UNIVERSITETET I AGDER

Neuroevolution of Actively Controlled Virtual Characters

SVEIN INGE ALBRIGTSEN & ALEXANDER IMENES

SUPERVISOR
Morten Goodwin

University of Agder, 2017
Faculty of Engineering and Science
Department of ICT

Abstract

Physics-based character animation offer an attractive alternative to traditional animation techniques, however, physics-based approaches often struggle to incorporate active user control of these characters. This thesis suggests a different approach to the problem of actively controlled virtual characters. The proposed solution takes a neuroevolutionary approach, using *HyperNEAT* to evolve neural controllers for a simulated eight-legged character, a previously untested character morphology for this algorithm. Using these controllers this thesis aims to evaluate the robustness and responsiveness of a control strategy that changes between them based on simulated user input. The results show that *HyperNEAT* is quite capable of evolving long walking controllers for this character, but also suggests a need for further refinement when operated in tandem.

Preface

Neuroevolution for Actively Controlled Virtual Characters is a project by Svein Inge Albrigtsen and Alexander Imenes as their master thesis in IKT508 at the University of Agder.

The project is built upon and uses a game engine that we have implemented and worked on in our previous projects, including our Bachelor thesis. Our Bachelor thesis focused in part on character animation, and this project continues along the same lines, but uses evolutionary algorithms to automate the animation and movement of the character. We have both long been fascinated by evolutionary simulations such as the work of Karl Sims [1], [2]. We have therefore found it very interesting to be able to learn about the diverse field of neuroevolution and to try it out on our own.

We would like to thank our supervisor and coordinator, Associate Professor Morten Goodwin for his guidance and support throughout both our master thesis and master degree.

Grimstad,

May 22, 2017

Svein Inge Albrigtsen and Alexander Imenes

Contents

List of Figures

List of Tables

Chapter 1.

Introduction

Character animation has become an important part of modern game development and the responsiveness and perceived realism of these animations play an important role in the games ability to provide an immersive experience to its audience.

While an experienced animator is often able to create lifelike and realistic looking animations, the limitation of these animations are that they are only applicable for the purpose that the animation portrays. When these animations are used in situations that are outside of their intended domain, however slightly, they start to fall short. For example, a running animation created for running on flat terrain is not likely to look equally perfect when used to portray running on bumpy terrain or up a flight of stairs.

Seeing as game characters are often interactive, the sheer number of possible variations of the actions they should be able to perform makes it challenging to create separate animations for every conceivable scenario. While techniques like motion capture makes it possible to quickly record a large number of animations, these animations still suffer from the same limitations as manually crafted animations. Furthermore, motion capture is mostly limited to humans and other creatures that can be persuaded to participate in motion capture [3].

Physics-based simulation offer an attractive alternative to traditional animation techniques, wherein each motion is the direct result of a physics simulation and is therefore physically realistic by definition [3], at least within the limitations of the physics simulation itself. Physics-based animations are commonly used to simulate passive phenomena

like objects, cloths, fluids and ragdolls. However, for more active animations most games still resort to kinematics-based approaches [3]–[5]. One of the commonly sited reasons for this is that physics-based simulated characters are notoriously difficult to control, as all movement has to be controlled by the application of torques and/or other forces [3]. One way of approaching this issue is to train controllers using machine learning, a technique that has shown promising results [1], [6]–[14]. However, such controllers are usually trained for one singular purpose, or action, and bare little application outside of that purpose.

While one approach could be to train multiple controllers and then switch between them, the actual implementations of this logic is sparse. It is therefore not known whether or not these controllers could handle the incessant switching that would be required inside a highly interactive environment like a game.

1.1. Problem Statement

This project aims to investigate the feasibility of combining individually trained neural controllers to form a control strategy for actively controlled virtual characters.

The proposed approach taken to this end utilizes neuroevolution to train a small set of neural controllers. These controllers will be used to generate joint torques for a physics-based simulated multi-legged character in order to produce motion for a corresponding set of target behaviours. Finally the efficiency of these controllers can be evaluated based on how robustly they perform when switching between them.

The main goal of this thesis is twofold. Firstly, evaluate whether or not *HyperNEAT* is capable of evolving gaits for an eight-legged character, which to the best of the authors knowledge has not been attempted before. Secondly, evaluate whether or not neural controllers trained for different behaviours can operate in tandem to produce robust interactive animation controllers.

1.2. Contributions

This thesis theorise an approach to physics-based character animation control using evolved artificial neural networks to produce joint torques for a physics-based simulated character. The contributions of this thesis mainly resides in the approach taken to train these networks, by using *HyperNEAT* to generate gaits for a previously untested, highly complex, eight legged character, and provide insight into the efficiency of constructing high-level controllers using such evolved networks.

Chapter 2.

Background

This chapter provides a brief introduction to some of the topics relevant to this thesis. Particularly legged locomotion, a topic that has gained a lot of interest in the field of robotics, and neuroevolution with focus on the algorithms used in this project, namely NEAT and HyperNEAT.

2.1. Legged Locomotion

Legged locomotion offers several benefits over wheeled locomotion. Particularly in the field of robotics where the prospect of increased manoeuvrability and mobility on difficult terrain [15] offer applications in a wide variety of domains, ranging from entertainment robots [16] to military or industrial transportation, the exploration of disaster areas [15] or even planetary exploration [17].

However, a major drawback for such robots is the time and difficulty involved in creating effective controllers for them. The high number of degrees of freedom for each leg and the coordination required to maintain balance makes this a very difficult task for human engineers. Further considering how sensitive such controllers are to changes, effectively requires these controllers to be recreated every time the robot is modified [18], it should be to no surprise that researchers have long tried to automate this process.

One alternative to manually engineering such controllers are to train them using machine learning. Evolutionary algorithms, typically involving the evolution of artificial neural

networks, i.e. neuroevolution, are commonly used for this end and can often create better gaits than those created by human engineers [2], [8], [10], [12]–[14], [18]–[22].

2.2. Neuroevolution

Neuroevolution is a class of machine learning techniques that uses evolutionary algorithms to generate artificial neural networks, drawing inspiration from the biological evolution of nervous systems in nature.

Most common, non-evolutionary, artificial neural networks are trained using back-propagation, a form of supervised learning that provides continuous feedback to the network. However, this form of learning imposes limitations on the network being trained. Firstly, the size and topology of the network must be specified in advance, i.e.: the number of hidden layers, the number of neurons in each layer and the connections between each neuron. Secondly, this form of training also depend upon available datasets of input-output pairs, where each output represents the optimal action or correct answer for a given input. This further limits the domains where such networks can be used to those cases where an optimal strategy for reaching the target goal is known.

Neuroevolution is an approach which aims to address these limitations. Similar to biology and natural selection, which is driven only by reproductive success, neuroevolution evaluates fitness based on some measure of overall performance. The main advantage of such an approach is that it facilitates learning with only sparse feedback and without any information about exactly what it should be doing at any given point. For example, when playing a game the optimal action at any given time may be ambiguous, as there might not be one single correct action, but rather multiple actions that can lead to a successful outcome. In such a scenario it would be impossible to judge the fitness of each individual action, but it would be trivial to judge the fitness of the sum of all actions once the game concluded and a victor was declared.

Early neuroevolution algorithms focused only on evolving the weights of each neuron's connections, however, the network topology also has a huge impact on the performance of the network [23], [24], though more advanced algorithms are also capable of evolving

this [25]. The algorithms that are able to evolve the network topology have been shown to hold a significant advantage over their fixed topology counterparts in a multitude of domains, with examples ranging from pole balancing [26], vehicle control [27] and collision warning systems [28] to various video game controllers [29], [30].

Neuroevolution is a very general learning technique with broad applicability and can be used for supervised, unsupervised and reinforcement learning tasks [31]. When compared to other reinforcement learning techniques, like temporal difference learning, neuroevolution has been shown to be slower, but eventually reaches higher performance [32], [33].

2.2.1. Algorithms

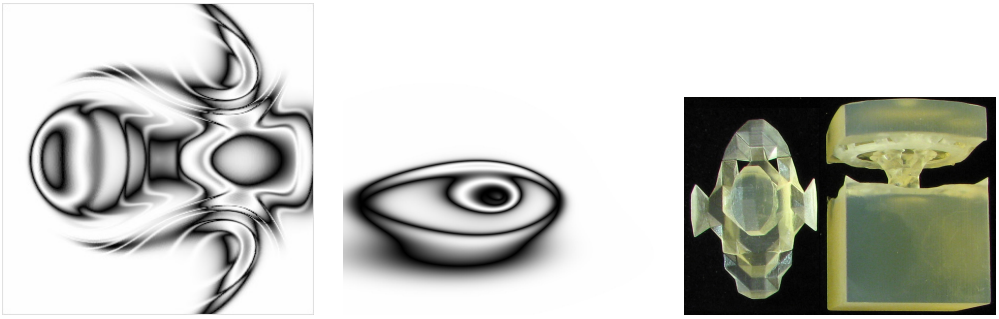
Neuroevolution algorithms work by taking a population of genotypes, i.e. a set of genetically encoded neural networks, and testing each one on the given task for a certain period of time. During this test period the fitness, i.e. the measure of how well the network performs, is recorded. Once the fitness for each member of the population has been obtained, offspring are generated by applying mutations and/or crossovers between the fittest individuals. These offspring then replaces the individuals with the lowest fitness, forming a new generation. This loop is then repeated until a network with a sufficiently high enough fitness is found.

Some algorithms, like the *NeuroEvolution of Augmented Topologies* or NEAT algorithm, allow the population to split into separate species. This facilitates the algorithm to explore new innovations that initially may have resulted in a lower fitness, but that could eventually lead to a better solution than the current optimum. Other species that are not improving, indicating it may be stuck at some local optima, or that are not performing well can go extinct. [25]

More recent research in neuroevolution algorithms have focused much on the encoding used for the genotype, i.e. the manner in which the algorithm stores the parameters (weights, connections, etc.) of the neural networks it evolves. Early algorithms, including the aforementioned *NEAT* algorithm, use a direct encoding. This form of encoding effectively employs a one-to-one mapping between each parameter and the value encoded in

the genome. The drawback of this approach is that similarities in the problem must be discovered separately [14], [31], [34], [35]. For instance, to evolve a controller for a legged robot, a solution that was effectively able to control one of its legs is likely going to be very similar to the solutions that control its other legs, yet with a direct encoding these solutions must evolve independently for each leg.

Conversely, biological organisms in nature does not use such an encoding scheme, but rather uses an indirect form of encoding [35], [36], wherein large parts of the information encoded in the genome are reused to make up the various parts of the organism. Drawing inspiration from nature, the development of indirect encoding schemes have become an important research topic in neuroevolution. The basic idea is that using such a scheme could enable the discovery of patterns in the problem and allow for the reuse of solutions across systems comprised of such patterns [14], [36]. By using an indirect encoding the genotype can be more compact than the artificial neural network it encodes, resulting in fewer variables for the algorithm to optimize [8], [37]. Multiple studies have also shown that indirect encodings are more effective at problems containing such regular patterns [8], [10], [20], [34]–[36].



