A Neuroevolutionary Approach to Evolve a Flexible Neural Controller for a Morphology Changing Quadruped Robot

Wonho Lee

Thesis submitted for the degree of
Master in Robotics and Intelligent Systems
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2020

A Neuroevolutionary Approach to Evolve a Flexible Neural Controller for a Morphology Changing Quadruped Robot

Wonho Lee

Abstract

Designing a gait controller for a morphology changing robot is a challenging problem due to the high degree of freedom. In 2013, Risi showed that by learning the relation between morphology and controller, it is possible to evolve a neural network-based controller that can walk on different morphologies.

This thesis aims to implement Risi's flexible controller for a robot platform that features a morphology changing physical robot, Dynamic Robot for Embodied Testing: DyRET, developed at the University of Oslo. Also, it compares which of the evolutionary algorithms (EA) would be the best fit for the task - among behavior diversity, fitness-based and multi-objective EA (MOEA), and combinations of different behavior descriptors for behavior diversity EA.

In particular, this thesis implements a flexible controller based on Hypercube-based NeuroEvolution of Augmented Topologies, designed to learn the relation between the morphology and the controller based on the previous work. The controller is evolved by Novelty search, fitness, and MOEA, combining both. Three behavior descriptors are implemented - max-min amplitude, duty factor, and the last position. The experiments are conducted on the simulation environment provided by DyRET.

The results show that it is possible to evolve a stable controller that can walk on various morphology for DyRET platform while not all the evolved controllers learned the relationship between the controller and the morphology. Also, the comparison of EA's revealed that fitness-based EA could produce controllers that are as good as the behavior diversity EA and MOEA for the given constraints. The experiment also indicates that combining all three behavior descriptors can generate most fit controllers while not statistically significant when comparing the walking distance.

Acknowledgements

I would like to express my appreciation to Kyrre Glette for sincere guidance and encouragement throughout the study.

Also, I would like to thank Sebastian Risi for providing me the inspiration for this thesis, and Jørgen Nordmoen for sharing his insights and thoughtful advice on DyRET platform.

The experiments were performed on resources provided by UNINETT Sigma2 - the National Infrastructure for High-Performance Computing and Data Storage in Norway.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In 2016, one of the best human players in the ancient board game of Go was defeated by Deepmind's AI, AlphaGo[48]. The game of Go was previously thought of as undefeatable by AI by many due to the complexity and long-term planning attribute of the game. By combining deep neural networks with reinforcement learning, this achievement marked a milestone in terms of AI research[42].

Does it mean that we have come up with an algorithm to think and act like us? Although the recent accomplishment in deep learning has provided a miraculous advance in many fields - from medical image analysis to playing games, these so-called artificial intelligence algorithms can only do one thing, for which they are designed. An AI system - sophisticated enough to defeat the best human player in 2000 years old game, does not have a common sense of a dog. The smooth-talking virtual personal assistants, such as Siri or Alexa, are essentially an extended interface that automates and integrates some trivial tasks.

Nearly 30 years ago, Randall D. Beer wrote that "intelligence exhibited by so far current AI systems is extremely narrow and brittle" as a critic to the classical AI methodology - which models intelligence as "manipulation of symbolic representation of the world"[1]. Despite the years of advance in the field, the statement holds somewhat true. Beer goes on and suggests an alternate view, *Intelligence as adaptive behavior*. He argues that the ability of autonomous agents - which can dynamically adjust its behavior repertoire in its interaction with its environment - is the essence of intelligence. Along with the argument, he conducts a computational neuroethological experiment building an artificial insect based on inspirations from biological neural architectures.

Evolved in millions of years, the human brain is the most sophisticated and complex system known so far. If it is the function of our brain which we are trying to replicate, why don't we start looking at its simplest form and question ourselves whether we could replicate that? Neurologist Daniel Wolpert explains that the purpose of the brain is to move around[50].

Provided by an example of sea squirt - a sea creature which floats and swims around controlled by its elementary brain in early stage, then consumes its own brain when it settles in one place - the augment seems to be very convincing. If it were appropriate to define a primary brain as a mere network with a handful of neurons designed to move around with a purpose, evolving such a system could perhaps reveal the least sign of intelligence - adaptive behavior.

Although it has not gained as much of attention as deep learning - where deep and complex artificial neural network is trained by backpropagation, there is an alternative approach to AI, known as Neuroevolution (NE)[46]. NE makes use of an evolutionary algorithm to evolve neural networks, drawing inspiration from nature's process of evolution. While some of the earlier approaches have evolved weight of fixed topology neural networks traditional evolutionary algorithm such as genetic algorithm, one of the exceptional successes within the field of NE is NeuroEvolution of Augmenting Topologies(NEAT)[47], which evolves artificial neural networks in terms of both of its topology and connection weights. Besides the evolving the neural network itself, NE also introduces indirect encoding - presenting genome as a neural network, instead of directly encoding the property of network into the genome. Hypercube-based NEAT(HyperNEAT) utilizes Compositional Pattern Producing Network(CPPN), which abstracts genomic information into a pattern generating network, evolved by NEAT[44]. In HyperNEAT the CPPN is used to encode the information of the substrate - a template of neural network with geometric information. Other remarkable success also includes Quality Diversity(QD) algorithms - such as Novelty Search(NS)[22], and Multi-dimensional Archive of Phenotypic elites(MAP-Elites)[28]. These algorithms promote the search of unique and novel behaviors, that may not have been able to discover by an objective optimization - where only fitness of the solution is considered for the evolution process.

Legged robots have been a fascinating subject for many researchers and engineers due to its difficulty in designing, which stems from a high degree of freedom in the movement. While there are commercially available legged robots[1] that are versatile and agile, most of the state-of-the-art legged robots are in a way or another *hand* designed by engineers. Nonetheless, as Beer states "Animals are evolved, not designed"[2]. NE has been tested and utilized extensively in the field of Evolutionary Robotics(ER), where the evolutionary approach is taken for confronting problems of designing a robot's sensory system, morphology, and control at the same time. Although researches in ER do not always revolve around the connectionist approach - using a neural network as its basic form of the evolved agent, there have been studies that use ideas from NE such as NEAT, HyperNEAT, and QD to evolve controller or morphology for legged robots[51, 22, 39, 18, 7, 21, 31].

One of the interesting accomplishment from ER is the work done by

[1]https://www.bostondynamics.com/spot

Risi[39], where he presents a novel idea of evolving a flexible gait controller using HyperNEAT - which evolved CPPN that encodes the information of the morphology of the robot alongside the neural controller. His work presents that it is indeed possible to evolve a gait controller that is flexible, i.e., can adapt to different morphologies, different leg lengths in this case.

This thesis will focus on evolving a flexible neural network-based controller for a quadruped inspired by Risi's work. DyRET: Dynamic Robot for Embodied Testing, is a robotic platform that features various morphology with adjustable leg length developed at University of Oslo by Nygaard et al.[34]. As DyRET is designed to test various ideas on controller and morphology; this would make an ideal platform to test Risi's flexible controller. Although it has already been shown to be possible, it is interesting to implement it on the DyRET platform for the following points:

- To reconfirm Risi's work.
- To investigate a better way of evolving the gait controller.
- To realize the flexible controller in a physical robot.

Even though the most interesting aspect of testing Risi's work on DyRET would be testing it on the physical robot, the primary focus in this thesis will be on a simulated environment. However this work will pave a way to realize the main idea - neural network-based flexible controller on a physical robot.

1.1 Research Questions

Implementing Risi's flexible neural gait controller on DyRET platform poses some challenges. The algorithms that are used in the original work should be implemented upon the correct understanding of each. While there will be some readily available packages for the task, such as NEAT and HyperNEAT, considering the uniqueness of the original work, many of its components - such as the multi-layered substrate and its corresponding CPPN should be implemented by the author.

In addition to implementing the original work on the DyRET platform, this thesis can also experiment on various evaluation methods and variations to see if there is any particular method that can accelerate the search process or improve performance of the evolved controller. The original work utilizes only novelty search as its reward mechanism coupled with a behavior descriptor which describes the trajectory of the robot.

With the considerations given above, the research questions for this thesis are outlined as the following;

- **Is it possible to evolve a neural network-based flexible gait controller for DyRET?** First and foremost, the goal of this thesis is to confirm whether Risi's flexible neural controller can be implemented on the simulated DyRET platform. Static controllers and flexible controllers will be evolved and compared with their interpolated leg configuration.

- **Which selection pressure works best? - diversity vs. fitness vs. both** The choice of evaluation in Risi's work is by novelty search. While it has been suggested that quality diversity works better when the search landscape of performance-oriented approach can be deceptive[22, 27], it would be interesting to confirm it and see any exception can be drawn from a particular implementation as this thesis. For this task, this thesis will look into comparing performance between novelty search, fitness only, and multi-objective evaluation combining both diversity and objective optimization.
- **Which behavior descriptor works best for evolving a gait controller for DyRET?** Risi only implements a behavior descriptor, a trajectory of robot movement. It would be interesting to see how other types of behavior descriptors would perform and compare combinations between them. For this purpose, this thesis suggests three different behavior descriptor - Max-min amplitude of joint output, duty factor, and the last position.

1.2 Contributions to the Research Community

Even though this thesis is not submitted to conveying any new idea in the research field, there are a few points that this thesis will contribute to the community.

First of all, this thesis will pave a way of implementing flexible neural controller for the DyRET platform. DyRET has been tested with various controller scheme such as high-level inverse-kinematics based position controller[35], and low-level gait controller parameterized to describe continuous first-order spline that presents a series of angle for each joints[29], and a network of oscillators as central pattern generator(CPG)[32]. Implementation of neural network-based gait controller will expand the catalog of DyRET's controller scheme.

Although Risi[39] has confirmed and demonstrated a successful result in his work, it has not been implemented into other platforms. This thesis will make it the first example.

Lastly, this thesis compares different methods to evolve a flexible neural gait controller - such as a different combination of behavior descriptors and different evaluation systems. By doing so, this thesis will be able to present alternative approaches and their results.

Chapter 2

Background

In this chapter, the theories behind evolving a gait controller is introduced. First, Neuroevolution is introduced, including an artificial neural network, a simple overview of evolutionary algorithms, behavioral diversity algorithms, and evolving neural networks. Then the second section provides a general overview of legged robot controllers with a focus on biologically inspired methods. In the end, previous works on the flexible gait controller are introduced.

2.1 Neuroevolution

2.1.1 Artificial Neural Network

As we are trying to model intelligence, it is only natural to start looking at the brain and its element, the neuron. In terms of functionality, a biological neuron consists of dendrites - where it takes input from the synapses and axon where electric signals are passed over to the dendrites of other neurons. The neuron fires a pulse signal down the axon if a certain threshold is reached by membrane potential from input synapses. Several factors describe the neuron's behavior, such as chemical reactions in synapses and electrochemical properties during the firing, to name a few. In other words, precise modeling of a biological neuron is complex.

In 1943, McCulloch and Pitt came up with a simplified mathematical model of neuron mainly focusing on the all-or-none principle[26]. The basic formulation of the mathematical model follows as such;

$$h = \sum_{i=0}^{m} w_i x_i \tag{2.1}$$

h denotes membrane potential level of an artificial neuron with m connected and, with a weight of w for ith synapses. x denotes the output of i th neuron. Then based on an activation function with its input as h, the neuron fires or doesn't fire. So it's a binary device.

