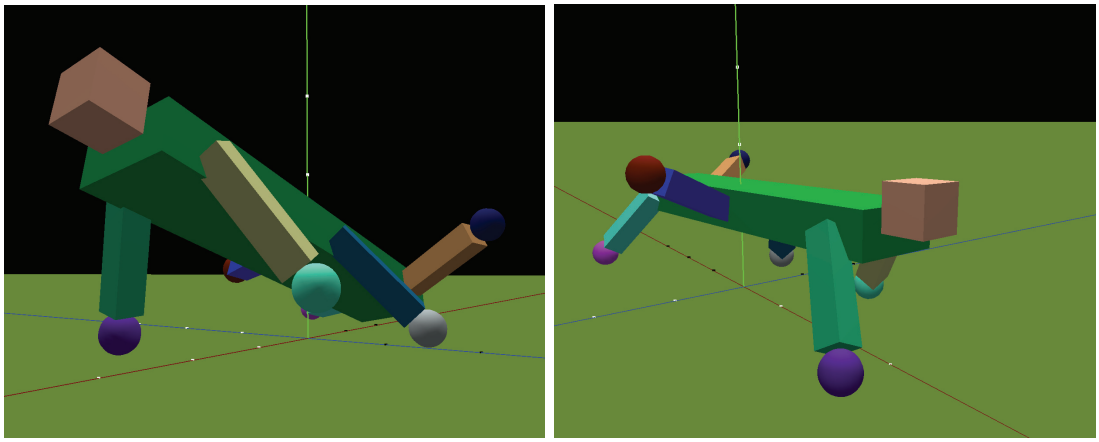


Figure 27: 3D visualization screenshots GEN0

	N	S	A	M
N	X	X	C	X
S	X	X	P	0
A	X	X	X	S
M	X	X	0	0

Table 8: Topology specification GEN0

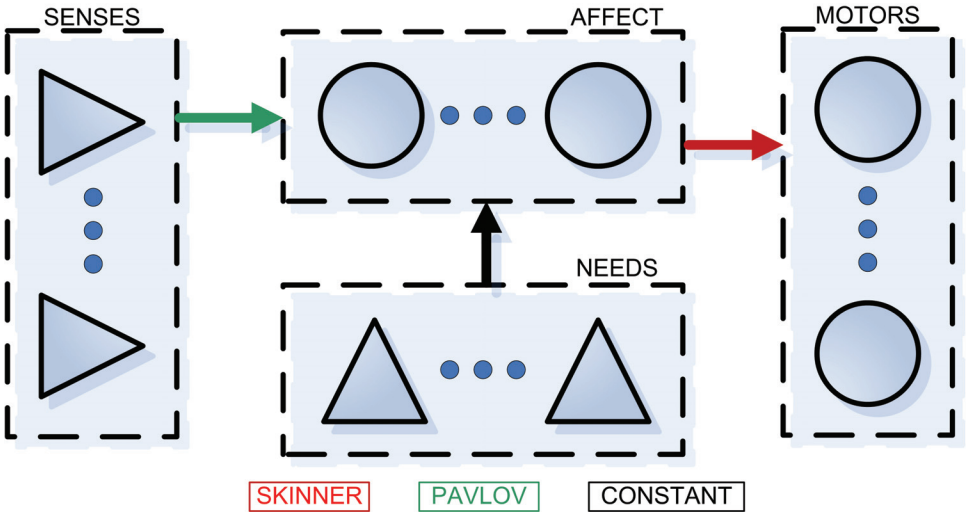


Figure 28: Topology structure GEN0

a reduced maximum horizontal angle of deflection should make the torso more highly elevated if the creature is resting its weight on the maximum horizontal joint angles. The positive effect of the other three parameter changes is not immediately evident, and the fact that the fittest creature of generation 0 had these changes may be incidental.

The lifetime visualization (Figure 27) shows that the creature initially learns to lift the back of the torso using the muscles at hindmost leg pair. Gradually, after this, the creature lifts the front of its torso using its two front legs. Coincidentally with lifting the front of its torso, however, the creature accidentally lifts one of the hindmost legs too, and thus falls backward. Soon after this, the creature once again lifts the back of its torso, and thus manages to elevate the entire torso in a quite balanced fashion. The creature now gradually learns to stabilize this body position, although two legs remain lifted toward the maximum upward joint angle. Because the creature has six legs, it can easily balance its torso in an upright position using in effect only four legs. The creature remains in this state for the rest of its lifetime: Four legs support the weight of the torso, while the two remaining legs are jiggled forth and back in a seemingly random fashion. The creature can be seen to rest some of the weight of its torso on the rather large maximum horizontal leg angles, and never learns to straighten its legs.

Generation 4: 16585

The best creature from generation 4 achieved a lifetime fitness of 16585, which is more than twice the fitness obtained at generation 0. The ANN topology is slightly changed compared to the default topology: An internal cluster has been introduced, to which the affect cluster connects by means of Skinner synapses, and which connects onto the motor cluster. Further, the motor cluster is intraconnected by means of Hume synapses, and the senses connect directly onto the motor cluster instead of going through the affect cluster. The resulting topology specification is shown in Table 10. The remaining GA output is shown in Table 11, where only those parameters whose values have changed compared to the initial GA input specification (Table 7) are listed.

Value type	Name	Description	Seed value	Value	Range
FLOAT	`std_stc`	Standard deviation parameter for Cauchy probability distribution used in stochastic perturbations	1	1.70	[0.1, 2]
BOOL	`in_ntrc`	Inclusion of neuronal trace element in neuronal activation function	true	false	
FLOAT	`lr_hume`	Learning rate for Hume synapses	50	77.7	[1, 200]
FLOAT	`alph_sk`	Trace controlling parameter for Skinner synapses	0.1	0.24	[0.01, 1]
FLOAT	`alph_hu`	Trace controlling parameter for Hume synapses	0.1	0.24	[0.01, 1]

Continued on next page

Table 11 :: Continued					
Value type	**Name**	**Description**	**Seed value**	**Value**	**Range**
FLOAT	`mod2val`	Parameter controlling the function describing MOD2 efficacy limit as a function of the number of synapses	7	5	[5, 100]
FLOAT	`b_l_ver`	Leg vertical base angle (°)	-20	-42.8	[-70, 0]
FLOAT	`b_l_hor`	Leg horizontal base angle (°)	0	20	[-20, 20]
FLOAT	`maxdefv`	Maximum vertical angle of deflection (°)	45	48.6	[20, 70]
FLOAT	`maxdefh`	Maximum horizontal angle of deflection (°)	45	36.4	[20, 70]
BOOL	`in_beln`	Inclusion of belly pain sensor needs	false	true	

Table 11: GA output GEN4

To summarize the above, we see that over the first five generations, there has been a considerable number of changes in the GA parameters. The most important are as follows: In the neuronal model, the average size of stochastic drive perturbations has increased. Such an increase of randomness should make initial learning faster, although behavioral stability over time may be adversely affected. In the mechanical model, the vertical and horizontal base angles, together with the corresponding maximum angles of deflection, have changed. Presumably, the new configuration on allowed joint angles is more suitable with respect to the torso height goal. Finally, the belly pain sensor needs have been included; as indicated earlier, they may be favorable in helping the creature to learn the initial behavior of getting the torso up from the ground.

When considering the lifetime visualization (Figure 29), the creature from generation 4 performs considerably better than the one from generation 0 discussed above; the creature rather quickly manages to get up from the ground, and soon achieves a quite good torso height. In this process of raising its torso, the creature must overcome opposing forces due to friction between its feet and the ground. Most importantly, static frictional forces must be overcome before the feet can slide toward the middle.[57] Interestingly, the creature seems to be taking advantage of the fact that static frictional forces are more easily overcome if, when pulling the feet inwards, the torso is simultaneously rocked slightly back and forth. This behavior is also interesting with respect to the emerging of synchronized oscillating patterns; some sort of oscillatory muscular control is probably needed for such synchronized leg movements to take place.

At the early stage of standing up, the creature does not distribute the weight of its torso equally among its six legs; the front of the torso is held lower than the back, making the hindmost pair of legs dangle freely in the air. With time, the creature learns to balance the torso in a better way, such that all six legs support the torso weight. The creature ends up in a propitious

[57] Static frictional force will increase to prevent relative motion up until some limit where motion occurs. Once the limit is exceeded, frictional forces drop, because the coefficient of static friction typically is larger than the coefficient of kinetic friction [TM03].

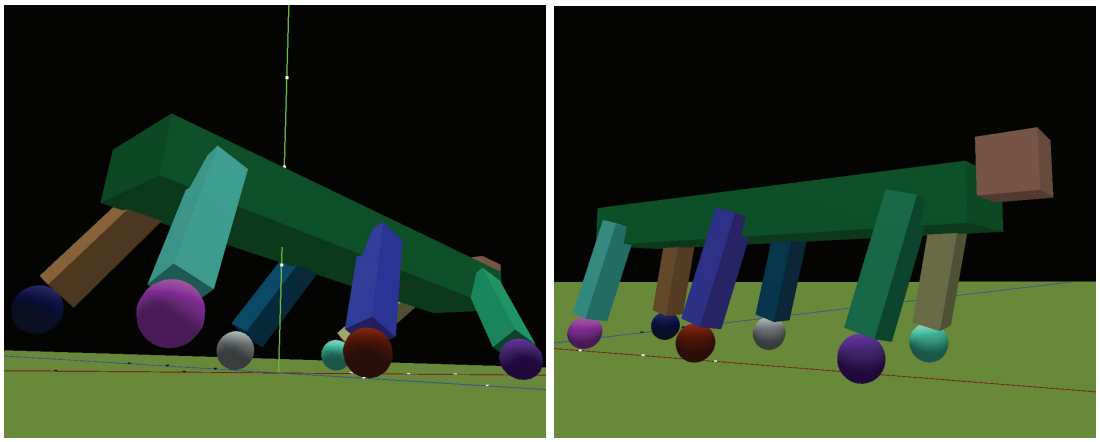


Figure 29: 3D visualization screenshots GEN4

	N	S	A	I1	M
N	X	X	C	X	X
S	X	X	0	0	P
A	X	X	X	S	S
I1	X	X	0	0	P
M	X	X	0	0	H

Table 10: Topology specification GEN4

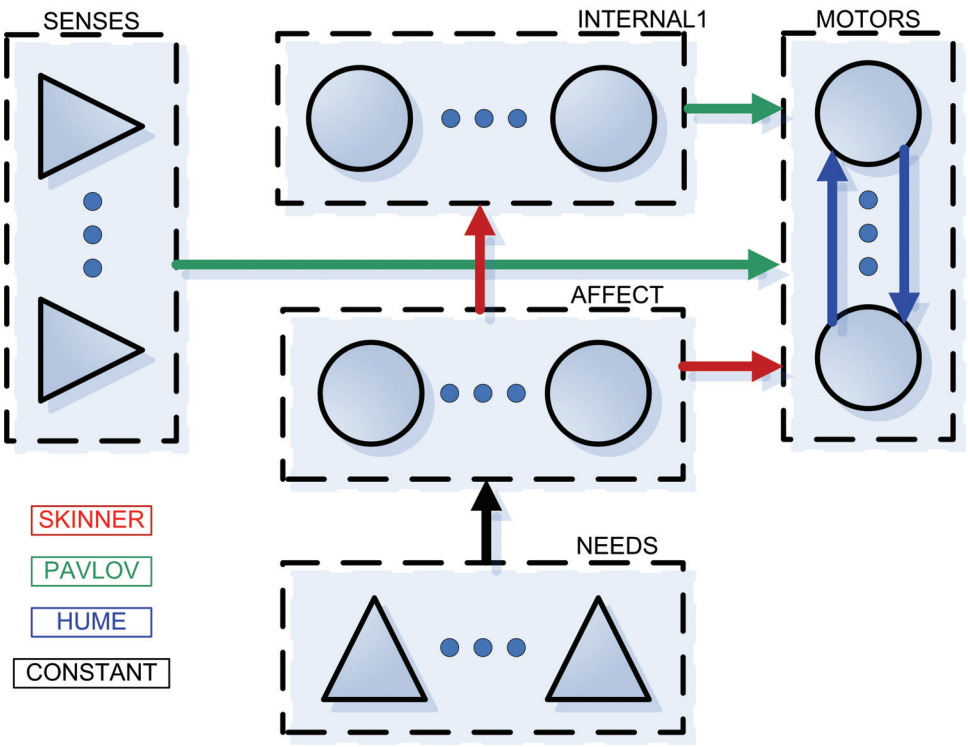


Figure 30: Topology structure GEN4

state where all six legs are held fairly straight downward, making the torso remain prominently elevated above ground. The creature never settles completely, however; throughout the rest of its lifetime, the creature seems to attempt to straighten its legs perfectly, but it never manages to stabilize any further. This tottering actually causes the creature to move slowly forward. Recalling that the only need implemented for this creature is the torso height need and belly pain sensor needs, this forward moving behavior is, of course, merely a matter of coincidence.[58]

Generation 9: 18265

The best creature from generation 9 achieved a lifetime fitness of 18265, which is within a 2% margin of the theoretical optimum of 18300 calculated earlier. The ANN topology is slightly changed compared to the default topology: The topology has one intraconnected internal cluster to which the affect cluster connects, and that connects onto the motor cluster. Further, the motor cluster is now intraconnected by means of Pavlov synapses. Finally, the senses now connect directly onto the motor cluster. The resulting topology specification is shown in Table 12.

The remaining GA output is shown in Table 13, where only those parameters whose values have changed compared to the initial GA input specification (Table 7) are listed.

Value type	Name	Description	Seed value	Value	Range
FLOAT	`std_stc`	Standard deviation parameter for Cauchy probability distribution used in stochastic perturbations	1	1.66	[0.1, 2]
BOOL	`in_ntrc`	Inclusion of neuronal trace element in neuronal activation function	true	false	
FLOAT	`alph_ne`	Trace controlling parameter for neuronal trace function	0.5	0.19	[0.01, 1]
FLOAT	`lr_skin`	Learning rate for Skinner synapses	2000	3016	[100, 10000]
FLOAT	`lr_pavl`	Learning rate for Pavlov synapses	500	719	[25, 2500]
FLOAT	`lr_hume`	Learning rate for Hume synapses	50	75	[1, 200]
FLOAT	`alph_sk`	Trace controlling parameter for Skinner synapses	0.1	0.24	[0.01, 1]
FLOAT	`alph_pa`	Trace controlling parameter for Pavlov synapses	0.1	0.42	[0.01, 1]
FLOAT	`alph_hu`	Trace controlling parameter for Hume synapses	0.1	0.24	[0.01, 1]
FLOAT	`b_l_ver`	Leg vertical base angle (°)	-20	-42.8	[-70, 0]
FLOAT	`b_l_hor`	Leg horizontal base angle (°)	0	20	[-20, 20]

Continued on next page

[58] The behavior might, however, be an after-effect of the way the creature learnt to raise its torso.

Table 13 :: Continued					
Value type	**Name**	**Description**	**Seed value**	**Value**	**Range**
FLOAT	`maxdefv`	Maximum vertical angle of deflection (°)	45	48.6	[20, 70]
FLOAT	`maxdefh`	Maximum horizontal angle of deflection (°)	45	20	[20, 70]
FLOAT	`alph_mu`	Trace controlling parameter for muscle force trace function	0.01	0.02	[0.001, 0.2]
BOOL	`in_beln`	Inclusion of belly pain sensor needs	false	true	
BOOL	`in_aves`	Inclusion of muscle length rate senses	false	true	
BOOL	`f_ave_s`	Scaling factor for muscle length rate senses	0.1	0.29	[0.01, 1]

Table 13: GA output GEN9

At generation 9, a considerable number of parameters have changed, where the most important are as follows: For the neuronal model, the randomness in neuronal drives has increased, making initial learning faster. Further, all three learning rates and corresponding trace-controlling parameters have been adjusted. For the mechanical model, the maximum horizontal angle of deflection has been minimized. The latter would be expected, because large horizontal joint angle deflections are not favorable with respect to the torso height goal. Also, the muscle length rate senses have been included, and their relative influence and intensity has been increased.

When considering the lifetime visualization (Figure 31), the creature from generation 9 performs very well. It quickly gets up on all six feet, learns to stabilize its torso, and never loses balance. The creature manages to keep its legs much straighter than what was seen for the creature of generation 4, thus causing the torso to be elevated considerably higher. Shortly after getting up, two of the creature's legs are dangling freely and randomly outwards. After some time, however, the creature learns to utilize one of these, the middle left leg, to support the torso weight, thereby allowing even better torso stabilization. The front right leg never settles, and keeps moving randomly outwards for the rest of the creature's lifetime.

Generation 18: 18603

At generation 18, there is a prominent peak in the graph for best fitness; the best creature from this generation achieved a lifetime fitness of 18603, which is exceedingly high considering the previously calculated approximate theoretical optimum of 18300. The ANN topology is only slightly different from the default topology: A direct Pavlov connection has been introduced from the senses to the motor cluster; other than that, the topology is unaltered. The resulting topology specification is shown in Table 14.

The remaining GA output is shown in Table 15, where only those parameters whose values have changed compared to the initial GA input specification (Table 7) are listed.

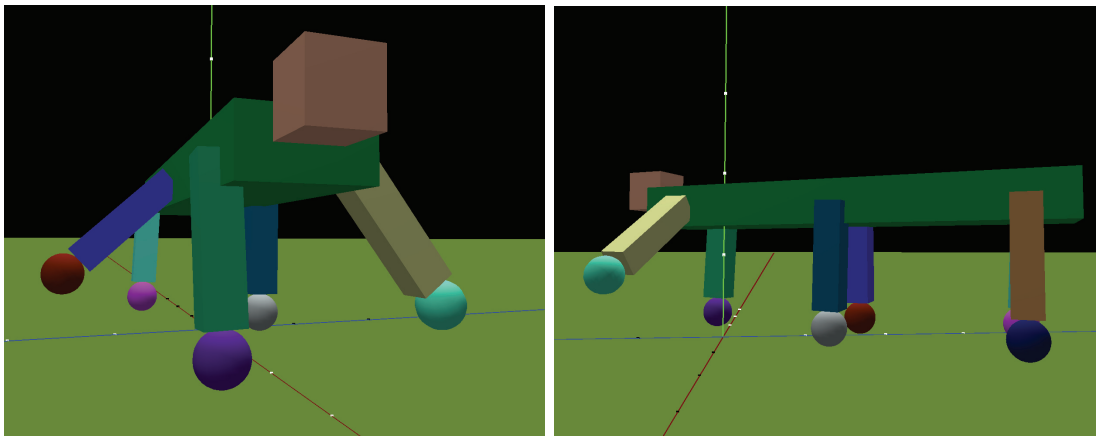


Figure 31: 3D visualization screenshots GEN9

	N	S	A	I1	M
N	X	X	C	X	X
S	X	X	P	0	P
A	X	X	X	S	S
I1	X	X	0	H	P
M	X	X	0	0	P

Table 12: Topology specification GEN9

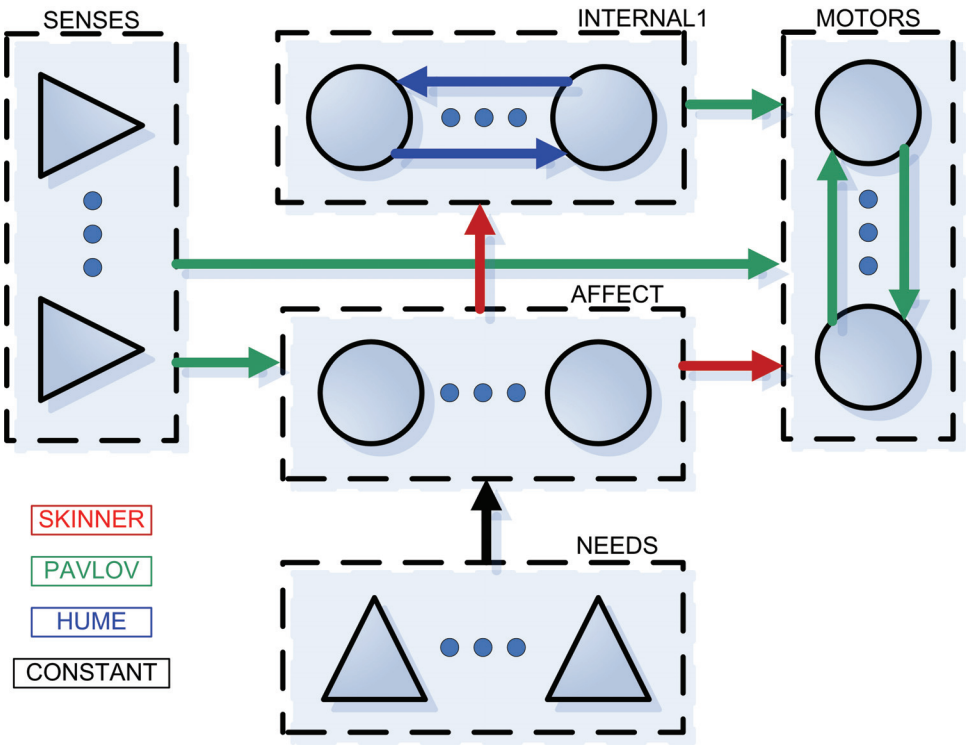


Figure 32: Topology structure GEN9

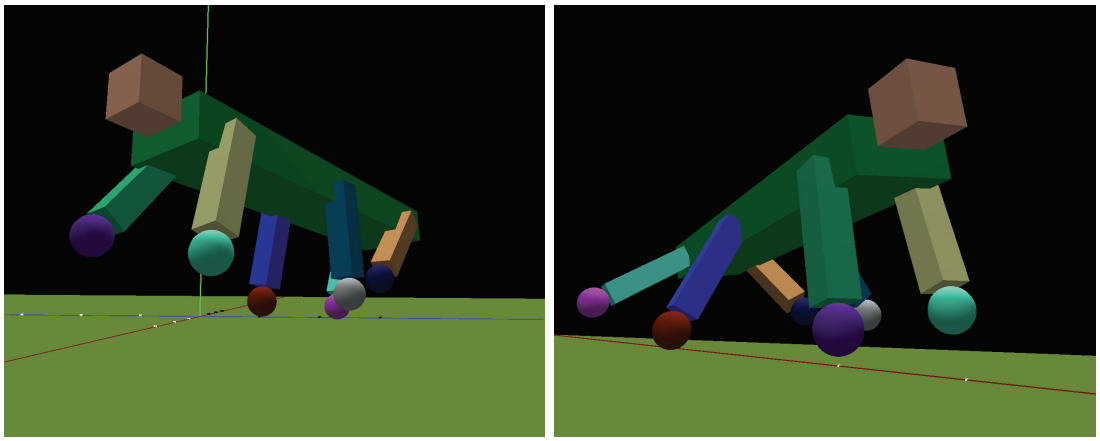


Figure 33: 3D visualization screenshots GEN18

	N	S	A	M
N	X	X	C	X
S	X	X	P	P
A	X	X	X	S
M	X	X	0	0

Table 14: Topology specification GEN18

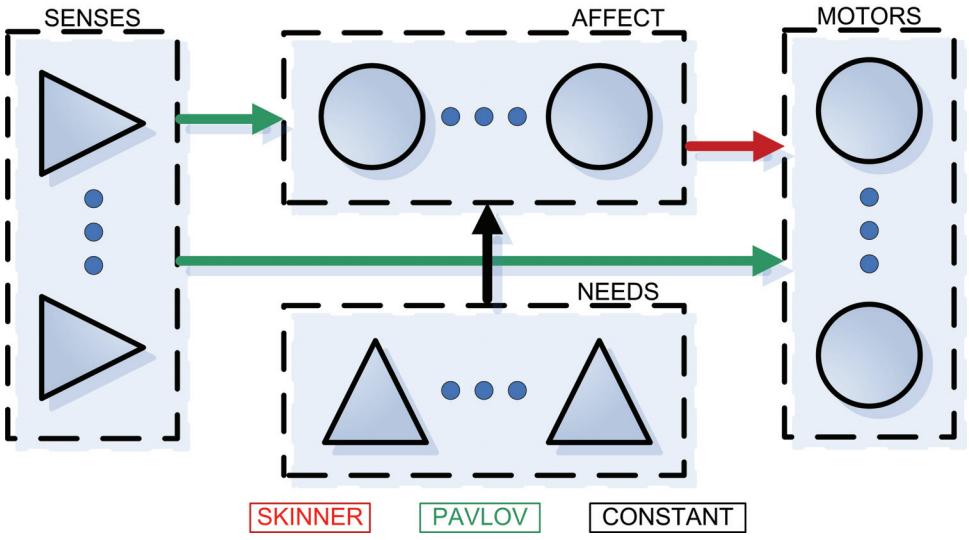


Figure 34: Topology structure GEN18

Value type	Name	Description	Seed value	Value	Range
FLOAT	std_stc	Standard deviation parameter for Cauchy probability distribution used in stochastic perturbations	1	1.19	[0.1, 2]
FLOAT	alph_ne	Trace controlling parameter for neuronal trace function	0.5	0.48	[0.01, 1]
FLOAT	lr_skin	Learning rate for Skinner synapses	2000	1082	[100, 10000]
FLOAT	lr_pavl	Learning rate for Pavlov synapses	500	725	[25, 2500]
FLOAT	alph_sk	Trace controlling parameter for Skinner synapses	0.1	0.16	[0.01, 1]
FLOAT	alph_pa	Trace controlling parameter for Pavlov synapses	0.1	0.01	[0.01, 1]
BOOL	newpavl	Use of modified Pavlov synaptic learning mechanism	false	true	
FLOAT	mod2val	Parameter controlling the function describing MOD2 efficacy limit as a function of the number of synapses	7	19.6	[5, 100]
INT	num_cns	Number of neurons in internal clusters	20	26	[10, 50]
FLOAT	b_l_ver	Leg vertical base angle (°)	-20	-44.1	[-70, 0]
FLOAT	b_l_hor	Leg horizontal base angle (°)	0	16.8	[-20, 20]
FLOAT	maxdefv	Maximum vertical angle of deflection (°)	45	49.2	[20, 70]
FLOAT	maxdefh	Maximum horizontal angle of deflection (°)	45	20	[20, 70]
FLOAT	alph_mu	Trace controlling parameter for muscle force trace function	0.01	0.02	[0.001, 0.2]
BOOL	f_hei_n	Scaling factor for torso height need	1	0.79	[0.1, 1]
BOOL	in_angs	Inclusion of muscle length senses	false	true	
BOOL	f_ang_s	Scaling factor for muscle length senses	1	0.86	[0.1, 1]

Table 15: GA output GEN18

As opposed to the only slightly changed topology specification, at generation 18 a considerable number of parameters have changed, where the most important are as follows: For the neuronal model, there has been a slight increase in randomness, making initial learning faster. Both active learning rates and the corresponding trace-controlling parameters have also been adjusted. Further, the value describing the shape of the function dictating summed incoming synaptic efficacy (MOD2) has increased considerably, thus raising the average maximum value

on synaptic efficacy vastly. For the mechanical model, the joint angle specifiers have been adjusted; most prominently, and not unexpectedly, the maximum horizontal angle of deflection has been minimized. Also, the muscle length senses have been included, and their relative influence and intensity have been decreased slightly.

As mentioned above, this best creature of generation 18 achieved an exceedingly high fitness value of 18603. The lifetime visualization (Figure 33) reveals how this is possible: The early learning phase for this creature is much more chaotic than what has been seen up until now; the early behavior seems random and erratic. As indicated in the previous paragraph, such chaotic behavior at an early stage may be regarded as trying out several behavioral options before settling at one specific pattern of movement.

After the creature in a quite chaotic way manages to get up on its feet and elevate its torso high up, at the point where previously described creatures have calmed down and stabilized their movement patterns, this creature simply continues its erratic behavior. Inspection of the lifetime visualization reveals that the creature continually performs seemingly random but vigorous leg movements that can best be described as small *jumps*.

The apparent randomness in these movements can, however, be questioned: The creature never falls, and almost every significant event of movement causes the torso to rise above what is normally regarded as maximum height. Had these movements been principally random in character, the creature never would have achieved the exceedingly high lifetime fitness of 18603. Hence, we must assume that, for this specific instance, the behavior, which at times resembles jumping, is at least partly caused by chiefly deterministic neural activity. The fact that this specific behavior emerged in this specific creature's lifetime, however, is probably due to randomness in the neuronal model (i.e. stochastic drive perturbations). The following supports the latter: When running identically specified creature simulations (i.e. such that all model parameters are equal), the set of observed behaviors is quite diverse; some creatures exercise behavior that is quite resemblant to what is described above, while other creatures are total failures that e.g. end up on their backs, wriggling their legs in despair. All creatures do show similar erratic tendencies, but the final behavioral results are, as indicated, of quite random character. The only thing differing between these simulations is the differences of neuronal drive perturbations due to randomness in the neuronal model. Consequently, we can conclude that this must be the cause for divergence. It can further be noted that the behavior of creatures with this specific configuration are seen to be particularly sensitive to and dependent on the actual course of stochastic perturbations, which is, of course, not optimal.

The behavior described above is anyhow highly interesting. Although chaotic, it exhibits prominent signs of synchronized behavior; when inspecting the creature's lifetime visualization, there are clear signs of ANN-controlled synchronization between pairs of legs. For instance, the two legs constituting the middle of the three leg pairs can recurringly be seen to be lifted outwards and subsequently and simultaneously thrown inwards at great speed, causing the torso to bump slightly above the normal maximum height. Another example is that the two front legs are repeatedly seen to be thrown forward, causing the front of the torso to elevate to the extent that the front legs are dangling freely above ground. For both these cases, similar but unsynchronized behavior would probably not have caused the creature to "jump" (i.e. cause an increase of torso height). The key aspect here is, of course, simultaneity; when two legs repeatedly perform simultaneous movement patterns that consistently cause the torso to elevate above normal maximum height, it is natural to assume that this synchronization of limbs must be caused by corresponding synchronization in ANN muscular control.

Generation 34: 18276

The best creature from generation 34 achieved a lifetime fitness of 18276. The ANN topology is somewhat modified compared to the default topology: An internal cluster has been added, and a direct Pavlov connection from senses to this cluster has been introduced. The affect cluster also connects onto the internal cluster by means of Skinner synapses. The internal cluster is intraconnected by means of Hume synapses, and it has a Pavlov connection to the motor cluster. The resulting topology specification is shown in Table 16.

The remaining GA output is shown in Table 17, where only those parameters whose values have changed compared to the initial GA input specification (Table 7) are listed.