(a) bilateral symmetry and repetition [38]

(b) imperfect symmetry [39]

(c) example with 3D shapes [40]

Figure 2.1.: Spatial patterns evolved with CPPNs showcasing different forms of regularities

One prominent algorithm commonly employed as an indirect encoding is the *Compositional Pattern Producing Network* or CPPN. A CPPN is a form of artificial neural network that differs in the set of activation functions it employs [31], [38], [39]. Where standard artificial neural networks often only contain sigmoid functions, CPPNs additionally incor-

porates a set of different functions to produce various types of patterns and regularities [31], [38]. Some examples include Gaussian, to produce symmetric patterns, and sinusoids to produce repeating patterns [31], [34], [41]. Composing these functions allow the CPPNs to produce patterns in n-dimensional space from a set of n inputs, where each set of inputs describes a point within that space [38]. For instance, to produce a two-dimensional image the CPPN may be queried with the (x, y) position of each pixel and have the output(s) specify the shade and/or the colour of the pixel [38], [40]. A three dimensional CPPN, i.e. a CPPN with three inputs, may be similarly queried to produce three-dimensional-shapes [40]. Figure 2.1 shows some examples of images and shapes generated using CPPNs.

What makes CPPNs ideal candidates for use as indirect encodings is that they span their entire domain of possible input values. Just like a CPPN may be queried for each pixel position in order to create a image, this sampling may be performed at any resolution, making it an efficient encoding for that image at an infinite resolution [38], [41]. Crucially, because CPPNs are structurally similar, algorithms for evolving normal artificial neural networks, such as NEAT, may be used to evolve CPPNs as well [34], [38], [39]. Furthermore, CPPNs are not limited to producing visual patterns as images or shapes, rather its output may be purposed for a multitude of different uses [39], [41], e.g. to describe the weights and connections of an artificial neural network [34].

One particular algorithm that utilizes CPPNs in this way is the *Hypercube-Based NeuroEvolution of Augmented Topologies* algorithm, or HyperNEAT for short [34]. Specifically, HyperNEAT makes use of CPPNs to paint connectivity patterns for large-scale artificial neural networks. The key insight that allows HyperNEAT to realize this is that connectivity patterns, i.e. connections between neurons, may be encoded as a function of each neuron's position in space. However, a connection between any two neurons cannot simply be described as a function of the position of either neuron, but rather requires the position of both neurons [39]. HyperNEAT solves this by representing the problem in a higher-dimensional space, by using a higher-dimensional CPPN to paint patterns in hyperspace. These patterns may then be projected back as connectivity patterns by querying the CPPN with the pairwise combination of every neurons position. For example, for two neurons situated on a two-dimensional plane at position (x_1, y_1) and (x_2, y_2) a four-dimensional CPPN may be queried with the combined position (x_1, y_1, x_2, y_2) in order to find the

weight of the connection between them [14], [34], [39]. HyperNEAT may similarly create connectivity patterns for network topologies expressed in three-dimensional space, by querying the CPPN with the combined three-dimensional inputs: $(x_1, y_1, z_1, x_2, y_2, z_2)$, i.e. using a six-dimensional CPPN. This is especially interesting as it suggests that topologies like those found in the brains of biological creatures theoretically exist within its search space [34].

Because neurons in HyperNEAT exists at precise points in space, aligning these neurons with the geometry inherent to the problem domain enables HyperNEAT to exploit the geometric relations in the problem [34], [37], [42], [43]. For example, in a game like chess or checkers the relative position of each square of the game board plays an important role in the understanding and mastery of the game, and being able to recognize the adjacency relationships between the squares of the board makes it possible to learn general tactics rather than actions tied to specific squares [42]. HyperNEAT's general encoding allows it to use this adjacency information to paint a connectivity pattern across the entire board rather than having to learn this information one square at a time.

Furthermore, HyperNEAT's abilities has been shown applicable in a variety of different domains: Controlling multi-agent teams in a predator-pray simulation [44], [45], playing checkers [37], [42], creating car controllers [46], [47], and creating controllers for legged robots [10], [12]–[14], [22], [35]. On the common RoboCup Keepaway soccer reinforcement learning benchmark HyperNEAT outperformed all previous learning algorithms [48]. HyperNEAT as also been shown to be able to play general Atari games from raw pixel data, even beating human high-scores on several games [49].

One limitation of HyperNEAT is that the location and role of each neuron connected by the CPPN must be decided beforehand by the user. This creates a new task for the user, wherein the user must manually position every input, output and hidden neuron in the network inside a separate space called the *substrate*. While it is often easy to find some way of positioning the input and output neurons in a way that represents the domain geometry, it is often more difficult to determine the placement and number of hidden neurons [50], [51]. An extension to the HyperNEAT algorithm called *Evolvable-Substrate HyperNEAT* or ES-HyperNEAT [50] alleviates this problem by using the CPPN itself to determine the number and placement of hidden neurons. As areas of the CPPN

with uniform weight ultimately encode very little information, ES-HyperNEAT attempts to best capture the information stored in the CPPN by searching and positioning neurons in regions of high variance [52].

Chapter 3.

Related Works

One of the earliest examples of physics-based character animation came in the inspiring work of Karl Sims in 1994 [1], which used genetic algorithms to evolve both the morphology and neural controllers of various simple three-dimensional creatures. In this work Sims managed to evolve his creatures to swim, walk, jump and follow a light source, and later evolved them to compete over the control of an object [2].

3.1. Bipedal

Reil and Husbands [7] evolves a fixed topology recurrent neural network for a three dimensional bipedal controller. Their model is a simplified lower-body humanoid with two joints per leg, a hip and knee joint. The evolved controller generates target joint angles that produce a stable cyclic walking motion for the biped.

Allen and Faloutsos [9] utilize the NEAT algorithm to evolve controllers for a more complex humanoid model. In addition to the hip and knee joints the evolved controllers also control both the spine and ankle joints of the model, by setting target joint angles. Further to promote symmetry the controller is constructed of two identical controllers reflecting the bilateral symmetry found in humans. However, while showing smooth and believable human-like motion, their controllers were generally not stable for more than a few meters before falling over.

Olson [11] uses a similar bipedal model to that of Reil and Husbands [7], but uses HyperNEAT for the optimization process. Further to facilitate oscillatory gaits the produced controllers makes use of a *Continuous Time Recurrent Neural Network* or CTRNN. Though the most novel approach in this work might be the use of the *Novelty Search* fitness calculation, that instead of rewarding a direct target goal simply rewards individuals that are different. The produced controllers show stable oscillatory gaits, but are on average not capable of walking further than previous bipedal experiments evolved with NEAT.

Hagenaars [5] uses NEAT to evolve a developmental hierarchy of controllers to generate joint torques for a full body humanoid character. Each controller, *posing*, *standing* and *reaching* is trained in succession, in increasing order of complexity, each building upon the previous lower level behaviour, creating a combined controller that can do all the various actions. The user may also toggle individual sub-controllers on or off changing the behaviour of the controller. As such this work is the most similar to the work presented here in.

3.2. Quadrupedal

Clune, Beckmann, Ofria and Pennock [10] compares HyperNEAT and FT-NEAT on producing gaits for a simple quadrupedal model. The model features a planar surface with four legs and three joints per leg for a total of 12 joints. Results showed that HyperNEAT were capable of producing smooth, natural looking gaits far outperforming those evolved by FT-NEAT.

Yosinski, Clune, Hidalgo, Nguyen, Zagal and Lipson [12] used HyperNEAT to train gaits for a physical 3D printed robot directly on the hardware. The robot, also known as the *QuadraTot*, features two joints per leg and a central joint rotating the entire upper body in relation to the lower for a total of nine joints. Results showed that gaits evolved with HyperNEAT outperformed the previous hand coded controller for the same robot.

Lee, Yosinski, Glette, Lipson and Clune [13] shows the benefit of using HyperNEAT in simulation to evolve gaits for the QuadraTot robot before transferring the controller to the physical robot.

Morse, Risi, Snyder and Stanley [14] uses an extended version of HyperNEAT to train gaits for a simple quadrupedal model. The model features a rectangular body with four legs and tree joints per leg for a total of 12 joints. The novel idea in this work is the *Single-Unit Pattern Generator*, or SUPG, HyperNEAT extension. This extension is a new type of neuron that produces temporal activation patterns that can be reset and repeated on various triggers such as a leg touching the ground. The SUPG extension aims to provide an alternative to other commonly applied methods for achieving oscillatory motion like CTRNNs or inputting sinusoids directly into the neural network. The results show controllers producing stable oscillatory gaits that are capable of continuous walking far beyond the training window.

3.3. Covariance Matrix Adaptation

A different approach to the problem of physics-based character animation that has resulted in a variety of impressive animation controllers [53]–[55] is using the *Covariance Matrix Adaptation* or CMA evolution strategy [56]. However, this is also purportedly very difficult to implement and do not mix well with common physics engines, and also highly computationally expensive where most implementations usually reports real-time performance for only a single character at a time [57].

Chapter 4.

Approach

This chapter details the approach and methods used in this project, most notably the setup of the physics simulation and design of the character model and substrate.

4.1. Multi-legged Character

This project uses a custom OpenGL-based game engine. This engine was built prior to this project and has been adapted to incorporate a physics engine and neuroevolution library in order to simulate and train the controllers needed for this thesis. The engine itself is mainly used for visualization and to control the flow of the physics-based simulation.

The Bullet3[1] physics engine is used to perform all rigid body simulation and collision detection required to simulate the character. The settings of Bullet has been tuned to approximate real-life physics as closely as possible in order to provide a realistic environment. A full list of these settings can be found in Appendix A Settings.

To evolve the controllers, the neuroevolution library MultiNEAT[2] is used. MultiNEAT features an implementation of NEAT, HyperNEAT and ES-HyperNEAT and is one of several highly recommended libraries with good track records [58]. The choice of MultiNEAT in particular was made due to language compatibility with the rest of the project code. In this project MultiNEAT is used for its implementation of HyperNEAT. While

[1] http://bulletphysics.org
[2] http://multineat.com

Figure 4.1.: Character in a upright position.

ES-HyperNEAT sounds like a very promising algorithm, it is still relatively new and have not previously been tested for evolving locomotion controllers.

In addition to these libraries a complete list of all libraries used in this project as well as the source code for the project itself can be found on the authors github repository[3].

One of the elements that was kept from the original game engine is the character model. This model is a arachnoid with a robotic theme, featuring eight legs, each with five different joints. It also has a head, an abdomen and a sternum, where the latter is what all the legs are connected to. This model was chosen as it had a sufficient complexity to provide a realistic example of a fully featured game character.