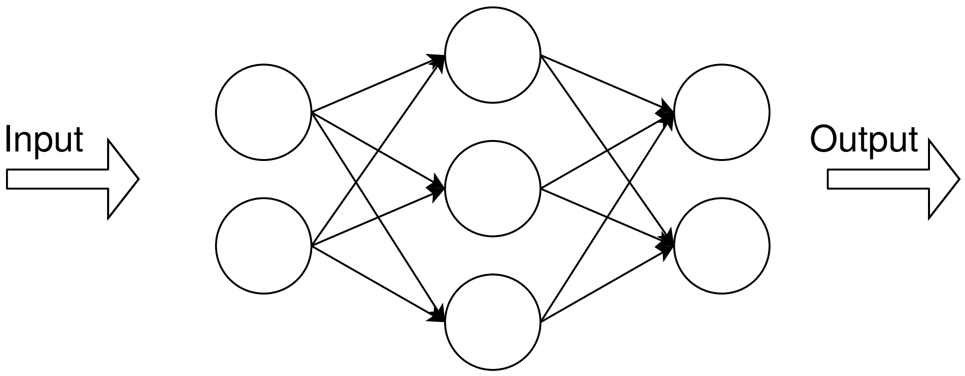


Figure 2.1: An ANN with hidden layer and input, output layer. The circles presents neuron and the arrows presents connectivity and its assosiated weights. Two input nodes and two output nodes with three hidden nodes are shown.

Perceptron is a mere collection of those McCulloch-Pitt neurons structured in layers, also known as a feed-forward neural network(FFNN). It has been proven that such a neural network with as little as a single hidden layer can approximate any function given that a sufficient number of hidden units are available, and activation is continuous and not constant[15]. The most basic and common form of the FFNN is fully-connected or dense layers where all the neuron at one layer is connected to each of the next and the previous layer(See figure 2.1). Computation in such a network happens iteratively by layer and in parallel by neurons on the same layer. While McCulloch and Pitt neuron model utilizes step function as its activation function, multi-layered perceptron requires activation function that can be differentiated when the chosen learning method is backpropagation. Therefore, the sigmoid function is often used as the activation function, but other functions such as tanh, relu functions are also common. The deep learning also employes such a network with some derivation - convolutional network, recurrent network, etc. - with deep structure.

The learning or training of an FFNN happens via adjusting its synaptic weights. The most common method used at the moment is backpropagation[25]. In the training process, the output is calculated for a given input, which we have some idea of how the result should be. In a supervised learning setup, it can the answer/solution for the given input. From the output from the network and the correct solution, an error term is calculated. This error term can be used to evaluate the fitness of the network in training, or it is used to update the weights via chain rule. As the term backpropagation suggests, it propagates backward from the output layer adjusting weights of each connection based on derivates of the activation function of each layer. Backpropagation has shown itself to be useful to train a neural network with some optimization tricks, but it requires a lot of data and computation resources. Another method includes the evolutionary algorithm. A population of random initial weights goes through iteration of evolution by mutation/cross-over and parent/survivor selection. For an FFNN, fixed topology ANN can be evolved via the evolutionary algorithm[49, 12, 1, 4]. More details of evolutionary algorithms will be discussed later in the chapter.

Continuous-Time Recurrent Neural Network

Continuous-Time Recurrent Neural Network (CTRNN), often referred as dynamical neural networks is a subset of ANN which models behavior of biological neurons with ordinary differential equation. The state equation of ith neuron is expressed as follows:

$$\tau_i \frac{dy_i}{dt} = -y_i + \sum_{j=1}^{N} w_{ji}\sigma_j(y_j) + I_i(t) \tag{2.2}$$

Where y is the state of the neuron, sometimes interpreted as an average of membrane potential of the neuron; σ_j is the activation function, usually a sigmoid, given as $\sigma_y(\xi) = \frac{1}{1+e^{\theta_j - \xi}}$, with θ as a bias that controls firing threshold; τ_i is time constant, which decides reactiveness of a neuron; w_{ij} is synaptic weight between neuron i and j; $I_i(t)$ is an external input to the neuron. For hidden or output neurons, this term will simply be zero.

CTRNN is a choice of modeling an artificial neural model for; first, its simplicity; second, it can be interpreted neuro-biologically with synaptic nonlinearities; third, they are known as universal approximator of smooth dynamics[2]. Also, CTRNN is dynamical in the sense that the state of the network changes temporarily varying over time. These properties make CTRNN an ideal candidate when modeling a nervous system such as a biologically inspired controller for a legged robot.

The inherent difference between CTRNN and FFNN is that CTRNN is dynamic, all the node in the network outputs a value continuously in each timestep, where as FFNN does not have temporal property associated with it. Unless it is implemented in a particular way that it interacts in real-time, FFNN is nevertheless a non-temporal function. Although it is possible to train CTRNN with backpropagation[13], the usual choice of a learning algorithm is EA, such as NEAT, HyperNEAT, and GA[1, 22, 39].

2.1.2 Evolutionary Algorithm

An evolutionary algorithm(EA) is a genre of computational algorithm inspired by biological evolution, survival of the fittest - Darwinism and genetic representation of phenomena. It can be considered as an optimization algorithm as it searches for an optimal solution in the multi-dimensional search landscape.

The process of a conventional EA follows as such. First, a population of the solution is initiated, either based on prior information on the problem or random basis. This population then undergoes an iterative evolution cycle. A new generation of the population is generated by a cross-over between two parent solutions or a mutation of a chosen solution in a stochastic manner. Now, the new set of the population goes under a selection scheme based on a fitness measure. The selection scheme can be focused on preserving diversity - which may help to find the global optima, or elitism which may result in faster convergence. The survivors are now the new initial population, which goes through the same cycle until the termination condition is met when a particular objective is achieved, i.e., good enough solution.

Based on variations of genotype representation, and evolution processes, there are commonly known EA methods available[11]. One of the first EA introduced is the genetic algorithm(GA), which had undergone various transformations and has established its canonical, simple genetic algorithm(SGA). In SGA, the genotype is represented in a list of binary values. It can either be understood as a list of logical true or false for specific criteria or a number presented in binary. The population of such bit strings undergoes a parent selection by fitness proportional roulette wheel, then recombination by 1-point cross over and stochastic bit-flip mutation creating a new generation. The SGA is notoriously simplest EA approach, but effective when binary representation is suitable, and also provides a benchmark score when a new EA is tested. Evolutionary strategy(ES) is another popular EA method, characterized by real-valued genotype representation with discrete/intermediate recombination, gaussian perturbating mutation with self-adaptive parameter tuning, and deterministic survivor selection. Other commonly used EA include genetic programming(GP) with genotype represented with a tree structure.

Multi-Objective Evolutionary Algorithm

The EAs introduced above search a solution to a given problem by optimizing a single objective - fitness function. However, it is not uncommon to face a problem that two or more, often conflicting, objectives should be satisfied in order to solve it. The easiest solution to this is scalarization, which aggregates the multiple objectives with some weighting into a single fitness score. This approach has a number of drawbacks, such as 1. required priori for setting weights, and 2.necessity to adjust the weight to different sets of problems[11].

Multi-Objective EA(MOEA) addresses such issues by the concept of *dominance*. A dominant solution is, for an example, for given two solutions and two objectives, the solution that is better in both objectives or at least as good as the other in one objective and better than the other object. This can be extended to multiple objectives; in this case, the dominant solution should have at least one objective higher than others, given that other objectives are equal. When a population of such solutions is present, the dominant solutions form a line on the cluster's outer edge. This line is called *pareto-front*.

While there are a number of MOEA available, it is worth looking into one of the most popular MOEA, Nondominated Sorting Genetic Algorithm II(NSGA-II)[10]. NSGA-II starts off by identifying Pareto-front in the population; then, the Pareto-front is incrementally excluded from the population. The excluded Pareto-fronts are grouped - the first Pareto-front identified will be the best solutions for the current population. NSGA-II introduces crowding distance metric defined by an average side length of cuboid formed from the two nearest neighbors in the same front. Then half of the population is used as survivors to generate the new population while keeping the parents. The survivors are incrementally chosen from the Pareto-fronts, and if the last Pareto-front group has to be divided, the crowding distance metric is used to determine which individuals are included. The figure 2.2 provides a graphical presentation of the survival selection process.

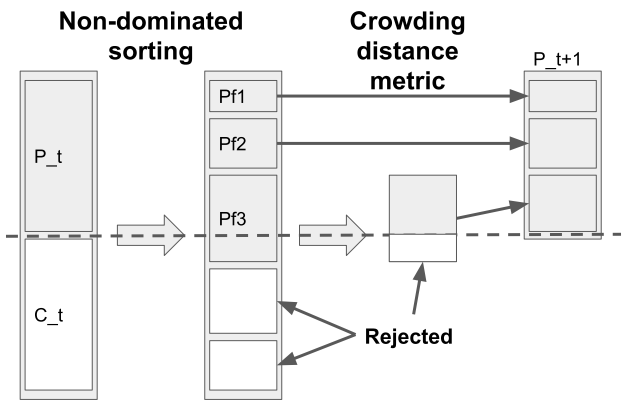


Figure 2.2: NSGA-II survivor selection procedure. P_t indicate survivor from last generation, C_t the child generated from P_t. Pareto-front groups are noted by Pf1,2,3.

2.1.3 Behavioral Diversity

The conventional EAs, which focus on optimizing an objective, has its pitfalls. The EA's search algorithm may end up stuck around a mediocre solution, a local optima in the fitness landscape. A better solution might be hidden behind a small bump in the fitness landscape. Consider a robot designed to find an escape route in a labyrinth. The robot may only try to go straight to the goal, If the absolute distance towards the goal is the only fitness measure. This condition will only create solutions that will go directly to the goal, although the optimal route goes away from the goal.

The solution to the illustrated problem is diversification. However, diversifying the genotype might create solutions that eventually does same thing in terms of its behavior - e.g., an EA designed to evolve a walking gait of a robot could have potentially many numbers of solutions that results in the robot falling immediately. The research community has suggested different methods to tackle this problem.

Novelty Search

Lehman and Stanley[22] suggested Novelty Search which sets aside the objective function completely and only embracing the behavioral novelty of genomes. Novelty Search replaces the fitness function in EA with a *novelty metric*, measuring how unique the behavior of an individual. This creates pressure to search for a new behavior in EA, making the algorithm possible to search for various solutions.

In order to *measure* the uniqueness of the behavior of each genome, Novelty Search in its simplest form utilizes k-nearest neighbor(kNN) algorithm, separating the population into k groups of genomes with respect to their behavioral distance. The novelty metric $\rho(x)$ is then calculated by taking average distance to the k-nearest neighbors, expressed as;

$$\rho(x) = \frac{1}{k}\sum_{i=0}^{k} d(x, \mu_i) \tag{2.3}$$

Where d denotes distance function between the behavior feature vector of ith nearest neighboring genome μ_i with respect to the target genome x. The number of nearest neighbors, k is defined empirically. The calculation of novelty metrics should take account of the current population and the archive of novel genomes. Taking consideration of only the current population may end up circulating around the behavioral feature landscape.

Based on the novelty metric value $\rho(x)$, the high scoring *novel* genomes, exceeding the predefined threshold ρ_{min}, are added into its permanent archive. ρ_{min} is adjusted along with the generations if multiple genomes are added in one generation; it can be increased and lowered if none is added in the given number of generations. Measuring novelty metrics should be done within the problem domain. The behavioral features need to be identified and quantified to fit the application.