Value type	Name	Description	Seed value	Value	Range
FLOAT	std_stc	Standard deviation parameter for Cauchy probability distribution used in stochastic perturbations	1	0.6	[0.1, 2]
FLOAT	alph_ne	Trace controlling parameter for neuronal trace function	0.5	1	[0.01, 1]
FLOAT	lr_pavl	Learning rate for Pavlov synapses	500	1033	[25, 2500]
FLOAT	lr_hume	Learning rate for Hume synapses	50	52.7	[1, 200]
FLOAT	alph_sk	Trace controlling parameter for Skinner synapses	0.1	0.33	[0.01, 1]
FLOAT	alph_pa	Trace controlling parameter for Pavlov synapses	0.1	0.08	[0.01, 1]
FLOAT	alph_hu	Trace controlling parameter for Hume synapses	0.1	0.33	[0.01, 1]
FLOAT	mod2val	Parameter controlling the function describing MOD2 efficacy limit as a function of the number of synapses	7	68.3	[5, 100]
INT	num_cns	Number of neurons in internal clusters	20	26	[10, 50]
FLOAT	b_l_ver	Leg vertical base angle (°)	-20	-43.4	[-70, 0]
FLOAT	b_l_hor	Leg horizontal base angle (°)	0	17.7	[-20, 20]
FLOAT	maxdefv	Maximum vertical angle of deflection (°)	45	49.2	[20, 70]
FLOAT	maxdefh	Maximum horizontal angle of deflection (°)	45	20	[20, 70]
FLOAT	alph_mu	Trace controlling parameter for muscle force trace function	0.01	0.03	[0.001, 0.2]
BOOL	f_hei_n	Scaling factor for torso height need	1	0.95	[0.1, 1]

Table 17: GA output GEN34

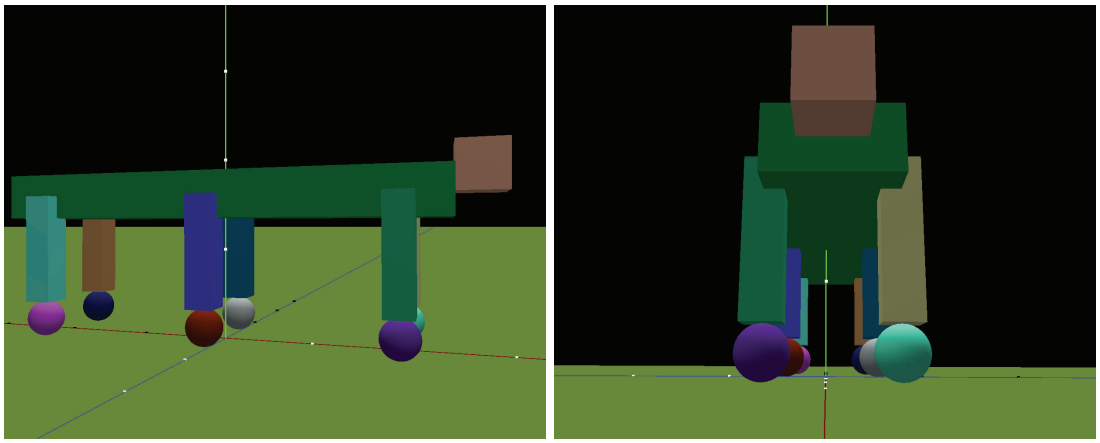


Figure 35: 3D visualization screenshots GEN34

	N	S	A	I1	M
N	X	X	C	X	X
S	X	X	P	P	0
A	X	X	X	S	S
I1	X	X	0	H	P
M	X	X	0	0	0

Table 16: Topology specification GEN34

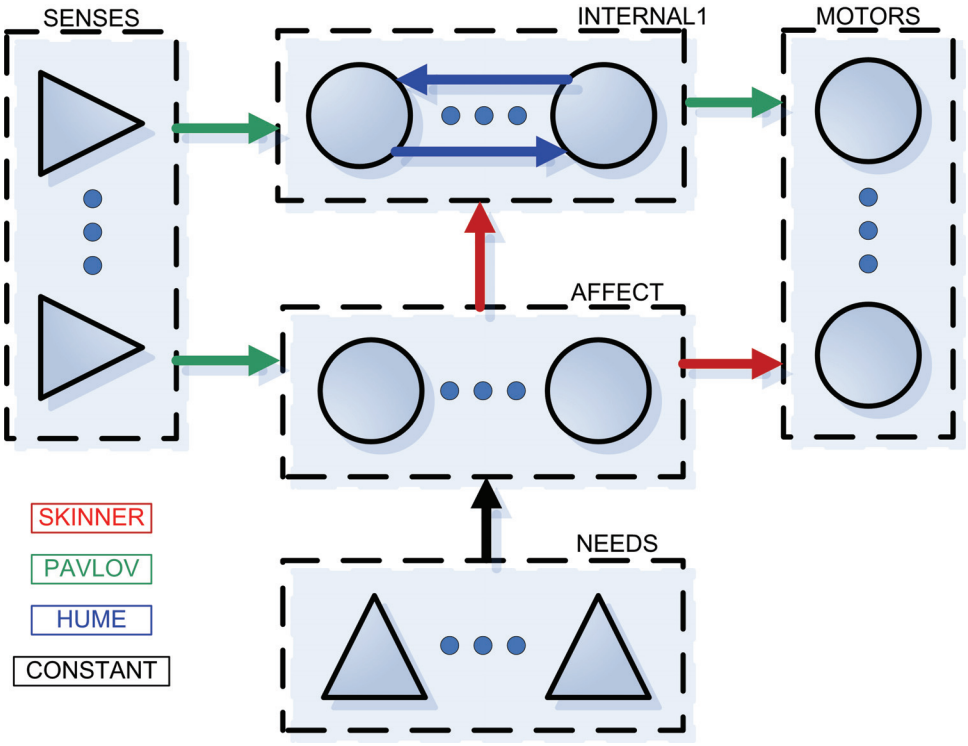


Figure 36: Topology structure GEN34

The changes in GA single value parameters are similar to those seen earlier; most importantly, neuronal model learning rates and mechanical model joint angle specifiers are adjusted to better suit the torso height need. Also, note that for this creature, the `mod2val` controlling the summed synaptic efficacy limit (MOD2) has been increased vastly, in effect virtually disabling the MOD2 divergence preventing learning mechanism modification, and thus allowing very high synaptic efficacies.

When considering the lifetime visualization (Figure 35), the creature from generation 34 exercises what prior to executing this experiment was considered perfect behavior. Early in its lifetime, the creature learns to lift its torso cleanly and stably using the back-and-forth rocking strategy (described above for generation 4), presumably to more easily overcome static frictional forces. Thence, it soon learns to stabilize the weight perfectly, with all legs pointing straight down. For the rest of its lifetime, it stands maximally upright, with virtually no movement.

4.2.3.3 Summary

As indicated by the fitness plots of Figure 26, the torso height need is solved successfully. The creature from generation 34 exercised perfect behavior, a behavior that characterized most of the creature from generation 24 and outwards. A few creatures even exceeded the calculated theoretical optimum fitness value of 18300; the absolute maximum fitness value of 18603 was achieved at generation 18: This creature exercised quite erratic but at the same time quite repetitive behavior that could best be described as jumping.

To further investigate the behavior of the creature from generation 18, identically specified creatures were simulated to see if the behavior seen was reproducible. Repeated simulations showed that results were partly random, in that some creatures were total failures, while others showed behavior that resembled the behavior seen for the creature that achieved the exceedingly high lifetime fitness of 18603. The latter indicates that, for this specific GA configuration, results are heavily dependent on the specific course of random samples for stochastic perturbations.

Table 18 gives an overview of the development in GA parameter values over the generations discussed above.

Name	**Seed value**	**GEN0**	**GEN4**	**GEN9**	**GEN18**	**GEN34**	**Range**
Neuronal model GA parameters							
`std_stc`	1	1	1.7	1.7	1.2	0.6	[0.1, 2]
`in_ntrc`	true	false	false	false	true	true	
`alph_ne`	0.5	0.5	0.5	0.19	0.48	1	[0.01, 1]
`in_dtrc`	false	true	false	false	false	false	
`lr_skin`	2000	2000	2000	3016	1082	2000	[100, 10000]
`lr_pavl`	500	500	500	719	725	1033	[25, 2500]
`lr_hume`	50	50	77.7	75	50	52.7	[1, 200]
`alph_sk`	0.1	0.1	0.24	0.24	0.16	0.33	[0.01, 1]
`alph_pa`	0.1	0.1	0.1	0.42	0.01	0.08	[0.01, 1]
`alph_hu`	0.1	0.1	0.24	0.1	0.24	0.33	[0.01, 1]
`newpavl`	false	false	false	false	true	false	

Continued on next page

Table 18 :: Continued							
Name	**Seed value**	**GEN0**	**GEN4**	**GEN9**	**GEN18**	**GEN34**	**Range**
`newhume`	false	false	false	false	false	false	
`mod2val`	7	7	5	7	19.6	68.3	[5, 100]
`num_cns`	20	20	20	20	26	26	[10, 50]
Mechanical model GA parameters							
`num_lps`	3	3	3	3	3	3	{2, 3, 4}
`b_l_ver`	-20	-20	-42.8	-42.8	-44.1	-43.4	[-70, 0]
`b_l_hor`	0	0	20	20	16.8	17.7	[-20, 20]
`maxdefv`	45	45	48.6	48.6	49.2	49.2	[20, 70]
`maxdefh`	45	36.4	36.4	20	20	20	[20, 70]
`alph_mu`	0.01	0.04	0.01	0.02	0.02	0.03	[0.001, 0.2]
`in_beln`	false	false	true	true	false	false	
`f_hei_n`	1	1	1	1	0.79	0.95	[0.1, 1]
`f_bel_n`	1	1	1	1	1	1	[0.1, 1]
`ex_angs`	1	1	1	1	1	1	[1, 2]
`ex_aves`	1	1	1	1	1	1	[1, 2]
`in_angs`	false	false	false	false	true	false	
`in_aves`	false	false	false	true	false	false	
`f_ang_s`	1	1	1	1	0.86	1	[0.1, 1]
`f_ave_s`	0.1	0.1	0.1	0.29	0.1	0.1	[0.01, 1]

Table 18: GA parameter development

Immediately emanating from the above is the fact that all of the best creatures had three leg pairs, i.e. six legs. Although creatures with four and eight legs have been evaluated, it is evident that six legs consistently proves to be the best configuration for solving the task at hand.

The most prominent and consistent changes are at the specifiers for joint angle ranges. Not unexpectedly, the maximum horizontal angle of deflection is minimized to 20°. Also, the increases of both vertical base angle and vertical maximum angle of deflection were expected.

The initial choice of letting the inclusion of belly pain sensor needs be optional seems to have been appropriate; some of the best creatures have utilized these needs, while others have not. For the last two creatures discussed (generation 18 and 34), the belly pain sensor needs are disabled.

Also evident from Table 18 is the fact that muscle length and muscle length rate senses seldom are included for the fittest creatures. This is as expected; the torso height need is probably adequate for learning the behavior of raising and stabilizing the torso. For later experiments incorporating more complex goal specifications, such as forward velocity, the inclusion of neutralized senses is expected to be necessary for appropriate and behaviorally efficient learning to take place.

4.3 Experiment 2: Head height

This section describes the system configuration and simulation results constituting Experiment 2: Head height. This experiment is a slightly modified variant of Experiment 1: Torso height; here, the goal is based on head height instead of torso height. The reason for including this experiment is that the head height goal of this experiment to a greater extent than the torso height goal discussed earlier requires the ANN to control leg pairs differently; to achieve maximum head height, the creature must learn to position its torso such that the front is pointing upward.

4.3.1 Goal specification, needs and fitness function

As indicated above, the system goal of this experiment is based on head height: The creature is rewarded, both in terms of need values and in terms of fitness evaluation, for keeping its head as high above ground as possible. The actual value used in these need and fitness calculations is based on the height of the creature's head, as follows:[59]

$$heightvalue = y,$$

where y is the world coordinate vertical height component of the head position, scaled to range within $[0, 1]$. As opposed to the experiment on torso height, as described in Section 4.2, the calculations on head height do not incorporate averaging of different points on the creature's head; the goal is simply for the creature to hold its head as high up as possible, regardless of head orientation.

For this experiment, the configuration of needs is as follows:

Need	Configuration	Included
Torso height need	CONST	No
Head height need	CONST	Yes
Belly pain sensor needs	GA	Optional
Velocity need	CONST	No
Extreme joint angle needs	CONST	No
Muscle force needs	CONST	No
Foot friction needs	CONST	No

As indicated, the most important need for this experiment is the head height need (as partially stated above and described in detail in Section 3.3.3.2), which wholly and fully incorporates the system goal of achieving maximum head height. In a similar manner to what was explained for the previous experiment (Experiment 1: Torso height, Section 4.2), the other needs are less important, or conflict the goal specification. The torso height need conflicts the goal of achieving maximum head height, because it favors a horizontally balanced torso; it is therefore excluded. The belly pain sensor needs may be helpful in the initial phase of getting the torso up from the ground. On the other hand, it may be necessary to keep some part of the creature's belly in contact with the ground to stably achieve and retain maximum head height. Therefore, because the suitability of belly pain sensor needs for the goal at hand is not immediately clear, the inclusion of this need type is a free parameter in the GA process. The rest of the needs are considered to conflict the main goal, and are therefore excluded from this experiment.[60]

[59] The head height need, as described by Eq. (34) of Section 3.3.3.2, is simply the additive inverse of this value.

[60] The same arguments as those given in Section 4.2 for Experiment 1 apply.

With the above inverted head height value, the fitness function can be specified as follows:

$$fitness^0 = 0 \tag{45}$$

$$fitness^t = fitness^{t-1} + \frac{t}{N} \cdot heightvalue, \tag{46}$$

This fitness evaluation function should ensure that, in the selection of a small number of individuals for survival and recombination from a large and diverse generation, those individuals whose parameter settings are propitious for achieving and retaining maximum head height are chosen. To summarize, the desired simulation result is that the creature synaptically learns the behavior of appropriately positioning its body (torso and legs), lifting its head high above ground, and stably, and in a balanced fashion, maintaining this body position.

4.3.2 Parameters and settings

This section specifies the initial setting of all variable parameters for Experiment 2, i.e. the *input* to the GA process defining both the size of the GA parameter space and the seed of the GA search. Table 19 lists the parameters together with their initial (seed) values and value ranges. The settings are equivalent to those of the previous experiment.

Value type	**Name**	**Description**	**Seed value**	**Range**
Neuronal model GA parameters				
FLOAT	`std_stc`	Standard deviation parameter for Cauchy probability distribution used in stochastic perturbations	1	[0.1, 2]
BOOL	`in_ntrc`	Inclusion of neuronal trace element in neuronal activation function	true	
FLOAT	`alph_ne`	Trace controlling parameter for neuronal trace function	0.5	[0.01, 1]
BOOL	`in_dtrc`	Inclusion of trace at neuronal drive values	false	
FLOAT	`lr_skin`	Learning rate for Skinner synapses	2000	[100, 10000]
FLOAT	`lr_pavl`	Learning rate for Pavlov synapses	500	[25, 2500]
FLOAT	`lr_hume`	Learning rate for Hume synapses	50	[1, 200]
FLOAT	`alph_sk`	Trace controlling parameter for Skinner synapses	0.1	[0.01, 1]
FLOAT	`alph_pa`	Trace controlling parameter for Pavlov synapses	0.1	[0.01, 1]
FLOAT	`alph_hu`	Trace controlling parameter for Hume synapses	0.1	[0.01, 1]
BOOL	`newpavl`	Use of modified Pavlov synaptic learning mechanism	false	
BOOL	`newhume`	Use of modified Hume synaptic learning mechanism	false	

Continued on next page

Table 19 :: Continued				
Value type	**Name**	**Description**	**Seed value**	**Range**
FLOAT	`mod2val`	Parameter controlling the function describing MOD2 efficacy limit as a function of the number of synapses	7	[5, 100]
INT	`num_cns`	Number of neurons in internal clusters	20	[10, 50]
		Mechanical model GA parameters		
INT	`num_lps`	Number of leg pairs	3	{2, 3, 4}
FLOAT	`b_l_ver`	Leg vertical base angle (°)	-20	[-70, 0]
FLOAT	`b_l_hor`	Leg horizontal base angle (°)	0	[-20, 20]
FLOAT	`maxdefv`	Maximum vertical angle of deflection (°)	45	[20, 70]
FLOAT	`maxdefh`	Maximum horizontal angle of deflection (°)	45	[20, 70]
FLOAT	`alph_mu`	Trace controlling parameter for muscle force trace function	0.01	[0.001, 0.2]
BOOL	`in_beln`	Inclusion of belly pain sensor needs	false	
BOOL	`f_hed_n`	Scaling factor for head height need	1	[0.1, 1]
BOOL	`f_bel_n`	Scaling factor for belly pain sensor needs	1	[0.1, 1]
FLOAT	`ex_angs`	Exponent in functions describing muscle length senses	1	[1, 2]
FLOAT	`ex_aves`	Exponent in functions describing muscle length rate senses	1	[1, 2]
BOOL	`in_angs`	Inclusion of muscle length senses	false	
BOOL	`in_aves`	Inclusion of muscle length rate senses	false	
BOOL	`f_ang_s`	Scaling factor for muscle length senses	1	[0.1, 1]
BOOL	`f_ave_s`	Scaling factor for muscle length rate senses	0.1	[0.01, 1]

Table 19: Experiment 2 - GA parameter settings

4.3.3 Experiment results

This section presents a representative selection of the results obtained from Experiment 2: Head height. The GA process was run with the standard settings described in Section 3.4: mutation rate $P = 10\%$, mutation amount $A = 20\%$, and 50 individuals per generation. Each creature was simulated for 50000 iterations, which translates into a real-time lifetime of 8.33 minutes. The simulations were stopped after 30 generations, implying that a total of 1500 virtual creatures were simulated.

4.3.3.1 Results overview

Figure 37 shows the progress in fitness over the first 55 generations (numbered 0-54), depicting the best, average, and worst fitness for each generation. The last seven generations showed no further development in fitness or behavior, and the corresponding fitness data are consequently not shown in the chart.

For a reference as regards head height fitness values, it can be noted that when the creature lies flat on the ground, the instantaneous fitness value is 0.125. Thus, over a lifetime of 50000 iterations, and with an average time-weighting of 0.5 (see Section 3.4.1.2), the lifetime fitness

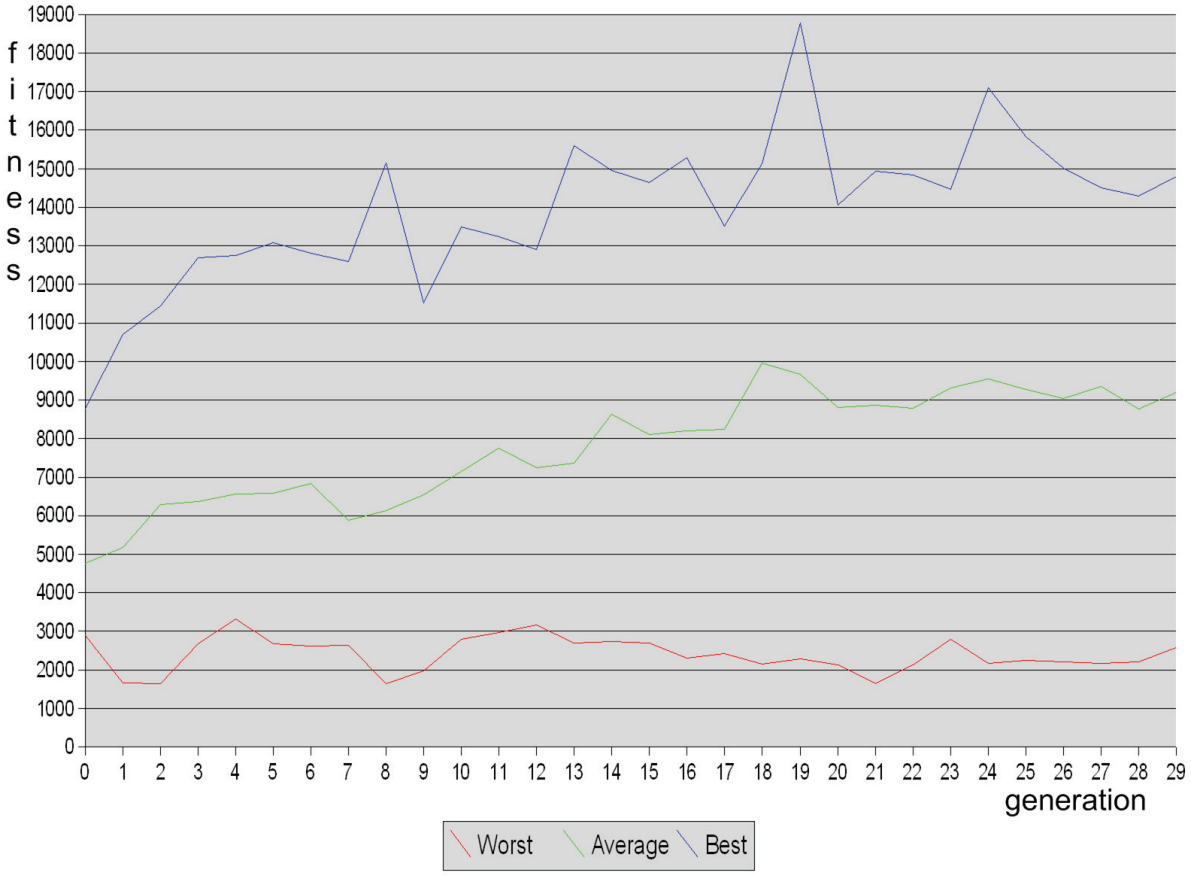


Figure 37: Fitness progress for Experiment 2: Head height

would be 3125. Similarly, when the creature stands maximally upright with its torso perfectly balanced horizontally (equivalent to what was regarded as "perfect" behavior for Experiment 1: Head height), the lifetime fitness value is 11563. Still, the creature should be capable of achieving even greater lifetime fitness values than this, by positioning its torso such that the front, where the head is attached, is pointing upward.[61]

The graph for best fitness shows that the head height goal is, indeed, successfully solved. There is a consistent improvement over the first six generations, with best fitness increasing from 8786 at generation 0 to 13085 at generation 5. After this point, the progress of best fitness becomes severely less deterministic. Although the graph is prominently jagged, there is a positive tendency over the next 10-15 generations. The absolute maximum fitness for the experiment is achieved at generation 19, where the fitness reaches 18797, clearly indicating that the head height goal is solved successfully.

The graph for average fitness shows a similar tendency, although the progress is somewhat less prominent. Finally, in a similar manner to the previous experiment, the progress in values for worst fitness is non-existent; the values are concentrated around 2000-3000, and never

[61] Calculating a theoretical optimum on lifetime fitness for this body positioning, similar to what was done for Experiment 1: Head height, would be guesswork, and is not attempted here.

consistently improve above this.

4.3.3.2 Analysis of specific creatures - GA output and qualitative descriptions

This section presents the lifetime visualizations (i.e. observable behavior) of the best creatures from a few selected generations. The selection has been made with the purpose of demonstrating the main developmental steps seen in the progress toward successful goal achievement. Corresponding lifetime fitness values are shown in the headings below.

Generation 0: 8786

The best creature from generation 0 achieved a lifetime fitness of 8786. The ANN topology of this creature, shown in Table 20, is identical to the default topology.

The GA single parameter output is shown in Table 21, where only those parameters whose values have changed compared to the initial GA input specification (Table 19) are listed.

Value type	Name	Description	Seed value	Value	Range
FLOAT	`b_l_ver`	Leg vertical base angle (°)	-20	-45.1	[-70, 0]
FLOAT	`maxdefv`	Maximum vertical angle of deflection (°)	45	50.8	[20, 70]

Table 21: GA output GEN0

As was seen for the best creature of the first generation from Experiment 1: Torso height, the changes compared to the initial seed are far from comprehensive, and probably not decisive as regards the level of goal achievement. Worth noting, however, is that the vertical joint angle deflection range has been modified; both the base angle has been lowered and the maximum deflection has been increased, effectively making the creature able to keep its legs straighter downward.

The lifetime visualization (Figure 38) shows that the creature initially lifts the back of its torso using its hindmost legs. This is in a way the opposite of what would be expected; by lifting the front of the torso instead, the head height need value would be considerably lower. Anyhow, the creature soon tries to raise the front of its torso too, thereby causing the head to elevate considerably higher. Throughout its lifetime, by trial and error, the creature gradually gets better at stabilizing its torso. Late in its lifetime, from time to time, the creature manages to stabilize its body such that the front of the torso and the head is pointing upwards. The creature does, however, never settle in this favorable position; it recurringly loses balance and falls or sinks toward the ground.

Generation 4: 12753

The best creature from generation 4 achieved a lifetime fitness of 12753. The ANN topology of this creature is considerably altered compared to the initial topology: Three internal clusters have been introduced, bringing along ten new clustral connections. The resulting topology specification is shown in Table 22.

The GA single parameter output is shown in Table 23, where only those parameters whose values have changed compared to the initial GA input specification (Table 19) are listed.

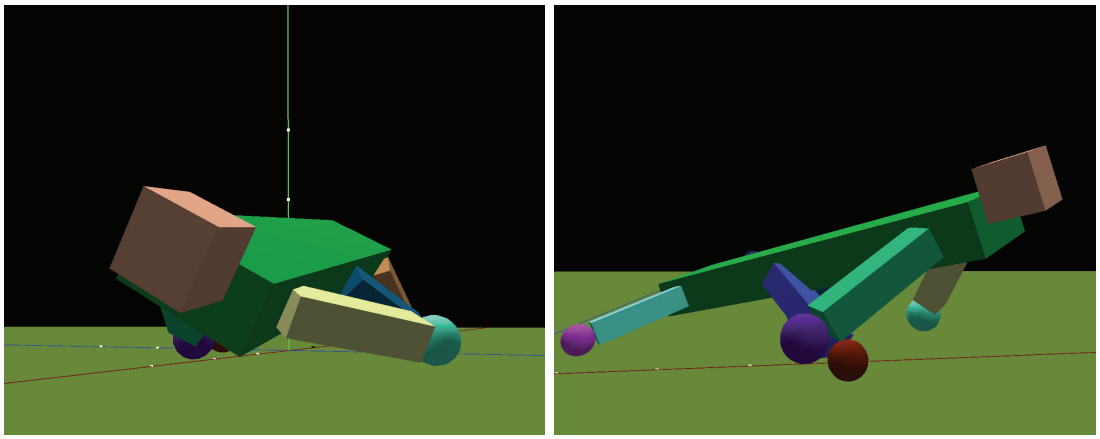


Figure 38: 3D visualization screenshots GEN0

	N	S	A	M
N	X	X	C	X
S	X	X	P	0
A	X	X	X	S
M	X	X	0	0

Table 20: Topology specification GEN0

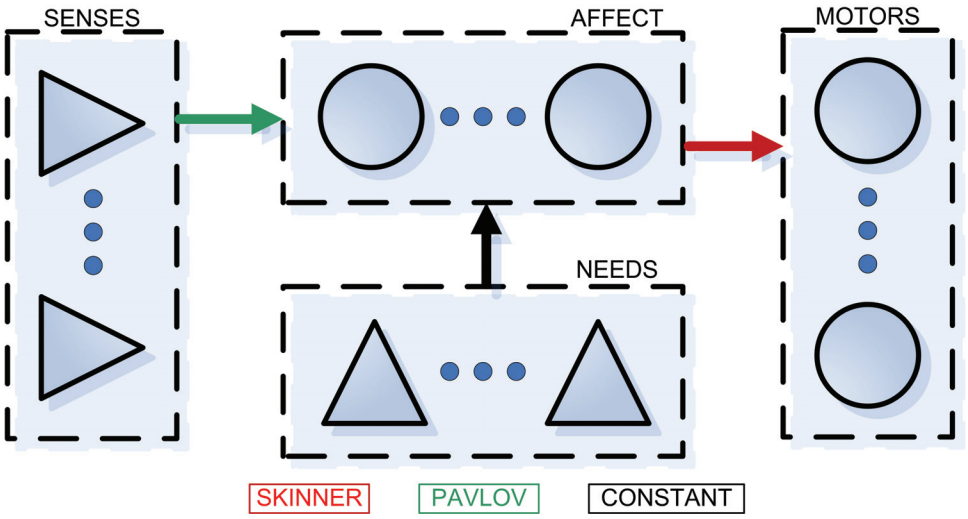


Figure 39: Topology structure GEN0

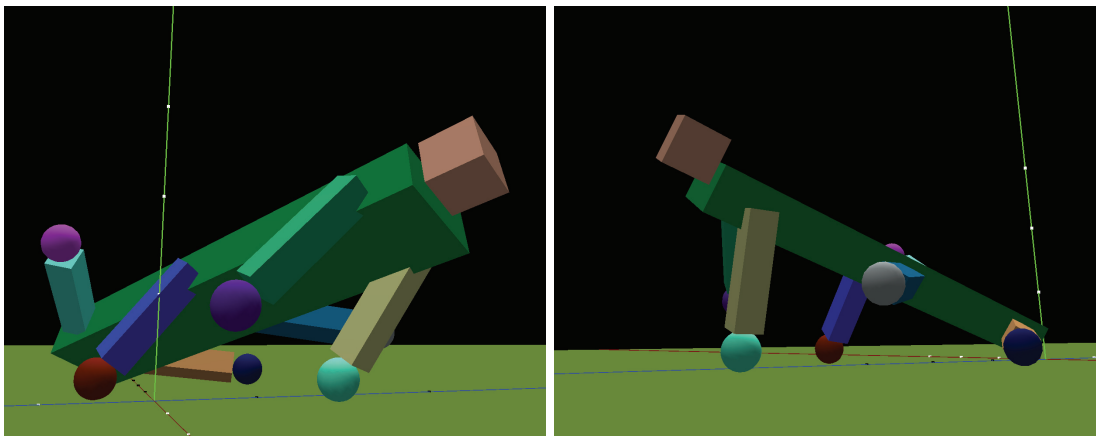


Figure 40: 3D visualization screenshots GEN4

	N	S	A	I1	I2	I3	M
N	X	X	C	X	X	X	X
S	X	X	P	P	0	0	0
A	X	X	X	S	0	0	S
I1	X	X	0	H	P	0	0
I2	X	X	0	P	0	0	P
I3	X	X	0	0	P	H	0
M	X	X	0	P	0	P	0

Table 22: Topology specification GEN4

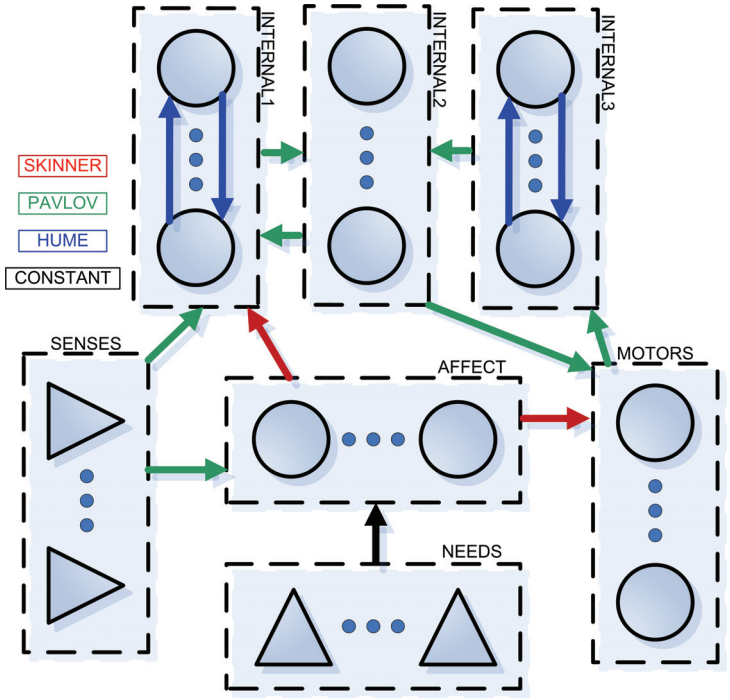


Figure 41: Topology structure GEN4

Value type	Name	Description	Seed value	Value	Range
FLOAT	`alph_ne`	Trace controlling parameter for neuronal trace function	0.5	0.57	[0.01, 1]
BOOL	`in_dtrc`	Inclusion of trace at neuronal drive values	false	true	
FLOAT	`lr_skin`	Learning rate for Skinner synapses	2000	1057	[100, 10000]
BOOL	`newpavl`	Use of modified Pavlov synaptic learning mechanism	false	true	
BOOL	`newhume`	Use of modified Hume synaptic learning mechanism	false	true	
FLOAT	`b_l_ver`	Leg vertical base angle (°)	-20	-34.6	[-70, 0]
FLOAT	`maxdefv`	Maximum vertical angle of deflection (°)	45	45.4	[20, 70]
BOOL	`in_beln`	Inclusion of belly pain sensor needs	false	true	

Table 23: GA output GEN4

The most prominent changes compared to the initial seed are that the learning rate for Skinner synapses has been halved and that the modified version of the Pavlov and Hume synaptic learning rates are used. For the mechanical model, note that the belly pain sensor needs are included.

The lifetime visualization (Figure 40) shows that the creature almost instantly raises the front of its torso, making the head elevate considerably above ground. Throughout the rest of its lifetime, however, there is not very much progress. The only improvement seen is that the front right leg, which provides the main support for the torso weight, is additionally straightened, thereby optimizing the current body position with respect to head height. Anyhow, because the body position obtained initially is favorable with respect to head height, and because the creature learns to maintain this position stably throughout its lifetime, the achieved fitness is considerably ($\approx 45\%$) higher than for the best creature of generation 0.

Generation 19: 18779

The best creature from generation 19 achieved a lifetime fitness of 18779, which was the maximum fitness obtained for the entire experiment. The ANN topology of this creature is slightly altered compared to topology shown for generation 4: The three internal clusters are retained, but there have been some changes in connectivity. The topology specification is shown in Table 22.

The GA single parameter output is shown in Table 25, where only those parameters whose values have changed compared to the initial GA input specification (Table 19) are listed.