All the joints of the character are defined as hinge joints, meaning that they can only rotate in one specific axis. Furthermore, all of these joints have limitations in how much they are able to move in their respective axis. This is to enforce realistic motions as the joints should not have a full 360 degree freedom of movement. Figure 4.2 showcases several legs in various different angles that are allowed, where Figure 4.3 shows each joint and their limits. As seen in Figure 4.3, Trochanter is the only joint able to rotate forwards or

[3] https://www.github.com/reewr/master

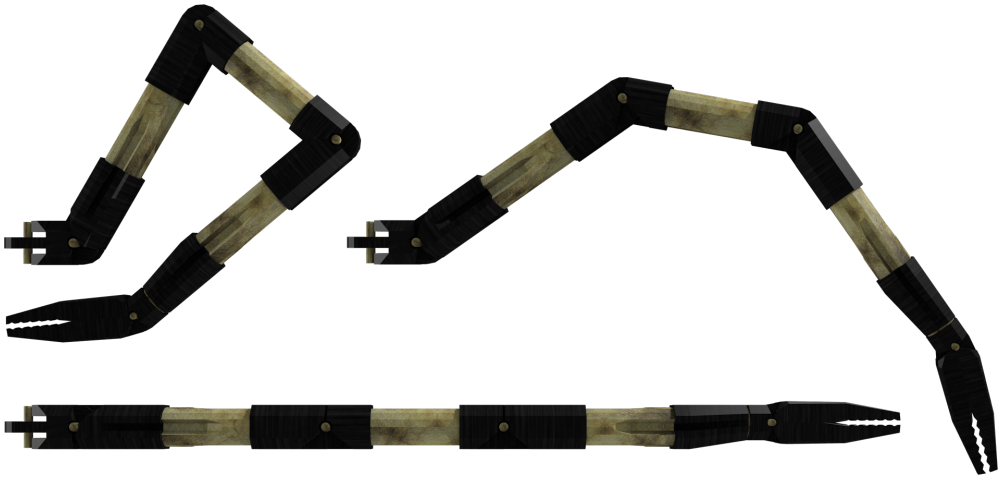


Figure 4.2.: One of the legs in various poses.

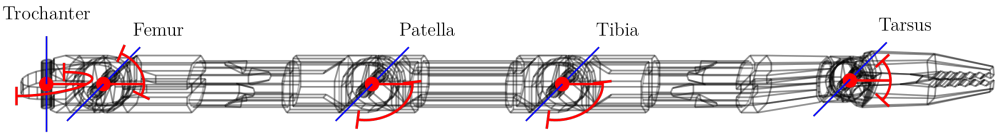


Figure 4.3.: Leg joint limits.

backwards in relation to sternum, where as every other joint can only rotate up and down in relation to sternum. The actual values of these limits can be seen in Table 4.1, where each row represents the angle going from left to right in Figure 4.3.

Name	Upper Limit	Lower Limit
Trochanter	-60	60
Femur	-20	60
Patella	-100	5
Tibia	-100	5
Tarsus	-35	35

Table 4.1.: Upper and lower limits of each joint on the legs of the character.

The character has four additional parts that are not mentioned in Table 4.1, the head, neck, hip and abdomen. These parts may also be rotated, however, these parts were not consid-

ered to contribute much in terms of the movement of the character and instead added more complexity to an already complex character. These joints have therefore been disabled and set to a constant angle that should not interfere with the movement of the character.

The dimensions of the character is roughly 4 units from its head to the back of the abdomen and has approximately 9 units in total leg span when fully stretched out.

Each part of the character has a weight of 1 mass unit, except for the sternum, abdomen and head which weighs 10 mass units, 8 mass units and 5 mass units respectively. These units were set to make sure that the centre of mass was not exactly at the centre of the character, but rather slightly tilted towards the back, in order to increase the perceived realism of the character.

4.2. Substrate

As mentioned in Chapter 2 Background, HyperNEAT uses a substrate to define the coordinates for input, hidden and output nodes. Designing the substrate is an important aspect of using HyperNEAT as it contributes to how it will determine symmetries and patterns [43].

The substrate is made to match the geometry of the character as closely as possible to make it easier for HyperNEAT to detect the symmetries of the legs. The substrate takes inspiration from previous research that has evolved gaits for Quadruped [10], [12], [13], [35], but is extended to account for the additional two legs on each side and the increased number of joints. The substrate with all its layers can be seen in Figure 4.4, which is defined with three two-dimensional 7×8 Cartesian grids. The first of these represent the input layer, the second is the hidden layer and the third is the output layer. All unmarked inputs/outputs are unused.

Each column in the substrate represents a leg, starting from the front left and going clockwise around the character. Each row, except for the two top most rows, represents the current angle of a joint normalized from a value between $[-\pi, \pi]$ to a value between $[-1, 1]$. The second row represents whether or not the tip of the leg is touching the floor where as the first row include the pitch, roll and yaw of the sternum. The first row also

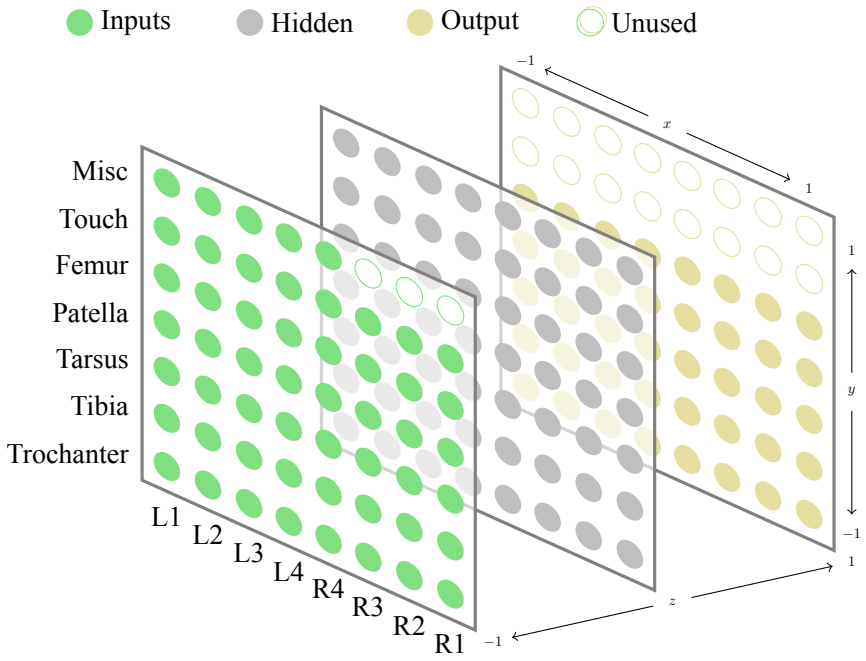


Figure 4.4.: The three-dimensional substrate of the character

include a sine and cosine wave to encourage periodic behaviour. All the inputs, hidden and output coordinates of the substrate are spread uniformly in the range of $[-1, 1]$, while trying to keep symmetry between opposite legs. To differentiate between the layers, the inputs, hidden and output layers are placed on different z-coordinates, where the input z-coordinate is -1, hidden z-coordinate is 0 and output z-coordinate is 1.

The outputs are expected to be of the range $[-1, 1]$. The current angle of the joint is subtracted from the output and the result of this is set as the new velocity of the joint. This simulates setting a target angle of the joint, allowing the networks to choose their desired angles of each joint.

4.3. Controllers

In order to create the control strategy, the neural controllers for each action has to be trained first.

The first controller is trained to make the character stand completely still in a standing position. While training, it will be awarded for not moving away from the initial starting position and for remaining still in a balanced pose. If the character falls at any point during the simulation, i.e. touches the ground with any vital parts, the simulation will be terminated and the character will be rewarded for the length of time it stayed alive. This mainly to discourage such behaviour by limiting the fitness of individuals that are unable to remain upright throughout the entirety of the simulation, promoting more stable postures.

The second controller is trained to make the character walk with a stable gait. Working under the assumption that the further the character walks the more stable the gait must be. Therefore, the fitness of the character will be the furthest distance travelled in one specific axis, in this case the positive z-axis, which is the axis the character is facing at the start of the simulation. As with the first controller, if the character falls, the simulation will end and the final fitness value will be the furthest distance travelled before it fell.

The particular reason for choosing these two actions was that together they form a minimum of actions required to test the control strategy. Walking gaits have been evolved for various legged creatures before and thus evolving this behaviour for the character used in this project should hopefully not be too much of a stretch, considering the characters heightened complexity. While it would easily be possible to not have a separate standing controller, by just locking all the joints of the character instead, this could possibly cause the character to fall over if it happened to stop in some unbalanced position. Thus the need for a separate standing controller that could account for such imbalance and other residual forces left over from the walking controller after a transition.

Both of the above controllers will be trained with HyperNEAT using the substrate described in the previous section and use the same input and output scheme as describe above.

The simulation process will be the same for both the controllers and they will only differ in the fitness functions that they use. All simulations will be run with 150 individual with randomized weights based on a random seed for 300 generations. Each character will be allowed to run for up to 10 seconds in real time, in addition to an un-simulated process of one second where the character is positioned into a standing pose that is equal across all simulations.

Due to the complexity of HyperNEAT and its underlying NEAT algorithm, it has a variety of different parameters to set, most of which have been heavily based on previous research for similar multi-legged characters [10], [12], [13], [35]. However, these may be subject to some degree of trial and error depending on the results. A full list of the NEAT/HyperNEAT parameters can be found in Appendix A Settings.

4.4. Control Strategy

Once these two controllers are in place, the control strategy can be evaluated by using the above controllers together. To do this, two experiments have been devised.

The first experiment is designed to measure the responsiveness of the controllers. This is done by using the controllers in sequence and measuring the time it takes to transition from standing to walking and vice versa.

The second experiment is aimed to test the robustness of the controllers under more intensive switching. In this experiment the controllers will be sequenced in a loop at random time intervals to see if the character starts displaying behaviours it was not train for.

Chapter 5.

Results

This chapter details the results of the standing controller, walking controller and the control strategy. It also details any deviations from the approach detailed in Chapter 4 Approach.

The results presented in this chapter uses image series to convey the movement and motion of the controller as correctly as possible, therefore the time between each image may be varied. The number of seconds into the simulation is displayed in the top corner. However, one can only convey so much motion through a set of images. For a more accurate representation of each controller, it is highly recommended to look at the corresponding videos on the authors github page[1], if possible. The page displays the videos in the same order as the image series appear in this chapter.

5.1. Standing Controller

The standing controller was evaluated based on its ability to stand still and not falling to the ground. While many of the evolved controllers were capable of keeping balance and remaining still very few of them did so in an intuitive manner.

In early generations of the simulation the dominating factor by far was the killing of individuals who fell. In the first generations rarely did even a single individual live long

[1] https://reewr.github.io/master

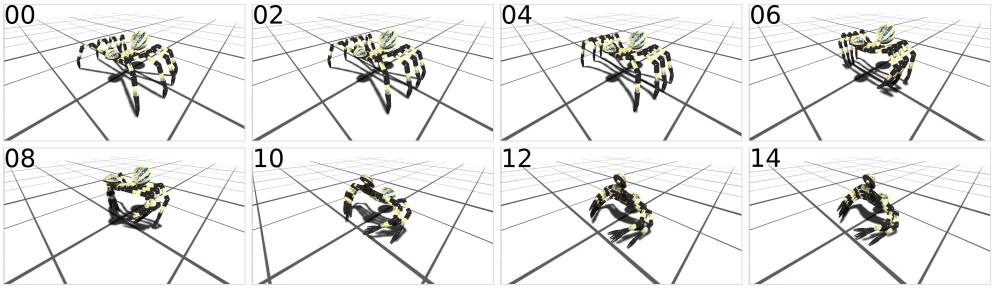


Figure 5.1.: Image series highlighting the first standing controller.

enough as to complete the full simulation duration. This led to a dramatic increase in simulation speed in the early stages as it allowed the simulation to be cut short and the next generation could be started.