The authors of the Novelty Search has shown that it outperforms ordinary objective-based EAs in deceptive applications such as maze search robot controller and bipedal walker problem. Both experiments are done with NEAT algorithm and CTRNN for the bipedal problem.

Mouret[27] has shown that Novelty Search combining fitness with novelty metrics into Pareto-based MOEA can perform better than the simple Novelty Search. This approach is a global competition in a sense that the evolution process optimizes for a single objective, fitness while diversifying solutions in the behavior space. Lehman and Stanley[23] demonstrated that Novelty Search paired with local competition (NSLC) can perform better than the global competition in terms of preserving the diversity, although at the cost of the absolute performance. It was shown by letting genomes only compete locally within a *niche-space*, defined by the nearest neighbors of the novelty metrics.

MAP-Elites

Another well-known approach is Multi-dimensional Archive of Phenotypic Elites(MAP-Elites), introduced by Mouret and Clune[28]. MAP-Elites works on discretized cells in the behavior feature space. Each cell contains one solution and evolves by randomly performing cross-over or mutation of cells and replacing the solution in the cell if the new one has a higher fitness. MAP-Elites effectively illuminates fitness potential in the behavior feature space and reveals relationships between the dimensions of interest and performance.

2.1.4 Evolving Artificial Neural Networks

A human brain, featuring roughly one hundred billion neurons and as many as a thousand trillion synaptic connections, is a result of many iterations of evolution. When we think of it, our first approach to replicate anything remotely resembles the behavior of a living creature would be precisely that of an evolutionary process and underlying structure of neural architecture. Previously, basic methods inspired by nature were introduced, ANN, and EA. Now, the primary focus is to present methods which attempt to evolve an ANN.

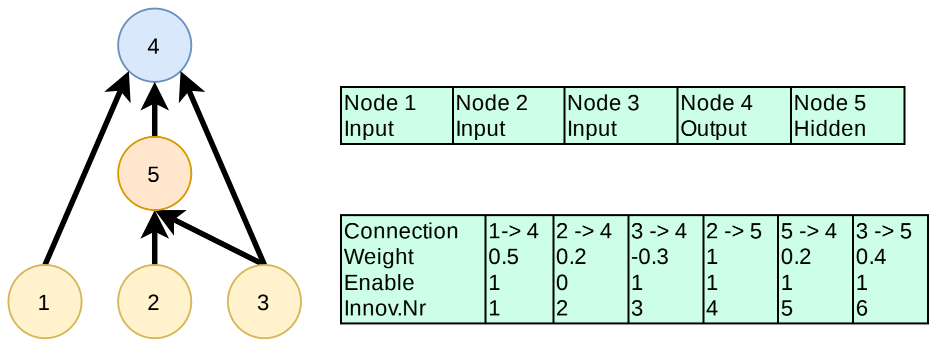
One of the popular attempts of applying EA was to evolve the weight of fixed topology FFNN via GA, as an alternative to the backpropagation. This method has shown itself to be valid in the reinforced learning setup for playing Atari games evolving deep neural networks[49]. NEAT is a method which showed promising result by employing direct encoding and speciation to track the ancestry and keep the divergence of the population. HyperNEAT is introduced several years later, introducing an indirect encoding scheme, CPPN. The HyperNEAT starts with a substrate, a two-dimensional mapping of a neural network with a coordinate system. The coordinates of each neuron and the connections are used as input for CPPN, and the output is the weight of the connection for the given synapse. The CPPN itself is evolved by NEAT. Although the ANN evolved by HyperNEAT has a fixed topology, there have been other variants introduced, one of which including Evolvable-Substrate HyperNEAT(ES-HyperNEAT).

NeuroEvolution of Augmenting Topologies

NEAT has been introduced by [47]. It evolves an ANN by updating its weights and by expanding its topologies through an evolutionary process. Prior to NEAT, there were various NE methods known as topology and weight evolving artificial neural networks(TWEANN), besides evolving only weights on the fixed topology of a neural network. It was one of first that showed the promising result on evolving both topologies and weights at the same time - especially on pole balancing benchmarking jests[47]. Pole balancing task had been traditionally known benchmark in the literature.

Genetic Encoding

In terms of representation, NEAT chooses a direct encoding for its genetic encoding scheme. Genomes in NEAT is composed of two genes, which are node genes and connectivity genes. Node genes list all the nodes(neurons), which can be connected and specifies the types of the node, input, output, hidden, or bias node. The connectivity gene includes a list of connections from the input node to the output node, weight of the connection, innovation number, and enable bit. The enable bit shows whether it is connected or not, disabled if there is a hidden node in between. The innovation number presents unique innovation, let it be a new node or connection within a generation. It is chronically incremented at the appearance. The use of the innovation number marks every gene's historical origin throughout the evolution process.



Node 1	Node 2	Node 3	Node 4	Node 5
Input	Input	Input	Output	Hidden

Connection	1-> 4	2 -> 4	3 -> 4	2 -> 5	5 -> 4	3 -> 5
Weight	0.5	0.2	-0.3	1	0.2	0.4
Enable	1	0	1	1	1	1
Innov.Nr	1	2	3	4	5	6

Figure 2.3: NEAT representation of phenotype and genotype

Mutation

NEAT evolves both weights and topology by mutation. The weights are mutated just like other NE systems, perturbation. In mutation of the topology, NEAT can either add a new connection between existing nodes or add a new node, dividing the current connection in half. Adding a new connection assigns a random weight. Adding a new node required more complication, where it needs to add an extra connection. In order to minimize the consequence of the new structure, which reduces fitness in most cases, the incoming connection is assigned a weight of 1, and the outgoing connection is assigned the same weight as the previous connection.

Cross-over

The genes of two networks can be lined out with corresponding connection with innovation number. Since each unique mutation in a generation is given an innovation number by keeping the list of the innovation of the generation, the system knows the historical origin of each gene, and how genes can be matched.

In the cross over, the two parent genes are line up by their matching gene - those with the same innovation number. The genes not shared by the two parents are called either disjoint or excess, based on whether they are within the boundary of the gene, disjoint, or the outside of the boundary. The offspring will randomly choose at all of the matching genes from each parent while including all the excess and the disjoint from the more fit parent. In the case of equal fitness, excesses and disjoints are inherited randomly between parents.

Speciation

Mutation in both weight and topology, followed by the cross over with innovation numbers, makes it possible to grow the population in diverse topologies. However, new structures are likely to get decreased fitness initially, which may lead to early extinction. NEAT implements speciation to protect the new structures by letting the niche individuals compete with each other.

The number of excesses measures the similarity and disjoint between two genes. the compatibility distance δ is expressed by the following equation

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \overline{W} \tag{2.4}$$

Where E and D, number of disjoints and excesses, N number of genes in the larger genome, $\overline{W}$, the average of weight differences of matching genes. The coefficients c_1, c_2 and c_3 are used to adjust the importance of the three factors, used as the parameters for NEAT.

An ordered list of species is maintained based on a compatibility threshold. At each generation, each genome is sequentially put into their species. Species are represented by a randomly chosen genome inside the species from the previous generation. For a given genome, there could be multiple species that fit under the threshold; in this case, the first species will hold that genome. If matching species are not found, it creates its own species.

NEAT uses explicit fitness sharing, where all the niche genome shares the same fitness. According to the distance between other species, the fitness of the species is adjusted to give a penalty to a species with a large population,

$$f'_i = \frac{f_i}{\sum_{j=1}^{n} \mathrm{sh}(\delta(i,j))} \tag{2.5}$$

Where $\delta(i,j)$ is the distance between two networks and the sharing function, sh() defined as

$$\mathrm{sh}(\delta(i,j)) = \begin{cases} 0, & \text{for} \quad \delta(i,j) > \delta_t \\ 1, & \text{for} \quad \delta(i,j) \leq \delta_t \end{cases} \tag{2.6}$$

Here δ_t is the predefined threshold value.

Since the denominator of the equation 2.5 will increase as the species has a larger population; it will get a larger penalty and get eliminated, resulting in balancing population distribution between the species.

The number of offspring for each species is proportional to the f_i', allowing species with better fitness producing more offsprings. The species with the lowest fitness is eliminated.

Initialization

NEAT initiates its population with a uniform distribution of minimalistic network consisting of only input nodes and output nodes. New architectures are generated throughout each generation, and speciation undergoes. This minimalistic approach ensures the search-space with fewer dimensions.

Hypercube-based NeuroEvolution of Augmenting Topologies

When it comes to comparing our attempt with an actual human brain, not only that ANN has much fewer neurons in orders of magnitudes, but also much less organization, and regularity. This implies that there is something missing in the methods mentioned so far. In DNA, massive structures are represented compactly; the repeated structure is presented by a single set of reused genes, in mapping from genotype to phenotype. Brain is often organized according to the geometry of the physical world as in an example of symmetry found in shape and location of ears, retinotopic map, and the corresponding eyes. Neurons located nearby in the brain are used to process the related information since more the distance means more the effort to maintain connectivity.

The ANN architectures and NE methods introduced so far ignore the structure in general. In a simple FFNN, the order of input neurons is something irrelevant to the learning algorithms, such as backpropagation or genetic algorithms. Although the convolutional neural network is capable of capturing spatial features from the given data, the location of the neurons remains abstract.

Stanley et al. suggested HyperNEAT addressing the problems by exploiting geometry and indirect encoding[45]. It employs connective CPPN, which is capable of presenting connectivity patterns as functions of cartesian space.

Compositional Pattern Producing Networks

Inspired by studies of developmental encoding in biology, CPPN tries to reproduce how extensive structural information is encoded into few genes by presenting repetitive patterns in Cartesian space.

CPPN produces a spatial pattern by composing essential functions. Stanley has shown that composed simple canonical functions can produce complex regularities and symmetries[44]. It can be interpreted such that these simple functions represent symmetry - gaussian, or division of discrete segments like fingers - sinusoid.

Functional wise, CPPN represents an indirect encoding of ANN. However, in terms of its structures, the two are comparable, the only difference being that ANN uses sigmoid in general, and CPPN includes many others. Because of this property, NEAT can be easily extended to evolve a CPPN. Instead of assigning a sigmoid function to hidden nodes, as in the vanilla NEAT cases, a random activation function can be assigned when a new node is created by mutation.

HyperNEAT

Instead of generating a spatial pattern, which takes an input of a 2d coordinate and outputs an intensity of that point, HyperNEAT uses connective CPPN which takes a coordinate of two nodes and outputs a weight for that connection(See figure 2.4). Thus producing connectivity patterns. The geometrical topology of the neurons is called a substrate. In vanilla HyperNEAT, the substrate is given a priori designed to fit the characteristic of the problem. Stanley has shown that the connective CPPN could easily generate important connectivity regularities such as symmetry, imperfect symmetry, repetition[45].

The capability of taking geometry into account when generating a connectivity pattern implies that the substrate can be configured to represent the location of sensors and motors in a robot model. Also by feeding 3d coordinates as an input to CPPN, connectivity pattern of a 3d substrate can be generated similar to the geometric regularity found in the brains.

Having an explicit function that generates a pattern of connections also means that an evolved CPPN can take a new node and connection without re-evolving the network and the behavior of that will fall into the generality of the rest of network. It also implies that it is possible to increase the resolution of the substrate, without further evolution, although it may display some artifacts that was not discovered before.

The evolutionary process in HyperNEAT will start with choosing an appropriate substrate for the problem. Then a population of CPPN is initiated with a minimal structure and random weight. The same iteration as NEAT evolution is done, only that fitness is measured by computing the substrate as a regular feed-forward network for evaluating the output.