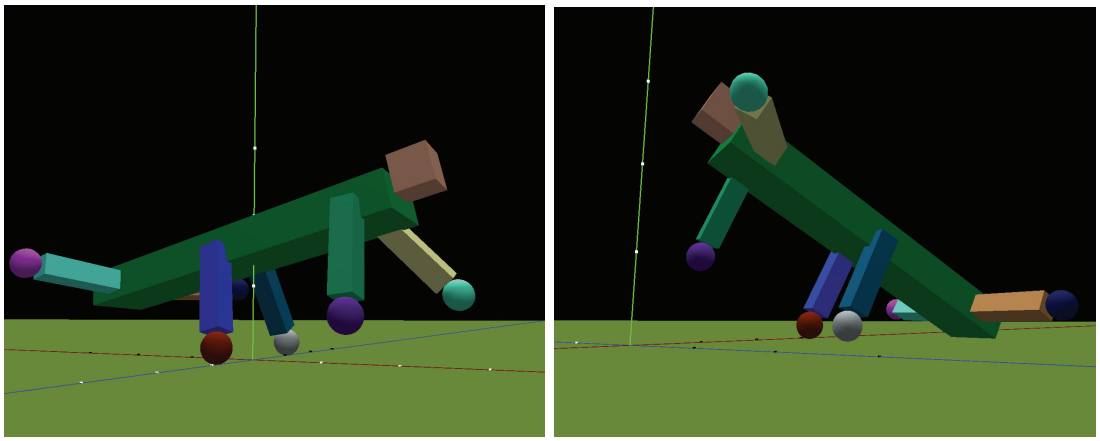


Figure 42: 3D visualization screenshots GEN19

	N	S	A	I1	I2	I3	M
N	X	X	C	X	X	X	X
S	X	X	P	0	P	0	P
A	X	X	X	S	0	S	0
I1	X	X	0	0	0	P	P
I2	X	X	P	P	H	0	P
I3	X	X	0	P	P	H	0
M	X	X	P	P	0	0	P

Table 24: Topology specification GEN19

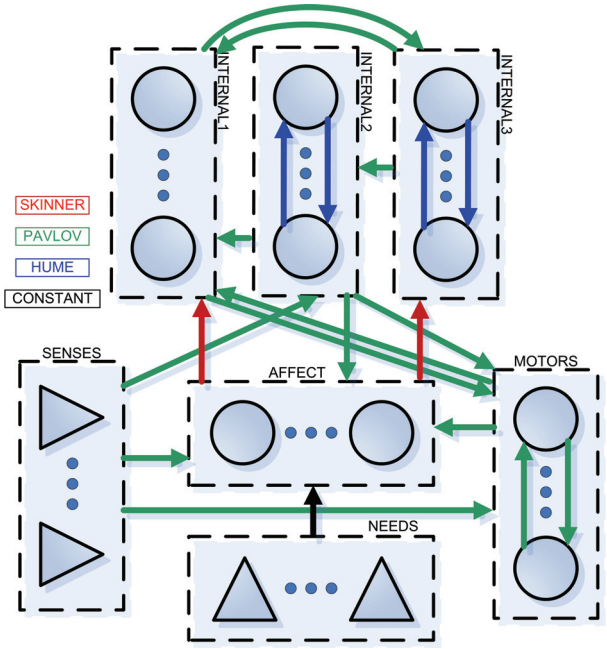


Figure 43: Topology structure GEN19

Value type	Name	Description	Seed value	Value	Range
FLOAT	`std_stc`	Standard deviation parameter for Cauchy probability distribution used in stochastic perturbations	1	1.40	[0.1, 2]
BOOL	`in_ntrc`	Inclusion of neuronal trace element in neuronal activation function	true	false	
FLOAT	`alph_ne`	Trace controlling parameter for neuronal trace function	0.5	0.21	[0.01, 1]
FLOAT	`lr_skin`	Learning rate for Skinner synapses	2000	100	[100, 10000]
FLOAT	`lr_hume`	Learning rate for Hume synapses	50	85.4	[1, 200]
FLOAT	`alph_sk`	Trace controlling parameter for Skinner synapses	0.1	0.28	[0.01, 1]
FLOAT	`alph_hu`	Trace controlling parameter for Hume synapses	0.1	0.12	[0.01, 1]
BOOL	`newpavl`	Use of modified Pavlov synaptic learning mechanism	false	true	
BOOL	`newhume`	Use of modified Hume synaptic learning mechanism	false	true	
FLOAT	`mod2val`	Parameter controlling the function describing MOD2 efficacy limit as a function of the number of synapses	7	9.9	[5, 100]
INT	`num_cns`	Number of neurons in internal clusters	20	10	[10, 50]
FLOAT	`b_l_ver`	Leg vertical base angle (°)	-20	-41.4	[-70, 0]
FLOAT	`b_l_hor`	Leg horizontal base angle (°)	0	13.5	[-20, 20]
FLOAT	`maxdefv`	Maximum vertical angle of deflection (°)	45	47.2	[20, 70]
FLOAT	`maxdefh`	Maximum horizontal angle of deflection (°)	45	20	[20, 70]
FLOAT	`alph_mu`	Trace controlling parameter for muscle force trace function	0.01	0.04	[0.001, 0.2]
FLOAT	`ex_aves`	Exponent in functions describing muscle length rate senses	1	1.15	[1, 2]
BOOL	`in_aves`	Inclusion of muscle length rate senses	false	true	
BOOL	`f_ave_s`	Scaling factor for muscle length rate senses	0.1	0.03	[0.01, 1]

Table 25: GA output GEN19

The most prominent changes compared to the initial seed are that the learning rate for

Skinner synapses has been minimized and that the modified version of the Pavlov and Hume synaptic learning mechanisms are still used. Also, the length rate senses have been included, and their intensity has been somewhat reduced.

The lifetime visualization (Figure 42) shows that the creature starts by synchronously using its two middle legs to elevate the torso. For a short time interval, these two legs are the only support for the torso weight. Soon, therefore, the torso falls backward while the middle legs remain straightened, causing the head to elevate high above ground. This body position is stabilized and optimized such that, for the rest of the creature's lifetime, the head remains prominently elevated above ground. Further, the position obtained seems to be very close to the optimal position; there is seemingly little room for improvements as regards head height.

Interestingly, there seems to be synchronized movement at the two hindmost legs; these are dangling freely in the air and not touching the ground, but they exercise seemingly deterministic and synchronized movement patterns that affect the balance of the torso. It is not known whether or not it is this muscular activity at the hindmost leg pair that keeps the torso from falling forward,[62] but the determinism and synchronization in the movement of the two hindmost legs seem apparent.

4.3.3.3 Summary

As indicated by the fitness plots of Figure 37, although performance as measured by best fitness is convergent to a lesser degree than what was seen for the Experiment 1, the head height need is solved successfully. The best creature from generation 19 achieved a lifetime fitness of 18779, which presumably is very close to the maximum of what is possible.

Table 26 gives an overview of the development in GA parameter values over the few generations discussed above.

Name	Seed value	GEN0	GEN4	GEN19	Range
Neuronal model GA parameters					
std_stc	1	1	1	1.40	[0.1, 2]
in_ntrc	true	true	true	false	
alph_ne	0.5	0.5	0.57	0.21	[0.01, 1]
in_dtrc	false	false	true	false	
lr_skin	2000	2000	1057	100	[100, 10000]
lr_pavl	500	500	500	500	[25, 2500]
lr_hume	50	50	50	85.4	[1, 200]
alph_sk	0.1	0.1	0.1	0.28	[0.01, 1]
alph_pa	0.1	0.1	0.1	0.12	[0.01, 1]
alph_hu	0.1	0.1	0.1	0.1	[0.01, 1]
newpavl	false	false	true	true	
newhume	false	false	true	true	
mod2val	7	7	7	9.9	[5, 100]
num_cns	20	20	20	10	[10, 50]

Continued on next page

[62] The head, which is connected to the front of the torso, causes the creature to be nose-heavy.

Table 26 :: Continued					
Name	**Seed value**	**GEN0**	**GEN4**	**GEN19**	**Range**
Mechanical model GA parameters					
`num_lps`	3	3	3	3	{2, 3, 4}
`b_l_ver`	-20	-45.1	-34.6	-41.4	[-70, 0]
`b_l_hor`	0	0	0	13.5	[-20, 20]
`maxdefv`	45	50.8	45.4	47.2	[20, 70]
`maxdefh`	45	45	45	20	[20, 70]
`alph_mu`	0.01	0.01	0.01	0.04	[0.001, 0.2]
`in_beln`	false	false	true	false	
`f_hed_n`	1	1	1	1	[0.1, 1]
`f_bel_n`	1	1	1	1	[0.1, 1]
`ex_angs`	1	1	1	1	[1, 2]
`ex_aves`	1	1	1	1.15	[1, 2]
`in_angs`	false	false	false	false	
`in_aves`	false	false	false	true	
`f_ang_s`	1	1	1	1	[0.1, 1]
`f_ave_s`	0.1	0.1	0.1	0.03	[0.01, 1]

Table 26: GA parameter development

The most prominent change made to the neuronal model is that the learning rate for Skinner synapses has been minimized. It is not clear why such an extreme reduction in learning rate has provided favorable results. The increase in `alph_sk` cancels some of the decrease of effective learning rate, but still the Skinner effective learning rate seems remarkably low. Further worth noting is the use of the modified Pavlov and Hume learning mechanisms; both of the two last creatures, which were the best ones examined, learn by these modified mechanisms.

As for the mechanical model, similarly to the previous experiment, all the best creatures examined had six legs. For the head height goal, this would be expected: with six legs, the back of the torso can rest at the ground while the middle leg pair is supporting the torso weight, thus allowing the head to elevate high above ground. Creatures with four or eight legs cannot as easily or efficiently maintain such body positioning stably. Apart from this, vertical and horizontal base angles and maximum angles of deflection have been adjusted to better suit the head height goal.

4.4 Experiment 3: Forward velocity

This section describes the system configuration and simulation results constituting Experiment 3: Forward velocity, which incorporates a system goal of achieving maximum forward velocity. For this task to be solved successfully, the creatures will have to develop (partly) synchronized and oscillatory walking patterns - gaits. As described in the introduction to this thesis, the main experimental goal of this thesis is to investigate the possibilities for such gait-like movement patterns to develop, i.e. to look for tendencies of repetitive, synchronized and oscillatory motor control. The two previous experiments, Experiment 1: Torso height and Experiment 2: Head height, have established the validity of the model and simulation system as a whole, and demonstrated that such ANN-controlled creatures are capable of solving simple mechanically originate problems. The principal theoretical and experimental interest, however, lies with the current forward velocity goal and the desired development of dynamic gait-like movement patterns.

4.4.1 Goal specification, needs and fitness function

As indicated above, the system goal of this experiment is based on forward velocity: The creature is rewarded, both in terms of need values and in terms of fitness evaluation, when the creature is moving forward relative to its own orientation. The value used in the fitness calculations is based on the relative forward velocity of the torso, as follows:[63]

$$velocityvalue = \nu,$$

where ν is the velocity in creature relative forward direction.

For this experiment, the configuration of needs is as follows:

Need	Configuration	Included
Torso height need	GA	Optional
Head height need	CONST	No
Belly pain sensor needs	GA	Optional
Velocity need	CONST	Yes
Extreme joint angle needs	GA	Optional
Muscle force needs	GA	Optional
Foot friction needs	GA	Optional

As indicated, the most important need for this experiment is the velocity need (as stated above and described in detail in Section 3.3.3.4), which wholly and fully incorporates the system goal of achieving maximum forward velocity. The head height need is excluded, as it is thought to provide nothing of value with respect to the system goal. As regards the other needs, the appropriateness is uncertain: The torso height need may help in keeping the torso elevated, the belly pain sensor needs may be propitious for getting the torso up from the ground, the extreme joint angle needs may assist in the development of oscillatory leg movements, muscle force needs may help in favorizing energy efficient movement patterns,[64] and the foot friction needs may contribute in learning to lift the feet instead of sliding them along the ground.

[63] Qualitatively, the velocity need, as described by Eq. (35) of Section 3.3.3.4, is equivalent to the additive inverse of this value.

[64] This need would be useful in the fine-tuning of gait-like movement patterns, should such patterns develop.

With the above velocity value, the fitness function can be specified as follows:

$$fitness^0 = 0 \tag{47}$$

$$fitness^t = fitness^{t-1} + \frac{t}{N} \cdot velocityvalue, \tag{48}$$

This fitness evaluation function should ensure that, in the selection of a small number of individuals for survival and recombination from a large and diverse generation, those individuals whose parameter settings are propitious for achieving forward velocity are chosen. To summarize, the desired simulation result is that the creature develops gait-like movement patterns such that it moves forward in a repetitive and (partly) synchronized fashion.

4.4.2 Parameters and settings

This section specifies the initial setting of all variable parameters for Experiment 3, i.e. the *input* to the GA process defining both the size of the GA parameter space and the seed of the GA search. Table 27 lists the parameters together with their initial (seed) values and value ranges.

Value type	**Name**	**Description**	**Seed value**	**Range**
Neuronal model GA parameters				
FLOAT	`std_stc`	Standard deviation parameter for Cauchy probability distribution used in stochastic perturbations	1	$[0.1, 2]$
BOOL	`in_ntrc`	Inclusion of neuronal trace element in neuronal activation function	true	
FLOAT	`alph_ne`	Trace controlling parameter for neuronal trace function	0.5	[0.01, 1]
BOOL	`in_dtrc`	Inclusion of trace at neuronal drive values	false	
FLOAT	`lr_skin`	Learning rate for Skinner synapses	200	[1, 1000]
FLOAT	`lr_pavl`	Learning rate for Pavlov synapses	20	[1, 100]
FLOAT	`lr_hume`	Learning rate for Hume synapses	20	[1, 100]
FLOAT	`alph_sk`	Trace controlling parameter for Skinner synapses	0.1	[0.01, 1]
FLOAT	`alph_pa`	Trace controlling parameter for Pavlov synapses	0.1	[0.01, 1]
FLOAT	`alph_hu`	Trace controlling parameter for Hume synapses	0.1	[0.01, 1]
BOOL	`newpavl`	Use of modified Pavlov synaptic learning mechanism	false	
BOOL	`newhume`	Use of modified Hume synaptic learning mechanism	false	
FLOAT	`mod2val`	Parameter controlling the function describing MOD2 efficacy limit as a function of the number of synapses	7	[5, 100]

Continued on next page

Table 27 :: Continued				
Value type	**Name**	**Description**	**Seed value**	**Range**
INT	num_cns	Number of neurons in internal clusters	20	[10, 50]
Mechanical model GA parameters				
INT	num_lps	Number of leg pairs	3	{2, 3, 4}
FLOAT	b_l_ver	Leg vertical base angle (°)	-20	[-70, 0]
FLOAT	b_l_hor	Leg horizontal base angle (°)	0	[-20, 20]
FLOAT	maxdefv	Maximum vertical angle of deflection (°)	45	[20, 70]
FLOAT	maxdefh	Maximum horizontal angle of deflection (°)	45	[20, 70]
FLOAT	alph_mu	Trace controlling parameter for muscle force trace function	0.01	[0.001, 0.2]
BOOL	in_hein	Inclusion of torso height need	false	
BOOL	in_beln	Inclusion of belly pain sensor needs	false	
BOOL	in_angn	Inclusion of extreme joint angle needs	true	
BOOL	in_mufn	Inclusion of muscle force needs	false	
BOOL	in_lfrn	Inclusion of foot friction need	false	
BOOL	f_hei_n	Scaling factor for torso height need	1	[0.1, 1]
BOOL	f_bel_n	Scaling factor for belly pain sensor needs	0.1	[0.01, 1]
BOOL	f_vel_n	Scaling factor for velocity need	1	[0.1, 1]
BOOL	f_ang_n	Scaling factor for extreme joint angle needs	0.1	[0.01, 1]
BOOL	f_muf_n	Scaling factor for muscle force needs	1	[0.1, 1]
BOOL	f_lfr_n	Scaling factor for foot friction need	0.2	[0.1, 1]
FLOAT	ex_angs	Exponent in functions describing muscle length senses	1	[1, 2]
FLOAT	ex_aves	Exponent in functions describing muscle length rate senses	1.0	[1, 2]
BOOL	in_angs	Inclusion of muscle length senses	true	
BOOL	in_aves	Inclusion of muscle length rate senses	false	
BOOL	f_ang_s	Scaling factor for muscle length senses	1	[0.1, 1]
BOOL	f_ave_s	Scaling factor for muscle length rate senses	0.1	[0.01, 1]

Table 27: Experiment 3 - GA parameter settings

The seed values for the scaling factors have been set by trial and error: The effective growth in synaptic efficacies has been monitored in real-time simulations where different combinations of needs and senses are enabled, and scaling factor values have been adjusted to the level where rate of synaptic growth seems sensible.

4.4.3 Experiment results

This section presents a representative selection of the results obtained from Experiment 3: Forward velocity. The GA process was run with the standard settings described in Section 3.4: mutation rate $P = 10\%$, mutation amount $A = 20\%$, and 50 individuals per generation. Each creature was simulated for 50000 iterations, which translates into a real-time lifetime of 8.33 minutes. The simulations were stopped after 46 generations, implying that a total of 2300

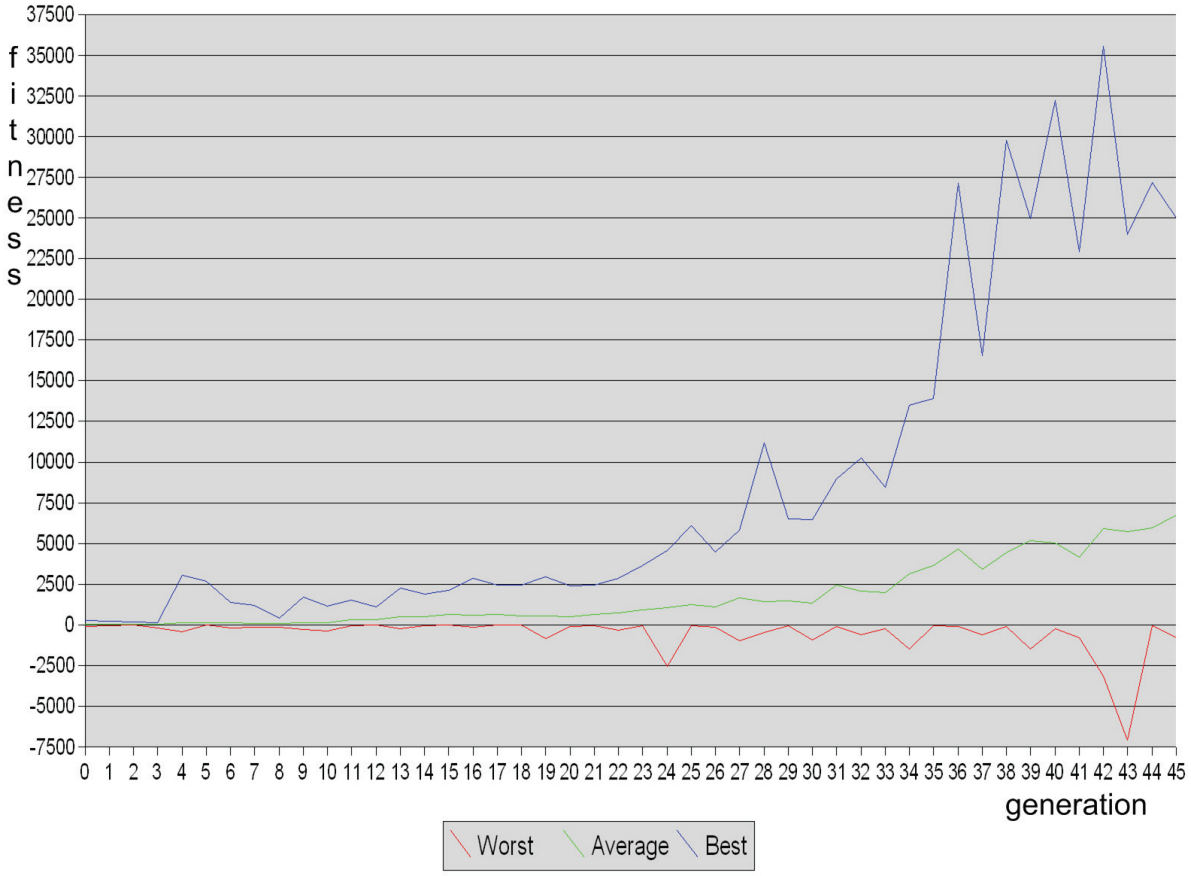


Figure 44: Fitness progress for Experiment 3: Forward velocity

virtual creatures were simulated.

4.4.3.1 Results overview

Figure 44 shows the progress in fitness over the 46 generations simulated (numbered 0-45), depicting the best, average, and worst fitness for each generation.

For a reference as regards forward velocity fitness values, note that if the torso is not moving in the horizontal plane (e.g. when lying absolutely still on the ground), the instantaneous fitness is 0. When the torso velocity is non-zero, if the forward component of the velocity vector in creature relative forward direction is positive, the instantaneous fitness is positive. Correspondingly, if this velocity component is negative, the instantaneous fitness is negative. Thus, a creature that, at average, is moving forward (backward) will obtain a positive (negative) lifetime fitness. Estimating the size that may be expected on lifetime fitness values is not at all trivial, but the following can be noted: If the creature learns a gait-like movement pattern such that the creature maintains an average forward velocity of 1.0 m/s (which is well within reach, should gaits develop), the lifetime fitness would be 25000.

As the chart clearly depicts, fitness values of such magnitudes do in fact occur. The maximum fitness of 35532, achieved at generation 42, is actually considerably higher than the above

approximate value expected for reasonably efficient gaits. When inspecting the lifetime visualizations of these fit creatures, however, the following behavior is seen: The creatures position all legs pointing backward and start moving them slightly back and forth. Leg deflections are very slight and leg movement soon becomes very fast, in effect making the legs *vibratory*. Such behavior is both biologically unrealistic and of very limited interest in the search for signs of gait development.

4.4.3.2 Analysis of specific creatures - GA output and qualitative descriptions

As described above, the behavior developed for the fittest creatures is far from what was hoped for. Although only achieving a fitness value one third of the absolute best fitness, the most promising behavior as regards the development of gaits was seen at generation 28. Consequently, this is the only creature examined in detail.

Generation 28: 11166

The best creature from generation 28 achieved a lifetime fitness of 11166. The ANN topology used is shown in Table 28.

The GA single parameter output is shown in Table 29, where only those parameters whose values have changed compared to the initial GA input specification (Table 27) are listed.

Value type	Name	Description	Seed value	Value	Range
FLOAT	`std_stc`	Standard deviation parameter for Cauchy probability distribution used in stochastic perturbations	1	1.40	[0.1, 2]
BOOL	`in_ntrc`	Inclusion of neuronal trace element in neuronal activation function	true	false	
FLOAT	`alph_ne`	Trace controlling parameter for neuronal trace function	0.5	0.39	[0.01, 1]
FLOAT	`lr_skin`	Learning rate for Skinner synapses	200	1	[1, 1000]
FLOAT	`lr_pavl`	Learning rate for Pavlov synapses	20	65	[1, 100]
FLOAT	`lr_hume`	Learning rate for Hume synapses	20	37	[1, 100]
FLOAT	`alph_pa`	Trace controlling parameter for Pavlov synapses	0.1	0.13	[0.01, 1]
FLOAT	`alph_hu`	Trace controlling parameter for Hume synapses	0.1	0.11	[0.01, 1]
BOOL	`newpavl`	Use of modified Pavlov synaptic learning mechanism	false	true	
BOOL	`newhume`	Use of modified Hume synaptic learning mechanism	false	true	

Continued on next page

Table 29 :: Continued					
Value type	**Name**	**Description**	**Seed value**	**Value**	**Range**
FLOAT	`mod2val`	Parameter controlling the function describing MOD2 efficacy limit as a function of the number of synapses	7	18.1	[5, 100]
FLOAT	`b_l_ver`	Leg vertical base angle (°)	-20	-35.3	[-70, 0]
FLOAT	`maxdefv`	Maximum vertical angle of deflection (°)	45	39.2	[20, 70]
FLOAT	`maxdefh`	Maximum horizontal angle of deflection (°)	45	43.8	[20, 70]
FLOAT	`alph_mu`	Trace controlling parameter for muscle force trace function	0.01	0.11	[0.001, 0.2]
BOOL	`in_beln`	Inclusion of belly pain sensor needs	false	true	
BOOL	`in_lfrn`	Inclusion of foot friction need	false	true	
BOOL	`f_bel_n`	Scaling factor for belly pain sensor needs	0.1	0.1	[0.01, 1]
BOOL	`f_lfr_n`	Scaling factor for foot friction need	0.2	0.42	[0.1, 1]
BOOL	`in_angs`	Inclusion of muscle length senses	true	true	
BOOL	`f_ang_s`	Scaling factor for muscle length senses	1	0.73	[0.1, 1]

Table 29: GA output GEN28

The most prominent change to the set of GA parameters is that the Skinner learning rate has been minimized to the value 1. Such a low Skinner learning rate implies that needs will have very little influence on behavior. This is unexpected, because need values are what steer behavior according to the system goal at hand. It might be that the small influence from need values is sufficient to produce forward movement. On the other hand, with such a low Skinner learning rate, the fact that the creature ends up moving forward may also be purely incidental.

The lifetime visualization (Figure 45) shows the following behavior: After an initial phase of chaotic and random behavior (approximately one fifth of the creature's lifetime), the creature positions all its legs such that they point slightly backward. Shortly thereafter, it starts moving some of its legs back/up and forth/down, at relatively high speed and in a quite fast-paced manner. The leg movements are energetic; the feet consistently hit the ground with high velocity, causing the creature to bump slightly forward. The creature retains this behavior throughout the rest of its lifetime.

Although the movement pattern developed to no considerable extent resembles the gaits seen in biological systems, there seems to be some synchronization going on between the different legs performing such movements: The two hindmost legs consistently hit the ground simultaneously, and are thus synchronized. Further, the front left leg and the middle right leg are synchronized in the same fashion, they move and hit the ground simultaneously. Finally, there is anti-synchronization between the internally synchronized pairs of legs just described; the front-

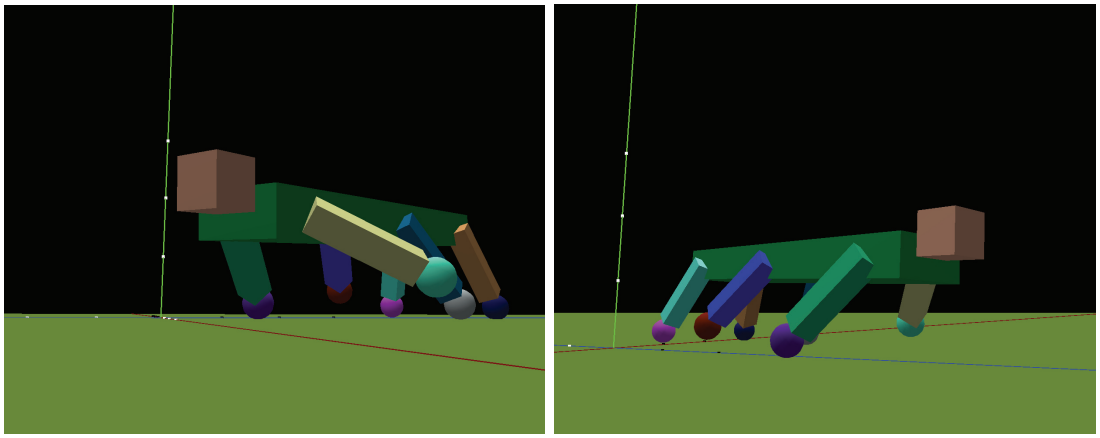


Figure 45: 3D visualization screenshots GEN28

	N	S	A	I1	I2	M
N	X	X	C	X	X	X
S	X	X	0	0	P	P
A	X	X	X	S	S	S
I1	X	X	0	0	0	P
I2	X	X	0	0	0	P
M	X	X	0	0	0	0

Table 28: Topology specification GEN28

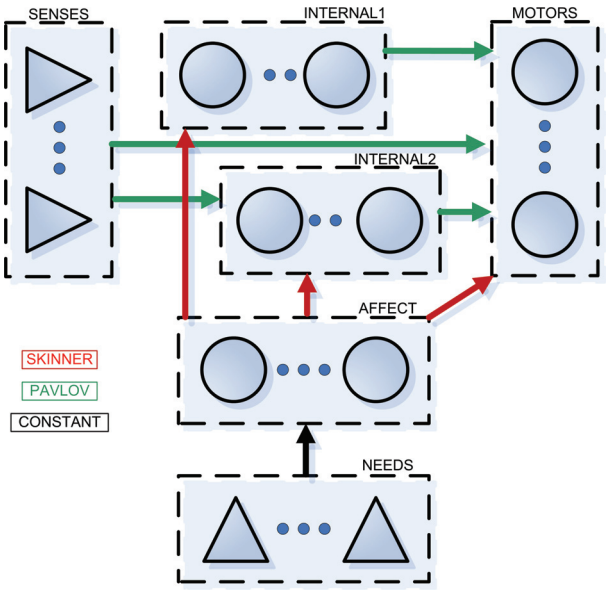


Figure 46: Topology structure GEN28

middle leg pair consistently and invariably hits the ground just before the hindmost leg pair does. This latter temporally shifted cross-synchronization is probably what is causing most of the propulsion; forward drive would presumably have been significantly lower had, for instance, all four legs been hitting the ground simultaneously.

4.4.3.3 Summary

To summarize the results from Experiment 3: Forward velocity, we can conclude that the system as a whole finds a rather good solution to the task at hand. The fittest creatures consistently achieve reasonably high forward velocities. These creatures do, however, seem to be taking advantage of a weakness in the mechanical model; legs are allowed to move forth and back in a vibratory manner. The latter is, of course, not biologically realistic, and thus far from what was hoped for.

In relation to the GA search process, the creatures exercising behavior characterized by vibratory leg movements may be regarded as being positioned in or near a local maximum in the GA search space. Further, it seems that the search is incapable of escaping this presumably suboptimal local maximum once it is discovered. To avoid such suboptimal convergence in the next experiments, therefore, it will be necessary to make some changes in the mechanical model. More specifically, vibratory leg movements should be disallowed.

Although no biologically resemblant gaits are seen, at generation 28 a movement pattern is discovered that is reasonably efficient for moving forward. Also, the evident synchronization is interesting, providing light indications that ANNs based on the mechanisms of Connectology may self-organize into synchronized movement patterns, and eventually into gaits. The exceedingly low value seen for the Skinner learning rate for this creature, however, questions the determinism in the development of this movement pattern; with such a low learning rate, the fact that the pattern causes the creature to move forward may be incidental.

Finally, therefore, to investigate the degree of determinism in the movement patterns developed, repeated simulations have been performed with creatures that are configured identically to the creature from generation 28. Results show that similar patterns do, in fact, develop quite deterministically in these creatures; all creatures investigated learnt partly synchronized and repetitive patterns of movement similar to that described earlier for the creature from generation 28. The degree of goal achievement as regards forward velocity, however, is fluctuating; the efficiency of the patterns developed as regards forward movement is highly variable.

4.5 Reassessment of model and simulation settings

The results from the previous experiment on forward velocity showed only few signs of synchronized movement patterns. The resulting behavior was far from what was hoped for; it appears that the problem was solved mainly by the GA process instead of by ANN synaptic learning during a creature's lifetime. Based on this, as a second attempt for gaits to develop, the configuration of the simulation system has been reassessed.

4.5.1 Neuronal model

Based on discussions at supervisor meetings, J. Hokland has suggested two changes to the model for synaptic learning. The original model of learning mechanisms was based on the following generic structure:

$$\Delta e_{ij}^t = \psi \beta_{ij} f_i(\Delta D_i^t) f_j(\Delta D_j^t), \tag{49}$$

where f_i and f_j represent the identity function or a synaptic trace of the presynaptic and postsynaptic delta drives, respectively.