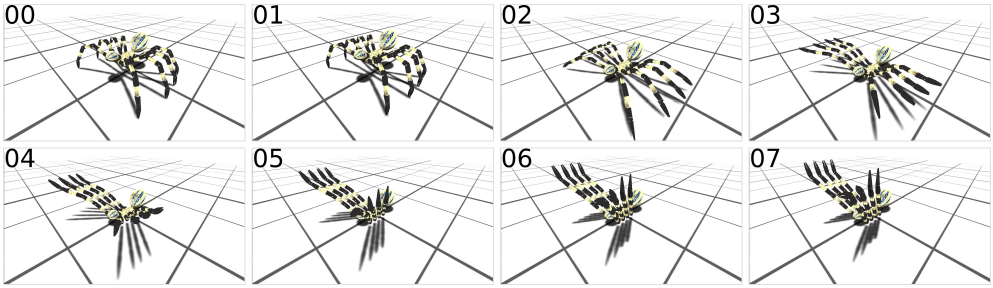


Figure 5.2.: Another image series highlighting the first standing controller.

Seeing as the controllers were directly rewarded for how long they remained alive it was not surprising that most of the controllers managed to learn that staying upright was a good strategy. However, many controllers had trouble going beyond that and often flailed about frantically in order to remain upright. An example of this can be seen in Figure 5.1. This controller tries to remain upright by pulling all its legs towards its body, however, it appears to do so with an equal force across all legs. Seeing as the character's centre of mass is slightly towards the back this symmetrical application of force leads the character to tilt, eventually falling backwards.

One observed phenomenon that became apparent after the controllers was run outside the training environment was that only slight, imperceptible differences in the simulation could lead to totally different results. As an example Figure 5.2 shows the exact same

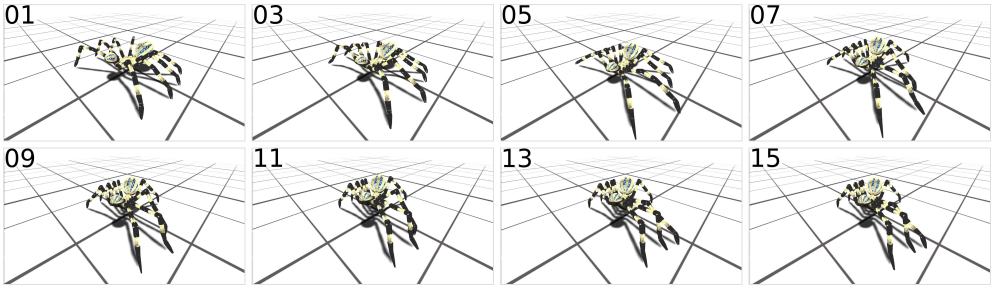


Figure 5.3.: Image series highlighting the second standing controller.

controller as seen in Figure 5.1, but display a radically different behaviour. This phenomenon stems from the fact that the physics engine is not deterministic due to rounding approximations and floating point errors that can make a huge impact later in the simulation. In order to reduce the impact of this effect on the training simulation, HyperNEAT was reconfigured to allow cloned individuals. This allowed identical individuals to exists simultaneously in the population, giving it multiple attempts to prove itself, decreasing the chance that individuals would go extinct due to these inconsistencies.

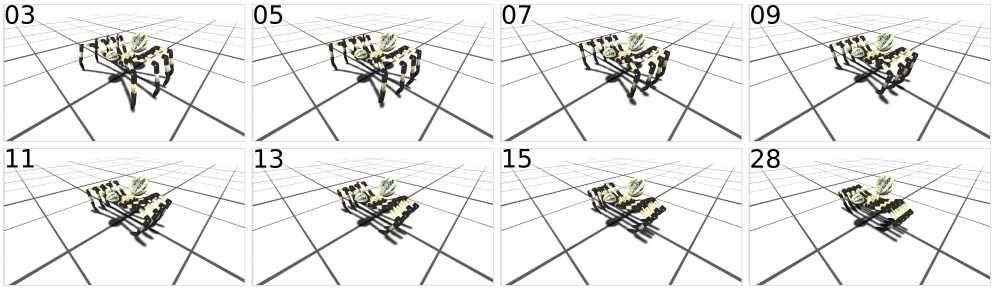


Figure 5.4.: Image series highlighting the third standing controller.

Other controllers manage to learn that certain poses were easier to balance than others. As such, a number of controllers evolved various static poses that they were able to hold near indefinitely. However, some of these poses looked more like a spider mannequin that had been randomly assembled by a tornado. One example of this can be seen in Figure 5.3. This controller does take some time to gather itself, but eventually converges to a stable pose. While certainly amusing, it did not meet the requirements of the control strategy, as it were considered unlikely that a walking controller could naturally transition into this

particular pose.

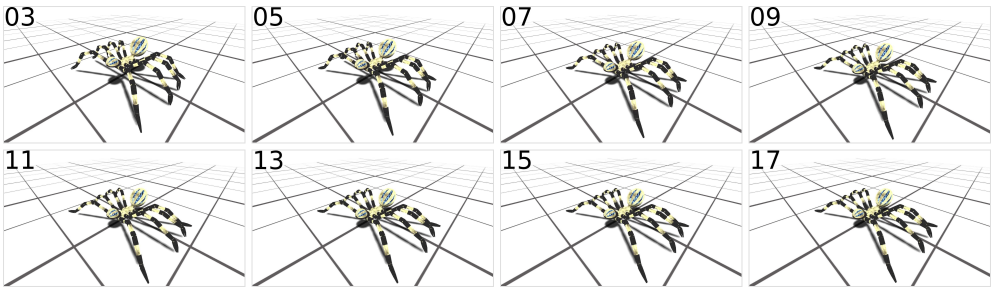


Figure 5.5.: Image series highlighting the champion standing controller.

Another example of these controllers managed to find a more intuitive resting position. The best one of these can be seen in Figure 5.4. This controller gathered its legs and manages to achieve a balanced resting position with its legs tucked in under itself. While this controller was the most balanced controller evolved during the project, it was considered too fragile for use with the control strategy. It tended to put all its legs very close together which could easily cause it to fall over if subjected to external forces, such as lingering forces left over from a different controller after a transition.

The final controller, which originated as a separate species in the same simulation as the previous example, was chosen as the champion standing controller. This controller did not achieve a perfectly balanced pose as it slowly descended towards the ground. However, it never did any drastic motions in order to position itself and were therefore judged to be the most compatible with other controllers in the control strategy.

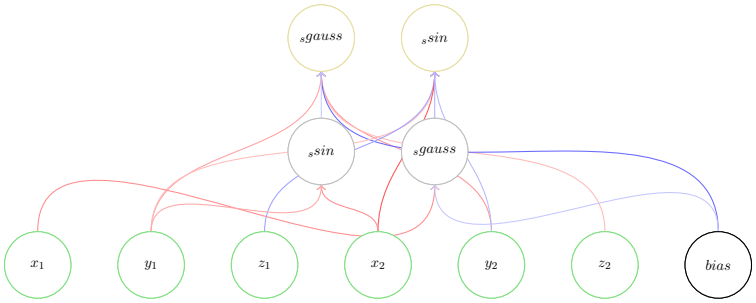


Figure 5.6.: The CPPN of the champion standing controller.

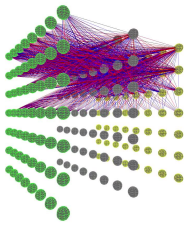


Figure 5.7.: The generated neural network for the champion standing controller.

Type	Number
Input Neurons	56
Output Neurons	56
Hidden Neurons	56
Input to Hidden	232
Input to Output	464
Hidden to Hidden	0
Hidden to Output	464
Output to Hidden	232
Output to Output	0
Looped Hidden	232
Looped Output	464
Total connections	2088

Table 5.1.: The type and number of connections in the final network for the champion standing controller.

As can be observed in Table 5.1 and Figure 5.7, the number of connections is low and they are mostly grouped around the top of the substrate. Since it was rewarded for not moving and only to keep itself from falling, it only had to use some of the joints to do exactly that. Seeing as the number of connections in a network will increase its sensitivity, this sensitivity would usually cause it to move around more. In this case it seemed that HyperNEAT discovered that reducing the number of connections in the network would make it better at standing still. An additional note is that most of the outputs that it is connected to are outputs that are not used. The downside of this type of network is that

its very hard for it to react to changes, unless those changes happen to hit the very few input nodes that its connected to.

5.2. Walking Controller

As mentioned in Chapter 4 Approach, the walking controller was evaluated purely based on how far it travelled in one specific axis while remaining upright. While all the evolved controllers were able to move forward to some degree, classifying all of them as *gaits* would be a stretch.

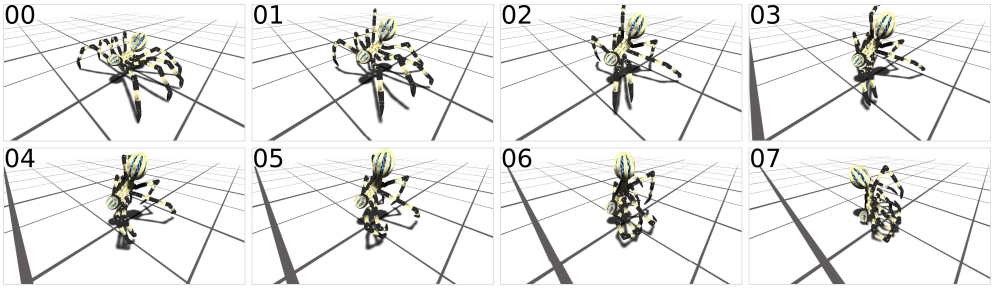


Figure 5.8.: Image series highlighting the gait of the first walker.

The controllers often quickly learnt that standing as far up as it could and then falling forward would give it a fairly decent fitness score. In some training runs it learnt that by throwing itself forward from the standing position it started in, it could easily cover a distance of 4-5 units. If evolution discovered any of these behaviours early in the simulation it would often outcompete other individuals and eventually dominate the population. Further evolution of these controllers would usually only get incrementally better at falling over or throwing themselves forward. An example of one of these controllers can be seen in Figure 5.8.

One of the more reasonable controllers that were able to evolve from the same type of behaviour can be seen in Figure 5.9. In this case, the controller flings itself forward with all its legs spread. However, unlike the previously mentioned examples, this controller learnt to move its legs into position to catch itself in time to avoid being killed due to falling. While this is an interesting result that definitely had an amusing gait, it was far

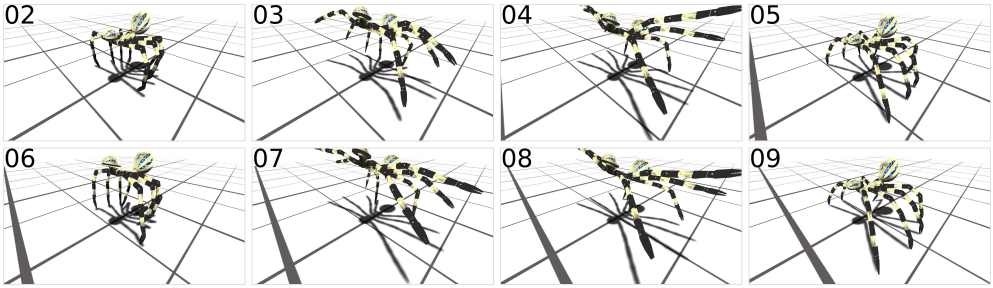


Figure 5.9.: Image series highlighting the gait of the second walker.

too unstable to be a candidate for the control strategy. Another thing to note about this controller is that while catching itself and preparing another jump, it often slid or turned unpredictably, meaning that it would very rarely travel in a straight line. Though in the cases where it did manage to move straight the controller was very capable of walking quite far.