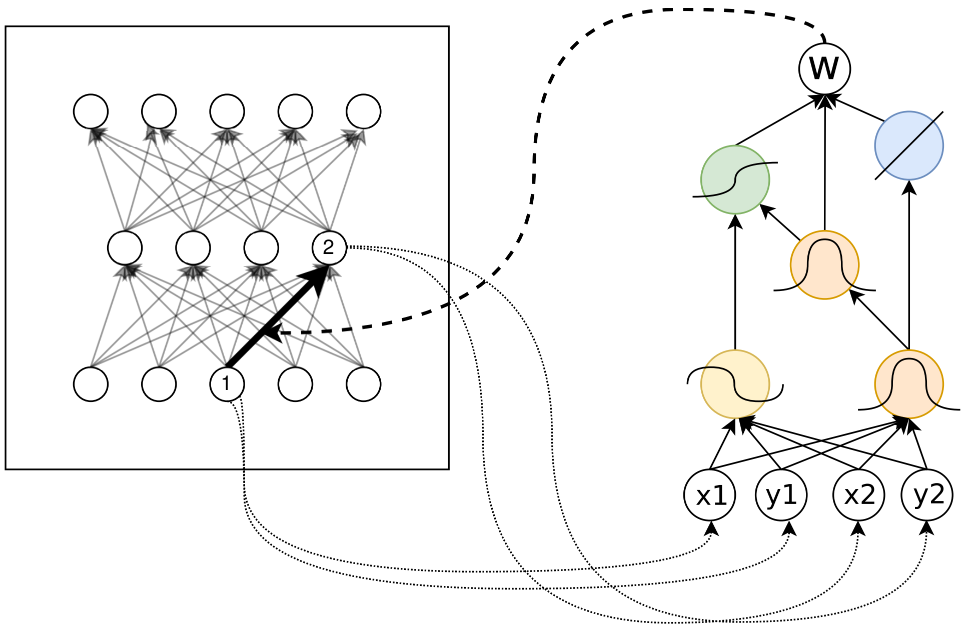


Figure 2.4: Weight of a connection between neuron 1 and 2 is queried by CPPN

Adaptive ES-HyperNEAT

There has been various approaches and derivations of HyperNEAT over the years, including adaptive, HyperNEAT, HyperNEAT-LEO, ES-HyperNEAT, HybrID, and HyperGP[9]. In this section, basic concepts on adaptive Evolvable-Substrate HyperNEAT is introduced which is capable of evolving more biologically plausible artificial neural network.

While NEAT could evolve the topology of the network using direct encoding, the HyperNEAT required a predefined substrate. Although HyperNEAT is designed to exploit the geometry, such as the location of sensors and motors as input and output nodes, it is not trivial to decide the locations of the hidden nodes. Risi et al. introduced a method called Evolvable-Substrate HyperNEAT, which could deduce the location of a new hidden node from CPPN, thereby evolving the substrate of HyperNEAT at the same time[38].

The insight behind the method is that CPPN takes in a 4d vector as an input(for 2d substrate), the location of the two nodes, and the intensity output of the network express the connectivity of the network, giving out the weight of that specific connection. Since the domain of the CPPN is continuous, any point in the output space is a possible connection in the substrate. Thus by sampling the CPPN with a specific interval, the area of interest, where CPPN outputs high intensity, can be found. This, in other words, maps the density of the connectivity information. Useful connectivity information is found by quadtree search by focusing on the variance of the density map, considering that a uniform gradient won't provide a meaningful connection. ES-HyperNEAT starts with searching for a possible connection from the input node by iteratively querying the possible connections until the output node is reached.

Another perspective is the adaptivity of an ANN. A plastic neural network is an ANN with variable weights opposed to the usual fixed weight ANN. These networks have shown to be more effective at solving problems than the fixed ANNs. By embedding Hebbian learning rule into the HyperNEAT, adaptive HyperNEAT can evolve a plastic neural network[40]. The CPPN incorporates Hebbian learning parameters as one of the outputs in the adaptive HyperNEAT.

By combining these two methods, Risi[38] has proposed adaptive Evolvable-Substrate HyperNEAT, which is capable of learning in a lifetime by plasticity and evolve substrate of the network from implicit information given by the CPPN. This unified approach gives the possibility to evolve more brain-like ANN by its indirect encoding with CPPN, plasticity with Hebbian learning at the same time evolving the substrate of the ANN itself.

2.2 Locomotive Controllers

Legged robots are more versatile than wheeled robots. It is capable of maneuvering through rough terrains and narrow alleys where wheeled vehicles or drones can't access. The development of a legged robot system poses more challenges than conventional wheeled ones. The most conventional approach is to develop a hand-crafted controller. One of the challenges is to coordinate a high degree of freedom, which makes locomotion possible. A simple insect-like quadruped will require at least 8 to 12 degrees of freedom, considering 2-3 motors on each leg. All of these motors need to work on harmony in order to make it walk, or even to let it stand. Another problem is that the developer needs to spend a considerably long time programing each and every maneuver, such as walking straight, turning, stopping, and so on. Even after all of these have been accomplished, the developed controller is confined to that specific morphology, with no chance of transferring the controller to the other body. Many attempts have been made to address those problems listed, an effort to develop an adaptive locomotion controller. However, most of the state-of-the-art legged robots are built in a traditional way.

2.2.1 The State of Art

The modular controller design is the most popular approach in designing a legged robot system, demonstrated by state-of-the-art systems such as [20]. In this approach, the system is broken down into smaller decoupled submodules. Each module is built on a certain template dynamics or some heuristic values. The template-dynamics-based control module approximates the next foot position. From the estimated foot positions, the next module calculates a trajectory. The last module moves the leg, following the trajectory with a proportional-integral-derivative controller.

Despite the state-of-the-art performance, this method inherits all the problems and challenges mentioned above.

Another promising approach is trajectory optimization. The control scheme is divided into planning and tracking. A trajectory is planned and optimized using rigid-body dynamics and numerical optimization. The tracking module makes sure that the trajectory is followed. Although this approach features a more automated way to develop a system, the current optimization techniques are not capable of managing such a complex model. In practice, approximation or a partial optimization is used.

Most recently, a data-driven method, based on reinforcement learning, has been shown to be sufficient for designing locomotive controllers. In these methods, a deep neural network is trained to form a control policy from a large set of data created by trial and error. Most of the related work has been focused on the simulated environment, but Hwangbo et al. have successfully implemented a trained RL agent on a physical system[16].

2.2.2 Biologically Inspired Methods

Consider a control scheme of a four-legged mammal. You can easily imagine hundreds of muscles coordinating themselves perfectly to generate a gait, or whatever they are up to for the moment - running, crawling, or resting. However, it may seem apparent that the brain itself does not actively control each and every single muscle. Instead, small neural circuits are found both in vertebrates and invertebrates, which directly governs signals controlling each muscle. These small neural networks are known as Central Pattern Generator(CPG), which are capable of generating rhythmic patterns without rhythmic signal input.

There were two approaches to how CPGs work, one relying solely on its sensory feedback and the latter, centrally generated patterns [5]. Now there is clear evidence that the rhythmic patterns are generated centrally, but the importance of sensory feedback has been demonstrated via various experiments(decerebrated cat, lamprey), showing that there is a tight coupling between them. One of the experiments showed a decerebrated cat on a treadmill managed to keep its walking gait, even freely changing to running gait depending on the speed of the treadmill[41]. In this case, keeping the gait steady has been mainly guided by its sensory feedback.

In the light of the gait transition, it has been shown that a very simple signal from the central nervous system can easily control the change of the gait. Electric stimulation in the brain stem can initiate rhythmic pattern generation. Simply changing the intensity of the signal can cause a gait change, in case of a cat, from walking to galloping, for a salamander, from walking to swimming[17].

From findings and description of the general principle of CPG, one can easily imagine a low-level controller scheme for a robot which requires complex and sophisticated movement. Modeling characteristics and behavior of CPG will benefit designing a locomotive controller for a legged robot.

Several different approaches have been made in an effort to model and implement the CPG in robotics. One of the popular approaches is the mathematical modeling of CPG with a set of coupled differential equations[17, 43]. An oscillator node, a differential equation, is placed on each joint with a set of parameters such as phase, amplitude. These parameters are then tuned either by hand based on prior experiments[17] or optimized by an algorithm[43]. These oscillator nodes form a network with weighted connections between them. Ijspeert has managed to implement this CPG approach to re-enact gait transition of salamander, from walking to swimming in an actual robot[17, 8]. Another CPG approach that puts emphasis on the importance of sensory feedback is Tegotae introduced by [37]. Tegotae utilizes sensory feedback for stable interlimb coordination.

Another popular approach is the connectionist models. These models usually incorporate a small network of a simplified neuron model such as leaky-integrator neurons, otherwise known as a continuous-time recurrent neural network(CTRNN). The parameters for the network are usually evolved by genetic algorithms. From the early '90s. Randal Beer and his colleagues have been extensively studying the implementation of CTRNN[1, 4] and analysis of its dynamical property[3, 6].

More recently, CPG was implemented in conjunction with CPPN which showed in a significant result in combination with Lamarckian evolution[19, 18]. In this method, CPG is implemented as two differential oscillators with feedback connections to each other. Then each CPG is mapped on a 2D substrate, which describes the morphology of the robot. The coordinates of the CPGs are fed into CPPN and produce weights after iteration of evolutions.

2.3 Previous Works

2.3.1 DyRET

DyRET: Dynamic Robot for Embodied Testing, is a robotic platform which features a quadruped with self-modifying morphology - meaning that the robot can change its morphology mechanically at the runtime. DyRET is developed at University of Oslo by Nygaard et al.[34] with a goal of bridging the reality gap - brought by a substantial difference between a simulation environment and the physical robot. The applications that DyRET is aiming for are automatic design, environment adaptation, automatic evaluation, and meta-studies[34]. Also, DyRET has been utilized in some experiments such as evolving fast and stable gaits in physical robot[33], adapting morphology and control to hardware limitations by evolving in real-world[36], Tegotae style feedback loop for robust locomotion using

CPG[32], and evolving gaits by combining incremental evolution and MAP-elites[29].

As being a platform that provides a physical robot with dynamic morphology, one of the main focuses on DyRET is to investigate the control-morphology problem. So far, the control systems that DyRET has been tested are

- A high-level inverse-kinematics based position controller, which calculates the trajectory of leg movement for given landing feet position[35],
- A low-level gait controller parameterized with amplitude, phase, duty cycle, offset, and gait period for each joint that describes a continuous first-order spline presenting destination angle for each joints[29].
- A network of oscillators as CPG coupled with feedback loop described by Tagotae[32].

2.3.2 Flexible Controllers

Evolving a gait controller that can be utilized in various morphologies is a challenge, as dynamics of gait changes with the change of the morphology. The longer legs on a robot would result in a higher center of gravity that the controller should deal with either by lowering the center with different gait or changing dynamics of movement - e.g., a larger swing of a leg. However, in nature, it seems that there is an intriguing relationship between morphology and control as seen by an example given by [24] - a new born foal that can walk almost immediately after birth. The approach should rather be that instead of learning to walk for a given morphology, the relationship should be learned[39].