The first change introduces a new element of nonlinearity by performing a nonlinear transformation on delta drives before including them as multiplicative contributions to the learning mechanisms:

$$\Delta e_{ij}^t = \psi \beta_{ij} f_i\left(sgn(\Delta D_i^t) \cdot \left|\Delta D_i^t\right|^k\right) \cdot f_j\left(sgn(\Delta D_j^t) \cdot \left|\Delta D_j^t\right|^k\right) \mid k \geq 1 \in \mathbb{Z}, \tag{50}$$

where $sgn(x)$ is the sign function. The effect of the above modification is that, when $k > 1$, small drive changes are suppressed and large drive changes are favorized. The purpose is to increase the impact that *significant events* have on learning compared to insignificant events. Thus, the larger k, the more prominent the favorizing of large delta drives. E.g. for the Skinner mechanism:

$$\Delta e_{ij}^t = -\beta_{ij} \min\left(sgn(\Delta D_i^t) \cdot \left|\Delta D_i^t\right|^k, 0\right) T_{ij}^t \tag{51}$$

$$T_{ij}^t = (1 - \alpha_{ij}) T_{ij}^{t-1} + \alpha_{ij} \cdot sgn(\Delta D_j^t) \cdot \left|\Delta D_j^t\right|^k \tag{52}$$

The corresponding changes in the Pavlov and Hume mechanisms are completely analogous, and consequently not shown here.

A second modification introduces a qualitative change to the synaptic trace functions. A new multiplicative element is included in the trace, as follows (for the Skinner synaptic trace):[65]

$$T_{ij}^t = (1 - \alpha_{ij}) T_{ij}^{t-1} + \alpha_{ij} \left(1 - \left|T_{ij}^{t-1}\right|\right) \Delta D_j^t, \tag{53}$$

The effect of this modification is that the trace of significant changes becomes more persistent: If the trace value at some point has become large, let us assume that it is close to 1 or -1, then the $(1 - |T_{ij}^{t-1}|)$ element will cause subsequent delta drives to have significantly lower influence on the trace value, i.e. it becomes more persistent. If the trace is close to zero, however, the new element will cancel out (approach one), thus allowing future delta drives to influence the trace value normally.

[65] An alternative formulation is $T_{ij}^t = (1 - \alpha_{ij}) \left|T_{ij}^{t-1}\right| T_{ij}^{t-1} + \alpha_{ij} \left(1 - \left|T_{ij}^{t-1}\right|\right) \Delta D_j^t$, i.e. such that the equation is symmetric in $\left|T_{ij}^{t-1}\right|$. Because the magnitude of trace values is small for the simulations of this thesis, however, $\left|T_{ij}^{t-1}\right|$ is left out for the first term of the equation, avoiding potential problems when $\left|T_{ij}^{t-1}\right|$ approaches zero.

4.5.2 Topology specification

Some of the creature instances described earlier had quite complex topologies with up to six internal clusters and many clustral connections. For the experiments performed henceforth, greater restrictions are put on ANN topology, meaning that the GA topology search is heavily constrained. Two prominent arguments support this choice: 1) Such large topologies are presumably not needed for sensible gaits to develop,[66] and 2) allowing such large topologies potentially puts heavy strain on computer simulations, causing running time to increase vastly.[67]

For the last two experiments, therefore, only two types of topologies are simulated. The topology specification matrices are shown in Tables 30-31, and the corresponding topology structures are shown in Figures 47-48. These two topology variants define two different search spaces for GA topology searches. Variant 1 has two optional connections, and thereby allows $2^2 = 4$ specific topologies, which is the space of allowed topologies for Experiment 4: Forward velocity v2. Variant 2 has six optional connections, and thereby allows $2^6 = 64$ specific topologies, which is the space of allowed topologies for Experiment 5: Forward velocity v3. The number of clusters is constant for both variants, meaning that the GA mutation process (Section 3.4.1.5) may no longer add or remove clusters.

4.5.3 GA parameter configuration

Section 4.5.1 discussed two optional modifications to the neuronal model that are included in the last two experiments. For the simulations performed henceforth, the use of these two modifications is determined by two additional GA parameters. The first modification, which specifies delta drive transformations, is described by the floating-point parameter k, where $k = 1$ is equivalent to excluding the modification. The second modification, which slightly modifies the synaptic trace function, is simply described by a boolean parameter denoting whether or not the new trace element is included in the corresponding equation. The two parameters are termed `ddexpon` and `mod_trc`, respectively:

Value type	**Name**	**Description**	**Seed value**	**Range**
FLOAT	`ddexpon`	Exponent in function transforming synaptic learning mechanism delta drives	1	$[1, 3]$
BOOL	`mod_trc`	Use of modified synaptic trace function with new persistence increasing element	false	

Both of the neuronal model modifications discussed above, and particularly the first, may require synaptic learning rates to be higher. Based on this, the maximum limits have been expanded for all three learning rate variables. Further, to restrict the GA process from suppressing one mechanism any other way than by topological modifications, the corresponding minimum limits have been increased:

[66] J. A. Scott Kelso provides several examples of small neuron-like systems capable of producing self-oscillating patterns [Kel95].

[67] Recall from Section 3.2.7 that the running time of ANN updates is linear in the number of neurons and synapses.

	N	S	A	M
N	X	X	C	X
S	X	X	(P)	P
A	X	X	X	S
M	X	X	X	(H)

Table 30: Constrained topology specification 1

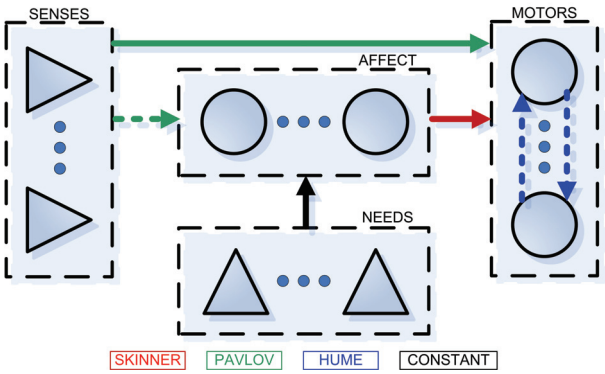


Figure 47: Constrained topology structure 1 (dotted connections are optional)

	N	S	A	I1	M
N	X	X	C	X	X
S	X	X	(P)	P	(P)
A	X	X	X	S	(S)
I1	X	X	X	(H)	P
M	X	X	X	(P)	(H)

Table 31: Constrained topology specification 2

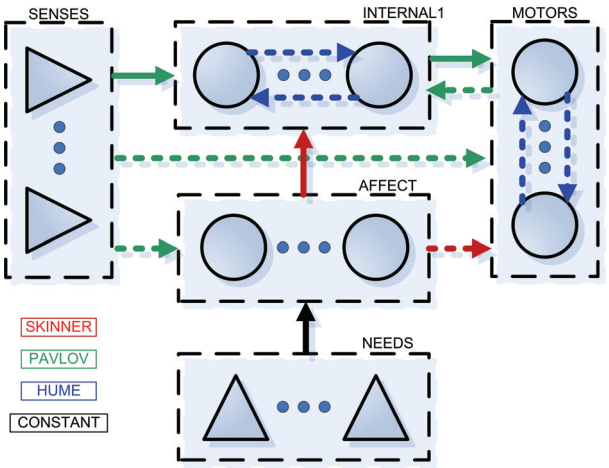


Figure 48: Constrained topology structure 2 (dotted connections are optional)

Value type	Name	Description	Old range	New range
FLOAT	`lr_skin`	Learning rate for Skinner synapses	[1, 1000]	[50, 10000]
FLOAT	`lr_pavl`	Learning rate for Pavlov synapses	[1, 100]	[5, 1000]
FLOAT	`lr_hume`	Learning rate for Hume synapses	[1, 100]	[5, 1000]

The experience gained from Experiment 3 indicates that the mechanical model needs some modifications; the vibratory leg movements characterizing the fittest creatures from Experiment 3 should not be allowed. Finally, therefore, to reduce the possibility for such movement patterns to emerge, the following changes have been made:

Value type	Name	Description	Old range	New Range
FLOAT	`maxdefv`	Maximum vertical angle of deflection (°)	[20, 70]	[45, 70]
FLOAT	`maxdefh`	Maximum horizontal angle of deflection (°)	[20, 70]	[45, 70]
FLOAT	`alph_mu`	Trace controlling parameter for muscle force trace function	[0.001, 0.2]	[0.001, 0.05]

The decrease in the maximum limit for `alph_mu` considerably lowers the maximum muscle rapidness, effectively implying that the maximum frequency with which muscle force may change from high to low is decreased. This should ensure that creatures not as easily will be able to perform vibratory leg movements by muscular control. The increase in minimum limits for both the vertical and horizontal maximum angles of deflection further ensure that creatures not as rapidly can bounce their legs back and forth off joint angle limits.

4.5.4 Outline

For the experiments performed henceforth, greater restrictions are put on ANN topologies: Only two types of topologies are used, one having no internal clusters, and the other having a single internal cluster. The GA process may no longer introduce new clusters, but can enable or disable selected clustral connections.

The two final experiments of this thesis are presented next. These are identically specified, except for ANN topologies, as follows:.

- Experiment 4: Forward velocity v2 uses topology variant 1 (Table 30, Figure 47)
- Experiment 5: Forward velocity v3 uses topology variant 2 (Table 31, Figure 48)

For both of the above, the seed topology has none of the optional connections activated. Further, the two neuronal model modifications are included as GA parameters, and GA parameter seeds are updated according to the parameter range modifications described above. Finally, for the last two experiments, the number of iterations has been reduced from 50000 to 15000, implying that the creatures' lifetime has been reduced from 8.33 to 2.50 minutes. This decision has been made based on discussions at supervisor meetings, and is justified as follows: For the simple topologies used henceforth, if a system configuration (ANN topology and GA parameter values) is discovered that allows some gait to develop, this gait is expected to emerge quickly, and well within the 15000 iterations simulated. The purpose of lowering the iteration count, of course, is to allow more creatures to be simulated within a reasonable time-interval, thereby arranging for a larger portion of the space of allowable system configurations to be explored.

4.6 Experiment 4: Forward velocity v2

This section describes the system configuration and simulation results constituting Experiment 4: Forward velocity v2. The goal specification, need configuration and fitness function are equal to those described for Experiment 3, and are consequently not discussed here.

4.6.1 Parameters and settings

This section specifies the initial setting of all variable parameters for Experiment 4, i.e. the *input* to the GA process defining both the size of the GA parameter space and the seed of the GA search. The settings are identical to those presented for Experiment 3, except for the modifications discussed in Section 4.5. Table 32 lists the parameters together with their initial (seed) values and value ranges. Changes compared to the input configuration of the previous experiment on forward velocity are shown in red.

Value type	Name	Description	Seed value	Range
Neuronal model GA parameters				
FLOAT	`std_stc`	Standard deviation parameter for Cauchy probability distribution used in stochastic perturbations	1	$[0.1, 2]$
BOOL	`in_ntrc`	Inclusion of neuronal trace element in neuronal activation function	true	
FLOAT	`alph_ne`	Trace controlling parameter for neuronal trace function	0.5	[0.01, 1]
BOOL	`in_dtrc`	Inclusion of trace at neuronal drive values	false	
FLOAT	`lr_skin`	Learning rate for Skinner synapses	200	[50, 10000]
FLOAT	`lr_pavl`	Learning rate for Pavlov synapses	20	[5, 1000]
FLOAT	`lr_hume`	Learning rate for Hume synapses	20	[5, 1000]
FLOAT	`alph_sk`	Trace controlling parameter for Skinner synapses	0.1	[0.01, 1]
FLOAT	`alph_pa`	Trace controlling parameter for Pavlov synapses	0.1	[0.01, 1]
FLOAT	`alph_hu`	Trace controlling parameter for Hume synapses	0.1	[0.01, 1]
FLOAT	`ddexpon`	Exponent in function transforming synaptic learning mechanism delta drives	1	$[1, 3]$
BOOL	`newpavl`	Use of modified Pavlov synaptic learning mechanism	false	
BOOL	`newhume`	Use of modified Hume synaptic learning mechanism	false	
BOOL	`mod_trc`	Use of modified synaptic trace function with new persistence increasing element	false	
FLOAT	`mod2val`	Parameter controlling the function describing MOD2 efficacy limit as a function of the number of synapses	7	[5, 100]

Continued on next page

Table 32 :: Continued				
Value type	**Name**	**Description**	**Seed value**	**Range**
Mechanical model GA parameters				
INT	`num_lps`	Number of leg pairs	3	{2, 3, 4}
FLOAT	`b_l_ver`	Leg vertical base angle (°)	-20	[-70, 0]
FLOAT	`b_l_hor`	Leg horizontal base angle (°)	0	[-20, 20]
FLOAT	`maxdefv`	Maximum vertical angle of deflection (°)	45	[45, 70]
FLOAT	`maxdefh`	Maximum horizontal angle of deflection (°)	45	[45, 70]
FLOAT	`alph_mu`	Trace controlling parameter for muscle force trace function	0.01	[0.001, 0.05]
BOOL	`in_hein`	Inclusion of torso height need	false	
BOOL	`in_beln`	Inclusion of belly pain sensor needs	false	
BOOL	`in_angn`	Inclusion of extreme joint angle needs	true	
BOOL	`in_mufn`	Inclusion of muscle force needs	false	
BOOL	`in_lfrn`	Inclusion of foot friction need	false	
BOOL	`f_hei_n`	Scaling factor for torso height need	1	[0.1, 1]
BOOL	`f_bel_n`	Scaling factor for belly pain sensor needs	0.1	[0.01, 1]
BOOL	`f_vel_n`	Scaling factor for velocity need	1	[0.1, 1]
BOOL	`f_ang_n`	Scaling factor for extreme joint angle needs	0.1	[0.01, 1]
BOOL	`f_muf_n`	Scaling factor for muscle force needs	1	[0.1, 1]
BOOL	`f_lfr_n`	Scaling factor for foot friction need	0.2	[0.1, 1]
FLOAT	`ex_angs`	Exponent in functions describing muscle length senses	1	[1, 2]
FLOAT	`ex_aves`	Exponent in functions describing muscle length rate senses	1.0	[1, 2]
BOOL	`in_angs`	Inclusion of muscle length senses	true	
BOOL	`in_aves`	Inclusion of muscle length rate senses	false	
BOOL	`f_ang_s`	Scaling factor for muscle length senses	1	[0.1, 1]
BOOL	`f_ave_s`	Scaling factor for muscle length rate senses	0.1	[0.01, 1]

Table 32: Experiment 4 - GA parameter settings

4.6.2 Experiment results

This section presents a representative selection of the results obtained from Experiment 4: Forward velocity v2. The GA process was run with the standard settings described in Section 3.4: mutation rate $P = 10\%$, mutation amount $A = 20\%$, and 50 individuals per generation. Each creature was simulated for 15000 iterations, which translates into a real-time lifetime of 2.50 minutes. The simulations were stopped after 202 generations, implying that a total of 10100 virtual creatures were simulated.

4.6.2.1 Results overview

Figure 49 shows the progress in fitness over the first 150 generations simulated (numbered 0-149), depicting the best, average, and worst fitness for each generation. Since the creatures

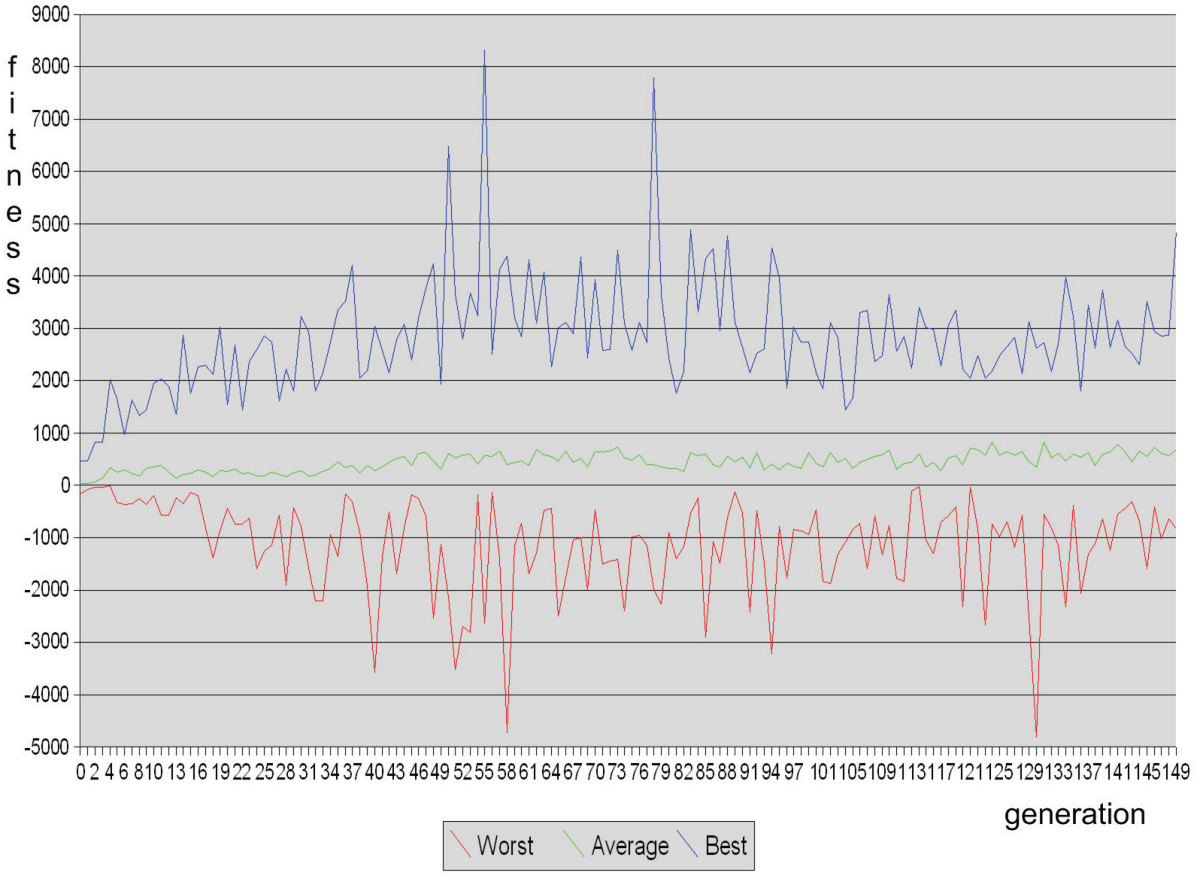


Figure 49: Fitness progress for Experiment 4: Forward velocity v2

of this experiments only live for 15000 generations (as compared to previous lifetime of 50000 generations), expectations on fitness values must be adjusted correspondingly.

The graph for best fitness shows that there is a quite consistently positive tendency in fitness values over the first third of the simulation; at generation 0 the best fitness value is 469, whereas at generation 48 the best fitness value has increased to 4233. In the middle third of the generations, there are three prominent peaks in the graph for best fitness, indicating that the behavior developed for the best creatures from generations 50, 55 and 78 was particularly favorable. The corresponding fitness values are 6482 (GEN50), 8323 (GEN55) and 7782 (GEN78). For the last third of the simulation performance degrades, and fitness values stabilize around 3000.

As regards average fitness, values increase slightly over the simulation period, indicating that behavior at average is somewhat better than the indifferent behavior of lying completely still on the ground.

The graph for worst fitness is quite similar to the graph for best fitness, only opposite, indicating that the spread in performance among the creatures of a generation is large. Further, as for the best fitness, there are a few prominent negative peaks in the graph for worst fitness. The latter is interesting, as it indicates that some creatures have developed behavior that is opposite of what would be expected according to the needs implemented in the model, i.e. they

are quite good at moving backward.

Worth noting in this respect as regards fitness values is that the time that creatures spend in the initial learning phase, i.e. the time elapsed before creatures discover the movement pattern that characterizes the rest of their lifetimes, may influence the fitness results obtained significantly. Thus, when analyzing fitness values, it is not necessarily the creature that obtained the absolute highest fitness value that developed the most efficient movement pattern.

4.6.2.2 Analysis of specific creatures - GA output and qualitative descriptions

This section presents the lifetime visualizations (i.e. observable behavior) of the best creatures from a few selected generations. The selection has been made with the purpose of showing the best movement patterns developed; in other words, the creatures represented by the best fitness peaks from the chart of Figure 49 are inspected. The creatures examined below do in fact develop gaits; after the initial learning phase, they consistently move forward with a permanent, synchronized and repetitive movement pattern. Screenshots are provided that attempt to show the gaits developed. To really get the idea of how the creatures are moving, however, the lifetime visualizations should be inspected.[68] Corresponding lifetime fitness values are shown in the headings below.

Generation 50: 6482

The best creature from generation 50 achieved a lifetime fitness of 6482. The ANN topology was identical to the seed topology (Figure 47), and is therefore not shown.

The GA single parameter output is shown in Table 33, where only those parameters whose values have changed compared to the initial GA input specification (Table 32) are listed.

Value type	Name	Description	Seed value	Value	Range
FLOAT	`std_stc`	Standard deviation parameter for Cauchy probability distribution used in stochastic perturbations	1	1.74	[0.1, 2]
FLOAT	`alph_ne`	Trace controlling parameter for neuronal trace function	0.5	0.81	[0.01, 1]
BOOL	`in_dtrc`	Inclusion of trace at neuronal drive values	false	true	
FLOAT	`lr_skin`	Learning rate for Skinner synapses	200	50	[50, 10000]
FLOAT	`lr_pavl`	Learning rate for Pavlov synapses	20	109	[5, 1000]
FLOAT	`alph_sk`	Trace controlling parameter for Skinner synapses	0.1	0.19	[0.01, 1]
FLOAT	`alph_pa`	Trace controlling parameter for Pavlov synapses	0.1	0.32	[0.01, 1]

Continued on next page

[68] They should be worth the effort. See Appendix A for instructions.

Table 33 :: Continued					
Value type	**Name**	**Description**	**Seed value**	**Value**	**Range**
BOOL	`newpavl`	Use of modified Pavlov synaptic learning mechanism	false	true	
FLOAT	`mod2val`	Parameter controlling the function describing MOD2 efficacy limit as a function of the number of synapses	7	25.7	[5, 100]
FLOAT	`b_l_ver`	Leg vertical base angle (°)	-20	-30.2	[-70, 0]
FLOAT	`b_l_hor`	Leg horizontal base angle (°)	0	17.6	[-20, 20]
FLOAT	`maxdefv`	Maximum vertical angle of deflection (°)	45	47.9	[45, 70]
FLOAT	`maxdefh`	Maximum horizontal angle of deflection (°)	45	48.9	[45, 70]
FLOAT	`alph_mu`	Trace-controlling parameter for muscle trace function	0.01	0.05	[0.001, 0.05]
BOOL	`in_beln`	Inclusion of belly pain sensor needs	false	true	
BOOL	`in_angn`	Inclusion of extreme joint angle needs	true	false	
BOOL	`f_vel_n`	Scaling factor for velocity need	1	0.49	[0.1, 1]
FLOAT	`ex_angs`	Exponent in functions describing muscle length senses	1	1.30	[1, 2]
BOOL	`f_ang_s`	Scaling factor for muscle length senses	1	0.95	[0.1, 1]

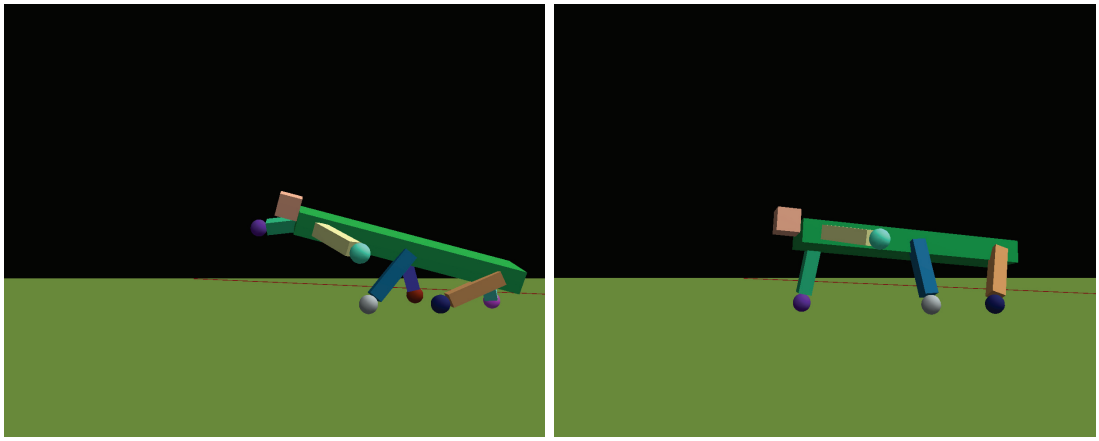
Table 33: GA output GEN50

The most prominent change to the set of GA parameters is the minimization of the Skinner learning rate and the exclusion of the extreme joint angle needs combined with the inclusion of the belly pain sensor needs.

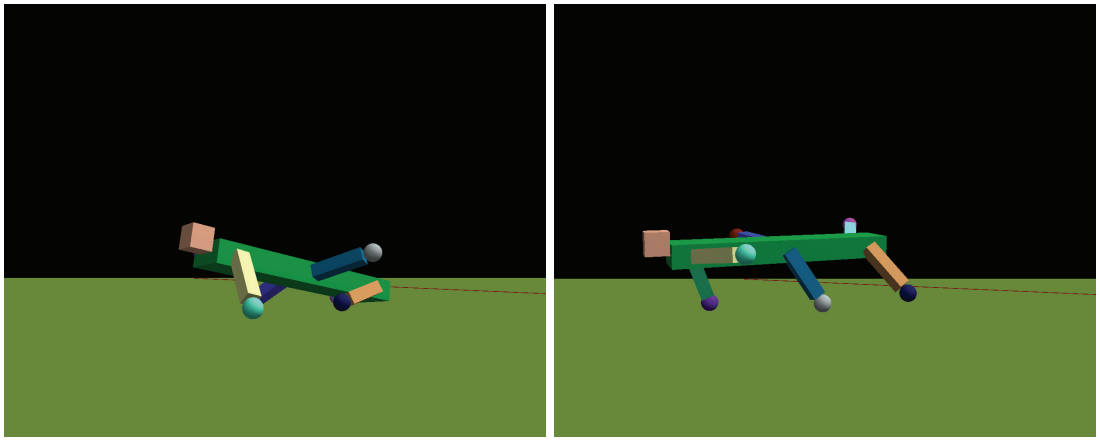
The lifetime visualization (Figure 50) shows that, after exercising chaotic behavior for almost half of its lifetime, approximately at iteration 7000 the creature discovers a movement pattern that it retains throughout the rest of its lifetime.

Initially, this pattern may seem random and chaotic, but by inspection it soon becomes clear that the pattern clearly is repetitive. By further inspection, several aspects of synchronism can be seen: For instance, the right middle leg consistently hits the ground just before the hindmost leg pair does. Further, the two middle legs move very similarly: They are consistently lifted up and forward to subsequently be put down on the ground and dragged backward, thereby causing propulsion. The (anti-)synchronism for the latter is seen in the fact that the right middle leg consistently performs this movement just before the left middle leg.

The gait is not perfectly executed, meaning that the creature from time to time falls out of the repetitive pattern of movement, thus intercepting propulsion. After each such failure, however, the creature almost immediately restarts and continues to move according to the gait it has learnt.



(a) 1 **(b)** 2



(c) 3 **(d)** 4

Figure 50: 3D visualization screenshots GEN50

The gait can be seen to improve a little in the time following its discovery (iteration 7000); based on subjective assessments average forward velocity seems to increase slightly and stably up until convergence at approximately iteration 11000.

Generation 55: 8232

The best creature from generation 55 achieved a lifetime fitness of 8232. The ANN topology was identical to the seed topology (Figure 47), and is therefore not shown.

The GA single parameter output is shown in Table 34, where only those parameters whose values have changed compared to the initial GA input specification (Table 32) are listed.

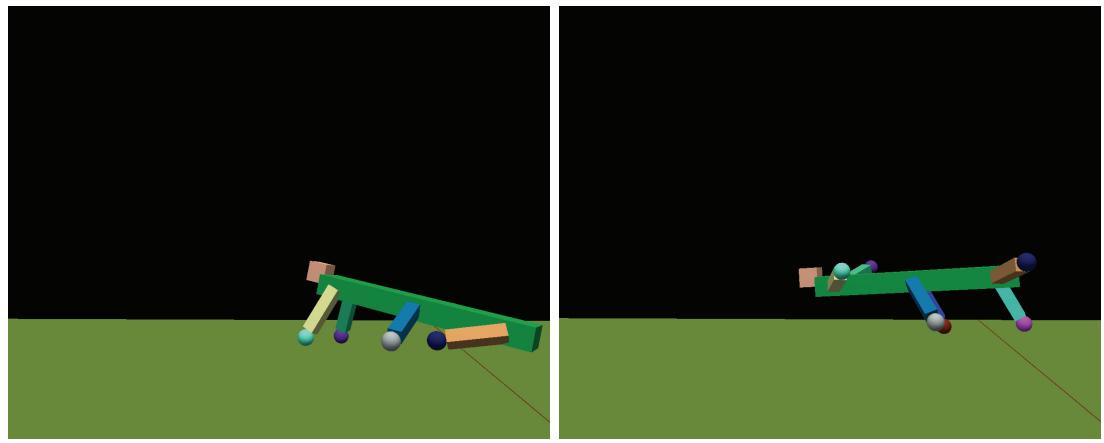
Value type	Name	Description	Seed value	Value	Range
FLOAT	std_stc	Standard deviation parameter for Cauchy probability distribution used in stochastic perturbations	1	1.74	[0.1, 2]
FLOAT	alph_ne	Trace controlling parameter for neuronal trace function	0.5	0.82	[0.01, 1]
BOOL	in_dtrc	Inclusion of trace at neuronal drive values	false	true	
FLOAT	lr_skin	Learning rate for Skinner synapses	200	50	[50, 10000]
FLOAT	lr_pavl	Learning rate for Pavlov synapses	20	109	[5, 1000]
FLOAT	alph_sk	Trace controlling parameter for Skinner synapses	0.1	0.39	[0.01, 1]
FLOAT	alph_pa	Trace controlling parameter for Pavlov synapses	0.1	0.32	0, 1]
BOOL	newpavl	Use of modified Pavlov synaptic learning mechanism	false	true	
BOOL	newhume	Use of modified Hume synaptic learning mechanism	false	true	
FLOAT	mod2val	Parameter controlling the function describing MOD2 efficacy limit as a function of the number of synapses	7	26.4	[5, 100]
FLOAT	b_l_ver	Leg vertical base angle (°)	-20	-30.6	[-70, 0]
FLOAT	b_l_hor	Leg horizontal base angle (°)	0	19.8	[-20, 20]
FLOAT	maxdefv	Maximum vertical angle of deflection (°)	45	47.9	[45, 70]
FLOAT	alph_mu	Trace controlling parameter for muscle force trace function	0.01	0.03	[0.001, 0.05]
BOOL	in_hein	Inclusion of torso height need	false	true	
BOOL	in_beln	Inclusion of belly pain sensor needs	false	true	
BOOL	in_angn	Inclusion of extreme joint angle needs	true	false	
BOOL	f_hei_n	Scaling factor for torso height need	1	0.71	[0.1, 1]
BOOL	f_vel_n	Scaling factor for velocity need	1	0.49	[0.1, 1]
FLOAT	ex_angs	Exponent in functions describing muscle length senses	1	1.30	[1, 2]

Table 34: GA output GEN55

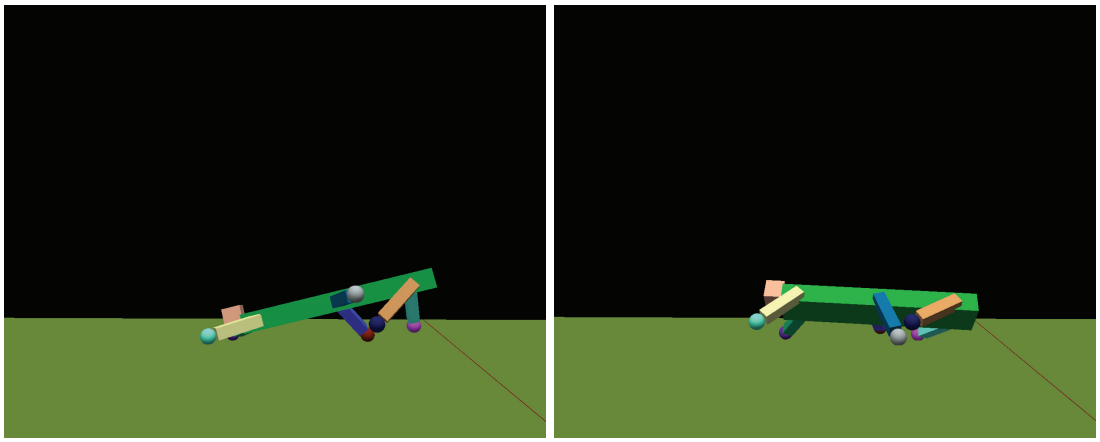
The most prominent change to the set of GA parameters is the minimization of the Skinner

learning rate and the exclusion of the extreme joint angle needs combined with the inclusion of the torso height need and belly pain sensor needs.