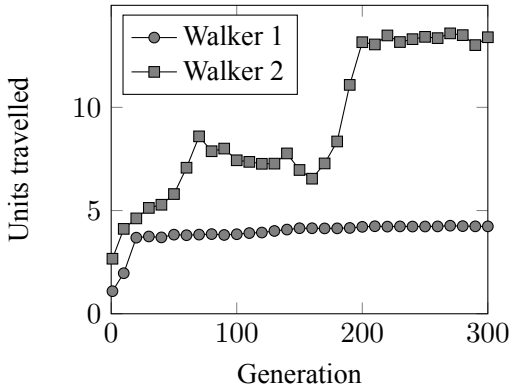


Figure 5.10.: The average distance travelled by the best individuals in the population of the 1st and 2nd walkers.

The above results were evolved with a fitness function that purely looked at the distance travelled and was given 10 seconds to walk. Figure 5.10 shows the distance travelled on average by the best individuals in the population for the two highlighted walkers.

One reoccurring issue that was seen in all the results up until this point was that the controllers often had no idea of what to do after the 10-second training period had passed. As a result the simulation duration was increased to 30 seconds. The idea behind this

change was to promote the evolution of a looped gait that could possibly go on for longer than the simulation period. However, as individuals that fell were killed, the simulation would almost never go on for the full 30 seconds. When the simulation time did get that high, it was usually due to an individual managing to balance in some odd stationary pose, eventually leading to its demise due to poor fitness.

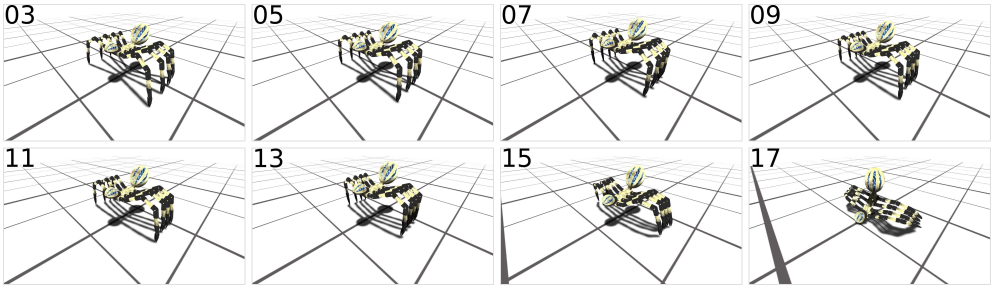


Figure 5.11.: Image series highlighting the gait of the third walker.

One controller that evolved from this experiment, can be seen in Figure 5.11. This controller positioned itself in a somewhat stable pose, before it started to produce a rapid, near imperceptible movement, with many of the joints. This caused the character to inch it forwards ever so slightly, usually ending with a spectacular bow.

This behaviour showcases one of the issues that comes with using neural networks. Neural networks can be very sensitive to change in the input it is given. While this is often good, in some cases this can lead to rapid changes in the output of the network. As in the example above the controller evolved to produce rapidly alternating outputs for many of the joints of the character. The controller could go from requesting the angle of the joint to be a large positive value followed by a large negative value the next update, before going back to a large positive value again. Since the joint can only move so far during an update, it meant that the joint would start going in one direction before suddenly changing to the opposite direction, causing what looks like vibrations. As the idea was to achieve a more realistic motion, these type of behaviours were discouraged by punishing the walking controller if a joint changed its direction of motion more than a certain number of times per second.

The new fitness function worked well to discourage rapid changes in the motion of each joint, and generally had no impact in the later stages of the simulation as the controllers

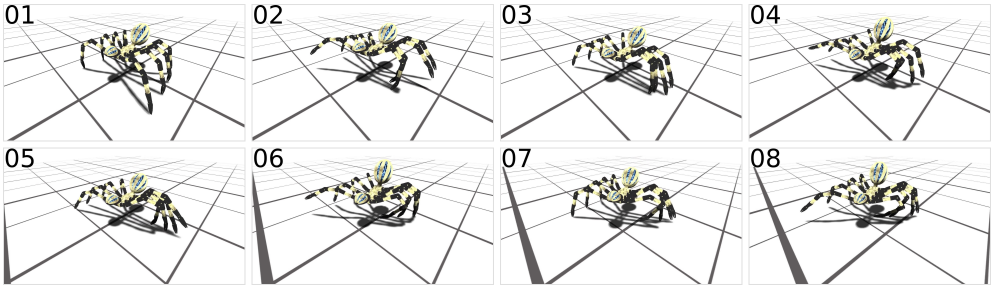


Figure 5.12.: Image series highlighting the gait of the fourth walker.

usually learnt rather quickly to stay away from such motions. One of the first results that showed a more reasonable gait was evolved after this new fitness function was added and the duration of the simulation was increased. The gait of this controller can be seen in Figure 5.12. The figure shows how the character would slightly turn itself and perform a sideways walk resembling a gait seen in various types of crabs. This method of walking enabled the character to walk quite far. In fact, this controller achieved one of the furthest distances travelled within the 30-second lifespan. However, this controller was also highly unstable and frequently fell over. Furthermore, it walked sideways and only performed well because it managed to fool the fitness calculator by turning itself to walk in the *correct* direction. As such it was therefore not an ideal candidate for a controller aimed to walk forward.

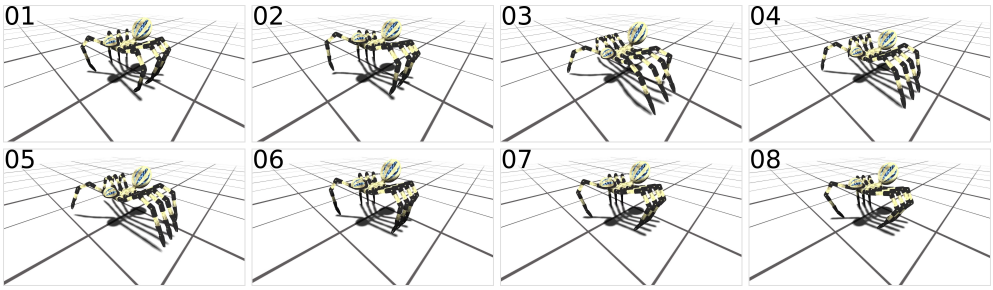


Figure 5.13.: Image series highlighting the gait of the fifth walker.

In order to create a control strategy, the controller needed to be able to make the character walk in a more stable forward walking motion that moves as straight as possible. To discourage the sideways motion displayed by the previous controller, another fitness

function was added to measure how far it moved into the different directions on the x axis. The further the character moved from 0 on the x axis in either direction, the worse fitness it would get.

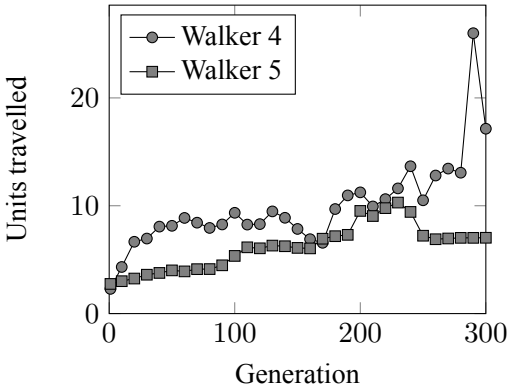


Figure 5.14.: The average distance travelled by the best individuals in the population of the 4th and 5th walkers.

One variation of the walking gaits that were evolved after these changes can be seen in Figure 5.13. This controller uses its legs in a symmetric motion, showing off that HyperNEAT has understood the symmetry of the model. In addition, it uses one of the legs as a way to keep itself up when moving, so not to fall forward.

Another reoccurring issue present in many of the results was that they were highly unstable. When tested outside the training simulation, i.e. in an environment where they were not immediately killed for falling, most of the controllers were not able to get themselves back off the ground when they did fall, usually just displaying uncoordinated spastic movements. As a final change to the approach, the controllers was allowed to keep going even after it had fallen. By allowing this it was believed that it would increase the likelihood that the controllers could learn how to get up even after falling flat on the ground. Two different variations of this change were implemented. First where touching the ground with specific pieces of the character, such as the sternum, abdomen and head, would suffer a penalty in the final fitness of the controller. Second did not incur any punishment except for the implicit penalty caused by friction.

One disadvantage of not terminating poorly performing individuals was the huge increase

in the time it took to execute the simulation. Even though the simulation is highly parallelized, processing batches of individuals in separate threads, the bottleneck of the simulation by far was the physics update. The physics simulation used in this project is rather detailed as to most correctly represent the complexities of the chain of joints in the character model. The amount of detail and error correction performed by the physics engine slows down the simulation quite significantly when used on as many as 150 individuals, even when parallelized. Not terminating any individuals required the physics to be run for every individual for the entirety of the 30 second simulation period, something which rarely ever happened in the previous experiments.

This was exemplified by the first experiment run with these new changes. This experiment was cancelled, after only 180 generations, when it had already been running three times as long as any prior experiment. In response to this the simulation duration was reduced to 15 seconds.

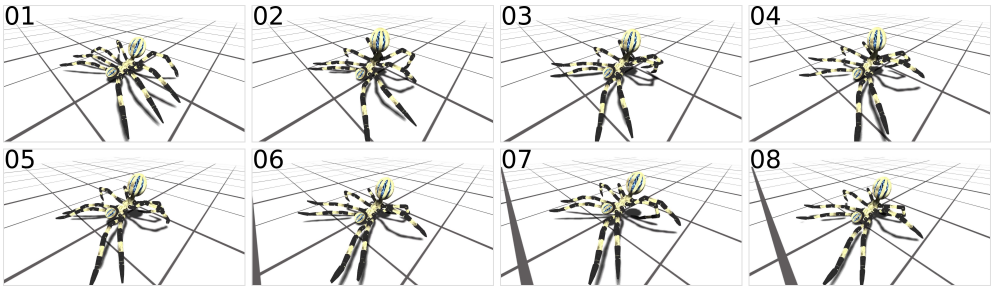


Figure 5.15.: Image series highlighting the gait of the sixth walker.

Even though the experiment was cancelled it exhibited a new and interesting motion, as can be seen in Figure 5.15. This controller was not punished for contacting the ground with any vital parts, which can clearly be seen in the resulting movement where it is resting against the ground for long periods of its gait. However, it had a more fluid motion than seen in the previous examples, and also achieved a higher stability by using two of its hind legs to keep it self above ground in a balanced pose. The walking motion displayed by this controller moves more consistently and was even capable of sustained walking far beyond the training period. It still had a problem walking in a straight line, but rather walked with a slight sideways tilt leading to a slightly curved heading.

All the controllers evolved after removing the falling restriction showed similar results. All of them where capable of walking substantially further than experiments run with this restriction. Many of were also capable of sustained walking after the now 15 second training period. However, all these controllers where categorically hugging the ground as an intricate part of their evolved gaits.