Besides the adaptive CPG-based control system by [24], which introduced a distributed oscillator system called Adaptive Ring Rules that can adapt its parameters to the change of morphology by matching sensory feedback of the previous morphology, in the field of ER, Risi's work[39] of evolving a single instance of gait controller for various morphologies is unique; although there have been studies in ER that evolves the morphology and the controller at the same time[21, 19, 18].

The flexible gait controller introduced by Risi[39] learns the relationship between the morphology and the controller by utilizing a CPPN as a function that takes the morphology as input and outputs a corresponding controller for that particular morphology. The suggested method implements HyperNEAT with a multi-module substrate, which is composed of submodules that controls each leg with inter-modular connections between specific nodes on each submodule. The CPPN takes in the input of a pair of 4D coordinates for connections between each node and the leg length input L. Then the output from the CPPN completes the substrate, which is expressed as CTRNN. The controller is evolved by novelty search with the trajectory of the robot as the behavior descriptor for measuring the distance between individuals. The experiment compares performance between the static controller and the flexible controller, where both controllers are evolved while evaluating three different leg length configurations. Risi reports that the flexible controller outperformed in the evaluation of interpolated morphologies that have not been seen during the learning. Also, Risi provides an analysis of how the evolved flexible controller learned the relationship between the morphology and the control by showing a correlation between L input variation on the output of each joint.

Chapter 3

Implementing a Flexible Neural Controller

Risi[39] has shown that it is possible to evolve a flexible controller which can adapt to different length of legs with HyperNEAT and a special CPPN which takes leg length as an input. As DyRET is a robot platform which is designed to change its leg length for different environments, implementing Risi's HyperNEAT on DyRET's platform can be interesting to confirm Risi's method and further investigate improvements and pitfalls of implementing it on an existing system. Furthermore, Risi implements only Novelty Search as its learning method for his experiments. As covered in the previous chapter, there are vast possible learning methods available within the boundary of EA, such as traditional fitness only evaluation, or MOEA combining multiple objectives. Also, implementing a behavior diversity based EA will require attention while choosing which behavior to look at and which one would be worthwhile to measure.

To test the research questions given in Chapter 1, an experiment environment is implemented using various tools that are openly available. In this chapter, details of implementing flexible neural controllers are described on each component of the final system, both from readily available packages and own implementations.

3.1 Overview

Figure 3.1 shows the overall flow of the final design to evolve a flexible neural controller. A population of genomes is initialized at the beginning. Each genome, CPPN in this case, generates weights, bias and time constants for the pre-defined substrate with CTRNN properties based on Beer[1, 4]. The generated CTRNN interacts with the simulation environment, which is set up with OpenAI's Gym environment, specifically implemented for DyRET[30]. For each iteration of the simulation time steps, the reward is calculated. The reward summarizes the status of the robot, whether the robot is moving or not, has fallen or not, etc.

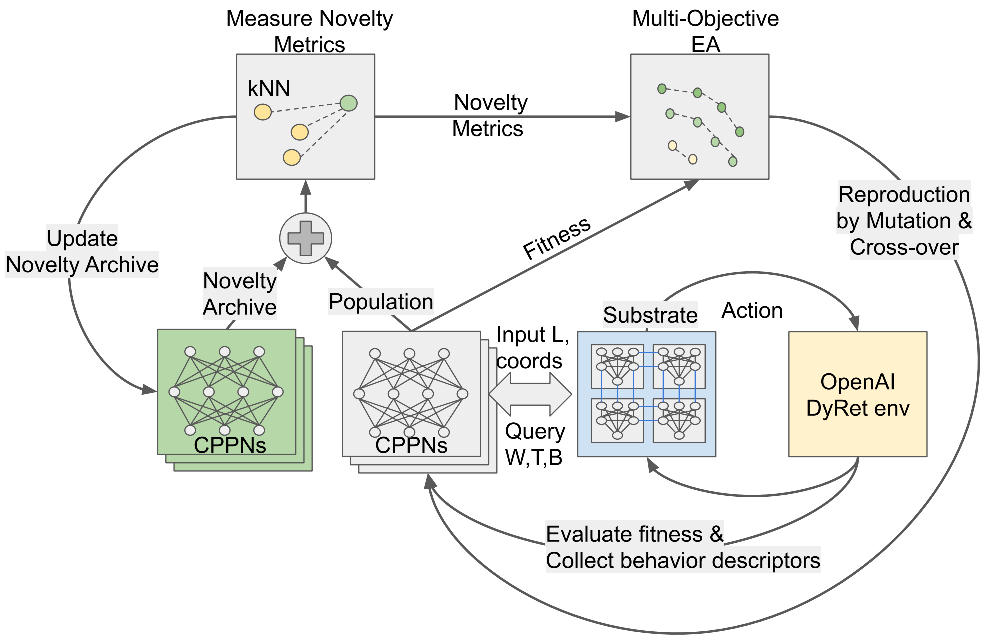


Figure 3.1: Overview of the system. Substrates query the CPPNs with the coordinates of nodes and leg length L, and CPPN returns weight, W, time constants, T, and bias, B for each connection.

The rewards in each iteration are accumulated as the fitness of the genome. At the same time, behavior descriptors are collected for each genome, which is evaluated by Novelty Search calculating *novelty metrics*, average distance of defined behavior descriptor to *k*-nearest neighbors. Outstanding genomes with higher novelty metrics than the threshold distance are collected in Novelty Archive. With the fitness and novelty metrics together, *pareto-front* is formed, and a new population is generated by NEAT's reproduction algorithm - mutation, cross over, and speciation. As the Novelty Archive is initialized, the population and the archive is merged when measuring novelty metrics.

3.2 Simulation Environment

The simulation environment is realized in an OpenAI Gym environment specifically for DyRET[30][1] The physics engine of this simulation environment utilizes *pyBullet*[2] which can simulate collision detection, soft and rigid body dynamics.

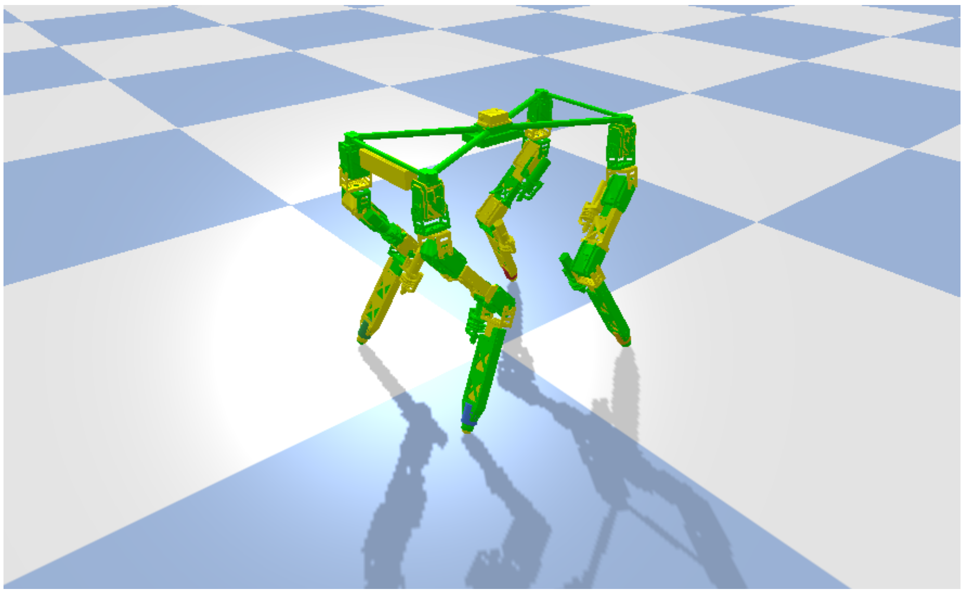


Figure 3.2: View of the simulation environment

3.2.1 Environment

The simulation environment comes with a various ready-made environment to test - walking and standing with pre-defined reward function. In this thesis, only the walking environment is used with its default reward function. The reward function will evaluate the robot based on whether the robot is standing upright, y-axis velocity, deviance velocity over a threshold, and a norm difference of force applied in each joint. The details of this reward function are discussed in section 3.4.2 on fitness function. Besides the default reward function, the environment which the robot is interacting with is flat surface as seen on figure 3.2.

3.2.2 Robot Interface

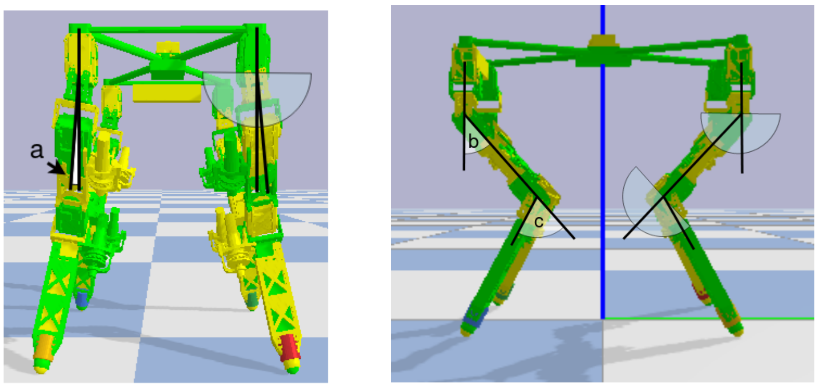
The DyRET- OpenAI Gym environment provides an interface to control the action of the robot under evaluation and an environment where the robot can interact. The robot is controlled by specifying the destination angle of each revolute joint and destination length of each prismatic joints in each time step. The default time step for the physics engine simulator *pyBullet* is set as 1/240 second.

In each simulation time step, new joint values are put into a velocity motor with some friction associated with it by specifying maximum force. All the revolute joints are set with maximum velocity, and maximum force value. However, prismatic joints are set without maximum force value, and

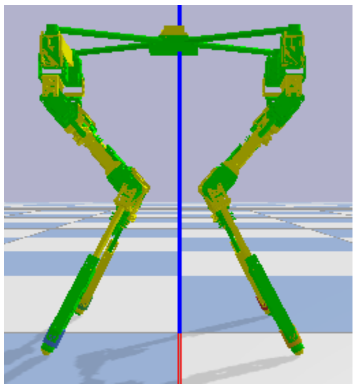
[1]https://github.uio.no/jorgehn/gym-dyret. At the time of writing, this Gym environment is not published for the wide public but only to those who have affiliation with University of Oslo.

[2]https://pybullet.org/wordpress/

unrealistically high maximum velocity value so that testing each leg length configuration can be done without delay.



(a) Front (b) Side



(c) Fully extracted prismatic joints

Figure 3.3: In the subfigure (a) and (b), full range of the joints angle are noted on the right side of legs, and angle *a*, *b* and *c* which notes the angle of the default pose. The subfigure (c) shows fully extracted prismatic joints with the same default pose.

Figure 3.3a and 3.3b shows the default pose used in the experiments. The full range of angles is shown with the zero points in the middle. Each revolute joints can rotate from -90 to 90 degrees with zero angle indicated in the middle of the half-circle. The input values for controlling the robot range from $[-\pi/2, \pi/2]$. The prismatic joints take an input range of $[0, 45]$ for the upper leg and $[0, 100]$ for the lower leg. The figure 3.3c shows DyRET setup with maximum input value for prismatic joints.