The lifetime visualization (Figure 51) shows that, in a completely analogous manner to the creature from generation 50, after exercising chaotic behavior for almost half of its lifetime (approximately at iteration 7000), this creature discovers a movement pattern that it retains throughout the rest of its lifetime.



(a) 1 **(b)** 2



(c) 3 **(d)** 4

Figure 51: 3D visualization screenshots GEN55

Compared to the pattern developed at generation 50, this movement pattern is far cleaner and less chaotic. The gait is based on repetitive jumps mainly caused by simultaneous thrusts at the two hindmost legs. Further, the front legs are used in a similar manner to produce jumps, and are further raised and thrown forward while airborne such that the creature lands on its feet without falling all the way down to the ground at landing.

The synchronism inherent in the gait developed is much more easily seen than what was the case for the previous creature discussed. Most importantly, the hindmost legs are synchronized in the thrusts causing the creature to jump forward. The front legs, which thrust and move in a circular manner, are synchronized similarly. The behavior seen at the middle leg pair seems

less deterministic; in fact, it would seem like the gait developed had been even more efficient had the creature had only four legs.

As for the creature from generation 50 discussed above, the creature from generation 55 also does not execute the gait developed perfectly. From time to time, the creature falls out of the repetitive movement pattern, thus intercepting propulsion. Further, as was seen for generation 50, after each such failure, the creature almost immediately restarts and continues to move according to the gait it has learnt.

The gait can be seen to improve somewhat over the course of the creatures lifetime; movements become increasingly synchronized, and, by and large, jumps become more efficient.

Generation 78: 7782

The best creature from generation 78 achieved a lifetime fitness of 7782. The ANN topology was identical to the seed topology (Figure 47), and is therefore not shown.

The GA single parameter output is shown in Table 35, where only those parameters whose values have changed compared to the initial GA input specification (Table 32) are listed.

Value type	Name	Description	Seed value	Value	Range
FLOAT	`std_stc`	Standard deviation parameter for Cauchy probability distribution used in stochastic perturbations	1	1.65	[0.1, 2]
FLOAT	`alph_ne`	Trace controlling parameter for neuronal trace function	0.5	0.01	[0.01, 1]
BOOL	`in_dtrc`	Inclusion of trace at neuronal drive values	false	true	
FLOAT	`lr_skin`	Learning rate for Skinner synapses	200	50	[50, 10000]
FLOAT	`lr_pavl`	Learning rate for Pavlov synapses	20	109	[5, 1000]
FLOAT	`alph_sk`	Trace controlling parameter for Skinner synapses	0.1	0.19	[0.01, 1]
FLOAT	`alph_pa`	Trace controlling parameter for Pavlov synapses	0.1	0.32	[0.01, 1]
BOOL	`newpavl`	Use of modified Pavlov synaptic learning mechanism	false	true	
BOOL	`newhume`	Use of modified Hume synaptic learning mechanism	false	true	
FLOAT	`mod2val`	Parameter controlling the function describing MOD2 efficacy limit as a function of the number of synapses	7	27.4	[5, 100]
FLOAT	`b_l_ver`	Leg vertical base angle (°)	-20	-39.9	[-70, 0]
FLOAT	`b_l_hor`	Leg horizontal base angle (°)	0	6.75	[-20, 20]

Continued on next page

Table 35 :: Continued					
Value type	**Name**	**Description**	**Seed value**	**Value**	**Range**
FLOAT	`maxdefv`	Maximum vertical angle of deflection (°)	45	50.1	[45, 70]
FLOAT	`alph_mu`	Trace controlling parameter for muscle force trace function	0.01	0.05	[0.001, 0.05]
BOOL	`in_hein`	Inclusion of torso height need	false	true	
BOOL	`in_angn`	Inclusion of extreme joint angle needs	true	false	
BOOL	`in_mufn`	Inclusion of muscle force needs	false	true	
BOOL	`in_lfrn`	Inclusion of foot friction need	false	true	
BOOL	`f_hei_n`	Scaling factor for torso height need	1	0.50	[0.1, 1]
BOOL	`f_muf_n`	Scaling factor for muscle force needs	1	0.73	[0.1, 1]
BOOL	`f_lfr_n`	Scaling factor for foot friction need	0.2	0.1	[0.1, 1]
FLOAT	`ex_angs`	Exponent in functions describing muscle length senses	1	1.30	[1, 2]
BOOL	`f_ang_s`	Scaling factor for muscle length senses	1	0.95	[0.1, 1]

Table 35: GA output GEN78

The parameter value changes are similar to those presented for the two creatures discussed previously. Interesting, however, is the inclusion of several optional needs: torso height, muscle force and foot friction.

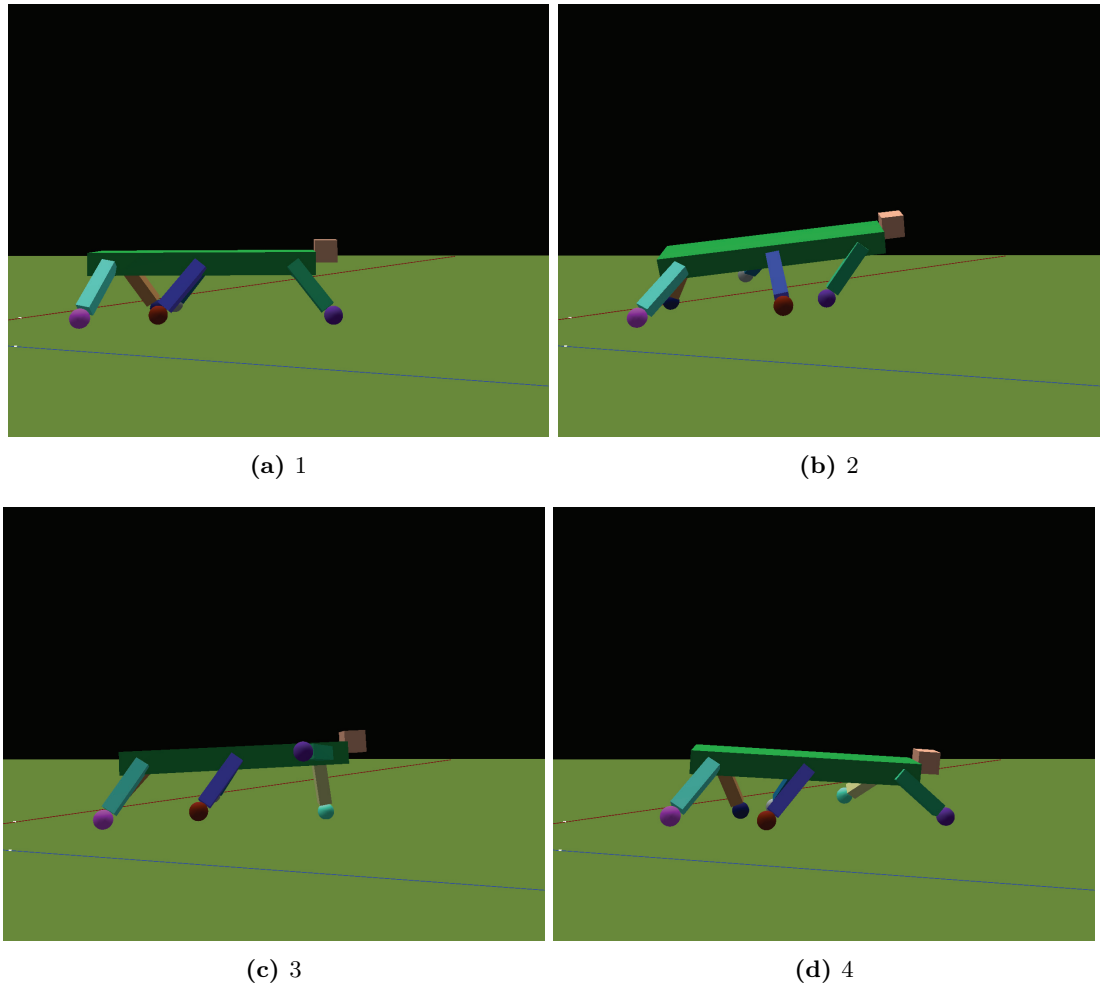
The lifetime visualization (Figure 52) shows that, as opposed to the creatures discussed previously, this creature very early in its lifetime discovers an efficient movement pattern, approximately at iteration 1700.[69]

The gait that this creature develops is the least chaotic and most biologically resemblant seen: Four of the six legs perform circular movements, resulting in a gait that, although being a bit awkward, most closely can be described as walking. The two remaining legs contribute little to propulsion; in fact, it appears that they are slowing the creature down. Their function as regards balance and stability may, however, be important in allowing the creature to maintain a reasonably steady forward velocity.

The synchronism inherent in the gait developed is easily seen: The two front legs, performing equal and repetitive circular movements, alternate in thrusting the ground, ensuring fairly smooth and continuous propulsion. More specifically, the two front legs can be said to be phase-inverted, meaning that when one leg is low and thrusting the ground the other is lifted high and is on its way forward. It appears that this highly deterministic alternation is important in maintaining both stability and forward velocity.

Continuing the analysis on synchronization, the hindmost left leg can be seen to contribute

[69] Recall that the creatures from generation 50 and 55 used almost half of their lifetime to discover efficient movement patterns (approximately at iteration 7000).



(a) 1 **(b)** 2

(c) 3 **(d)** 4

Figure 52: 3D visualization screenshots GEN78

to increasing and maintaining forward velocity by repetitively thrusting toward the ground. By further inspection, it is evident that this hindmost left leg consistently performs this movement at the exact same time as the front right leg performs the same movement. This latter synchronism seems to be crucial and dominant as regards obtaining forward velocity.

Finally, once again, the movement of the middle leg pair seems considerably less deterministic, and obviously contributes less to the creature's propulsion. As noted earlier, however, these middle legs may be important for keeping the creature from losing balance and falling to the ground.

Although performance is a little varying, the creature never falls out of the gait it has learnt; from approximately iteration 1700 and throughout the rest of its lifetime, the creature moves forward with the same repetitive movement pattern. Over the first few thousand iterations after gait discovery, the movement pattern can be seen to steadily increase in efficiency, whereafter behavior converges at a fairly propitious pattern resulting in a steady and reasonably high forward velocity.

4.6.2.3 Summary

To summarize the results from Experiment 4: Forward velocity v2, we conclude that some creatures did indeed develop gaits, i.e. repetitive, synchronized and deterministic movement patterns. Reasonably efficient gaits were, however, only seen for very few of the 10100 creatures simulated: A large part of the creatures showed promising tendencies, but only a few managed to develop fully functional gaits resulting in high fitness values.

When re-running the creatures that exercised the best behavior, i.e. when simulating creatures with identical configurations as those given by the creatures from generations 50, 55 and 78 discussed above, results are fluctuating. There is, indeed, a clear tendency that chiefly deterministic movement patterns develop, but the efficiency of these patterns is highly varying: Identically specified creatures vary from developing quite efficient behavior resemblant to that seen earlier, to developing behavior that causes no forward velocity worth mentioning, or even move backward. At average, behavior seems a little better than indifferent behavior (lifetime fitness values are usually greater than zero), but only few runs result in fitness values approaching those obtained earlier.

Considering the fact that groups of simulated creatures are identically specified internally, these findings show that the behavior developed heavily depends on the specific course of random samples encountered in the calculations on neuronal drive values.

Table 36 gives an overview of the development in GA parameter values over the few generations discussed above.

Name	Seed value	GEN50	GEN55	GEN78	Range
Neuronal model GA parameters					
std_stc	1	1.74	1.74	1.65	[0.1, 2]
in_ntrc	true	true	true	true	
alph_ne	0.5	0.81	0.82	0.5	[0.01, 1]
in_dtrc	false	true	true	true	
lr_skin	200	50	50	50	[50, 10000]
lr_pavl	20	109	109	109	[5, 1000]
lr_hume	20	215	215	145	[5, 1000]
alph_sk	0.1	0.19	0.39	0.19	[0.01, 1]
alph_pa	0.1	0.32	0.32	0.32	[0.01, 1]
alph_hu	0.1	0.01	0.01	0.35	[0.01, 1]
ddexpon	1	1	1	1	[1, 3]
newpavl	false	true	true	true	
newhume	false	false	true	true	
mod_trc	false	false	false	false	
mod2val	7	25.7	26.4	27.4	[5, 100]
Mechanical model GA parameters					
num_lps	3	3	3	3	{2, 3, 4}
b_l_ver	-20	-30.2	-30.6	-39.9	[-70, 0]
b_l_hor	0	17.6	19.8	6.75	[-20, 20]
maxdefv	45	47.9	47.9	50.1	[45, 70]
maxdefh	45	48.9	45	45	[45, 70]

Continued on next page

Table 36 :: Continued					
Name	**Seed value**	**GEN50**	**GEN55**	**GEN78**	**Range**
alph_mu	0.01	0.05	0.03	0.05	[0.001, 0.05]
in_hein	false	false	true	true	
in_beln	false	true	true	false	
in_angn	true	false	false	false	
in_mufn	false	false	false	true	
in_lfrn	false	false	false	true	
f_hei_n	1	1	0.71	0.50	[0.1, 1]
f_bel_n	0.1	0.1	0.1	0.1	[0.01, 1]
f_vel_n	1	0.49	0.49	1	[0.1, 1]
f_ang_n	0.1	0.1	0.1	0.1	[0.01, 1]
f_muf_n	1	1	1	0.73	[0.1, 1]
f_lfr_n	0.2	0.2	0.2	0.1	[0.1, 1]
ex_angs	1	1.30	1.30	1.30	[1, 2]
ex_aves	1	1	1	1	[1, 2]
in_angs	true	true	true	true	
in_aves	false	false	false	false	
f_ang_s	1	0.95	1	0.95	[0.1, 1]
f_ave_s	0.1	0.1	0.1	0.1	[0.01, 1]

Table 36: GA parameter development

Consider first the neuronal model. The randomness in neuronal drive is consistently increased compared to the seed, causing initial learning to be faster, while at the same time decreasing the level of determinism. Further, the trace function at neuronal drives is activated for all of the three best creatures. All synaptic learning rates and the corresponding trace-controlling parameters have been adjusted; most prominently, the Skinner learning rate has been minimized.

An interesting thing worth noting is that the modified Pavlov learning mechanism is used in all of the three best creatures, and the modified Hume learning mechanism is used in two of these three. This might indicate that these variants on the Pavlov and Hume learning mechanisms are propitious with respect the development of gaits. Further worth noting is that the new parameters introduced in Section 4.5 remain at their default values, meaning that none of the qualitative modifications made to the neuronal model were included for any of the three best creatures.

As for the mechanical model, customary adjustments on joint angle range specifiers have been made. Further, the muscular alpha controlling the rapidness of muscles is consistently increased, and for two of the three creatures it has been maximized to the upper limit of 0.05.

Further worth studying is the exclusion or inclusion of the different types of needs. Most prominently, the extreme joint angle needs are excluded from all three creatures. Both the height need and the belly pain sensor needs are included for two of three creatures, whereas the muscle force needs and the foot friction needs are included for the last one.

Lastly as regards the parameter changes, it should be noted that all three creatures continued the default settings as regards the inclusion of senses: For all three creatures, the muscle length

senses are included, whereas the muscle length rate senses are excluded. These results, together with the corresponding above results on needs, provide clear indications as regards what needs and senses are favorable with respect to the development of gaits for forward velocity.

4.7 Experiment 5: Forward velocity v3

This section describes the system configuration and simulation results constituting Experiment 5: Forward velocity v3. The goal specification, need configuration and fitness function are equal to those used in previous experiments on forward velocity (Experiments 3 and 4), and are therefore not discussed here.

Further, the GA input configuration is identical to the previous experiment, and is consequently not repeated here. In summary, the only thing separating this experiment from the previous is the topology specification i.e. the seed topology and the set of allowed topologies (as described in Section 4.5.2).

4.7.1 Experiment results

This section presents an overview of the results obtained from Experiment 5: Forward velocity v3. The GA process was run with the standard settings described in Section 3.4: mutation rate $P = 10\%$, mutation amount $A = 20\%$, and 50 individuals per generation. Each creature was simulated for 15000 iterations, which translates into a real-time lifetime of 2.50 minutes. The simulations were stopped after 125 generations, implying that a total of 6250 virtual creatures were simulated.

4.7.1.1 Results overview

Figure 53 shows the progress in fitness over the 125 generations simulated (numbered 0-124), depicting the best, average, and worst fitness for each generation.

The graphs for best, average, and worst fitness throughout the experiment show similar tendencies as those described for Experiment 4: Forward velocity v2. For best fitness, there is a consistent increase over approximately the first 50 generations. At generation 0 the best fitness is 772, whereas at generation 50 this value has increased to 3634. Although the graph remains jagged for the rest of the generations simulated, values chiefly stabilize above 2500, and they never drop to the levels seen at the early generations. The absolute maximum fitness value of 5279 is achieved at generation 74. Although reasonably high, this value is considerably lower than the maximum value of 8323 seen for the previous experiment.

As regards average fitness, values increase slightly over the simulation period, indicating that, at average, behavior is somewhat better than the indifferent behavior of lying completely still on the ground. This is fully equivalent to what was seen for Experiment 4.

Finally, as was also seen for Experiment 4, the graph for worst fitness is similar to the graph for best fitness, only opposite. This indicates that the spread in performance among the creatures of a generation is large, in effect implying that some creatures developed quite efficient patterns for moving backward.

In summary, then, fitness graphs indicate that the forward velocity goal is solved reasonably successfully, although the best creatures from this experiment achieved considerably lower fitness values than those seen for the previous experiment.

4.7.1.2 Analysis of specific creatures - GA output and qualitative descriptions

As depicted by the fitness chart of Figure 53, the creatures of this experiment never obtain as high fitness values as those that were obtained for the previous experiment. Not unexpectedly, visual inspection shows that the movement patterns developed are not efficient to the extent seen for the previous experiment. Anyhow, similar tendencies of deterministic and propitious

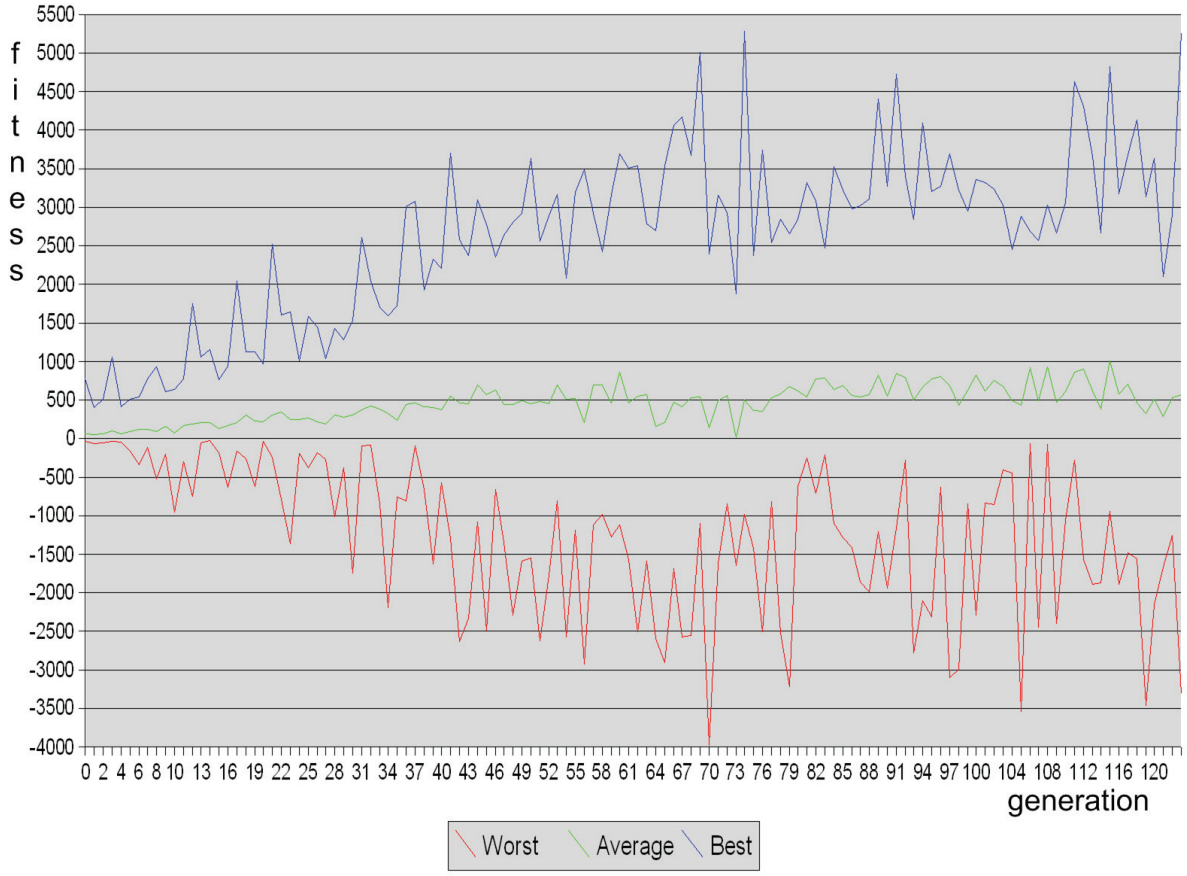


Figure 53: Fitness progress for Experiment 5: Forward velocity v3

movement patterns are clearly visible. The creature that achieved the absolute maximum fitness is discussed below.

Screenshots are provided that attempt to show the gait developed. However, as was also pointed out for the previous experiment, to really get the idea of how the creature is moving, the lifetime visualization should be inspected.[70] The creature's lifetime fitness value is shown in the heading below.

Generation 74: 5279

The best creature from generation 74 achieved a lifetime fitness of 5279. The ANN topology is slightly changed compared to the seed topology; three new clustral connections have been introduced: The senses now additionally connect directly onto the affect cluster, the affect cluster additionally connects directly onto the motor cluster, and the motor cluster is now intraconnected. The resulting topology is shown in Figure 54.

The GA single parameter output is shown in Table 38, where only those parameters whose values have changed compared to the initial GA input specification (Table 32) are listed.

[70] See Appendix A for instructions.

	N	S	A	I1	M
N	X	X	C	X	X
S	X	X	P	P	0
A	X	X	X	S	S
I1	X	X	0	0	P
M	X	X	0	0	H

Table 37: Topology specification GEN74

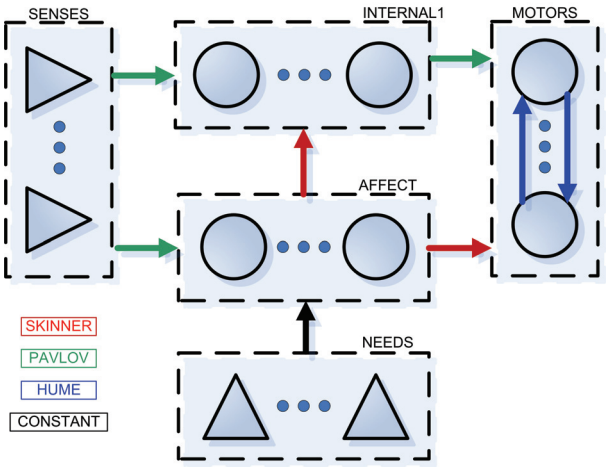


Figure 54: Topology structure GEN74

Value type	Name	Description	Seed value	Value	Range
FLOAT	`std_stc`	Standard deviation parameter for Cauchy probability distribution used in stochastic perturbations	1	2	[0.1, 2]
BOOL	`in_ntrc`	Inclusion of neuronal trace element in neuronal activation function	true	false	
FLOAT	`alph_ne`	Trace controlling parameter for neuronal trace function	0.5	0.35	[0.01, 1]
BOOL	`in_dtrc`	Inclusion of trace at neuronal drive values	false	true	
FLOAT	`lr_skin`	Learning rate for Skinner synapses	200	50	[50, 10000]
FLOAT	`lr_pavl`	Learning rate for Pavlov synapses	20	101	[5, 1000]
FLOAT	`lr_hume`	Learning rate for Hume synapses	20	5	[5, 1000]

Continued on next page

Table 38 :: Continued

Value type	Name	Description	Seed value	Value	Range
FLOAT	alph_sk	Trace controlling parameter for Skinner synapses	0.1	0.01	[0.01, 1]
FLOAT	alph_pa	Trace controlling parameter for Pavlov synapses	0.1	0.19	[0.01, 1]
FLOAT	alph_hu	Trace controlling parameter for Hume synapses	0.1	0.08	[0.01, 1]
FLOAT	ddexpon	Exponent in function transforming synaptic learning mechanism delta drives	1	1.02	[1, 3]
BOOL	newpavl	Use of modified Pavlov synaptic learning mechanism	false	true	
BOOL	newhume	Use of modified Hume synaptic learning mechanism	false	true	
BOOL	mod_trc	Use of modified synaptic trace function with new persistence increasing element	false	true	
FLOAT	mod2val	Parameter controlling the function describing MOD2 efficacy limit as a function of the number of synapses	7	41.8	[5, 100]
INT	num_cns	Number of neurons in internal clusters	20	18	[10, 50]
FLOAT	b_l_ver	Leg vertical base angle (°)	-20	-39.1	[-70, 0]
FLOAT	b_l_hor	Leg horizontal base angle (°)	0	20	[-20, 20]
FLOAT	maxdefv	Maximum vertical angle of deflection (°)	45	40.9	[45, 70]
FLOAT	maxdefh	Maximum horizontal angle of deflection (°)	45	54	[45, 70]
FLOAT	alph_mu	Trace controlling parameter for muscle force trace function	0.01	0.04	[0.001, 0.05]
BOOL	in_hein	Inclusion of torso height need	false	true	
BOOL	in_beln	Inclusion of belly pain sensor needs	false	true	
BOOL	in_lfrn	Inclusion of foot friction need	false	true	
BOOL	f_hei_n	Scaling factor for torso height need	1	0.56	[0.1, 1]
BOOL	f_vel_n	Scaling factor for velocity need	1	0.44	[0.1, 1]
BOOL	f_ang_n	Scaling factor for extreme joint angle needs	0.1	0.29	[0.01, 1]
BOOL	f_lfr_n	Scaling factor for foot friction need	0.2	0.21	[0.1, 1]

Continued on next page

Table 38 :: Continued					
Value type	**Name**	**Description**	**Seed value**	**Value**	**Range**
FLOAT	`ex_angs`	Exponent in functions describing muscle length senses	1	1.18	[1, 2]
BOOL	`f_ang_s`	Scaling factor for muscle length senses	1	0.76	[0.1, 1]

Table 38: GA output GEN74

Most importantly, both the Skinner learning rate and the Hume learning rate have been minimized. Further worth noting is that all needs except the muscle force needs are included in the model.

The lifetime visualization (Figure 55) shows that the creature gradually develops a repetitive movement pattern, starting at approximately iteration 3000 and improving throughout the creature's lifetime.

The pattern developed is not as efficient as any of the three gaits that were discussed for the previous experiment. Behavior is based on vigorous thrusts performed by three, and at times four, of the legs from the hindmost and middle leg pairs causing the creature to jump forward. Performance is degraded by the fact these movements are not consistently successful; only from time to time does the creature succeed in performing jumps that brings it forward at any considerable velocity. Sometimes, on the other hand, the creature performs a series of successful jumps in a row; when this happens, the creature seemingly moves faster than any of the creatures seen earlier.

The synchronism in the movement pattern is mainly visible in the simultaneous thrusts performed at the three or four back legs. Other than this, behavior is partly chaotic; especially the two front legs are moving in a more random manner that does not seem to contribute much to the forward velocity goal.

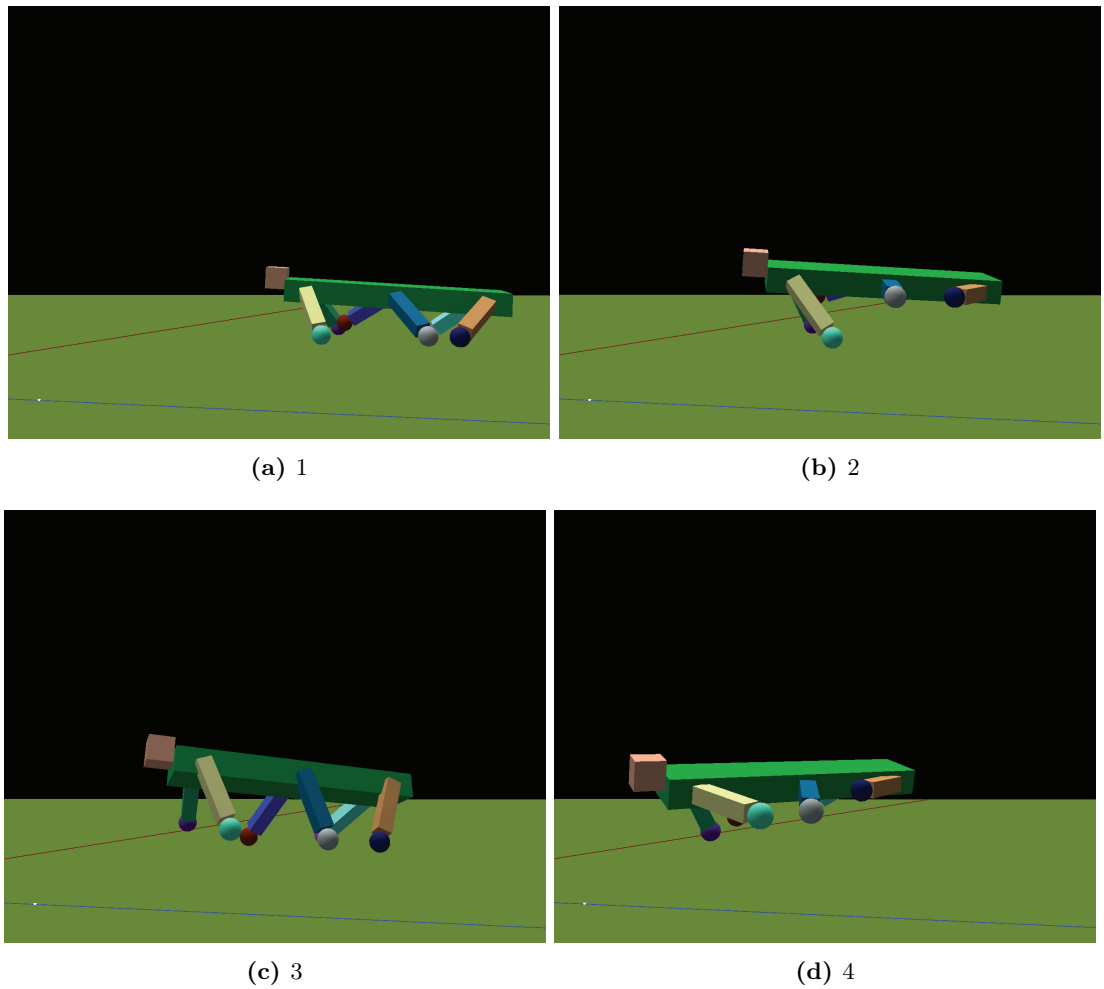
4.7.1.3 Summary

To summarize the results from Experiment 5: Forward velocity v3, we can conclude that the increase of topological complexity has not been favorable as regards the crude level of goal achievement: The best creatures from the previous experiment, where the topologies had no internal clusters, performed considerably better than all creatures from the present experiment, where the topology had one internal cluster.

Anyhow, the results obtained are important in another respect: They show tendencies not previously seen of gait-like movement patterns developing in ANNs incorporating more complex recurrent topologies. For the best creature from this experiment (GEN74), the motor cluster was intraconnected by means of Hume synapses. At the opposite, none of the best creatures from Experiment 4 incorporated Hume synapses at all; in fact, all their topologies were purely feedforward, meaning that all synaptic connections were directed from inputs to outputs.

In conclusion, therefore, Experiment 5 has provided important positive indications on the suitability of recurrent networks in general, and of the Hume learning mechanism in particular, as regards the development of sensible goal-oriented patterns of movement.