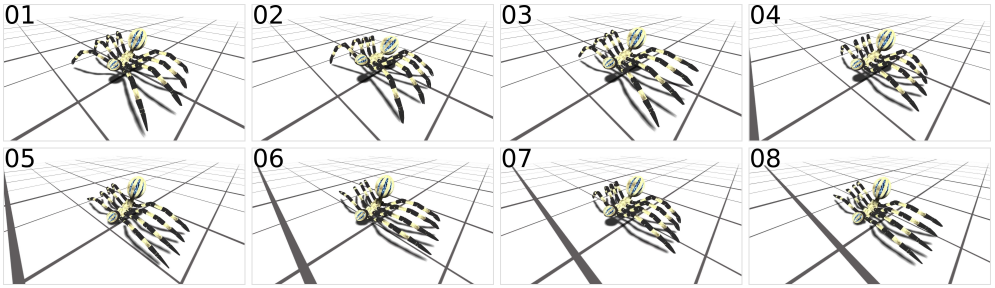


Figure 5.16.: Image series highlighting the gait of the seventh walker.

One of the controllers that exemplifies this kind of movement is the controller seen in Figure 5.16. In the first seconds, the movement seems very reasonable, however, it quickly falls forward and uses that to move itself by scraping the head against the ground, even when punished for it. It still manages to move forward when using the legs in a movement similar to a swimming octopus. Due to how it scrapes the ground, it was not able to stay straight for very long, since it would occasionally lean over to one of the sides and turn gradually into that direction.

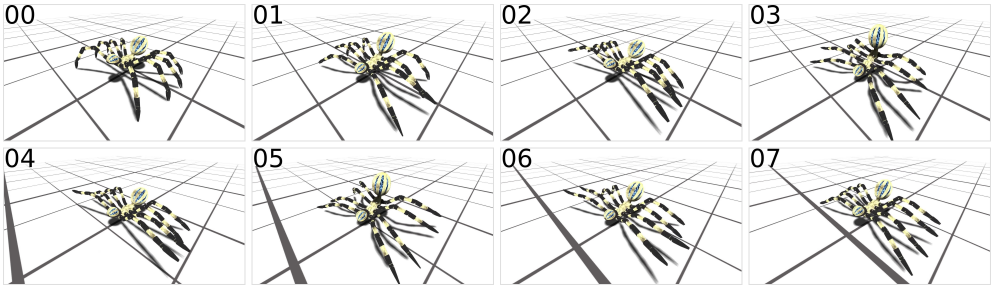


Figure 5.17.: Image series highlighting the gait of the champion walker.

The final controller, also designated the champion walker, can be seen in Figure 5.17. This controller, which was not punished for touching the ground, displayed the most pur-

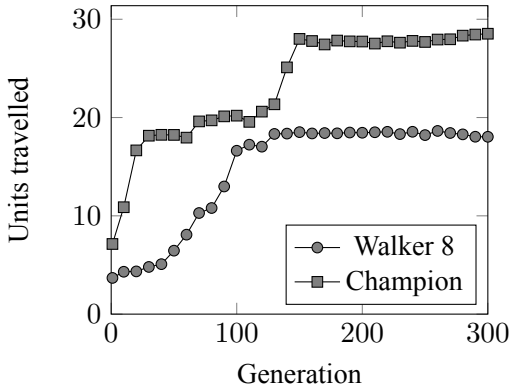


Figure 5.18.: The average distance travelled by the best individuals in the population of the 7th and champion walkers.

poseful and straight movement of all the controllers. The gait exhibit a strong asymmetric tendency in two distinct groups, consisting of the four front and four back legs, resembling a form of gallop. This controller was also capable of sustained walking far beyond the training period, in addition to being stable.

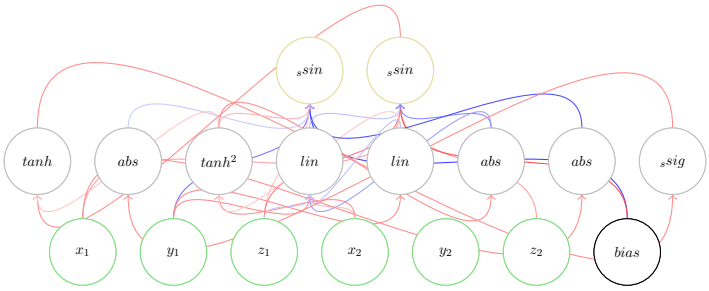


Figure 5.19.: The CPPN of the champion walker.

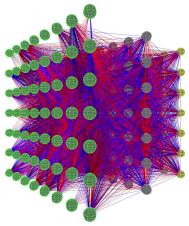


Figure 5.20.: The generated neural network for the champion walker.

Type	Number
Input Neurons	56
Output Neurons	56
Hidden Neurons	56
Input to Hidden	1582
Input to Output	1596
Hidden to Hidden	0
Hidden to Output	1617
Output to Hidden	1617
Output to Output	0
Looped Hidden	1610
Looped Output	1617
Total connections	9639

Table 5.2.: The type and number of connections in the final network for the champion walker.

The CPPN of the champion walker can be seen in Figure 5.19 with all its neurons and weights. The final generated neural network for the champion can be seen in Figure 5.20, where the number of connections is shown in Table 5.2. The table also shows the number of neurons, as determined by the pre-defined substrate shown in Figure 4.4 in Chapter 4 Approach.

5.3. Control Strategy

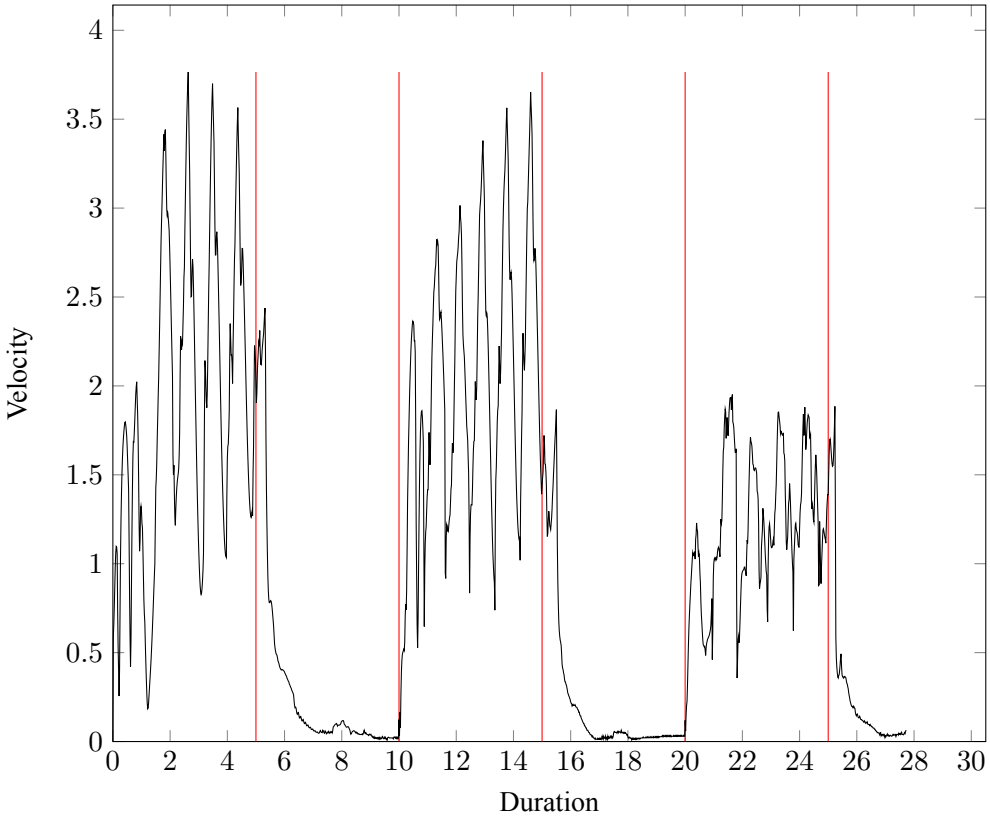


Figure 5.21.: Velocity of the character over time. The transitions between walking and standing are indicated by the red vertical lines.

The proposed control strategy was evaluated based on its responsiveness and robustness when switching between the two controllers. For the first experiment the walking and standing controllers was sequenced in a loop with 5 second time intervals, and the velocity of the character was recorded at each update. Figure 5.21 shows the result of this experiment.

However, measuring the actual responsiveness of this experiment proved to be more difficult than anticipated, as the standing controller never ever stopped completely. That is to say, its velocity never reached exactly 0, but rather slowed down to a near stop without

stopping entirely. Although setting a threshold velocity for measuring its responsiveness when switching from walking to standing it became possible to measure the time it took to perform the switch. Depending on how strict this threshold value were set the average response time were measured at 0.41 seconds for a lenient threshold velocity of 1 and 1.54 seconds when the threshold were set to a strict 0.1.

The same measurement was used for the switch from the standing to the walking controller. The walking controller did not have a constant speed, but rather displayed a periodically varying phase of movement and rest. However, measuring the responsiveness with a threshold value proved to be similarly effective, even when the threshold was set higher than the low-points of the resting period of its gait. The average response time for this switch was measuring at 0.09 seconds for a lenient threshold velocity of 0.5 and 0.65 seconds for a more strict threshold value of 1.5.

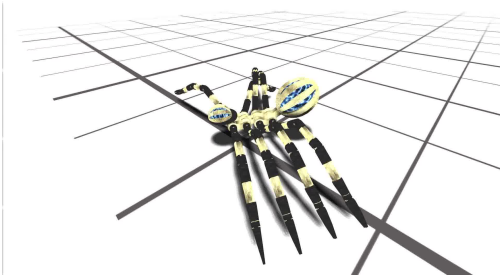


Figure 5.22.: Untrained behaviour seen during transitions.

However, as can be seen in Figure 5.21, the speed of the walking controller reduced noticeably after the second switch from the standing controller. The reason for this was that it managed to lock one of its forelegs under two of its hind legs, as can be seen in Figure 5.22, a behaviour that was never observed when running the controller standalone. For this reason the second control strategy experiment was abandoned. Seeing as even this slow periodic switching between the walking and standing controllers was enough to judge that the robustness of the control strategy did not meet expectations.

Chapter 6.

Discussion

While the results of the control strategy did leave a lot to be desired, it is hard to fully conclude that the strategy itself was a failure. This mainly due to the controllers it used, which was not particularly robust to begin with. Thus to draw any final conclusion on the topic, it will have to be tested with a set of more robust controllers.

Both the standing and walking controllers suggested much room for improvement, especially during transitions. The standing controller had a highly unconnected substrate, suggesting that it could not react very well to differences in the input. As such, it may have looked acceptable when starting from an already standing position, however, when transitioned from the walking controller, it was not able to react and just slid down to the ground. The walking controller did not handle transitions particularly well either, sometimes interlocking legs when starting from previously unseen starting positions left over from the standing controller.

These results suggest that more extensive training is required for both the controllers before they can be tested together. These issues mostly seem to stem from the controllers inability to adapt to situations it has not encountered during training. The decision to start all training simulations from the same identical standing position may have been a contributing factor in this regard. Training the controllers from a random starting position, or in sequence as per their intended use case, could possibly increase the controllers robustness over a wider range of scenarios. However, it is unknown whether or not this added variation would ultimately help the evolution of the controllers or lead to their

demise. Seeing as too much non-determinism during training could lead generally well-performing individuals to go extinct if they just happened to be unlucky.