3.3 Controller

The controller scheme implemented in this thesis is based on Risi's work [39], which is evolved by HyperNEAT with the multi-layered substrate as its topology with the characteristics of CTRNN. The figure 3.4 illustrates the flow of the controller scheme. The substrate of the CTRNN network in the

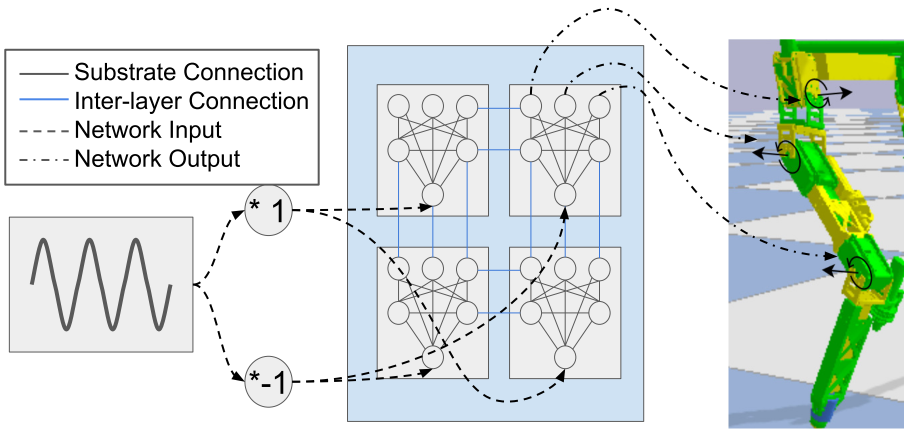


Figure 3.4: Controller scheme. The input from sinusoid and output of CTRNN are illustrated with dotted arrows. Each layers - modularized substrate controls each leg segment layed out geometrically.

middle takes input values from a sinusoidal wave. Then each leg module substrate outputs each destination angle value. The coordinates of each leg module correspond to the geometrical position of each leg. Simply put, the upper left module for the front left leg, the lower-left module for the back left leg, and vice versa. The coordinates of the output nodes in each leg module also correspond to which joint they output - the innermost nodes for the uppermost joint and the outermost nodes for the lowermost joints. For the details on the substrate, refer to section 3.4.1.

3.3.1 Neural Controller

CTRNN is implemented by uzilizing CTRNN node provided in *neat-python* package [3]. This package implements CTRNN by the following equation.

$$\tau_i \frac{dy_i}{dt} = -y_i + f_i\Big(\beta_i + \sum_{j \in A_i} w_{ij} y_j\Big) \tag{3.1}$$

where τ_i is the time constant of i-th neuron, y_i is the membrane potential of neuron i, f_i the activation function of neuron i, β_i the bias, A_i set of connected input neurons to neuron i, and w_{ij} the connection weights between neuron j and i. The CPPN for each genome provides w_{ij}, τ_i and β_i for each node and connections. The activation function, f_i is sigmoid function and A_i is defined by the substrate.

This differential equation is computed using the forward Euler method given as;

[3]https://neat-python.readthedocs.io/

$$y_i(t + \Delta t) = y_i(t) + \Delta t \frac{dy_i}{dt} \tag{3.2}$$

Where Δt is the time step specified when the network is instantiated. The time step value is empirically set as 5/240 s, which is five times larger then the time step of the physics simulator.

Note that the equation 3.1 implemented in this package is slightly different than the equation 2.2 introduced in the chapter 2, section 2.1.1, where the aggregation and weight multiplication happens after applying the activation function to the output of the connected neuron. Despite this difference, the principal concept of CTRNN holds in both equations - continuity, recurrence, and, consequently, the dynamic property of CTRNN.

3.3.2 Input and Output

While Risi's[39] implementation uses the current state of hip joint angle as input to the network - effectively making it a closed-loop controller, the input for CTRNN used in this thesis is a sinusoidal wave. The rationale behind this convention is having a potential extendability towards evolving an agent that can drive the evolved controllers. The input sinusoid is given by

$$Input = \sin(i/\text{freq} \cdot \pi) \tag{3.3}$$

Where i is each time step in the simulator, defined as $1/240s$, freq is a parameter, empirically defined as 32. This gives frequency of $\omega = \frac{2\pi}{2 \cdot 240 \cdot 32}$, which equals $f = 15.360$ kHz. Adjusting this frequency usually results in the change of the output frequency. Furthermore, the sinusoidal wave is multiplied with -1 to the input for both front right and back right leg. This constraint helps to evolve a stable quadruped trot gait.

The network's output is constrained with the direction of each swing and the stable standing position as a null point. Since the activation function for each node is the sigmoid function, the output range is confined to $[0, 1]$. Then the output values are adjusted to its null position, as noted as angle a, b and c by the figure 3.3a and 3.3b. The directional constraints multiply -1 for specific joints so that they only revolute further from their null position. These constraints help in search of stable gait. Figure 3.5 shows each joint output after constraints plotted against a timescale.

3.4 Implementing Evolutionary Algorithm

3.4.1 HyperNEAT

HyperNEAT is implemented based on the NEAT algorithm from *neat-python* package. While keeping the default NEAT evolution process

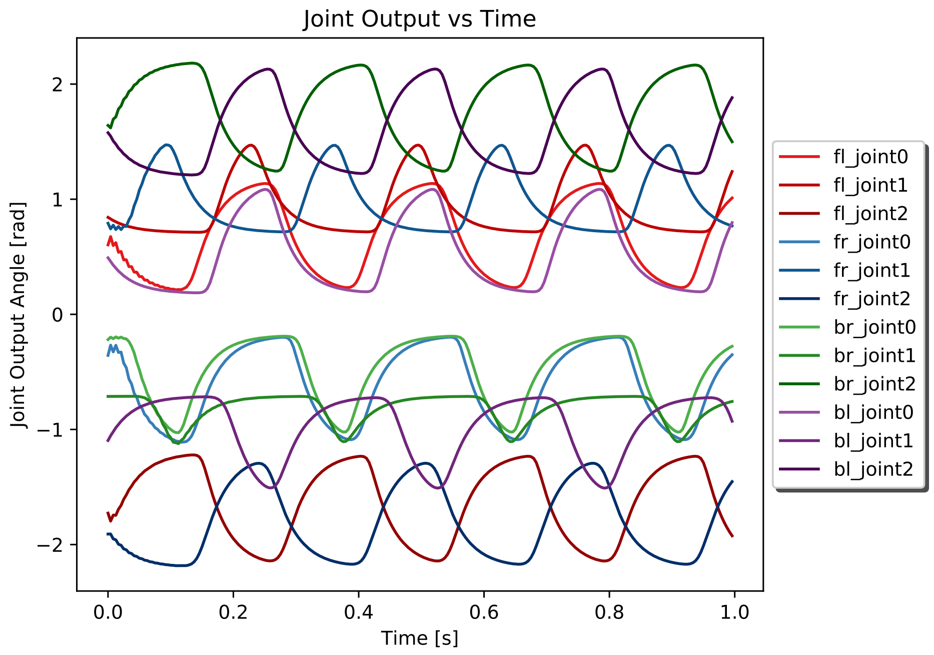


Figure 3.5: Example of joint output from CTRNN after applying the constraints. Naming conventions for each output are fl for front left, fr for front right, br for back right, and bl for back left. Joint 0 is the hip joint, 1 for the middle joint, and 2 for the lower most joint.

provided in the package, all the NEAT genome is now considered as CPPN, and it generates weights, time constants, and bias of the substrate.

The substrate is inspired by Risi[39], a modularized substrate where each module controls each leg. The result is a four-dimensional substrate. The geometric positions of each neuron are described by the 4D coordinates. The first two coordinates (xm, ym) denote the position of the leg-module, and two following coordinates (x, y) represent the position of each neuron in the leg-module. Besides inter-modular connections and intra-layer connections, i.e., hidden node to hidden node connection, all connections are one-directional, going from the input layer to the output layer. See figure 3.6 for the layout of the substrate.

Two sets of the four-dimensional coordinates, along with the leg length configuration, L are used as input for CPPN, see figure 3.7. The CPPN outputs connection weights, W, between the neurons, time constants, T, and biases, B. By convention, when querying CPPN for T and B, only the first set of four-dimensional coordinates are provided as input for CPPN while keeping the rest of the coordinate as zeros. This is due to the fact that the weights are described as connections and the time constants and biases are the properties of each individual neuron. The activation functions for CPPN nodes are gaussian, sinusoid, absolute and sigmoid, randomly chosen during the mutation stage of the NEAT algorithm while the default

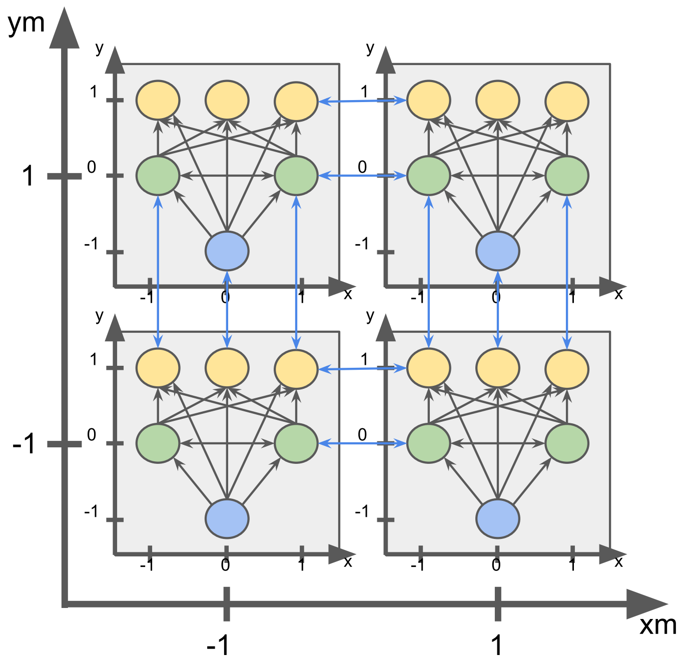


Figure 3.6: The Substrate design. Blue node indicates input nodes, green for hidden and yellow for output. Weights between connections and time constants, bias for each node are queried from CPPN.

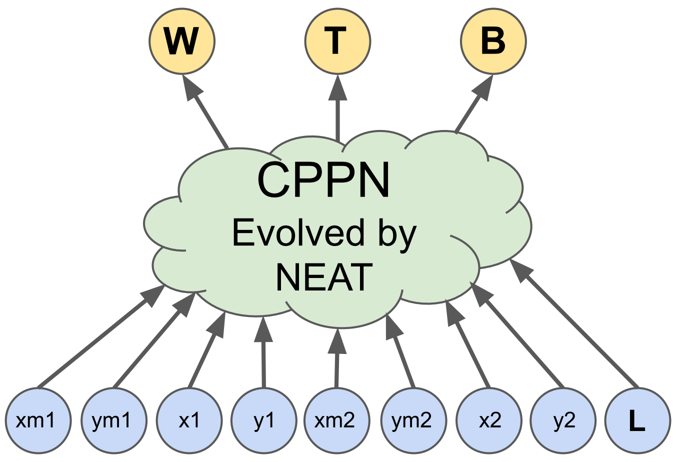


Figure 3.7: The CPPN features 9 inputs, 2 sets of 4d coordinates and leg configuration, L. 3 outputs are W, weights, T, time constants and B, bias. The hidden layers are evolved by NEAT, adding node and connection with random activation function.

activation function is sigmoid.