To investigate the degree of determinism in the development of this behavioral pattern, identically specified creatures as that of generation 74 have been run. Analogous to what



(a) 1 **(b)** 2

(c) 3 **(d)** 4

Figure 55: 3D visualization screenshots GEN74

was seen for the previous experiment, these re-runs result in similar tendencies as regards the development of partly deterministic movement patterns, but the efficiency of these patterns as regards the forward velocity goal is highly variable. Thus, even with the increase of network size and complexity, actual behavioral results depend heavily on the randomness in the calculations on neuronal drive values.

5 Discussion

5.1 Results

This section discusses the results obtained throughout the experimental work with this thesis.

5.1.1 Results summary

Experiment 1 of Section 4.2 showed that the torso height goal, which involved the raising and stabilization of the creature's torso, was successfully solved. Results on best fitness were convergent, with final behavior characterized by creatures that raised and stabilized their torso perfectly. A few creatures achieved even better fitness values by performing seemingly quite random but yet effective behavior characterized by repetitive jumps causing the torso to elevate above the normal maximum height. The synchronism and repetitiveness inherent in the behavior of these creatures provided the first indications on the development of partly synchronized and repetitive movement patterns.

Experiment 2 of Section 4.3 examined the head height goal, which was a variant on the torso height goal involving maximum elevation of the creature's head. Although in a less convergent manner than what was seen for the previous experiment, the head height goal was solved successfully. Successful creatures were characterized by supporting the weight of their body at the middle legs while stably and consistently keeping their head as high above ground as possible.

Experiment 3 of Section 4.4 concerned the main experimental goal of this thesis: the development of gaits for forward velocity. As expected, this goal proved to be far more difficult to fulfill, and no sensible or efficient gaits developed. The experiment revealed an evident weakness in the mechanical model: Toward the end of the experiment results had stabilized at a type of behavior characterized by vibratory leg movements causing creatures to move forward in an unrealistic manner. Earlier in the simulations, however, promising tendencies were seen of creatures developing partly synchronized and repetitive movement patterns, thus indicating the potential for gaits to develop.

Based on the weaknesses seen in Experiment 3, the model and simulation settings were reassessed (Section 4.5): Two optional qualitative modifications were introduced to the neuronal model, heavier restrictions were put on the set of allowed topologies, and some quantitative changes were made to the mechanical model to disallow the vibratory behavior seen earlier.

Experiment 4 of Section 4.6 continued investigating the forward velocity goal. With the new restrictions on the mechanical model, the GA search process proceeded differently, and several examples of reasonably efficient movement patterns were seen: The best creatures deterministically moved according to synchronized and repetitive gaits, causing them to obtain forward velocity in a quite consistent and continuous manner. Repeated simulations on identically specified creatures did, however, reveal that the efficiency in the movement patterns developed depended heavily on the specific course of random samples encountered in the calculations on neuronal drive values.

Experiment 5 of Section 4.7 was identical to Experiment 4, except that a more complex type of network topology was used. Results showed that, also for this configuration, synchronized and repetitive movement patterns developed, causing some creatures to successfully move forward in a reasonably efficient manner. With a crude view to forward movement, behavior was never successful to the degree seen for the best creatures of Experiment 4. The experiment did,

however, provide new and important findings, in that the best creatures incorporated recurrent networks with Hume-based learning at the motor cluster. Whilst all the best creatures from Experiment 4 incorporated ANNs based on the feedforward structure, the best creatures from this experiment showed that gait-like movement patterns could develop in multi-layer recurrent networks. Dependence on the course of random samples seemed similar to that seen for Experiment 4; in simulating identically specified creatures repetitive movement patterns consistently developed, but the degree of efficiency as regards forward velocity was highly variable.

5.1.2 Methods of analysis

The results of this thesis have been analyzed in three ways: 1) inspection of progress in fitness values by means of fitness graphs, 2) comparisons of GA input and output parameter values, and 3) inspection of creature lifetime visualizations. All of these have provided important information on different aspects of the simulation results.

One important aspect is, however, missing: The analysis of ANN drive and efficacy values. Because of the nature of the simulations performed herein, with creatures being simulated batch-wise and inspected at a later time, decent analysis of the course in ANN drive and efficacy values has been out of reach. Consider first the amount of data that must be stored continually for post-analysis of ANN dynamics to be possible: For a six-legged creature incorporating a small sized ANN with three needs, one sense and one internal cluster, over a single lifetime of 50000 iterations the storage requirements for neuronal drive values and synaptic efficacy values alone would exceed 100 megabytes. When further considering the effort required in decently time-analyzing such amounts of data, it soon becomes evident that attempting such an approach for this thesis would have heavily affected the width of experimental coverage.

5.2 Neuronal model

This section discusses selected properties of the neuronal model.

5.2.1 Randomness in neuronal drives

The stochastic perturbations on neuronal drives were presented in Section 3.2.5.2, and have been used throughout the simulations of this thesis. Since all synaptic efficacies are initialized to zero, this randomness is absolutely necessary for initial learning to take place.

Such randomness does, however, pose some difficulties. When running identically specified creature simulations, i.e. such that all parameters describing the simulation system are set identically, the actual results obtained may at times be divergent. For instance, when GA fitness statistics depict that some specific creature achieved a high fitness value, running the exact same creature again may provide very different results. Decreasing the size of stochastic perturbations does, of course, reduce this problem. However, because the extent of initial behavioral deflections is determined by the amount of randomness, the decrease of randomness simultaneously causes initial learning to be slower. The trade-off between behavioral determinism and rate of initial learning thus becomes evident. The introduction of linear decline in randomness with lifetime seeks to overcome this trade-off. Still experiments show that the course of behavior is partly, and at times heavily, dependent on the actual stochastic samples encountered.[71]

[71] This lack of determinism between identical simulation runs was the main reason for implementing the post-visualization program provided with this thesis, that allows the actual behavior of the fittest creatures to be

5.2.2 Mapping need values to affect neurons

Affect systems, as presented in Section 3.2.6.1, allow needs, and thus potentially the behavior required to reduce them, to be anticipated in a more direct fashion based on senses. Originally, need values were mapped roughly onto corresponding affect neuron drives by synaptic transmission and subsequent transformation through the neuronal activation function. The nonlinearity introduced by this transformation has proved to be highly unfavorable, because the effective learning rates of the Skinner synapses connecting affect neurons to the internal network structure become dependent on the corresponding current need value ranges. More specifically, preliminary experiments have shown that the transformation causes the initial rate of learning, i.e. when need values typically are high, to be reduced considerably, in effect significantly increasing the time spent in the early learning phase.

To overcome this unwanted dependence on need value ranges, the following strategy has been suggested: Need values are transformed through an inverse activation function, such that the synaptic input to the affect neuron that, when considered in isolation, would have caused the exact same drive value is found. This value is then included in the customary calculations on synaptic drive, i.e. included in the summed synaptic input and transformed through the neuronal activation function. In that way, changes in affect neuron drives will mirror changes in need values exactly, while still allowing other neurons (e.g. sense inputs) to synaptically influence activity at affect neurons.

5.2.3 Neuronal model parameter granularity

Important in determining the effective learning rate are the learning rate parameters β_{ij}. For the simulations herein, these have been specified at the level of learning mechanisms, giving rise to a total of three different learning rates (`lr_skin`, `lr_pavl` and `lr_hume`).

An inherent property of the learning mechanisms of Connectology is that the size of synaptic change is proportional to the size of changes in presynaptic and postsynaptic drive values. In connection with ANN inputs, presynaptic drive changes are dictated by changes of need or sense values. In the experimental context of this thesis, the following difficulty has been experienced: The values of different types of needs and senses often tend to change at highly different rates, thus causing effective learning rates to vary considerably among the set of needs or senses providing feedback to the ANN. For instance, the values representing the forward velocity need used herein tend to change much more rapidly than the corresponding positional needs such as the torso height or head height needs. Similarly, muscle length rate senses, which are based on joint angular velocity, tend to change more rapidly than muscle length senses, which are based on joint angle.

For both of the examples just mentioned, one of the needs/senses is based on the derivative of the measure on which the other is based. Rate of change necessarily increases with the degree of the derivative. Had then, in addition, a need or sense based on acceleration (i.e. the second derivative of some measure) been included in the model, the problem would have been even more prominent.

For the simulations of this thesis, the problem has been addressed by the introduction of scaling factors on needs and senses. These describe the relative reciprocal intensities, and thereby allow overly dominant need or sense values to be toned down. A side effect, however,

inspected properly.

is that the actual value range also is modified, meaning that the actual need and sense values that drive behavior are reduced correspondingly.

A possible solution might be to introduce separate learning rates for each need and sense type. The Skinner synapses connecting the forward velocity need input could, for instance, learn by a different learning rate than the Skinner synapses connecting the torso height need input. Analogously, the Pavlov learning rates for the muscle length rate and muscle length senses could be different. Such a configuration could solve the problem of inherently different rates of change in different types of needs and senses quite elegantly.

The potential drawbacks would, however, have to be considered: Such a separation on specific synapses with regard to the type of needs or senses particular input values originate from would require a more complex programmatic neuronal model, incorporating a finer granularity on synaptic learning rates.[72] Further, in connection with affect systems, this would require one need input cluster for each need type, and a corresponding affect cluster for each of these. Also, if separate Skinner and Pavlov learning rates were to be used for the different need and sense types implemented, one could argue that it would be natural to, in general, specify learning rates at the level of specific clustral connections. Another disadvantage is therefore the introduction of even more simulation parameters.[73] Taking this chain of thought even further, one can imagine a learning rate granularity at the level of single synapses. Such a strategy would, however, make manual or GA-determined value specification intractable, and would require some automatic approach to tuning these parameters, e.g. based on the characteristics of the presynaptic and postsynaptic drive dynamics.

When considering a finer granularity on synaptic learning rates, it is natural to also consider a similar change with respect to the trace controlling parameters for the synaptic traces of drive differentials. In a similar manner to the learning rates discussed above, for the simulations herein, these are separated only by means of synapse type, i.e. such that a total of three values are required for a complete specification (`alph_sk`, `alph_pa`, `alph_hu`). Similar to the above, parameter granularity could be at the level of clustral connections, or possibly at the level of single synapses. As argued above, the last strategy would require automatic mechanisms for tuning of parameter values.

In a broader view, neuronal model parameters such as the size of stochastic perturbations in neuronal drives (`std_stc`), the trace controlling parameters for the neuronal trace element in the activation function (`alph_ne`) and the trace controlling parameter for the trace at neuronal drives (implicitly $5\cdot$ `alph_mu`) could also be considered with respect to finer parameter granularity, e.g. such that different groups of neurons could incorporate different degrees of randomness or different lengths of neuronal drive traces. Considering the increase of complexity, however, the potential benefits from such an approach should be assessed carefully before simulations are attempted.

The strategy of finer granularity on synaptic or neuronal parameters depicted above has not been attempted for the simulations of this thesis; time-limitations, combined with the necessary increase of model and simulation system complexity, has made it intractable.

[72] Although of limited theoretical importance, this would in addition suggest an engineerably unfavorable coupling of neuronal model and mechanical model concepts.

[73] If only need and sense related learning rates are specified specifically and scaling factors of the type discussed above are in use, however, the latter drawback disappears; there is no need for both separate learning rates and scaling factors, in effect making the number of parameters unaltered.

5.2.4 Divergence preventing modifications

The MOD1 and MOD2 divergence preventing learning mechanism modifications of Section 3.2.5.4, also examined in a previous simulation study [Axe06], have been used throughout the simulations of this thesis. Different from previous simulations, however, is the function describing the limit on maximum summed synaptic efficacies in MOD2. In [Axe06], a function that was linear in the number incoming synapses was used, but this function proved incapable of sensibly limiting rapidly growing synapses when networks grew large. Consequently, an alternative logarithmically shaped limit function has been suggested for this thesis. As far as analysis of ANN values has been possible, this new function seems to appropriately limit synaptic growth, also for larger networks.

5.2.5 Modified Pavlov and Hume synaptic learning mechanisms

Section 3.2.5.3 presented variants on the original equational specifications for the Pavlov and Hume synaptic learning mechanisms, which implied moving the positivity check on the presynaptic trace of drive differentials from the trace equation to the learning mechanism equation. The use of these modified Pavlov and Hume mechanisms has been optional for all simulations performed, implying that the GA search process has been free to determine whether or not to use the modified versions of the equations.

Interestingly, a large majority of the best creatures simulated have incorporated these new equations: The best creature from Experiment 1: Torso height used the new Pavlov variant, the best creature from Experiment 2: Head height used both the new Pavlov and Hume variants, the best creature from Experiment 3: Forward velocity used both the new Pavlov and Hume variants, all of the three best creatures from Experiment 4: Forward velocity v2 used the new Pavlov variant whereas two of these also used the new Hume variant, and, finally, the best creature from Experiment 5: Forward velocity v3 used both the new Pavlov and Hume variants. To summarize, almost all of the most successful creatures utilized the new variants on the Pavlov and Hume mechanisms, providing strong indications that these new equations may be superior to the original ones.

5.2.6 Model reassessments

Section 4.5 presented two possible modifications to the equations describing synaptic learning, designed to 1) favorize significant events by nonlinearly transforming delta drives and 2) increase the persistence of (early) trace changes by including in the trace function an additive inverse of the trace itself. For Experiment 4: Forward velocity v2, none of the three best creatures incorporated any of these last modifications. For Experiment 5: Forward velocity v3, the best creature incorporated both of these modifications, although the delta drive transformation was only marginally nonlinear (i.e. very slight inclusion of modification 1) above).

The appropriateness of these model reassessments is therefore uncertain; no conclusion can be drawn based on present results. Intuitively, both modifications seem favorable, and it would therefore be unwise to write them off entirely without further investigating their impact on learning.

5.3 Mechanical model

This section discusses selected properties of the mechanical model.

5.3.1 Rigid-body dynamics

The mechanical model of this thesis is based on rigid-body dynamics, and creatures based on such purely rigid body parts are not biologically realistic. Ideally, one would have fully biologically realistic mechanical models on which neuronal theories could be tested, but for now that remains impracticable. The question whether fairly simple mechanical models of the type used herein are adequate for the task at hand can of course be raised. Chiel and Beer argue that behavior emerges from close interactions of nervous system and body, and that specific mechanical properties of the body are crucial in the development of adaptive behavior [CB97]. For our purposes, the most important such mechanical property seems to be the filtering of neuronal signals inherent in muscles,[74] which was addressed in Section 3.3.2. Based on the promising tendencies seen thus far, it is believed that the realism inherent in the mechanical model used herein is sufficient for gaits to develop. It is, however, probable that results could improve with increasing body realism and complexity.

5.3.2 Muscle model

Initially, the muscle model of this thesis was adopted directly from the presumably incomplete model provided in [Kan00]. Preliminary simulations showed that, using this muscle model, creatures could exercise oscillating movement patterns governed solely by passive muscle forces. Based on meetings with science experts Gertjan Ettema and Beatrix Vereijken [EV07], therefore, the model was revised such that, most importantly, the size of the passive components constituting a part of the muscle force calculations were reduced considerably. This revised muscle model has provided creature behavior that, by visual inspection, seems much more plausible.

The time-dependence introduced by letting effective muscle force be determined by a trace function has proved to be necessary: Calculating muscle forces directly by the active and passive instantaneous components allow muscularly controlled limbs to move in an unrealistically fast-paced manner. Further, it has been shown that the value of the trace-controlling parameter `alph_mu` should be in a quite limited range (approximately $(0, 0.05]$) for sensible movement patterns to emerge. It was just this tightening of the upper limit on `alph_mu` that allowed gait-like movement patterns to develop for the last two experiments.

5.4 Genetic algorithms

This section discusses selected properties as regards the use of genetic algorithms to search the simulation parameter space.

5.4.1 Input configuration

The input configuration, which specifies the set of parameters with respect to parameter types (constants or variable/GA), value types (integral, floating-point or boolean) and value ranges (minimum and maximum limits), wholly and fully defines the space within which favorable settings are searched for. Specifying this configuration properly is therefore of greatest importance, as it determines which settings are at all discoverable. To assure that potentially favorable parameter settings are not precluded, value ranges must be large enough for all potentially propitious parameter values to be explored. At the same time, experiments contained

[74] Quoting [CB97]: "Muscle acts as low pass filter of motor neuronal outputs, that is, it filters out the high frequency components of the neural outputs." (p. 553)

herein have proved that restricting and tightening these same limits can be necessary to obtain the results searched for. The latter is related to the disallowing of certain parameter settings that in an unwanted manner ease the task at hand, or, more formally, to remove certain local maxima in the parameter space that represent unrealistic or otherwise unwanted system states. For instance, for gait-like movement patterns to develop for the forward velocity goal of this thesis, the lower limits on vertical and horizontal leg deflection angles had to be raised and the upper limit on the parameter controlling the rapidness of muscular movements had to be lowered. In summary, the setting of the GA input configuration must be considered very carefully; value ranges must be set such that all potentially favorable settings are allowed while, at the same time, restricting unrealistic or otherwise unfavorable system states by tightening specific limits on the same value ranges. Finding (close to) optimal values for such initial configurations is difficult, and the uncertainty inherent in this manually specified input configuration is thus considered one of the most prominent weaknesses and difficulties with using the GA search approach.

5.4.2 Seed

The approach taken for the GA search process of this thesis is to initialize the GA search by means of specifying a starting point, the seed.[75] The specifying of this seed denoting the initial values of all variable parameters is of highest importance with respect to GA search results. The parameter space within which favorable solutions is searched can be considered to contain a multitude of local maxima, where each represents a locally optimal setting of GA parameters with respect to the goal at hand. Implicitly, therefore, the seed, which represents an entry point in parameter space, heavily influences the probability for discovering different local maxima, because, in general, local maxima that are close to the seed are more probable of being discovered.[76] In summary, the specification of the seed is therefore of highest importance with respect to the results that can be expected, and the inherent uncertainty in the appropriateness of its specification is another weakness and difficulty with using the GA search approach.

5.4.3 Searching ANN topologies

Also included in the GA search process of this thesis is the search for ANN topologies. The original approach taken was to allow networks to grow freely, letting the GA mutation process add or remove clusters at will. For some simulations this topology search resulted in inappropriately large networks with up to six internal clusters. Such large topologies are unwanted for two reasons: Firstly, the different goal specifications of the experiments contained herein are all of limited complexity, and such large networks are not expected to be necessary for the tasks to be solved successfully. Secondly, extensively large networks put heavy strain on computer simulations, causing running times to increase vastly. For the last experiments performed, ANN

[75] This choice has been made based on discussions at supervisor meetings; the traditional approach of generating the first GA generation by random selection is not considered appropriate with respect to the nature of the search task at hand.

[76] The GA search process implemented herein in principle allows relatively large leaps in parameter space, potentially allowing comparingly small local maxima to be escaped. Anyhow, in practice the positioning of the seed seems to be heavily decisive as regards what solutions are discovered. Distance in such spaces can be measured by means of L^P-norms, such as the L^2-norm (Euclidean distance, denoting by $d_{x,y}$ the distance between two points x and y in space, and by $\mathcal{D}$ the set of dimensions): $d_{x,y} = \sqrt{\sum_{i \in \mathcal{D}} (x_i - y_i)^2}$.

topologies were therefore restricted to two simple variants with maximally one internal cluster. These last experiments also provided the best results, and thereby strongly suggest that topologies of relatively limited complexity are adequate for the system goals examined herein.

5.4.4 Method suitability

The above made evident a few noteworthy difficulties with using the GA process in search of propitious parameter settings. In spite of these drawbacks, however, the GA search is still considered the best alternative. The most basic approach would be to search the parameter space manually, i.e. by trial and error. The size of the parameter space, however, makes manual search overly time-consuming and, most importantly, makes the probability of discovering truly propitious parameter settings small. When considering automated approaches, due to the high dimensionality of the parameter space, non-heuristic approaches such as exhaustive search are left intractable. In comparing different heuristic approaches to search, then, when considering the degree of biological inspiration in the systems simulated, the GA search process seems like a natural choice: The ultimate goal of the type of work performed herein is to be able to simulate neurally controlled systems that as closely as possible resemble the biological systems they imitate. In the process of adjusting and fine-tuning these systems themselves, then, it seems very natural to adopt the biological principles of the corresponding evolutionary processes that take place in nature.

In summary, the GA search process is considered the most suitable for simulations of the type performed herein. The process has indeed provided much better results than those obtained by manual setting of parameter values.[77] The above difficulties must however be considered, implying that the specifying of the GA input configuration and seed must be carefully addressed and accommodated to the task at hand. As regards the specifying of the seed, manual search by trial and error seems sensible. In conclusion, therefore, a combined strategy is recommended such that the seed from which the automatic GA search process is initiated is searched for by trial and error based on manual inspection of simulation results.

5.5 Program system

This section discusses selected properties of the computer simulations system developed for the experimental work of this thesis. For complete structural specifications and programmatic details, the reader is referred to Appendix C: Program Structure.

5.5.1 Parameter model and parametrization

The parameter model is built to be entirely extensible and flexible. For flexibility, if a decision is made to change a parameter from free (GA) to constant (CONST), or to change its seed value and/or value range, only one line of code needs to be changed. As regards extensibility, if some new parameter is needed to accommodate future changes to the neuronal or mechanical model, the only code changes necessary as regards the handling of simulation parameters are to add the parameter of interest to the generic and dynamic set of simulation parameters (specify name, value and possibly value ranges, one line of code), and to fetch the parameter in the portion of source code where it is needed (by lookup on parameter name, one line of code). All

[77] Compare generation 0 (which includes the seed) to the best results obtained for an experiment, and the progress due to the GA process becomes evident.

structures handling parameters are dynamic, implying that no changes need to be made to the static program structure as far as the handling of simulation parameters is concerned.

Further, the programmatic neuronal and mechanical models currently implemented are fully parametrized, allowing most properties to be changed, manually or dynamically during simulation. For instance, ANN topologies including clusters, neurons and synaptic connections, are built programmatically based on a generic topology specification matrix that can be freely specified. Other examples include the structure and appearance of the creature's body; the number of legs and the size of all body parts can very easily be changed, arranging for free experimentation with different types of creatures.

5.5.2 Extensibility and flexibility in general

Due to the degree of uncertainty in the underlying field of research, the program system has been built with extensibility and flexibility in mind. For instance, in spite of the fact that the creatures of this thesis only have one type of joints - the universal joints connecting the legs to the torso - a separate structure is included in the program system that allow new joint types, such as knee-joints, a.k.a. hinge joints, to be added as needed.[78]

5.5.3 Simulating multiple creatures simultaneously

Should it be of interest to simulate multiple creature simultaneously in future experiments, a very limited amount of changes would be needed in the program system. With simultaneous is meant experiments where multiple creatures coexist in the same dynamics world at the same time, effectively making creatures able to physically affect each other. A creature's behavior could then possibly become adaptive with respect to the behavior of other coexisting creatures.

5.6 Theoretical contributions

This section gives a short summary of the theoretical contributions made to the theory of Connectology through the work with this thesis. These are given in bullets below:

- A new function describing the limit on summed synaptic efficacies for the MOD2 divergence preventing learning mechanism modification has been suggested and validated. When networks grow large, this new logarithmically shaped function seems to limit synaptic efficacies more appropriately than the original limit function did.

- A new strategy for mapping need values onto affect neuron drives has been suggested and validated. This strategy removes the originally inherent dependence on need value range in the effective learning rates for Skinner synapses connecting affect neurons to the internal network structure.

5.7 Conclusion

The experimental work of this thesis has demonstrated the potential of ANNs based on the synaptic learning mechanisms of Connectology to perform determined muscular control on

[78] New joint types must, however, map onto one of the ODE joint types (Appendix B), but the latter are numerous and probably cover all types of interest.

structurally simple but mechanically realistic bodies acting in a physically realistic environment. Introductory simulations incorporating simple goals based on torso height and head height proved the validity of the theoretical and executional model. Later simulations incorporated the forward velocity goal, which required entirely different and much more complex behavioral patterns to emerge. These latter simulations clearly demonstrated the potential for gait-like movement patterns to develop: Several different variants of synchronized and repetitive movement patterns were seen, some of which were quite efficient with respect to the forward velocity goal. Sensible movement patterns emerged in both simple feedforward networks and more complex recurrent networks. For both, however, performance was degraded by the randomness inherent in the calculations on neuronal drives; for some system configurations, identically specified simulations gave highly divergent results, making evident the need to increase the level of determinism in the neuronal model.

In spite of this relative lack of determinism, the above results obtained on gait development are considered to be of great importance; they represent the only examples known to us of a fully unsupervised and self-organized artificial neural system that synaptically learns synchronized and repetitive motor control. The results are naturally also very important to the theory of Connectology as such, as they provide significant positive indications on the appropriateness and suitability of the contained learning mechanisms for biologically realistic learning.

The parameter space has been searched by means of genetic algorithms, an approach that has provided considerably better results than those obtained from manually specifying parameter values. Important to address carefully, however, is the specifying of GA input configuration (parameter types and ranges) and seed values (initial values for all parameters). These specifications have been seen to heavily affect simulation results, and must be addressed carefully and accommodated to the system goal at hand.

The GA output results obtained for the best creatures on forward velocity show clear tendencies as regards favorable values on selected model parameters for synchronized and repetitive movement patterns to develop. Of greatest interest are the tendencies on the setting of neuronal model parameters for the four creatures that developed the most efficient gaits; these settings provide indications on propitious parameter values that may be valid in general, i.e. also when other environmental contexts are involved. The most prominent among these tendencies are summarized below:

- There is a consistent and prominent increase of randomness of neuronal drive for all four creatures. Such an increase makes initial learning faster, but is also seen to reduce the degree of behavioral determinism.

- The tracing on neuronal drive values, which effectively introduces inertia into neuronal drives, is included for all four creatures. The averaging effect on neuronal drives may be important in the development of oscillatory patterns of neuronal activity.

- Opposite of the seed specification, the Skinner learning rate is consistently set considerably lower than the Pavlov learning rate. Further, with the inclusion of Hume synapses for the last experiment, the Hume learning rate is minimized to one tenth of the Skinner learning rate.

- The trace controlling parameter for the Skinner synapse is consistently set lower than the corresponding Pavlov and Hume parameters, in effect making the Skinner synapse react more slowly to changes of postsynaptic drive values.

- The alternative specifications of the Pavlov and Hume learning mechanism equations, where the positivity check on presynaptic delta drives is moved from the trace equation to the actual learning mechanism equation, are included for almost all of the best creatures seen. This indicates that these new formulations are superior to the old ones.
- The new persistence increasing element introduced for the synaptic trace equations was not included for any of the best creatures seen. Intuitively, however, this modification seems favorable, and further experimental work should be carried out before concluding on its appropriateness.
- The exponent in the function transforming delta drives is consistently minimized to one, in effect removing the transformation. Similar to the above, however, a nonlinear transformation intuitively makes sense, and the matter should thus be looked into more carefully before conclusions are drawn.

Finally, a flexible and extensible program system has been built that allows freely specified simulations of ANN-controlled mechanical creatures to be carried out. Simulations can be run in real-time, with real-time 3D visualization of creature behavior, or in batch-mode incorporating the GA search process. For the latter, behavior can be inspected carefully by means of post-visualization of creature lifetimes. The program system is designed for extensibility and flexibility, and should thereby serve as a solid foundation on which further connectological simulations on animal motion behavior can be based.

5.8 Further work

This thesis has demonstrated the potential for sensible movement patterns to develop in neural networks with synaptic learning based on the mechanisms of Connectology. As a general remark, however, the space of possible simulation configurations has not been fully covered, and it is most certainly possible that future simulations may provide better results, e.g. in terms of more efficient gaits. Thus, to thoroughly investigate the potential for motion behavior inherent in ANNs based on the theory of Connectology, a general recommendation is to perform further mechanically based ANN simulations of the type examined herein. In addition to increasing the knowledge as regards the development of gaits, such simulations could provide further general knowledge on connectological ANNs - knowledge that could be useful in later simulations using the ANNs of Connectology in other contexts and for other purposes.

Simulation results have showed that the behavior developed often is heavily dependent on the actual course of random samples encountered in the calculations on neuronal drive values. This tendency is unfavorable; equally specified simulations should to a greater extent provide equal or equivalent results. For future experiments, therefore, effort should be put on reducing the degree of randomness in the model, thus hopefully increasing determinism in the results obtained. Such a reduction of randomness would also be highly propitious in connection with the use of genetic algorithms for parameter search; the non-determinism seen for the simulations performed herein implies that the landscape in which fitness maxima are sought undergoes continuous transformation, causing the search for optimal parameter settings to become more difficult and less reliable.

One difficulty that has become evident through the simulations of this thesis is the lack of ability to analyze the course in neuronal drives and synaptic efficacies. The only means by which the performance of simulated creatures has been evaluated are: 1) the quantitative and

objective fitness measure and 2) the qualitative and subjective inspection of creature lifetime visualizations. To better understand the neural network dynamics that cause favorable behavior, and to better understand what kind of neural network dynamics hinder even more favorable behavior to emerge, it would have been highly useful to be able to perform real-time or post-simulation analysis and inspection of neural network time-series data.

For future simulation experiments of the type performed herein, effort should be placed at developing tools for analysis and inspection of neural network data. Such tools should ideally provide the user with a superior overview of current ANN state and values, and at the same time allow for specific details to be inspected at will. A considerable amount of work is probably needed to develop such tools of high quality. Their potential value in the analysis and understanding of ANN dynamics and quantitative as well as qualitative manner of operation should, however, be worth the effort.

Through the work with this thesis, making qualitative changes to the three learning mechanisms of Connectology has been considered. Such changes may e.g. assert themselves in the temporal ordering of presynaptic and postsynaptic events or in the direction of synaptic change. Based on the promising tendencies seen for the current set of learning mechanisms, however, it is recommended to further investigate the present learning model before reconsidering the qualitative properties of the mechanisms. It is also important to keep in mind that the qualitative properties of the current mechanisms of Connectology are based on well-known observable psychological principles, and the intuition behind each mechanism is easily available and re-examinable by means of simple analysis. By altering the qualitative functioning of the learning mechanisms, parts of this aspect on intuition and psychological foundation would disappear, thus undermining one of the most appealing qualities of Connectology. To summarize, further experimental work is recommended before any form of comprehensive qualitative change to any of the learning mechanisms is considered.

The mechanical model used for the simulations of this thesis is of limited complexity. Most importantly, all body parts are purely rigid, and legs are non-articulated. Although this thesis has assumed and to some degree proved articulated leg structures not to be necessary for gaits to develop, such leg structures most certainly dominate in nature; presumably, they allow gaits to be more efficient. For future experiments, therefore, increasing the mechanical model complexity and realism may provide better results than those obtained here. A natural first step would be to make legs articulated by introducing knee-joints. Chances are that, as the biological realism of the mechanical model increases, gaits may develop that more prominently resemble the gaits seen in biological systems.

References

[AHS85] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*, 9, 1985.

[Ala92] Jarmo T. Alander. On optimal population size of genetic algorithms. Technical report, Laboratory of Information Processing Science, Department of Computer Science, Helsinki University of Technology, 1992.

[All07] AllPsych. Psychological dictionary. http::/allpsych.com, 2007.

[Axe06] Vebjørn W. Axelsen. Operant conditioning - minimizing environmental needs through synaptic learning. *NTNU - TDT4745 Knowledge systems*, 2006.

[Bäc93] Thomas Bäck. Optimal mutation rates in genetic search. In Stephanie Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms*. Morgan Kaufmann, 1993.

[Bee90] Randall D. Beer. *Intelligence as adaptive behavior: an experiment in computational neuroethology*. Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[CB97] Hillel J. Chiel and Randall D. Beer. The brain has a body: adaptive behavior emerges from interactions of nervous system, body and environment. *Trends in Neuroscience*, 1997.

[CS92] Patricia S. Churchland and Terrence J. Sejnowski. *The Computational Brain*. The MIT press, 1992.

[CT03] Travis C. Collier and Charles Taylor. Self-organization in sensor networks. Technical report, UCLA Department of Organismic Biology, Ecology, and Evolution, 2003.

[EV07] Gertjan Ettema and Beatrix Vereijken. Discussion meetings on mechanical modeling, 2007.

[Fre95] S. Freud. Project for a scientific psychology. *The Standard Edition of the Complete Psychological Works of Sigmund Freud*, 1, 1895.

[GR00] Stanley Gotshall and Bart Rylander. Optimal population size and the genetic algorithm. Technical report, School of Engineering, University of Portland, 2000.