6.1. Character Complexity

Much of the previous research into evolving gaits on multi-legged characters have focused on characters with four legs or less. There have been multiple studies showing success for bipedal and quadrupedal walkers using various algorithms, including HyperNEAT. However, the authors have not found any research that has shown or tried to use HyperNEAT with as many as eight legs. In addition to this, previous research has often had between two to three joints per leg, unlike the character in this thesis that has five joints. Due to the complexity of the character, it was expected that it would be more difficult for HyperNEAT to evolve a stable gait for it.

However, HyperNEAT proved quite capable of producing gaits for this highly complex eight-legged character. While most of them did not resemble what you would expect from eight-legged creatures in nature, they all managed to find amusing ways of walking forwards, some even managing to walk far beyond their training period. However, if the aim was to replicate gaits found in nature it is likely that a more complex fitness function would be required, maybe one that could optimize for concepts like pain or energy conservation.

On the other hand, if the character was replaced by a less complex character, it is believed that HyperNEAT would more easily have been able to evolve more stable gaits and it may have led to more success when creating the control strategy. However, as one of the motivations behind the control strategy was to possibly use it within a game, using a less complex character would have made it harder to verify whether or not this is possible with an actual game character.

Chapter 7.

Conclusion

This thesis set out to investigate the feasibility of combining individually trained neural controllers to form a control strategy that could be used to actively control virtual physics-based characters. To this end the neuroevolution algorithm *HyperNEAT* was used to evolve neural controllers for a previously untested domain of eight-legged locomotion. To create and evaluate the control strategy, two target neural controllers were trained for two different behaviours, standing and walking. The newly trained neural controllers were combined to form the control strategy and evaluated based on its robustness and responsiveness when switching between them.

The presented results show that *HyperNEAT* was successfully able to evolve gaits for a highly complex eight-legged character. The resulting gaits showed that they were quite capable of walking long distances, even beyond the training period. However, when combined within the control strategy the results suggest a need for further refinement as the controllers were still not robust enough to operate in tandem with each other.

Chapter 8.

Future works

This chapter describes various concepts that have not been tested within this thesis. Many of the described chapters have shown promising results for legged locomotion, where as some have yet to be tested.

8.1. Novelty Search

The controllers described in this thesis has been trained using an objective-based search where a fitness function evaluates how well a certain genome is performing. For instance, the walking controller was awarded for walking the furthest distance possible.

Novelty search is a different way of searching for a solution to a problem. Unlike objective-based search, novelty search is not searching for the solution to the problem, and instead of rewarding performance it rewards individuals who can be characterized differently from everyone else. It can therefore succeed where objective-based search often fails as it rewards stepping stones where objective-based searches often do not [59].

Characterizing the walking controller described in this thesis could be done by storing every x and z value at various intervals of the simulation. This would mean that two characters who walked who ended up at the same final coordinate by different paths would be characterized differently [11].

Novelty search would make it possible to discover gaits that, while possibly less effective, can be more natural to the character, and may be able to walk longer and more stable. This is because novelty does not search for a specific solution to a goal, but may instead stumble onto the solution of the goal as it is trying to evolve characters that can be characterized differently from each other.

Replacing the current objective-based search used in this thesis with a novelty based one would be of interest, as it may be possible to create a controller that is not only more stable, but may also look more natural.

8.2. Single-Unit Pattern Generators

One of the many challenges in generating stable oscillatory gaits is trying to give the network an intuitive understanding of time. This is often done by inputting pattern generating functions into the network, such as the sinusoids, which is the approach taken in this thesis, or by using a CTRNN. Another method of achieving this, that has shown great promise, is the usage of *Single-Unit Pattern Generators* or SUPG for short.

The SUPG is a new type of neuron that is able to produce a flexible temporal activation pattern that can be reset and repeated on a special trigger. It takes advantage of the CPPNs ability to encode spatial patterns, with a natural appearance, to instead use this to encode patterns over time. Unlike the sinusoids, SUPGs do not repeat a patterns over a set interval, but instead generate patterns until a trigger happens resetting the cycle. [14]

The trigger of the SUPGs can be customized, but as an example, one trigger might be a leg touching the ground. This means that when this happens, all the SUPGs that are connected to that leg is reset and starts the pattern over again.

Controllers using SUPGs have shown to be able to walk with a stable oscillatory gait and has even shown to be able to continue walking for far beyond the training window [14].

8.3. ES-HyperNEAT

Evolvable-Substrate HyperNEAT is a more recent alteration of the HyperNEAT algorithm, as mentioned in Chapter 2 Background. Unlike HyperNEAT, ES-HyperNEAT alleviates the problem of having to decide on the number and position of the hidden neurons, as this can often be a difficult task. ES-HyperNEAT takes advantage of the CPPN and its information by searching for regions of high variance and placing neurons in those regions.

While ES-HyperNEAT has not been used, to the best of the authors knowledge, to evolve legged gaits, it would be interesting to see what effect the evolvable substrate would have on the controllers and if it could evolve better gaits than shown in this thesis.

Appendices

Appendix A.

Settings

This appendix describes the settings that are used for Bullet Physics. It also describes the settings used for the various experiments, due to the numerous amount of different configurations that are available for HyperNEAT.

A.1. Bullet Physics

As mentioned in Section 4, the weight of the model were 1 mass unit for each piece except for the abdomen, eye and sternum which weigh 8, 5 and 10 mass units, respectively.

All of the pieces were set to have a friction of 0.84, which simulates the friction of metal. The number of iterations were set to 25, allowing a more accurate representation of the physics while not be too resource demanding.

The Error Reduction Parameter (ERP) specifies what proportion of the joint errors will be fixed during the next simulation step. An ERP of 0 will apply no correcting force and the bodies will eventually drift apart during the simulation. An ERP of 1 will attempt to fix all joint errors, but will fail to fix everything due to some internal approximations in Bullet. A value of 1 is not recommended. The value that was used for the simulations was 0.8, in order to increase the accuracy of the many joints that the model uses. Default value is 0.2.

The Constraint Force Mixing (CFM) determines how hard or soft constraints will be. The higher the value of CFM, the softer the constraints will be and the more it will be possible to violate the constraint by pushing on it. The value used for the simulations was 0.00001, which allows the constraints in the simulation to be violated by a small fraction. This was set in order to create a smoother simulation due to the complexity of the model.

The gravity of the world was set to 9.81 simulating the gravity that is experienced on Earth.

All other settings that was not specifically mentioned in this section was left at their default values.

A.2. HyperNEAT

The settings of HyperNEAT varied from experiment to experiment. Some of the important settings is included in this chapter. The settings for all the other experiments can be found on the authors github account[1].

A.2.1. Substrate

Table A.1.: Substrate parameters.

Name	Value
Allow input to hidden links	true
Allow input to output links	true
Allow hidden to output links	true
Allow hidden to hidden links	true
Allow output to hidden links	true
Allow output to output links	true
Allow looped to hidden links	true
Allow looped to output links	true

[1] https://github.com/reewr/master

A.2.2. NEAT

Table A.2.: NEAT/HyperNEAT parameters.

Name	Value
Basic	
PopulationSize	150
DynamicCompatibility	true
MinSpecies	10
MaxSpecies	15
AllowClones	true
GA	
YoungAgeTreshold	5
YoungAgeFitnessBoost	1.1
SpeciesMaxStagnation	15
StagnationDelta	0.0
OldAgeTreshold	30
OldAgePenalty	1.1
SurvivalRate	0.25
CrossoverRate	0.7
OverallMutationRate	0.25
InterspeciesCrossoverRate	0.0001
MultipointCrossoverRate	0.75
EliteFraction	0.2
Structural Mutation	
MutateAddNeuronProb	0.03
SplitRecurrent	true
SplitLoopedRecurrent	true
MutateAddLinkProb	0.2
MutateAddLinkFromBiasProb	0.0

Name	Value
MutateRemLinkProb	0.1
MutateRemSimpleNeuronProb	0.1
LinkTries	32
RecurrentProb	0.5
RecurrentLoopProb	0.25
Mutation	
MutateWeightsProb	0.8
MutateWeightsSevereProb	0.5
WeightMutationRate	0.96
WeightMutationMaxPower	0.5
WeightReplacementMaxPower	1.0
MutateActivationAProb	0.1
MutateNeuronActivationTypeProb	0.15
ActivationFunction_SignedSigmoid_Prob	1.0
ActivationFunction_UnsignedSigmoid_Prob	0.0
ActivationFunction_Tanh_Prob	0.0
ActivationFunction_TanhCubic_Prob	0.0
ActivationFunction_SignedStep_Prob	0.0
ActivationFunction_UnsignedStep_Prob	0.0
ActivationFunction_SignedGauss_Prob	1.0
ActivationFunction_UnsignedGauss_Prob	0.0
ActivationFunction_Abs_Prob	0.0
ActivationFunction_SignedSine_Prob	1.0
ActivationFunction_UnsignedSine_Prob	0.0
ActivationFunction_Linear_Prob	1.0
ActivationFunction_Relu_Prob	0.0
ActivationFunction_Softplus_Prob	0.0

Bibliography

[1] K. Sims, 'Evolving virtual creatures', in *Proceedings of the 21st annual conference on computer graphics and interactive techniques*, 1994, pp. 15–22.

[2] K. Sims, 'Evolving 3D morphology and behavior by competition', *Artificial Life*, vol. 1, no. 4, pp. 353–372, 1994.

[3] T. Geijtenbeek and N. Pronost, 'Interactive character animation using simulated physics: A state-of-the-art review', *Comput. Graph. Forum*, vol. 31, no. 8, pp. 2492–2515, Dec. 2012.

[4] T. Pejsa and I. Pandzic, 'State of the art in example-based motion synthesis for virtual characters in interactive applications', *Computer Graphics Forum*, vol. 29, no. 1, pp. 202–226, 2010.

[5] M. Hagenaars, 'Hierarchical development of physics-based animation controllers', Master's thesis, Utrecht University, 2014.

[6] R. Grzeszczuk and D. Terzopoulos, 'Automated learning of muscle-actuated locomotion through control abstraction', in *Proceedings of the 22Nd annual conference on computer graphics and interactive techniques*, 1995, pp. 63–70.

[7] T. Reil and P. Husbands, 'Evolution of central pattern generators for bipedal walking in a real-time physics environment', *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 159–168, Apr 2002.

[8] V. K. Valsalam and R. Miikkulainen, 'Modular neuroevolution for multilegged locomotion', in *Proceedings of the genetic and evolutionary computation conference gECCO 2008*, 2008, pp. 265–272.

[9] B. F. Allen and P. Faloutsos, 'Evolved controllers for simulated locomotion', in *Proceedings of the 2Nd international workshop on motion in games*, 2009, pp. 219–230.

[10] J. Clune, B. E. Beckmann, C. Ofria, and R. T. Pennock, 'Evolving coordinated quadruped gaits with the hyperNEAT generative encoding', in *2009 iEEE congress on evolutionary computation*, 2009, pp. 2764–2771.

[11] R. S. Olson, 'A step toward evolving biped walking behavior through indirect encoding', Master's thesis, University of Central Florida, 2010.