3.4.2 Fitness Function

The fitness function is designed to consider both the stability and mobility of the robot. In general, the genome will score higher fitness if the robot stands upright as much as possible at the same time moving along the y-axis and moves the furthest away from the starting point of the robot. The fitness function $f(g)$ for a genome g used in this thesis is given as the following;

$$f(g) = \sum_{i=0}^{I-1} (100 \cdot \mathrm{H}_i + \mathrm{U}_i + V_{y_i} + V_{d\,i} + \Delta\tau_i) \tag{3.4}$$

$$+ 10^5 \cdot ||p_{(x,y)}(I)|| - 10^5 \cdot \text{Penalty} \tag{3.5}$$

The first terms, eq.3.4 are pre-defined in the DyRET - OpenAI Gym environment, accumulated through I timesteps for each i-th iteration, and defined as following;

- H : If the robot is healthy or not, based on the center height z and roll, R and pitch P

$$\mathrm{H} = \begin{cases} 1, & \text{if} \quad z > 0.75 \quad \text{and} \quad |R| < 0.7 \quad \text{and} \quad |P| < 0.7 \\ -1, & \text{otherwise} \end{cases} \tag{3.6}$$

  This term describes whether the robot is in a fallen state, or in low position which is not desired as the quadruped gait for DyRET requires the mass center of the robot to be sufficiently high.

- U : How upright the robot is standing from roll R and pitch P. 0 if the robot stands upright.

$$\mathrm{U} = -(R^2 + P^2) \tag{3.7}$$

  This gives credit for stable gaits.

- V_y : Velocity reward as high linear velocity along the y-axis is desired.

- V_d : Deviance reward that discounts for motion deviating from the current velocity as constant velocity is desired. It is given by

$$V_d = \begin{cases} -|V_y - Va_y| & \text{if} \quad Va_y \geq 0.3 \\ 0, & \text{otherwise} \end{cases} \tag{3.8}$$

  Where Va_y is an accumulate velocity to calculate deviance, given by

$$Va_{y_i} = V_y \cdot dt + Va_{y_{i-1}} \cdot (1 - dt) \tag{3.9}$$

  dt is the default simulator timestep at 1/240 s.

- $\Delta\tau_i$: Norm of applied torque difference between the previous timestep τ_0 and the current torque τ for each joint j, given as

$$\Delta\tau = -1 \cdot \sqrt{\sum_{all\,j} (\tau_j - \tau_{0\,j})^2} \tag{3.10}$$

  This term discounts for unwanted jitters in the movement.

The second terms of the fitness function, eq.3.5 is a bonus added at the end of the simulation, based on absolute distance traveled from the starting position and penalty given for being unstable or not moving. The details are given as;

- $p_{(x,y)}(I)$: The final position of the robot
- Penalty : True if the Healthy reward, H has been negative or has not been moving for 100 timesteps. Whether the robot is moving is evaluated by checking the norm difference of each joint output value from the previous values.

The most decisive factor that would create pressure in the optimization process is the healthy point H throughout the simulation due to the weight given as 100, and the second terms, whether the robot has been moving, and if so how far it has moved from the start. With these factors, EA will give enough pressure for each genome to create a controller that is constantly moving, trying to get away from the starting point as far as possible while trying to keep stable gaits.

3.4.3 Novelty Search

The core concept of Novelty Search is to replace the fitness with *Novelty Metrics*, giving a higher score to a genome that is unique in terms of its behavior. The first step in implementing Novelty Search is to introduce a method to measure the behavioral distance between genomes and replace it with the existing fitness function. Then the EA should be modified for Novelty Search by fitting the population's behavior into *k*NN classifier, which will measer the distance between each other. The genomes with high novelty metrics will be permanently archived and used to compare with the population along the evolutionary process.

Behavior descriptors

The first step is to define the behavior descriptors, which would be significant to distinguish between genomes. Three behavior descriptors are defined - Max-min amplitude of each joint, duty factor, and the last position of the robot.

Max-Min amplitude of each joint

This descriptor measures difference between the maximum and the minimum amplitude of each joint. This is useful when differentiating between robots that move more lively - in a sense that it fully utilizes the range of each revolute joint angles and the others, which moves more quiet using least of energy. For each joint, the maximum output and the minimum output is recorded. In the end, the difference between the two values is used. This results in a (1×12) vector.

Duty factor

Duty factor is a percentage of the total cycle when a given foot is on the ground. This descriptor is useful to distinguish between different gaits - such as trotting gaits, running, bounding gaits, and so on. To measure it, first, the history of each foot touching the ground is recorded in each simulation time step. This would result in a 2-D array with a size of 4 times a number of simulation iterations. Yet, for simplicity, the four values - whether the feet are touching the ground or not, are encoded into binary. That is, if all four feet are on the ground, the value for that time step will be 15 - since $1111_2 = 15$. The recorded data for this descriptor is a 1-D vector with the length as the number of evaluation time iteration.

Last position

This is the last position of the robot at the end of the simulation run. The descriptor such as this one will give certain pressure to explore more once the moving genomes are populated in its population pool. It is the same behavior descriptor as the one which was introduced when Lehman and Stanley suggested Novelty Search[22] on their maze navigator robot experiment. The recorded value is simply (x, y) coordinate of the robot at the end of the evaluation.

Behavior Descriptor	Data type
Max-Min Ampl.	(1×12) vector
Duty Factor	$(1 \times \text{num\_iter})$ vector
Last Pos.	(1×2) vector

Table 3.1: Summary of data types for each behavior descriptors. The variable num_iter corresponds to the number of time steps in the simulation.

Novelty Metrics

With the behavior descriptors defined above, Novelty metrics can be measured. However, since each of the behavior descriptors is expressed by a vector (See table 3.1), it need to be converted into a scalar value to distinguish between the descriptors and their effects - making them into a compact set of features. In this thesis, the norm of each vector is calculated to present a scalar value for each behavior descriptor, except the last position. The reasoning behind this is that the last position descriptor will end up only showing the absolute distance - which would limit the diversity of it, but for other two descriptors with much longer vector length, it is still possible to differentiate the behaviors to a certain degree, despite the losing some of the finer details in the information.

The consequence of this simplification for each behavior descriptor can be summarized as the following.

- For Max-min amplitude descriptor, the smaller value would correspond to the behaviors with more limited movements, and on the other side of the scalar, the robot with the most active joints movements.
- For Duty factor descriptor, the highest value possible is one which corresponds to behaviors that don't lift its foot at all - either not moving at all or gliding around by exploiting the physics engine. The smallest possible value is 0, which corresponds to a flying robot, which is not possible.
- For last position descriptor, it is not affected as it is not converted to a scalar value.

These simplified behavior descriptors are now four features that kNN classifier can be fitted on. It is also possible to use these features in any combination to see the effect of each behavior descriptor.

Once k-nearest neighbor classifier is fitted with the behavior descriptors of each genome, novelty metrics is simply measured by averaging the distance of kNN groups(see equation 3.11). For kNN classifier, *scikit-learn*[4] machine learning package is utilized. The chosen kNN classifier uses the 'Ball Tree' algorithm, which is a tree-based algorithm, faster compared to a brute force method and better at handling of higher dimension than the 'K-Dimensional Tree' algorithm. The rest of the parameters for the kNN classifier is default besides the size of k.

$$\rho(x) = \frac{1}{k}\sum_{i=0}^{k} d(x, \mu_i) - 10 \cdot \text{Penalty} \tag{3.11}$$

In addition to the novelty metrics measured with the behavior descriptors, it also evaluates if the robot is moving or not by adding negative rewards. In particular it is done by adding $-10 \cdot \text{Penalty}$, which is introduced in section 3.4.2. This convention promotes behaviors that are actually moving around.

Novelty Archive

Novelty search without an archive can end up circulating similar solutions from newly generated individuals in each generation. Novelty archive is implemented to benefit the behavioral diversity from the Novelty search fully.

Following Mouret[27], a genome is added to the novelty archive if the novelty metric to that genome is higher than a threshold, ρ_{min}. If there are more than four genomes archived in the same generation, ρ_{min} is multiplied by 1.05, increasing the barrier to limit the similar genomes being added. Contrarily, ρ_{min} is decreased by multiplying with 0.95 if there are no new genomes are added for the duration of four generations.

[4]https://scikit-learn.org/stable/

The size of the novelty archive is also one thing to consider. As the size of the archive grows; the computation cost of the evaluation will also increase. The kNN classifier will take most of the computation time as the size grows, specifically for the chosen 'Ball Tree' method, computational complexity is $O[D\log(N)]$ for D dimension(number of features)[5]. To prevent the computation growing logarithmically, the size of the archive is limited as the same size of the population. It is done by sorting all the genomes in the archive by its novelty metrics or fitness. Then leave out the ones that fall out of the limit. If the fitness of the genome is used, then this will create some pressure for evolving more fit genomes - based on the fitness function.

3.4.4 Multi-Objective Evolutionary Algorithm

The implementation of the Multi-Object Evolutionary Algorithm is done at its simplest form, utilizing only the Pareto-front with the elitist approach. It is realized by implementing Pareto-front selection at the end of the evaluation of each generation before the reproduction. The fitness and novelty metrics of each genome are the two objectives defined. Compared to NSLC by Lehman and Stanley[23], this formation can be interpreted as Novelty Search with global competition inspired by Mouret[27]. While this approach imposes less pressure than NSLC in the search process, the choice is based on [23], which showed that it performs better in terms of the absolute fitness.

Once all genomes are evaluated with their fitness and novelty metrics, Pareto-front is formed containing both the current population and the novelty archive. Then the Pareto-front groups are included by their dominance rank until the number of genomes is larger than half of the population. The result is that the number of parent genomes is always slightly larger than the one fold of the population. Once the parent genomes are chosen, they go through random cross-over and mutation defined by the default NEAT algorithm.

[5]https://scikit-learn.org/stable/modules/neighbors.html#classification

Chapter 4

Experiments and Results

4.1 Overview

Experiments are designed to answer the corresponding research questions given in chapter 1.

First, the effect of behavior descriptors is evaluated. In total, three behavior descriptors are implemented - max-min amplitude of joint movements, duty factor, and last position of the robot. The seven possible combinations are evolved by MOEA, and their performances are compared. This comparison will give some direction on which combination of the behavior descriptor is best suited for other experiments.

Second, it is tested how the different evaluation schemes affect the result in the performance of the evolved controllers. While the implemented system employs MOEA with fitness and novelty search as its default evaluation scheme, the original method suggested by Risi uses only Novelty Search. The comparison between novelty search, only fitness, and MOEA is examined.

At last, it should be confirmed that the method introduced in the previous chapter can indeed generate walking gaits on DyRET platform. It should also be confirmed that Risi's method[39] can be reproduced, such that the method can generate the flexible walking gait for various lengths of legs. The static controller and the flexible controller are evolved by MOEA, and the performances are compared.

General Experiments Setup

Ahead of running the experiments, there are few more technical details that need to be defined and implemented. It includes how the leg configuration should be defined, what is testing criteria - i.e., how to define which of the evolved controller performs better than the other. Also, a description of the computation will be given.

Parameters

There are a couple of different parts regarding the parameters used in these experiments. First, it is NEAT parameters. NEAT is used to evolve CPPN, and it is the primary evolutionary algorithm in the experiments. Also, the evolved controllers need to interact with the simulation environment that requires matching parameter setup as the CTRNN needs the time step to calculate its output on the continuous time scale. Besides, novelty search and MOEA have their own parameters that need to be defined. Many of the parameters regarding EA are inspired by Risi[39] to reproduce the results from his work.