[Heb49] Donald O. Hebb. *The Organization of Behavior: A Neuropsychological Theory*. Wiley, June 1949.

[Hok97] J. Hokland. A hebbian model of classical and instrumental conditioning. Technical report, NTNU, Department of Computer and Information Sciences, 1997.

[Hok98] J. Hokland. A hebbian model of conditioning and concept learning. Technical report, NTNU, Department of Computer and Information Sciences, 1998. Draft, April 15.

[Hok06] J. Hokland. *Connectology: Research Programme for Brain-Psychology*. First edition, 2006. Draft, January 10.

[Hop82] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79, 1982.

[Hop84] J. J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences*, 81, 1984.

[Hum88] D. Hume. *A Treatise of Human Nature*. Oxford University Press, 1888.

[Kan00] Eric R. Kandel. *Principles of Neural Science*. McGraw-Hill Education, June 2000.

[Kel95] J. A. Scott Kelso. *Dynamic Patterns: The self-organization of Brain and Behavior*. The MIT press, 1995.

[Khr07] Khronos. OpenGL & OpenGL Utility Specifications. http://www.opengl.org, 2007.

[Klo88] A. Harry Klopf. A neuronal model of classical conditioning. *Psychobiology*, 16, 1988.

[Kol06] Bessel A. Van Der Kolk. Clinical implications of neuroscience research in ptsd. *Annals of the New York Academy of Sciences*, 2006.

[LSA84] T. Lanthorn, J. Storm, and P. Andersen. Current-to-frequency transduction in ca1 hippocampal pyramidal cells: Slow prepotentials dominate the primary range firing. *Experimental Brain Research*, 53(2), 1984.

[MRtPRG86] J. L. McClelland, D. E. Rumelhart, and the PDP Research Group. Parallel distributed processing: Explorations in the microstructure of cognition. *Psychological and Biological Models*, 2, 1986.

[Pav28] I. P. Pavlov. *Lectures on Conditioned Reflexes*. Liveright, 1928.

[Pia71] J. Piaget. *Biology and Knowledge - an Essay on the Relations between Organic Regulations and Cognitive Processes*. University of Chicago Press, 1971.

[Rip87] B. D. Ripley. *Stochastic Simulation, Chapt. 3: Random Variables*. 1987.

[Smi06] Russel Smith. Open dynamics engine v0.5 user guide. http://www.ode.org, 2006.

[Smi07] Russel Smith. Open dynamics engine - home. http://www.ode.org/, 2007.

[Tan06] M. Tanner. *Tools for Statistical Inference, Chapt. 6: Markow Chain Monte Carlo: The Gibbs Sampler and the Metropolis Algorithm*. 2006.

[TM03] Paul A. Tipler and Gene Mosca. *Physics for scientists and engineers: Extended Version*. W. H. Freeman, 2003.

[Yon02] Wu Yongwei. A Fast String Implementation for STL Map. http://www.geocities.com/yongweiwu/mstring.htm, 2002.

A System user guide

Two separate program systems are delivered with this thesis, the `Creatures` simulation system and the `CreatureVisualizer` post-visualization system. This appendix provides basic user guides for both. The `CreatureVisualizer` system, which is used for inspection of simulation results, is considered first.

A.1 The CreatureVisualizer system

The `CreatureVisualizer` program system is used to inspect the behavior of the creatures described in Section 4; it allows 3D post-visualization of creature lifetimes. It is a pure visualization system, and contains no semantics for performing neuronal or mechanical model simulations.

A.1.1 Startup

Visualizations are started as follows:

```
CreatureVisualizer.exe <path and name of file containing visualization data>
```

The example below shows how to start the lifetime visualization of the best creature from generation 55 of experiment Forward Velocity v2.

```
CreatureVisualizer.exe forwardvelocity_v2\visdata_gen55.dat
```

A.1.2 User interface

During visualization, the user can move the camera freely around in the 3D scene. Additionally, the animation speed can be altered, and the user can jump to the beginning or end of the simulation. All available user inputs are listed below.

```
Simulation positioning and animation speed
-------------------------------------------------
Go to start of simulation:      Home
Go to end of simulation:        End
Increase animation speed:       PageUp
Decrease animation speed:       PageDown
Set animation speed to 0:       Del

3D scene navigation
-------------------------------------------------
Look around:                    Mouse movement
Increase camera forward speed:  W
Decrease camera forward speed:  S
Stop camera:                    Space
Roll left:                      A
Roll right:                     D

Auxiliaries
```

```
-----------------------------------------------
Toggle wire frame rendering:     V
```

A.2 The Creatures system

The `Creatures` system is the actual program system that performs the neuronal and mechanical model simulations within a GA search process. The program structure for this system is described in Appendix C. The system has two modes: With and without real-time visualization. In addition to generation fitness statistics and, simulation runs output complete simulation configurations and complete visualization data for the best creatures of each generation. The latter allows creature lifetimes to be inspected later using the `CreatureVisualizer` program system presented above.

A.2.1 Startup

Based on recommendations given at supervisor meetings, and considering the limited amount of time available, no user interface has been built for this system. Thus, simulations are configured directly in the source code. An example of a creature configuration files is shown below:

```
//###TOPOLOGY###
int numClusters = 4;
int** matrix = new int*[numClusters];
for (int i = 0; i < numClusters; ++i)
   matrix[i] = new int[numClusters];
matrix[0][0] = -1;
matrix[0][1] = -1;
matrix[0][2] = 4;
matrix[0][3] = -1;
matrix[1][0] = -1;
matrix[1][1] = -1;
matrix[1][2] = 0;
matrix[1][3] = 2;
matrix[2][0] = -1;
matrix[2][1] = -1;
matrix[2][2] = -1;
matrix[2][3] = 1;
matrix[3][0] = -1;
matrix[3][1] = -1;
matrix[3][2] = -1;
matrix[3][3] = 0;
startindividual->setTopology(numClusters, matrix);
//###GA PARAMETERS###
startindividual->addGAInt("num_cns", 41, 10, 50);
startindividual->addGAInt("num_lps", 3, 2, 4);
startindividual->addGAFloat("alph_pa", 0.319311, 0.01, 1);
startindividual->addGAFloat("std_stc", 1.74763, 0.1, 2);
startindividual->addGAFloat("lr_hume", 214.759, 5, 1000);
startindividual->addGAFloat("alph_ne", 0.816079, 0.01, 1);
```

```
startindividual->addGAFloat("maxdefh", 0.78534, 0.78534, 1.22173);
startindividual->addGAFloat("alph_sk", 0.39212, 0.01, 1);
startindividual->addGAFloat("mod2val", 26.4114, 5, 100);
startindividual->addGAFloat("lr_pavl", 108.878, 5, 1000);
startindividual->addGAFloat("f_hed_n", 0.954172, 0.1, 1);
startindividual->addGAFloat("f_muf_n", 0.805587, 0.1, 1);
startindividual->addGAFloat("f_ang_n", 0.537023, 0.01, 1);
startindividual->addGAFloat("f_hei_n", 0.71009, 0.1, 1);
startindividual->addGAFloat("f_bel_n", 0.1, 0.1, 1);
startindividual->addGAFloat("f_vel_n", 0.491545, 0.1, 1);
startindividual->addGAFloat("f_lfr_n", 0.146021, 0.1, 1);
startindividual->addGAFloat("ex_angn", 18.1457, 3, 20);
startindividual->addGAFloat("lr_skin", 50, 50, 10000);
startindividual->addGAFloat("ddexpon", 1, 1, 3);
startindividual->addGAFloat("b_l_ver", -0.534272, -1.22173, 0);
startindividual->addGAFloat("b_l_hor", 0.34572, -0.34907, 0.34907);
startindividual->addGAFloat("f_ave_s", 0.954624, 0.1, 1);
startindividual->addGAFloat("f_ang_s", 1, 0.1, 1);
startindividual->addGAFloat("ex_aves", 1.333, 1, 2);
startindividual->addGAFloat("ex_angs", 1.30243, 1, 2);
startindividual->addGAFloat("alph_hu", 0.01, 0.01, 1);
startindividual->addGAFloat("alph_mu", 0.0334141, 0.001, 0.05);
startindividual->addGAFloat("maxdefv", 0.836598, 0.78534, 1.22173);
startindividual->addGABool("mod_trc", 0);
startindividual->addGABool("in_dtrc", 1);
startindividual->addGABool("in_ntrc", 1);
startindividual->addGABool("newhume", 1);
startindividual->addGABool("newpavl", 1);
startindividual->addGABool("in_mufn", 0);
startindividual->addGABool("in_angn", 0);
startindividual->addGABool("in_hein", 1);
startindividual->addGABool("in_beln", 1);
startindividual->addGABool("in_lfrn", 0);
startindividual->addGABool("in_aves", 0);
startindividual->addGABool("in_angs", 1);
//###CONST PARAMETERS###
startindividual->addCONSTInt("fitfunc", 1);
startindividual->addCONSTFloat("mx_l_ba", -1.39626);
startindividual->addCONSTFloat("m_ba_cf", 1);
startindividual->addCONSTFloat("m_fo_cf", 0.333333);
startindividual->addCONSTFloat("m_up_cf", 0.333333);
startindividual->addCONSTFloat("m_dw_cf", 1);
startindividual->addCONSTFloat("frc_leg", 1);
startindividual->addCONSTFloat("zero_sg", 0.15);
startindividual->addCONSTFloat("mx_l_fo", 1.39626);
startindividual->addCONSTFloat("timstep", 0.01);
```

```
startindividual->addCONSTFloat("mx_l_up", 0.87266);
startindividual->addCONSTFloat("frc_tor", 0.3);
startindividual->addCONSTFloat("len_lgs", 1.2);
startindividual->addCONSTFloat("maxf_mu", 5);
startindividual->addCONSTFloat("mx_l_dw", -1.39626);
startindividual->addCONSTBool("in_mod1", 1);
startindividual->addCONSTBool("in_mod2", 1);
startindividual->addCONSTBool("stc_dec", 1);
startindividual->addCONSTBool("in_hedn", 0);
startindividual->addCONSTBool("in_veln", 1);
startindividual->addCONSTBool("metropo", 1);
```

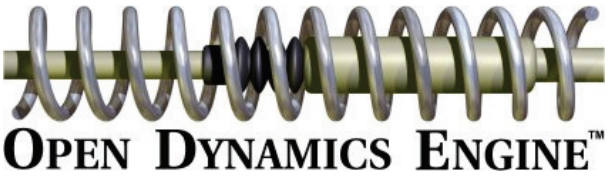
To start a simulation based on such a specification, the entire block of code is pasted into the right spot in the `runGA()` method of the `Controller` class, replacing the current simulation configuration, as shown below:

```
void Controller::runGA()
{
[...]
Individual* startindividual = new Inidividual();

// REPLACE ALL OF THIS
...
<current simulation configuration>
...
// END REPLACE

Individual::setInstance(startindividual);
[...]
}
```

A.2.2 User interface

The user interface in real-time visualization mode is the same as that described in Section A.1.2 above; except for the part on simulation positioning and animation speed, the user interface for `Creatures` is identical to that of `CreatureVisualizer`.[79]

[79] Additionally, s a quick demonstration of the capabilities of ODE, random sized cuboids and spheres can be dropped into the scene at real-time using the `1` and `2` keys, respectively. Dropping a multitude of such objects that collide with the creature and each other nicely demonstrates the realism in the ODE mechanics simulation.



B Open Dynamics Engine

Open Dynamics Engine (ODE) is the physics engine of choice for this thesis.

> *ODE is an open source, high performance library for simulating rigid body dynamics. It is fully featured, stable, mature and platform independent with an easy to use C/C++ API. It has advanced joint types and integrated collision detection with friction. ODE is useful for simulating vehicles, objects in virtual reality environments and virtual creatures. It is currently used in many computer games, 3D authoring tools and simulation tools. [Smi07]*

This appendix covers the basics of using ODE for simulating rigid-body virtual creatures of the type described in Section 3.3. The information contained herein is based on the official ODE manual [Smi06]. A lot of ODE details are, naturally, excluded from the description given here; for a complete specification of the workings of ODE, the reader is referred to the official manual.

B.1 ODE concepts and initialization

ODE is based on the following basic logical concepts:

- Body: represents a rigid-body object in the dynamics simulation
- Joint: connects two bodies and restrains their relative movement
- World: is a container for bodies and joints
- Geom: represents a collision object in the dynamics simulation
- Space: is similar to the world concept, except that it applies to collisions instead of dynamics (i.e. contains geoms)

As indicated by the above list, the dynamics simulations and the collision detection/response functionality are clearly separated concepts in ODE, allowing users to build and use their own collision models instead of the one provided with ODE. For the simulations herein, however, the default collision system included with ODE is used.

ODE supports a number of body types, of which two are used herein: Box and Sphere.[80] From these, composite objects can be built by attaching pairs of simple bodies by means of joints. There are several joint types in ODE, and the simulations of this thesis use two: Fixed and Universal.[81] The specific use of the two body types and the two joint types used was discussed in Section 3.3.1, and is not repeated here. All bodies and joints are contained in a

[80] Other body types implemented by ODE include Capsule, Cylinder and Trimesh.

[81] Other joint types implemented by ODE include Ball and Socket, Prismatic and Rotoide, Hinge, Slider and Hinge-2.

world. For our purposes, there is only one world containing all objects that are part of the dynamics simulation.

For collision detection, each body must be associated with a corresponding geom. Geom types correspond closely to the body types listed above, such that a body of one type is associated with a geom of the corresponding type. All geoms are contained in a collision space. For our purposes, there is only one space containing all objects that are subjected to collision detection and response.

B.1.1 World and space initialization

The dynamics and collision simulations are initialized as follows:

```
worldID = dWorldCreate();
dWorldSetGravity(worldID, 0.0f, -9.81f, 0.0f);
spaceID = dSimpleSpaceCreate(0);
contactGroup = dJointGroupCreate(0);
```

Here, a dynamics world is created with earthly gravity. Further, a collision space is created together with a contact group used in the collision detection routine.[82] ODE is pervaded by global ID's such as the world ID `worldID` and space ID`spaceID` which are saved for later reference above.

For the virtual creature simulations of this thesis, something representing the ground is needed for the virtual creature to be on. The ground is modeled as a mathematical plane (i.e. entirely and perfectly flat, and of infinite extent), as follows:

```
dCreatePlane(spaceID, 0.0f, 1.0f, 0.0f, 0.0f);
```

making the ground plane coincident with the XZ-plane.

B.1.2 Adding bodies and geoms

A box-shaped body with an associated box-geom is created as follows:

```
dMass mass;
bodyID = dBodyCreate(worldID);
dMassSetBox(&mass, density, lx, ly, lz);
dBodySetMass(bodyID, &mass);
dBodySetPosition(bodyID, posx, posy, posz);
geomID = dCreateBox(spaceID, lx, ly, lz);
dGeomSetBody(geomID, bodyID);
```

where `density` is the mass density of the box and `lx`, `ly` and `lz` are the width, height and depth, respectively. The mass concept, which has not yet been discussed, is simply a detaching of the geometric properties of rigid-body objects from the body concept.

A sphere-shaped body is created in the same fashion, except that lines three and six above are replaced by

```
dMassSetSphere(&mass, density, radius);
geomID = dCreateSphere(spaceID, radius);
```

respectively, where `radius` is the radius of the sphere.

[82] This contact group contains the contact points between pairs of bodies that are currently (i.e. at the current time step) colliding.

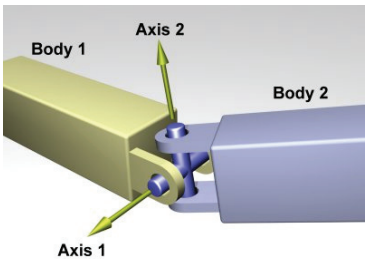


Figure 56: Universal joint (image courtesy of ODE Manual/Russel Smith)

B.1.3 Connecting bodies by joints

When creating fixed joints, the two bodies are fixed to the position and orientation they had at the moment of attachment. For instance, in connecting the head of our virtual creature to the torso:

```
dJointID torsoHeadJointID = dJointCreateFixed(worldID, 0);
dJointAttach(torsoHeadJointID, torsoBodyID, headBodyID);
dJointSetFixed(torsoHeadJointID);
```

After this, the relative position and orientation between the head and the torso will remain fixed throughout the entire simulation.

Universal joints are a bit more complex, see Figure 56. When connecting a leg of our virtual creature to the torso using a universal joint, the following is performed:

```
dJointID torsoLegJointID = dJointCreateUniversal(worldID, 0);
dJointAttach(torsoLegJointID, torsoBodyID, legBodyID);
dJointSetUniversalAnchor(torsoLegJointID, anchorX, anchorY, anchorZ);
```

where `anchorX`, `anchorY` and `anchorZ` are the X, Y and Z components of the anchoring point in world (absolute) coordinates. In addition to the anchoring point, the following must be specified for universal joints:

```
dJointSetUniversalAxis1(torsoLegJointID, axis1X, axis1Y, axis1Z);
dJointSetUniversalAxis2(torsoLegJointID, axis2X, axis2Y, axis2Z);
dJointSetUniversalParam(torsoLegJointID, dParamLoStop, loStop1);
dJointSetUniversalParam(torsoLegJointID, dParamHiStop, hiStop1);
dJointSetUniversalParam(torsoLegJointID, dParamLoStop2, loStop2);
dJointSetUniversalParam(torsoLegJointID, dParamHiStop2, hiStop2);
```

Here, `axis1` and `axis2` specify the two axes of the universal joint, as shown in Figure 56. Further, `loStop1` and `hiStop1` specify minimum and maximum angles of deflection for rotation about Axis1, and `loStop2` and `hiStop2` specify minimum and maximum angles of deflection for rotation about Axis2. All angles are given in radians, and are relative to the initial anchoring positions and orientations.

B.2 Simulating dynamics, collisions and movement

As mentioned in the previous section, the dynamics simulations and the collision detection/response functionality are isolated concepts in ODE, and they are thus handled separately at runtime, as

well. Additionally, our simulations involve the control of objects through applying torques/forces at joints.

B.2.1 Performing dynamics and collision calculations

ODE approximates real world dynamics by means of time step discretization. For each time step, the entire dynamics world is updated according to the laws of physics or, more specifically, Newtonian dynamics. One such update is calculated by the following routine:

```
dWorldStep(worldID, timestepsize);
```

where `worldID` is the ID of the world to calculate (as mentioned above, for our simulations there is only one world containing all objects), and `timestepsize` is the size of the time step in seconds. For the simulations performed herein, the time step size is $0.01s = 10ms$.

ODE also implements another alternative to the above dynamics step function termed `dWorldQuickStep`. QuickStep uses another type of mathematical solver, and is quite a bit faster, but not as accurate as `dWorldStep`. Specifically, for singular or close to singular systems the use of `dWorldQuickStep` instead of `dWorldStep` is not recommended.

> *For large systems this is a lot faster than dWorldStep(), but it is less accurate. QuickStep is great for stacks of objects especially when the auto-disable feature is used as well. However, it has poor accuracy for near-singular systems. Near-singular systems can occur when using high-friction contacts, motors, or certain articulated structures. For example, a robot with multiple legs sitting on the ground may be near-singular. ([Smi06] p. 17)*

The point of the above quote, strongly suggesting not to use `dWorldQuickStep` for simulating the virtual creatures of this thesis, has in fact been confirmed experimentally: For certain configurations, the use of `dWorldQuickStep` introduces clearly visible instability and inexplicable creature-wide oscillations. Consequently, `dWorldQuickStep` is not used herein.

As mentioned earlier, collision detection and response is detached from the dynamics simulation, and must be initiated explicitly:

```
dSpaceCollide(spaceID, 0, &collideCallback);
```

The collision callback function `collideCallback` provided as an argument to `dSpaceCollide` is the one discussed in Section B.2.2 below. Also, equivalently to the above `worldID` in the above world stepping function, there is only one collision space containing all geoms, here represented by `spaceID`.

Finally, for each iteration, the contact group containing the contacts between objects that were colliding at the now finished time step must be cleared:

```
dJointGroupEmpty(contactGroup);
```

B.2.2 Collision callback

ODE requires the user to provide a collision callback function where diverse properties of the *contacts* between colliding bodies is specified, and these are listed in the global *contact group* for the collision system. The entire function is too verbose for appropriate inclusion here; it is provided in its entirety in the source code provided with this thesis.

B.2.3 Applying muscle forces

As described in Section 3.3.2, the legs of the virtual creature move as a result of torques applied at universal joints. The torques are calculated in logical muscle functors, and are applied in the following manner:

```
dJointAddUniversalTorques(torsoLegJointID, torque1, torque2);
```

Here, `torque1` and `torque2` are the torques applied for rotation about `axis1` and `axis2` (Figure 56), respectively, as calculated from current and recent activity at the corresponding vertically and horizontally oriented antagonistic muscle pairs.

B.3 Support functions

ODE provides a great amount of information about the current simulation state including the state of all bodies and joints. This information can, of course, be used for various purposes as the user finds appropriate. For our simulations, as described in Sections 3.3.3-3.3.4, the need and sense values providing the ANN with feedback from the mechanical model are based on information about the current and/or recent state of simulation bodies and joints. Examples follow: The torso height need is based on the world coordinates of four different points on the creature's torso; the velocity need is based on torso velocity in creature relative forward direction; the muscle length rate senses are based on joint angular velocities at torso-thigh joints.

The most important support functions used for this thesis giving information about *bodies* are:

```
dBodyGetPosition(bodyID);
dBodyGetRotation(bodyID);
dBodyGetLinearVel(bodyID);
```

which return the world coordinate position, the world coordinate rotation, and the world coordinate linear velocity of the given body, respectively.

Correspondingly, the most important support functions used giving information about *joints* are:

```
dJointGetUniversalAngle1(jointID);
dJointGetUniversalAngle2(jointID);
dJointGetUniversalAngle1Rate(jointID);
dJointGetUniversalAngle2Rate(jointID);
```

which return the joint angle for the two universal axes, and the joint angular velocity for the two universal axes, respectively.

B.4 Deinitialization and cleanup

When the lifespan of a virtual creature ends, some cleanup operations must be performed to remove all ODE objects and corresponding data structures from memory before a new creature is initialized. ODE provides some convenience functions for this purpose; at the end of a creature's lifespan the following is performed:

```
dJointGroupEmpty(contactGroup);
dSpaceDestroy(spaceID);
dWorldDestroy(worldID);
dCloseODE();
```

Here, the contact group for collisions is emptied, the collision space with all contained geoms is destroyed, the dynamics world with all contained bodies and joints is destroyed and ODE is closed, inducing potential additional cleanups.

C Program structure

This appendix reviews the static structure of the entire program system developed for the simulations of this thesis. Program structure is diagrammed by means of simplified UML class diagrams. By simplified is meant that a considerable amount of detail is left out in the diagrams, such as constructors and destructors, getters and setters, selected attributes, and function arguments. The reason for lowering the level of detail in the diagrams presented here is that reduced size diagrams are believed to communicate the important aspects of the program structure in a better way than semantically and syntactically complete diagrams can; too much detail can overshadow important parts, making valuable information hard to discover. Further, instead of having a full UML class diagram of the entire system, something that would be too large and overwhelming for sensible presentation on paper, several smaller diagrams depicting the structure of different logical modules are presented. To begin with, an overview diagram showing the main classes of the system is presented. Every diagram presented thereafter contains within it some class to be found in the overview diagram.

C.1 System overview

The UML class diagram providing an overview of the program system is given in Figure 57.

C.1.1 Summary

As described in Section 3, there are two main model components in the simulation system: the neuronal model and the mechanical model. These are interfaced by the `NNSimulator` and `Creature` classes, respectively. The `Controller` class controls the main simulation loop, acting as a mediator between the two model classes; it initiates the neuronal and mechanical step functions which update the model states, and controls the data flow of need and sense signals from `Creature` to `NNSimulator` and of motor activation signals from `NNSimulator` to `Creature`. Finally, the `Individual` class provides the `NNSimulator` and `Creature` with all parameters needed for simulation, including GA parameters, constant parameters and ANN topology specifications.

C.1.2 Class details

C.1.2.1 Controller

The `Controller` class contains the executional semantics of the main simulation loop; as described in the summary above, it initiates update calls and performs data flow management. Simulations are initialized and deinitialized by the functions `initializeSimulation()` and `clearSimulation()`, respectively. Additionally, it contains the GA semantics, i.e. the genetic algorithm itself, and thereby controls selection, recombination and mutation. Further, some ODE specific properties are to be found here, including the collision callback function and global world, space and floor IDs (see Appendix B for ODE details). The semantics for 3D visualization are also to be found here, including the `drawScene()` function which controls the actual 3D rendering. The details of the graphics model are presented in Section C.5.

C.1.2.2 NNSimulator

The `NNSimulator` class represents an interface toward the neuronal model of Section 3.2, and contains vectors of references to all clusters, neurons and synapses constituting the ANN of

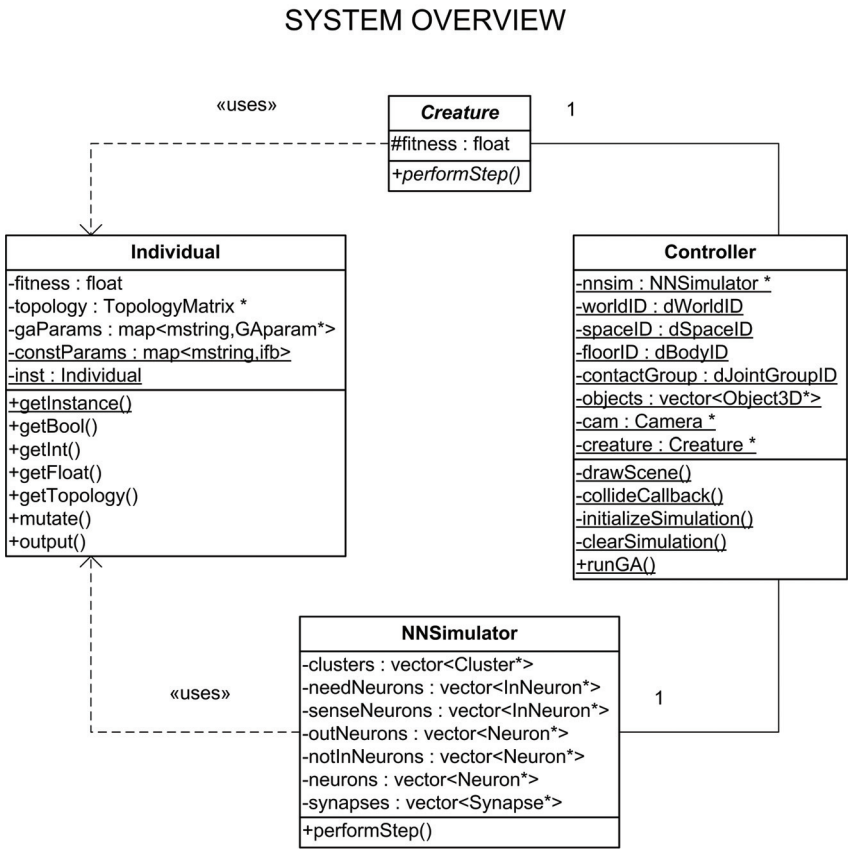


Figure 57: UML class diagram: system overview

a simulation run. It contains the semantics for setting up the ANN of a simulation based on the topology specification provided in `Individual` (contained in the constructor, not showed in Figure 57), and performing the neuronal and synaptic updates for one time step (contained in the `performStep()` function). The programmatic structure of the neuronal model is presented in more detail in Section C.2.

C.1.2.3 Creature

The `Creature` class is an abstract base class acting as an interface toward the mechanical model of Section 3.3.[83] Similarly to the `NNSimulator` class, the `Creature` class has a `performStep()` function which performs the mechanical updates for one time step. The specifics of this update function are left to subclasses of the `Creature` class containing the actual implementation of some creature. The programmatic structure of the mechanical model is presented in more detail in Section C.3.

C.1.2.4 Individual

The `Individual` class represents and interface toward the parameter model; it contains within it all variable and constant parameters used in the neuronal and mechanical model simulations, including GA parameters, constant parameters and ANN topology specifications. There is only one `Individual` object instance available to the rest of the program at any time, accessible through the static function `getInstance()`. The `mutate()` function performs mutation on GA parameters and topology specifications. Finally, the `output()` function prints the complete contents of an individual (i.e. the values and limits of all GA parameters, the values of all constant parameters, and the specification of the ANN topology) to a text file which can be used later to visualize a simulation with the exact same settings. The programmatic structure of the parameter model is presented in more detail in Section C.4.

[83] The term *abstract* implies that the class cannot be instantiated, i.e. that there can never exist run-time object instances of the class.

C.2 Neuronal model

The UML class diagram depicting the static structure of the programmatic neuronal model is given in Figure 58.

C.2.1 Summary

The main functional components of the neuronal model of Section 3.2, clusters, neurons and synapses, are realized through the three classes `Cluster`, `Neuron` and `Synapse`. In addition, there is the `NNSimulator` class, which acts as an interface toward the rest of the program system. The `NNSimulator` holds vectors of references to different groups of neurons and synapses, and through these controls ANN initialization and neuronal and synaptic updates. The `Neuron` class is subclassed by the `InNeuron` and `AffectNeuron` classes, which provide functionality specific to need and sense inputs and affect neurons, respectively. The `Synapse` class, which is abstract, is subclassed into the three synaptic learning mechanisms Skinner, Pavlov and Hume, as presented in Section 3.2, plus the constant neuronal connection type, realized through the classes `SynSkinner`, `SynPavlov`, `SynHume` and `SynConstant`, respectively.

C.2.2 Class details

C.2.2.1 NNSimulator

The `NNSimulator` class has already been briefly described in Section C.1; it acts as the interface of the neuronal model toward the rest of the program system. The semantics for generating the ANN topology (creating clusters and connecting them based on the current topology specification) are contained programmatically in the constructor of this class (not shown in the diagram). The class contains several vectors holding references to the clusters, neurons and synapses constituting the ANN: `clusters` holds all clusters, `needNeurons` holds all need inputs, `senseNeurons` holds all sense inputs, `outNeurons` holds all motor outputs, `notInNeurons` holds all neurons except inputs (needs and senses), `neurons` holds all neurons and inputs, and `synapses` holds all synapses. These lists are used when performing ANN updates; as described earlier, updates are performed in a sense that is equivalent to updating all neurons and synapses simultaneously (synchronous ANN updates, see Section 3.2.7), but programmatically, every neuron and every synapse is iterated and updated explicitly and sequentially. As described in Section C.1, the neuronal model (`NNSimulator`) receives need and sense input values from and provides motor activation signals to the mechanical model (`Creature`), all of which is communicated through the `Controller` class.

C.2.2.2 Cluster

The `Cluster` class represents artificial neuronal clusters, and holds a vector of the neurons it contains. It further contains semantics for producing intra-cluster connections among its own neurons, along with semantics for producing inter-cluster synaptic connections between its own neurons and the neurons of some destination cluster. This collective functionality is exposed through the `interconnect()` method.