[12] J. Yosinski, J. Clune, D. Hidalgo, S. Nguyen, J. C. Zagal, and H. Lipson, 'Evolving robot gaits in hardware: The hyperNEAT generative encoding vs. parameter optimization', in *In proceedings of the 20th european conference on artificial life*, 2011, pp. 890–897.

[13] S. Lee, J. Yosinski, K. Glette, H. Lipson, and J. Clune, 'Evolving gaits for physical robots with the hyperNEAT generative encoding: The benefits of simulation', in *Applications of evolutionary computation: 16th european conference, evoApplications 2013, vienna, austria, april 3-5, 2013. proceedings*, A. I. Esparcia-Alcázar, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 540–549.

[14] G. Morse, S. Risi, C. R. Snyder, and K. O. Stanley, 'Single-unit pattern generators for quadruped locomotion', in *Proceedings of the 15th annual conference on genetic and evolutionary computation*, 2013, pp. 719–726.

[15] C. Ridderström, 'Legged locomotion: Balance, control and tools — from equation to action', PhD thesis, The Royal Inst. of Technology, 100 44 Stockholm, Sweden, 2003.

[16] G. S. Hornby, S. Takamura, J. Yokono, O. Hanagata, T. Yamamoto, and M. Fujita, 'Evolving robust gaits with aIBO', in *Proceedings 2000 iCRA. millennium conference. iEEE international conference on robotics and automation. symposia proceedings (cat. no.00CH37065)*, 2000, vol. 3, pp. 3040–3045 vol.3.

[17] S. Colombano, F. Kirchner, D. Spenneberg, and J. Hanratty, 'Exploration of planetary terrains with a legged robot as a scout adjunct to a rover', in *Space 2004 conference and exhibit*, 2004.

[18] G. S. Hornby, S. Takamura, T. Yamamoto, and M. Fujita, 'Autonomous evolution of dynamic gaits with two quadruped robots', *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 402–410, June 2005.

[19] G. S. Hornby, S. Takamura, O. Hanagata, M. Fujita, and J. Pollack, 'Evolution of controllers from a high-level simulator to a high dOF robot', in *Evolvable systems: From biology to hardware: Third international conference, iCES 2000 edinburgh, scotland, uK, april 17–19, 2000 proceedings*, J. Miller, A. Thompson, P. Thomson, and T. C. Fogarty, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 80–89.

[20] G. S. Hornby, H. Lipson, and J. B. Pollack, 'Generative representations for the automated design of modular physical robots', *IEEE Transactions on Robotics and Automation*, vol. 19, no. 4, pp. 703–719, Aug 2003.

[21] N. Kohl and P. Stone, 'Machine learning for fast quadrupedal locomotion', in *Proceedings of the 19th national conference on artifical intelligence*, 2004, pp. 611–616.

[22] S. Risi and K. O. Stanley, 'Confronting the challenge of learning a flexible neural controller for a diversity of morphologies', in *Proceedings of the 15th annual conference on genetic and evolutionary computation*, 2013, pp. 255–262.

[23] C. Igel, 'Neuroevolution for reinforcement learning using evolution strategies', in *Evolutionary computation, 2003. cEC '03. the 2003 congress on*, 2003, vol. 4, pp. 2588–2595 Vol.4.

[24] J. Togelius, T. Schaul, J. Schmidhuber, and F. Gomez, 'Countering poisonous inputs with memetic neuroevolution', in *Proceedings of the 10th international conference on parallel problem solving from nature — pPSN x - volume 5199*, 2008, pp. 610–619.

[25] K. O. Stanley and R. Miikkulainen, 'Evolving neural networks through augmenting topologies', *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, Jun. 2002.

[26] F. Gomez, J. Schmidhuber, and R. Miikkulainen, 'Accelerated neural evolution through cooperatively coevolved synapses', *Journal of Machine Learning Research*, pp. 937–965, 2008.

[27] L. Cardamone, D. Loiacono, and P. L. Lanzi, 'Evolving competitive car controllers for racing games with neuroevolution', in *Proceedings of the 11th annual conference on genetic and evolutionary computation*, 2009, pp. 1179–1186.

[28] N. Kohl, K. Stanley, R. Miikkulainen, M. Samples, and R. Sherony, 'Evolving a real-world vehicle warning system', in *Proceedings of the 8th annual conference on genetic and evolutionary computation*, 2006, pp. 1681–1688.

[29] J. Schrum, I. V. Karpov, and R. Miikkulainen, 'UT^2: Human-like behavior via neuroevolution of combat behavior and replay of human traces', in *Proceedings of the iEEE conference on computational intelligence and games (cIG 2011)*, 2011, pp. 329–336.

[30] J. Schrum and R. Miikkulainen, 'Evolving multimodal behavior with modular neural networks in ms. pac-man', in *Proceedings of the 2014 annual conference on genetic and evolutionary computation*, 2014, pp. 325–332.

[31] S. Risi and J. Togelius, 'Neuroevolution in games: State of the art and open challenges', *CoRR*, vol. abs/1410.7326, 2014.

[32] S. M. Lucas and J. Togelius, 'Point-to-point car racing: An initial study of evolution versus temporal difference learning', in *2007 iEEE symposium on computational intelligence and games*, 2007, pp. 260–267.

[33] S. Whiteson, M. E. Taylor, and P. Stone, 'Empirical studies in action selection for reinforcement learning', *Adaptive Behavior*, vol. 15, no. 1, pp. 33–50, March 2007.

[34] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, 'A hypercube-based encoding for evolving large-scale neural networks', *Artif. Life*, vol. 15, no. 2, pp. 185–212, Apr. 2009.

[35] J. Clune, K. O. Stanley, R. T. Pennock, and C. Ofria, 'On the performance of indirect encoding across the continuum of regularity', *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 3, pp. 346–367, June 2011.

[36] K. O. Stanley and R. Miikkulainen, 'A taxonomy for artificial embryogeny', *Artificial Life*, vol. 9, no. 2, pp. 93–130, 2003.

[37] J. Gauci and K. O. Stanley, 'Autonomous evolution of topographic regularities in artificial neural networks', *Neural Comput.*, vol. 22, no. 7, pp. 1860–1898, Jul. 2010.

[38] J. Secretan, N. Beato, D. B. D Ambrosio, A. Rodriguez, A. Campbell, and K. O. Stanley, 'Picbreeder: Evolving pictures collaboratively online', in *Proceedings of the sIGCHI conference on human factors in computing systems*, 2008, pp. 1759–1768.

[39] K. O. Stanley, 'Compositional pattern producing networks: A novel abstraction of development', *Genetic Programming and Evolvable Machines*, vol. 8, no. 2, pp. 131–162, Jun. 2007.

[40] J. Clune and H. Lipson, 'Evolving 3D objects with a generative encoding inspired by developmental biology', *SIGEVOlution*, vol. 5, no. 4, pp. 2–12, Nov. 2011.

[41] K. O. Stanley, 'Exploiting regularity without development', in *Proceedings of the aAAI fall symposium on developmental systems*, 2006.

[42] J. Gauci and K. O. Stanley, ‘A case study on the critical role of geometric regularity in machine learning’, in *Proceedings of the 23rd national conference on artificial intelligence - volume 2*, 2008, pp. 628–633.

[43] J. Clune, C. Ofria, and R. T. Pennock, ‘The sensitivity of hyperNEAT to different geometric representations of a problem’, in *Proceedings of the 11th annual conference on genetic and evolutionary computation*, 2009, pp. 675–682.

[44] D. B. D’Ambrosio and K. O. Stanley, ‘Generative encoding for multiagent learning’, in *Proceedings of the 10th annual conference on genetic and evolutionary computation*, 2008, pp. 819–826.

[45] D. B. D’Ambrosio, J. Lehman, S. Risi, and K. O. Stanley, ‘Evolving policy geometry for scalable multiagent learning’, in *Proceedings of the 9th international conference on autonomous agents and multiagent systems: Volume 1 - volume 1*, 2010, pp. 731–738.

[46] J. Drchal, O. Kapral, J. Koutník, and M. Šnorek, ‘Combining multiple inputs in hyperNEAT mobile agent controller’, in *Artificial neural networks – iCANN 2009: 19th international conference, limassol, cyprus, september 14-17, 2009, proceedings, part iI*, C. Alippi, M. Polycarpou, C. Panayiotou, and G. Ellinas, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 775–783.

[47] J. Drchal, J. Koutnik, and M. Snorek, ‘HyperNEAT controlled robots learn how to drive on roads in simulated environment’, in *2009 iEEE congress on evolutionary computation*, 2009, pp. 1087–1092.

[48] P. Verbancsics and K. O. Stanley, ‘Evolving static representations for task transfer’, *J. Mach. Learn. Res.*, vol. 11, pp. 1737–1769, Aug. 2010.

[49] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone, ‘A neuroevolution approach to general atari game playing’, *IEEE Transactions on Computational Intelligence and AI in Games*, 2013.

[50] S. Risi, J. Lehman, and K. O. Stanley, 'Evolving the placement and density of neurons in the hyperneat substrate', in *Proceedings of the 12th annual conference on genetic and evolutionary computation*, 2010, pp. 563–570.

[51] S. Risi and K. O. Stanley, 'Enhancing es-hyperneat to evolve more complex regular neural networks', in *Proceedings of the 13th annual conference on genetic and evolutionary computation*, 2011, pp. 1539–1546.

[52] S. Risi and K. O. Stanley, 'An enhanced hypercube-based encoding for evolving the placement, density, and connectivity of neurons', *Artif. Life*, vol. 18, no. 4, pp. 331–363, Oct. 2012.

[53] J.-c. Wu and Z. Popović, 'Terrain-adaptive bipedal locomotion control', *ACM Transactions on Graphics*, vol. 29, no. 4, pp. 72:1–72:10, Jul. 2010.

[54] I. Mordatch, M. de Lasa, and A. Hertzmann, 'Robust Physics-Based Locomotion Using Low-Dimensional Planning', *ACM Transactions on Graphics*, vol. 29, no. 3, 2010.

[55] T. Geijtenbeek, M. van de Panne, and A. F. van der Stappen, 'Flexible muscle-based locomotion for bipedal creatures', *ACM Transactions on Graphics*, vol. 32, no. 6, 2013.

[56] N. Hansen, 'The cMA evolution strategy: A comparing review', in *Towards a new evolutionary computation: Advances in the estimation of distribution algorithms*, J. A. Lozano, P. Larrañaga, I. Inza, and E. Bengoetxea, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 75–102.

[57] T. Geijtenbeek, N. Pronost, and A. van der Stappen, 'Simple Data-Driven Control for Simulated Bipeds', in *Eurographics/ACM sIGGRAPH symposium on computer animation*, 2012, pp. 211–219.

[58] E. C. (EPlex) Research Group at the University of Central Florida, 'NEAT software catalog', 2017. [Online]. Available: http://eplex.cs.ucf.edu/neat_software/#HyperNEAT. [Accessed: 01-Feb-2017].

[59] J. Lehman and K. O. Stanley, ‘Novelty search and the problem with objectives’, in *Genetic programming theory and practice iX*, R. Riolo, E. Vladislavleva, and J. H. Moore, Eds. New York, NY: Springer New York, 2011, pp. 37–56.