C.2.2.3 Neuron

The `Neuron` class represents artificial neurons. It contains attributes that are important in the neuronal simulations performed herein, including drive values (`drive`, `previousDrive` and

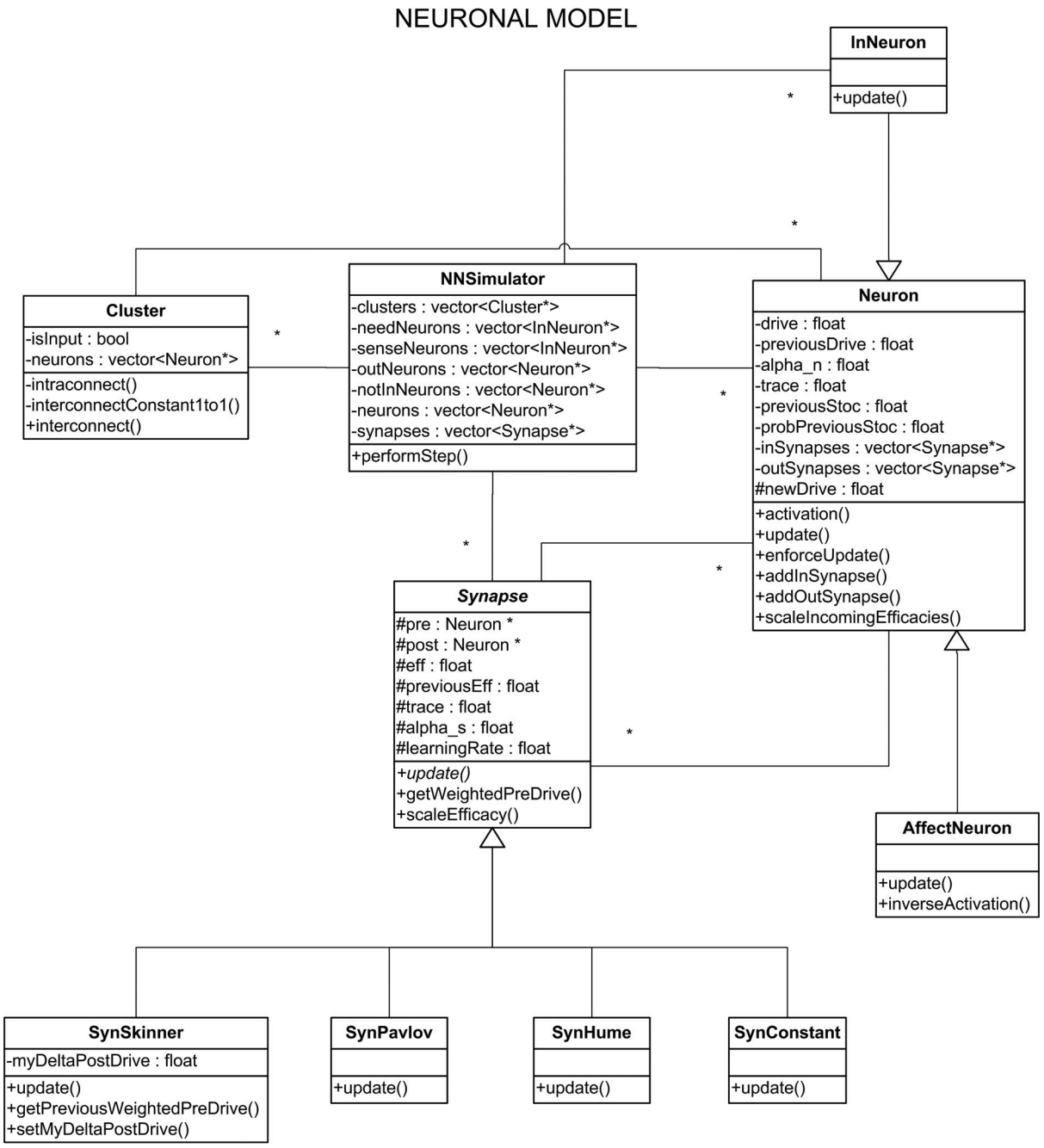


Figure 58: UML class diagram: neuronal model

newDrive), neuronal traces (trace and alpha_n) and stochastic samples (previousStoc and probPreviousStoc). Further, it offers basic neuronal functionality; the update() function performs neuronal updates based on synaptic inputs and stochastic perturbations, as calculated by the neuronal sigmoid activation function implemented in activation(). The function scaleIncomingEfficacies() is the implementation of the MOD2 synaptic learning mechanism modification presented in Section 3.2.5.4. The neuron also provides the functionality needed in setting up the ANN at initialization; the incoming and outgoing synapses to a neuron are kept track of by the vectors inSynapses and outSynapses, respectively, and the functions addInSynapse() and addOutSynapse() allow for synapses to be registered one by one.

C.2.2.4 InNeuron

The InNeuron class is a subclass of the Neuron class which overloads the update() function with an alternative implementation that allows the drive value to be set directly instead of being calculated through the activation() function. This functionality is necessary for need and sense inputs, whose drive values are set directly from the corresponding need or sense values provided by the mechanical model.

C.2.2.5 AffectNeuron

The AffectNeuron class is a subclass of the Neuron class which overrides the update() function with an alternative implementation specific to affect neurons (see Section 3.2.6.1). The function inverseActivation(), which is used in connection with update(), is simply an inverted version of the sigmoid neuronal activation function.

C.2.2.6 Synapse

The Synapse class is an abstract class representing the generic properties of artificial synapses. It contains important attributes that characterize the synapses of this thesis, including synaptic efficacies (eff and previousEff), synaptic traces (trace and alpha_s), learning rates (learningRate) and references to presynaptic and postsynaptic neurons (pre and post). The function getWeightedPreDrive() is a convenience function used when calculating the weighted synaptic incoming drives to a neuron, and the function scaleEfficacy() is used to scale the synaptic efficacy as of MOD2 (Section 3.2.5.4). The function update() is the most important synaptic function, as it determines how changes of synaptic efficacies (i.e. synaptic learning) are performed. The function is pure virtual in Synapse, meaning that each subclass must implement its own update semantics.

C.2.2.7 SynSkinner

The SynSkinner class is a subclass of Synapse representing Skinner synapses. The function update() contains a programmatic implementation of the Skinner synaptic learning mechanism. Further, the attribute myDeltaPostDrive and the functions getPreviousWeightedPreDrive() and setMyDeltaPostDrive() are used in connection with the MOD1 synaptic learning mechanism modification presented in Section 3.2.5.4, which only applies to Skinner synapses.

C.2.2.8 SynPavlov

The SynPavlov class is a subclass of Synapse representing Pavlov synapses. The function update() contains a programmatic implementation of the Pavlov synaptic learning mechanism.

C.2.2.9 SynHume

The `SynHume` class is a subclass of `Synapse` representing Hume synapses. The function `update()` contains a programmatic implementation of the Hume synaptic learning mechanism.

C.2.2.10 SynConstant

The `SynConstant` class is a subclass of `Synapse` representing constant neuronal connections. The function `update()` is empty, coinciding with the agreed property of constant neuron connections that they transfer the exact presynaptic values by means of activation function inversion instead of by synaptic transmission (i.e. learning is irrelevant).

C.3 Mechanical model

The UML class diagram depicting the static structure of the programmatic mechanical model is given in Figure 59.

C.3.1 Summary

The main components of the programmatic mechanical model are the `Creature` abstract class defining the module's interface toward the rest of the program system, and the `Nlegged` subclass of `Creature` defining the actual structure of one type of creatures. Additionally, the classes `Muscle` and `MusclePair` define the biologically inspired muscle model put to use throughout this thesis.

C.3.2 Class details

C.3.2.1 Creature

The `Creature` class is an abstract class defining the interface of the mechanical model toward the rest of the program system. It contains the attribute `fitness` which represents the fitness of the creature,[84] and the function `performStep()` which is aimed at containing semantics for applying muscle forces, calculating need and sense values, and performing fitness evaluation. The function is pure virtual in `Creature`, meaning that any subclass of `Creature` must implement its own motion, feedback and fitness evaluation semantics.

C.3.2.2 Nlegged

The `Nlegged` class is a subclass of `Creature` implementing the mechanical model of Section 3.3. The constructor (not shown in the diagram) specifies how a virtual creature consisting of a torso, a head and an arbitrary number of leg pairs (represented implicitly by `numLegs`) is built using ODE primitives and joints (see Appendix B for details on ODE). The function `performStep()` overrides the corresponding virtual function in `Creature`, and specifies how muscle forces are applied, how need and sense values are calculated, and how fitness evaluation is performed. The class contains within it references to the creature's torso (`torsoID`), feet (`feet`) and joints (`joints`); these are i.a. used in calculating need and sense values. It further contains data structures for keeping track of the feet's positions in the XZ-plane (`feetXpos`, `feetZpos` and `feetOnGround`); these are used in calculating the values for foot friction needs, as described in Section 3.3.3.7. Finally, the `Nlegged` class has a vector with references to objects of the `MusclePair` class, two for each leg.[85] Muscle pairs are used for determining what forces to apply at different joints in the `performStep()` function.

C.3.2.3 MusclePair

The `MusclePair` class represents a muscle pair, i.e. a pair of `Muscle` objects (detailed below). The two `Muscle` references are termed `agonist` and `antagonist`, alluding the agonist-antagonist naming of opposite muscles in biological systems. It further has a reference `jointID` to the joint over which its two muscles operate, making it possible for the `MusclePair` class to internally

[84] Instantaneous fitnesses are accumulated in this attribute; recall from Section 3.4.1.2 that the fitness function is accumulative.

[85] As described in Section 3.3.2, there are four muscles controlling each leg; two horizontally oriented and two vertically oriented. Consequently, two muscle pairs are needed for muscular control along these two dimensions.

fetch the ODE joint state parameters needed for muscle force calculations. For the latter, the axis of rotation about the universal joint must be known; the `MuscleType` attribute `type` specifies this directionality. Note also that extensions of this principle allow for muscle pairs to be put to use in other joint types as well, such as in hinge joints having a single axis of rotation.

C.3.2.4 Muscle

The `Muscle` class represents the biologically inspired muscle constructs presented in Section 3.3.2 and used throughout this thesis. The attributes `maxF` and `traceF` represent the maximum force this muscle can generate and the traced effective muscle force returned as instantaneous force, respectively. The function `getForce()` is the actual programmatic implementation of the muscle model presented in Section 3.3.2; it calculates and returns effective muscle force based on current and previous motor activation and mechanical state. The function `getCurrentNormalizedForce()` returns instantaneous effective muscle force as calculated by `getForce()`, but normalized to range within the interval $[0, 1]$. In other words, it provides the instantaneous degree of force generation at a muscle relative to its maximum muscle force, which coincides with the value of the muscle force needs described in Section 3.3.3.6.

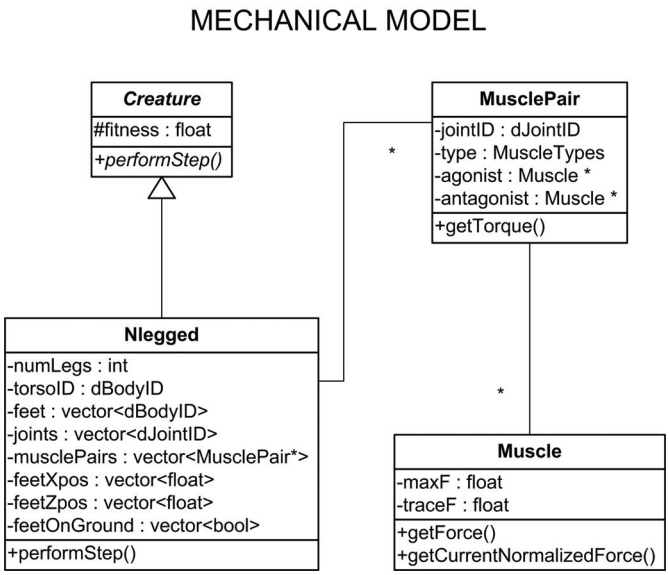


Figure 59: UML class diagram: mechanical model

C.4 Parameter model

The UML class diagram depicting the static structure of the programmatic parameter model is given in Figure 60.

C.4.1 Summary

The parameter model provides the rest of the program system with values for the numerous parameters needed in the simulations. The interface toward the rest of the system is the `Individual` class. It provides functionality for accessing all parameters needed: constant parameters, GA parameters and ANN topology specification. Constant parameters are simply represented by their values, while the somewhat more complex GA parameters are represented generically by the abstract class `GAparam`. The latter is inherited by three different types of GA parameter classes: `BoolGAparam`, `IntGAparam` and `FloatGAparam`, representing boolean, integral and floating-point GA parameters, respectively. Finally, the ANN topology specifications are represented by the `TopologyMatrix` class, to which an `Individual` object has one single reference.

C.4.2 Class details

C.4.2.1 Individual

At any time in the simulation, only one instance of the `Individual` class is active, providing the rest of the program system with the parameter values constituting the current simulation settings. This single instance, held in the static attribute `inst`, is accessed through the function `getInstance()`, which is public and static, and therefore globally available.[86] The actual parameters are represented by the two hash maps `constParams` and `gaParams`, holding constant parameters and GA parameters, respectively. Both hash maps use short strings as keys, meaning that any parameter can be looked up by specifying a keyword/parameter name. In addition to these, the ANN topology specification is represented by the attribute `topology`, which is a reference to an object of the `TopologyMatrix` class.

The first individual - the mother of all individuals - is initialized at simulation start-up by specifying all simulation parameters manually, one by one: The `Individual` class provides functionality for adding constant and GA parameters to the hash maps discussed above, as well as functionality for setting the topology specification matrix (none of these are shown in the diagram). After this initialization procedure, all parameters remain constant throughout the lifespan of one virtual creature (i.e. one simulation "instance"). Subsequent individuals are generated by means GA recombination and mutation (see Section 3.4) based on these initial parameters. The set of parameters (total number and names) and their properties, including their parameter type (GA or constant), value type (boolean, integral or floating-point) and range (minimum and maximum values), never change after initialization; it is only the value of each parameter that is subject to change. Thus, the setting up of the first individual not only determines the initial value of each parameter; it does in effect specify the contents and static properties of the entire parameter set for all subsequent individuals and generations.

Parameters are accessed through the functions `getBool()`, `getInt()` and `getFloat()`. All these functions implement the following semantics: a keyword/parameter name is provided as an argument, and the value of corresponding parameter is returned, regardless of whether the

[86] This structure incorporates central elements from the widely used *singleton* design pattern.

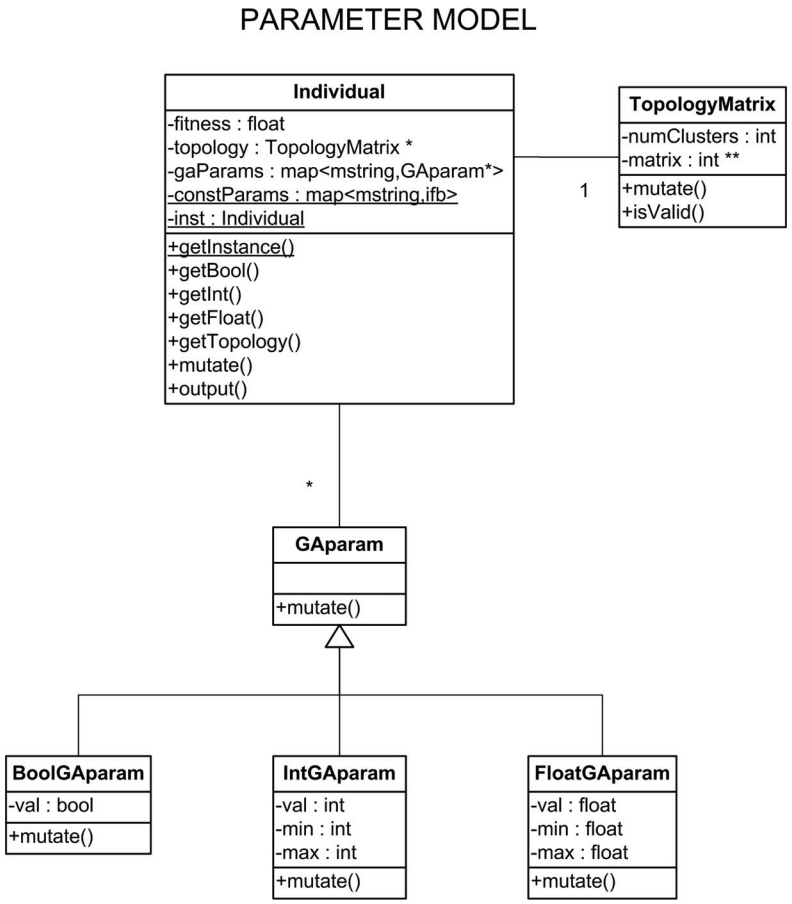


Figure 60: UML class diagram: parameter model

provided keyword pointed to a constant or GA parameter. This implies that the rest of the program system only needs to know the name (short string hash key) and type (boolean, integral or floating-point) of the parameters needed, and does not need to care whether specific parameters are constant or not. The latter also implies that parameters very easily can be changed between being constant and GA parameters; no changes are needed elsewhere in the program system. Also, new parameters can very easily be added, because their static structure including their number, names or types, is not represented anywhere except in the initialization of the first individual. The topology specification is accessed through the function `getTopology()`.

A call to the function `mutate()` initiates a mutation of the entire individual, including all GA parameters and the ANN topology specification, based on a given mutation probability and amount. The `output()` function outputs the entire contents of the individual, including all constant and GA parameters and the entire ANN topology specification, to a text file for later reference, insertion and visualization.

C.4.2.2 GAparam

The class `GAparam` is an abstract class representing the generic properties of GA value parameters. It specifies a pure virtual function `mutate()` which must be overridden by each inheriting base class.

C.4.2.3 BoolGAparam

The class `BoolGAparam` represents boolean GA parameters. It has an attribute `val` holding the boolean value of the parameter. The function `mutate()` specifies how boolean parameters are mutated in the context of genetic algorithms.

C.4.2.4 IntGAparam

The class `IntGAparam` represents integral GA parameters. It has an attribute `val` holding the integral value of the parameter, plus the attributes `min` and `max` holding the minimum and maximum allowed values, respectively. The function `mutate()` specifies how integral parameters are mutated in the context of genetic algorithms.

C.4.2.5 FloatGAparam

The class `FloatGAparam` represents floating-point GA parameters. It has an attribute `val` holding the floating-point value of the parameter, plus the attributes `min` and `max` holding the minimum and maximum allowed values, respectively. The function `mutate()` specifies how floating-point parameters are mutated in the context of genetic algorithms.

C.4.2.6 TopologyMatrix

The class `TopologyMatrix` represents an ANN topology specification in the context of genetic algorithms. It has two attributes: `numClusters` specifying the number of clusters in this topology, and `matrix` specifying the synaptic connections between clusters.[87] For each entry in the matrix, the row index represents the presynaptic cluster, and the column index represents the postsynaptic cluster. A matrix entry of zero indicates no connection, whereas positive values represent the different connection types through which clusters may connect: Skinner synapses

[87] The connectivity matrix is equivalent to the *cluster diamond* of Connectology [Hok06], which specifies synaptic connections between the clusters of (biological) neural systems.

(1), Pavlov synapses (2), Hume synapses (3) and constant connections (4). Finally, a negative entry (-1) states that such a connection is invalid (e.g. between from the affect cluster to the need input cluster).

`TopologyMatrix` has two important functions. The function `mutate()` performs a mutation on the topology specification, which may involve 1) an increase or decrease of the number of clusters and 2) changes of connectivity. The semantics for the `mutate()` function are described in Section 3.4.1.5. The function `isValid()` checks whether the current topology specification is valid. The criteria dictating the semantics of `isValid()` functions are described in Section 3.2.6.2.

C.5 Graphics model

The UML class diagram depicting the static structure of the graphics model is given in Figure 61.

C.5.1 Summary

The graphics model is a basic Open Graphics Library (OpenGL) based 3D graphics system capable of rendering and handling simple graphics primitives and navigating a camera through the scene in real-time. `Object3D` is the abstract base class representing the generic properties of graphics objects, and the subclasses `Ball` (representing a sphere) and `Box` (representing a cuboid) are the two object types currently implemented in the model. A graphics object consists of vertices and quad-shaped faces, represented by the structs `Vertex` and `Face`, respectively. The `Camera` class represents the camera from which the scene rendering viewpoint is calculated. Camera rotation is based on mathematical quaternions, represented by the class `Quaternion`. Quaternions are appropriate for free flight navigation through the scene, which is the least restrictive form of navigation, and the most suitable for our purposes. Finally, the `Controller` class controls the actual rendering, and updates the camera position and rotation based on interactive user input (mouse and keyboard). The `Controller` also gathers visualization data for all 3D objects (i.e. combined positional and rotational matrices) and outputs these to file.

In connection with the latter, in addition to the real-time visualization system discussed here, a stand-alone program system for post-visualizing the lifetime of a virtual creature is included with this thesis. This latter system, which reads the above-mentioned visualization data from file, is simply a cleaned and purged version of the graphics model presented here, and it is thus not discussed separately.

C.5.2 Class details

C.5.2.1 Controller

The `Controller` class, as described earlier, plays several important roles in the program system. In the context of the graphics model, it controls the graphics rendering. `Controller` holds a vector `objects` of references to `Object3D` graphics objects which is used in the function `drawScene()` to initiate the actual rendering of 3D objects. The `Controller` also holds a reference `cam` to the scene `Camera`, which is used in `drawScene()` to perform the geometric transformations needed to obtain a camera relative rendering viewpoint. The state of the camera, including its rotation and speed, can be changed by means of interactive user input (mouse and keyboard), and the actual callbacks that cover this functionality are found in the `Controller` class. The `drawScene()` function thus performs a complete camera relative rendering of all graphics objects, and is called once for each frame. In addition to the objects and camera discussed above, the `Controller` holds some some less important scene elements, including two stationary light sources and lines depicting world coordinate axes.

C.5.2.2 Object3D

The `Object3D` class represents the generic properties of graphics objects. It has three attributes `colr`, `colg` and `colb` depicting the object's RGB color, and arrays of `vertices` and `faces` describing its geometric properties. Additionally, each graphics object has a reference by means of a global ODE ID `bodyID` to its corresponding ODE body (see Appendix B for ODE details). This

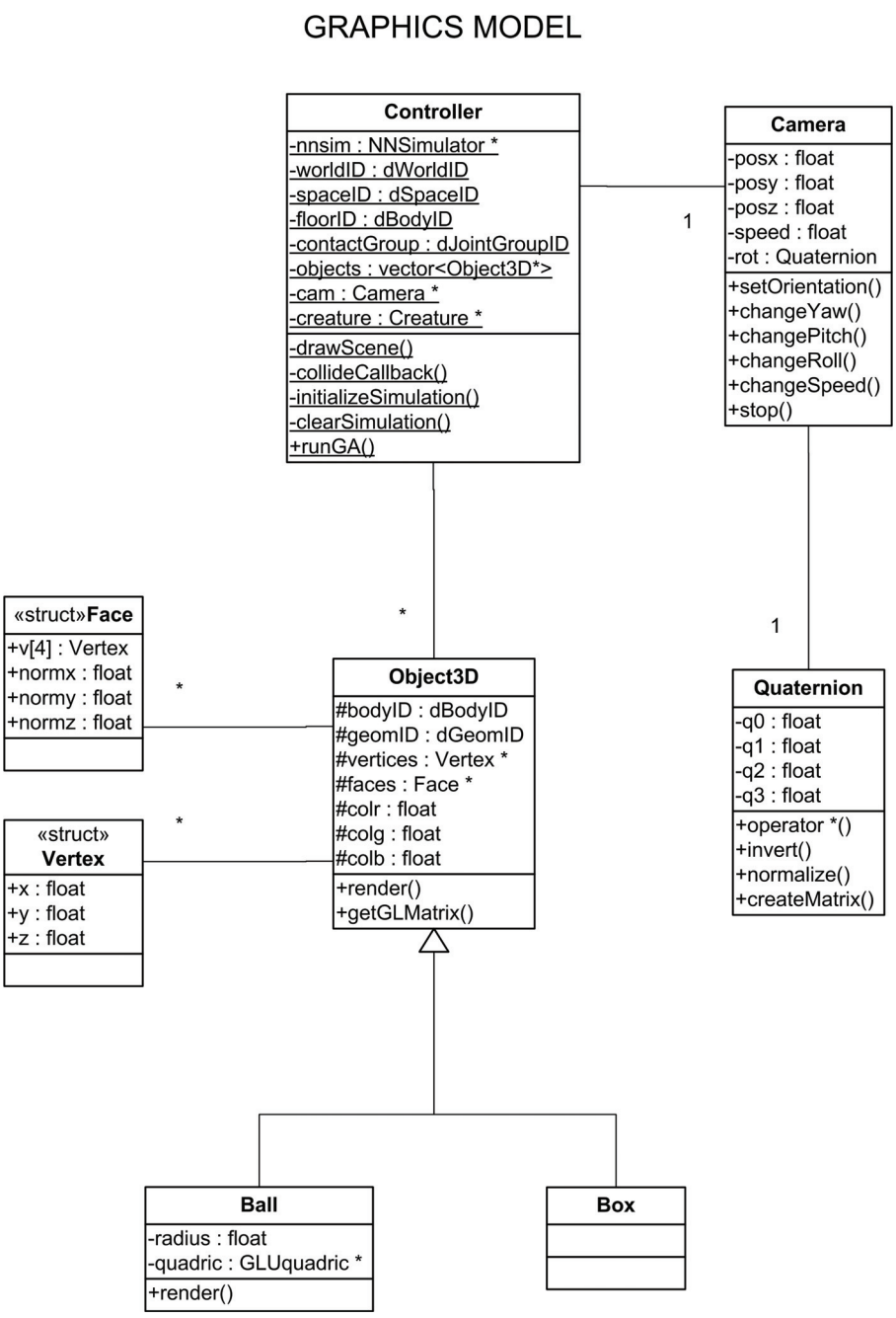


Figure 61: UML class diagram: graphics

reference can be used to fetch the object's position and rotation, which are, of course, needed in the rendering procedures, from the ODE physics engine.[88] The `Object3D` class contains a function `render()`, which is a generic rendering function that, using basic OpenGL functions, draws every quad face contained in `faces` with vertex positions adjusted to match the position and rotation of the corresponding ODE body. Finally, the class has a function `getGLMatrix()`, which gathers positional and rotational data, and returns a 4-by-4 matrix which can be used directly as an OpenGL matrix in post-visualization to specify this object's position and rotation.

C.5.2.3 Ball

The `Ball` class is a subclass of `Object3D` representing spherical graphics objects. It has an attribute `radius` which is the radius of the sphere. The `Ball` object is not rendered using the generic rendering function found in its base class; instead it overrides the `render()` function, and draws a sphere using convenience functions for drawing quadrics found in the OpenGL Utility Library (GLU, see [Khr07]).[89]

C.5.2.4 Box

The `Box` class is a subclass of `Object3D` representing cuboidal graphics objects. It has no attributes or functions in addition to those found in the base class, except its constructor (not shown in the diagram) where its entire geometric structure (vertices and faces) is set up based on given length, width and depth. `Box` objects are rendered by the generic rendering function found in the base class.

C.5.2.5 Camera

The `Camera` class represents the camera which depicts the user's viewpoint into the graphics scene. The camera has a position (`posx`, `posy` and `posz`), a rotation (`rot`) and a speed value (`speed`). These three quantities determine the current camera viewpoint and can be used to calculate subsequent viewpoints. The function `setOrientation()`, which is called from within the `drawScene()` function in `Controller`, performs the actual OpenGL geometric transformations that set the current orientational viewpoint based on position and rotation. The rotation of the camera `rot` is held in a `Quaternion` object. The `Camera` further provides functions for altering its position and rotation: `changeYaw()`, `changePitch()` and `changeRoll()` accumulate incremental changes of rotation along the three rotational axes, while `changeSpeed()` and `Stop()` change the camera speed incrementally and sets the speed to zero, respectively.

[88] One could at this point argue that, ideally, the graphics model should be completely detached from and independent of the physics engine, e.g. such that positional and rotational information is provided from elsewhere as arguments to the rendering procedures. The latter configuration would, however, in addition to introducing data flow overhead, require another global data structure holding the mappings between ODE objects and graphics objects. Alternatively, each creature could be responsible for rendering its own graphics objects, but this implies a mixing of mechanical model semantics and 3D visualization procedures, which is unfavorable considering the fact that most of the time the simulation system is run without visualization of creature behavior. To summarize, therefore, the chosen solution is considered to incorporate a propitious combination of structural and semantical simplicity and high performance.

[89] The sphere could, of course, have been implemented using the default face structure, and rendered using the generic rendering function in the base class, but 3D graphics and visualization are not the challenging or interesting parts of this thesis, and the use of utility library functions saves some development time.

C.5.2.6 Quaternion

The `Quaternion` class represent mathematical quaternions. Quaternions are four element vectors {`q0`, `q1`, `q2`, `q3`}. Quaternion mathematics needed for our purpose include multiplication (`operator*()`), inversion (`invert()`) and normalization (`normalize()`). Finally, an OpenGL rotation matrix can be generated from the quaternion using the `createMatrix()` function.

C.6 Utilities and auxiliaries

This section describes utilities and auxiliary structures used in the program system. A UML class diagram depicting the static structure of the utility classes, unions and enumerations is shown in Figure 62.

C.6.1 Summary

Utility classes and auxiliary structures such as unions and enumerations are used in the program system for clearness and convenience. They are not an important part of the program structure or architecture.

C.6.2 Class/union/enumeration details

C.6.2.1 ODEMath

The class `ODEMath` provides a static convenience function `getGLMatrix()` for converting the ODE-internal rotation matrices to the standard column major OpenGL format. The function is used in the rendering procedures of the graphics model, where ODE object rotations must be mirrored in OpenGL object rotations.

C.6.2.2 RandomSampler

The class `RandomSampler` provides three static convenience functions used in connection with randomness in the program system (i.e. for stochastic drive perturbations in neurons and for GA parameter mutations). The functions `Gaussian()` and `Cauchy()` are simply implementations of the Gaussian and Cauchy probability distributions, respectively. The function `BoxMuller()` is an implementation of the Box Muller Algorithm for sampling random values from the Gaussian probability distribution (Algorithm 1 of Section 3.2.5.2).

C.6.2.3 mstring

The union `mstring` is used for speedup in hash map lookups where strings are used as keys, as is the case the `Individual` class of the parameter model (type is `map<mstring, GAparam>`, see Section C.4). The use of a union with types `char[8]` and `ULONGLONG` as the key type in hash maps allows lookups to be based on integer comparisons instead of string comparisons, which is, of course, way faster. The implementation of `mstring` is due to Wu Yongwei [Yon02].

C.6.2.4 ifb

The union `ifb` has types `int`, `float` and `bool`, making it possible to keep all constant single value parameters in a single hash map of type `map<mstring, ifb>`.

C.6.2.5 MuscleTypes

The enumeration `MucleTypes` lists the joint types (with axes) to which muscles in the mechanical model of the program system can be connected. At the moment, there are only two types: `UNIVERSAL1` representing muscles acting at axis1 of universal joints, and `UNIVERSAL2` representing muscles acting at axis2 of universal joints. The enumeration can, of course, very easily be extended to cover other joint types, such as the hinge.

C.6.2.6 ClusterIndices

The enumeration `ClusterIndices` lists the cluster connection matrix row/column indices for need inputs, sense inputs, affect cluster and motor cluster, i.e. for the clusters types that are always contained in ANN topology specifications. The use of an enumeration for this purpose makes source code dealing with the ANN topology specifications (`TopologyMatrix`) less error prone and more readable.

C.6.2.7 EntryTypes

The enumeration `EntryTypes` lists all possibilities for entries into the cluster connection matrix used as ANN topology specifications. The use of an enumeration for this purpose ensures increased source code readability, consistency and reliability.

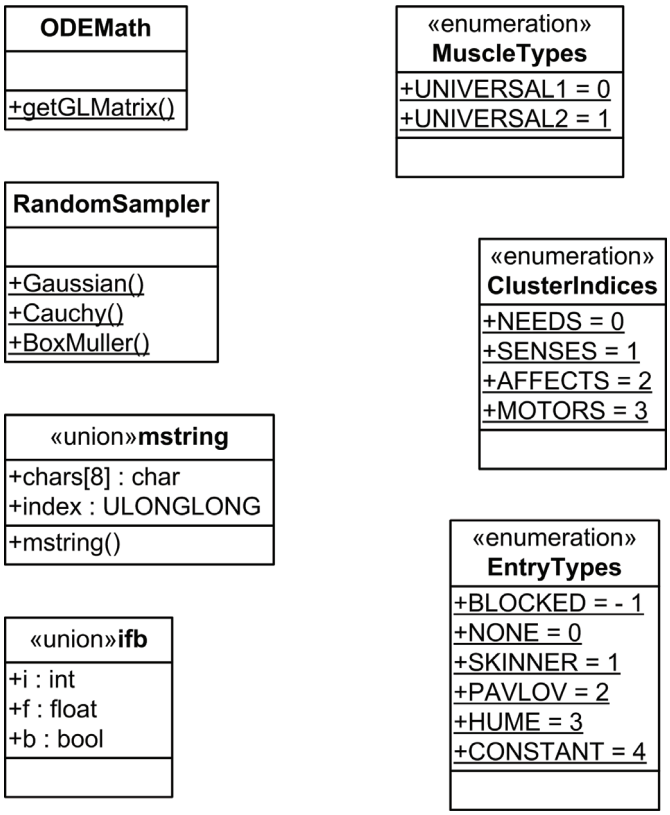


Figure 62: UML class diagram: utilities and auxiliaries