NEAT/CPPN Parameters The table A.1 in the appendix provides the overview all the paramater setup needed by the *neat-python* package to run the NEAT algorithm. The important parameters worth mentioning are the following;

- Population size 300
- The default activation function is sigmoid with 0.5 chance of mutation probability to gaussian, sinus and absolute value.
- Bias are randomized from $N(0,1)$ in a range of $[-1,1]$ with a muation rate 0.6.
- Weight are randomized from $N(0,1)$ in a range of $[-5,5]$ with a mutation rate 0.6
- Node add/delete probability is 0.1.
- Connection add/delete probability is 0.2.
- Neural networks are feedforward network initialized as fully-connected neural network.
- Size of elite population is 150.

CTRNN Parameters CTRNN is generated by *neat-python* package and comes with a set of parameters that need to be set up in accordance with the simulator's time step which is set at 1/240 s.

- time_step parameter for *neat-python* CTRNN is set same as the simulator at 1/240
- advance_time parameter for *neat-python* CTRNN - which decides how much time to advance before returning output, is empirically set for 5/240.
- Time constant queried by CPPN has a range of [0.1,2].
- Bias queried by CPPN has a range of [-3,3].
- Weights queried by CPPN have a range of [-5,5].

- Activation function is sigmoid.

Novelty Search Parameters For Novelty Search, there are only a couple of parameters that can be adjusted in the simplest form. These parameters are

- k-value from kNN, which determines the number of the nearest neighbors.
- ρ_{min}, a threshold to be added to the novelty archive.
- Size of the novelty archive. While this is not strictly following the original algorithm by limiting the size of the novelty archive, it is set for 300 to limit the computation time increasing uncontrollably.

These parameters affect the behavioral diversity, and these are the only parameters that are altered through the experiments conducted in the following sections.

Leg Configuration

The method suggested by the previous work[39] uses a single value for the leg configuration input value, L for CPPN when building the substrate. The experiment by Risi is tested on a simulation environment with a quadruped, which has a uniform length for the femur(upper leg) and the tibia(lower leg). However, DyRET has different leg lengths for the femur and tibia and also comes in a different range of lengths that can be extended. In addition, DyRET can change the length of each leg segment as it needs to.

While it would be interesting to evolve a flexible controller that can adapt to any given leg length on each leg, this work simplifies the leg configuration of DyRET. It is done by introducing a convention, where L, the leg configuration input for CPPN is zero when the legs are compressed to their minimum and two when extended to their maximum. Then everything else in-between is interpolated linearly between $[0, 2]$.

The experiments are done with three leg configurations 1. Fully compressed, 2. At half length, 3. Fully extended. During the evaluation process, the fitness is averaged among the three performance. As for behavior descriptors that are collected during the three evaluation, the max-min amplitude and the last position are averaged, and duty factor is simply appended after each other. For the interpolated performance evaluation, 13 leg configurations are used, interpolated in 13 iterations between $[0, 2]$.

Performance Evaluation

Despite the fact that the fitness function is designed to evaluate the performance of the controllers, there are few factors that can result in delusions, when determining the actual performance of the controller.

First of all, the implemented fitness function(see Eq.3.4) increases iteratively as the simulation runs, yet the traveled distance bonus term has a fixed weight. The consequence of this is that the balance between these two factors depend on how long the simulation will last. In other words, if the first term of the fitness function, Eq.3.4 provides good enough stability and accumulates reward that is large enough, then the second terms, Eq.3.5, which consists of a bonus for moving far, may not be significant compare to the accumulated rewards of the first terms. Also, another possibility is the EA exploiting the physics engine. As was pointed out in literature in EA(e.g., [11]), it is an inherent problem to EAs and in ER particular. In some of the experiments conducted in this thesis, there have been few cases where the evolved controller was capable of gliding around the surface without lifting any of its feet. On an additional note, the controllers evolved without using fitness function will need an alternative to measure its performance.

Therefore, the traveled distance is measured for the span of the simulation. It is measured by following the center of the robot in x, and y plane, (x, y), and calculating the deviation from the previous coordinates. To filter out jitters in the movement, it is only sampled every 0.5 seconds, and it is measured by meter [m]. This way, it measures the trajectory of traveled distance since the beginning of the evaluation. This will not, however, prevent or be able to filter out the controllers that are evolved to exploit the physics engine.

When comparing performance from two or more experiments Mann-Whitney U-Test(MWU) is used. MWU is a non-parametric test that is used for null hypothesis testing. It is useful when data samples are small, and the distribution of the data is assumed to be the same. In addition, when there are multiple statistical tests are performed on the same data, p-values are corrected with Holm-Benferroni method[14].

While MWU works well with small sample data, it still needs a sufficient size of a data sample. To balance out with the cost of computation of creating data - fully-evolved controllers, in this case, eight of the same experiment are evaluated in parallel. In the case of comparing two experiment results with eight data samples, MWU can give a p-value down to 0.0002, which is sufficient to confirm the difference between the two data set. To compare the performance using MWU, the best performance genomes are used from each evolution of eight parallel runs. MWU is provided by Python package *Pingouin*.[1]

Parallel Computation

Evolving a controller through hundreds of generations over a population consisting of hundreds of controllers poses a challenge in terms of computational cost. However, EAs are particularly suited for parallel computation as each genome can be evaluated separately. In the experiments, two parts

[1]https://pingouin-stats.org/index.html

can be explicitly parallelized. The first is the evaluation of each genome - fitness evaluation and collecting behavior descriptors. Each controller is evaluated in a separate instance of the simulation environment. The second phase is when measuring novelty metrics. Once kNN is fitted with the new behavior descriptors, the novelty metrics calculation - averaging the distance of k-nearest neighbors for each genome, can be parallelized with a handle to the kNN classifier.

While there exists a vast possibility of implementing parallelism such as MPI, OpenMP, the experiments are implemented with the multiprocessing package provided by python[2]. Multiprocessing is process-based parallelism - which means that each process has its own memory space, and the processes are confined to a single instance of an operative system. Even though the computation is confined only to a single system, the benefit is the simplicity of implementation. Since the system has access to the population - each genome containing behavior descriptors and kNN instance, the communication between processes does not need to be considered like it would with MPI implementation. The system used for the experiments provides 40 cores at 2.5 MHz. Running a standard evolution - MOEA with population size 300 for 600 generations and 200 simulation time steps each, takes about 9 hours wall-clock time utilizing all 40 cores without hyperthreading technology.

4.2 Behavior Descriptors

This experiment is designed to test the behavior descriptors introduced in section 3.4.3. The focus points in the experiment are to see if there are any descriptor or a particular combination of them that results in better performance and faster convergence in the evolution.

4.2.1 Setup

In this experiment, the three behavior descriptors - Max-min amplitude, duty factor and last position, are compared in every combination by evolving flexible controller. They are seven possible combinations - All, Ampl&Duty, Ampl&Pos, Duty&Pos, and each of descriptors by themselves. The chosen EA for this experiment is MOEA with default parameters described in the previous section 4.1 except for kNN parameter which is set for $k = 5$ and $\rho_{min} = 0.3$.

4.2.2 Results

The resulting evaluation of the experiment is summarized on the table 4.1. Both the mean best fitness of at the end of the run and the performance evaluation measuring the traveled distance are presented. At a glance, the

[2]https://docs.python.org/3/library/multiprocessing.html

	Mean Best Fitness	**Distance Travelled**
All	$\mu = 9.5e5$ $\sigma = 1.3e5$	$\mu = 6.61$ $\sigma = 2.14$
Ampl&Duty	$\mu = 7.3e5$ $\sigma = 2.1e5$	$\mu = 5.48$ $\sigma = 1.65$
Ampl&Pos	$\mu = 7.9e5$ $\sigma = 0.5e5$	$\mu = 6.47$ $\sigma = 1.23$
Duty&Pos	$\mu = 8.3e5$ $\sigma = 1.0e5$	$\mu = 8.18$ $\sigma = 3.78$
Ampl	$\mu = 7.1e5$ $\sigma = 1.4e5$	$\mu = 6.53$ $\sigma = 0.85$
Duty	$\mu = 7.4e5$ $\sigma = 0.3e5$	$\mu = 5.62$ $\sigma = 1.42$
Pos	$\mu = 9.0e5$ $\sigma = 2.1e5$	$\mu = 5.52$ $\sigma = 2.04$

Table 4.1: Behavior descriptor experiment result, showing mean and standard deviation that are not depicted in the boxplot 4.1

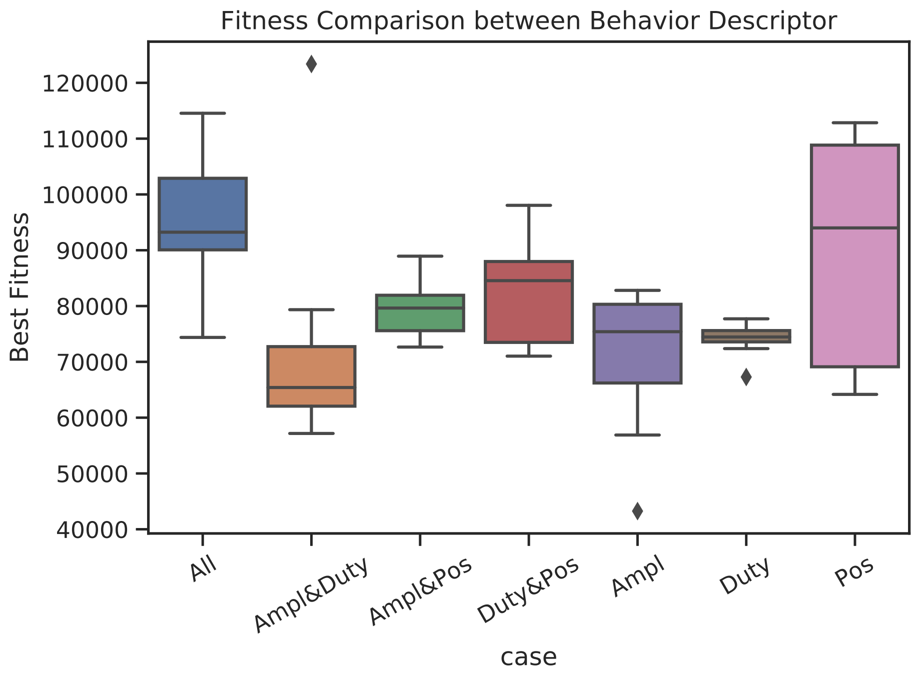


Figure 4.1: Fitness comparison of behavior descriptor comparision

case All and Pos performed best in terms of fitness, and Duty&Pos could walk furthest by 8.18 meters on average followed by All and Ampl only runs.

The differences in the best fitness for each case are shown in figure 4.1. As expected, utilizing all the descriptors resulted highest in terms of fitness, while using only the last position performed as good as using all despite the wide deviation in the resulting fitness.

Pairwise MWU test in the table A.4 shows a contradictory result for corrected p-values than the boxplot 4.1. All the statistically significant comparisons ($p < 0.05$) are not significant longer after correction. This is due to the relatively small size of the data and the large number of comparisons. However, in this section, the uncorrected p-values will be discussed as the goal of this experiment is to spot the tendency among the behavior descriptors with small evaluations.

The uncorrected p-values in table A.4 shows the same result as reading the boxplot - the case All is significantly better than all other cases except for the case Pos, the last position descriptor. Also, When Pos is paired with Ampl or Duty, they have higher fitness than Ampl, and Duty is paired. Ampl paired with Pos has higher fitness than Duty alone. At the same time, the last position descriptor is not significantly different from any other combination.

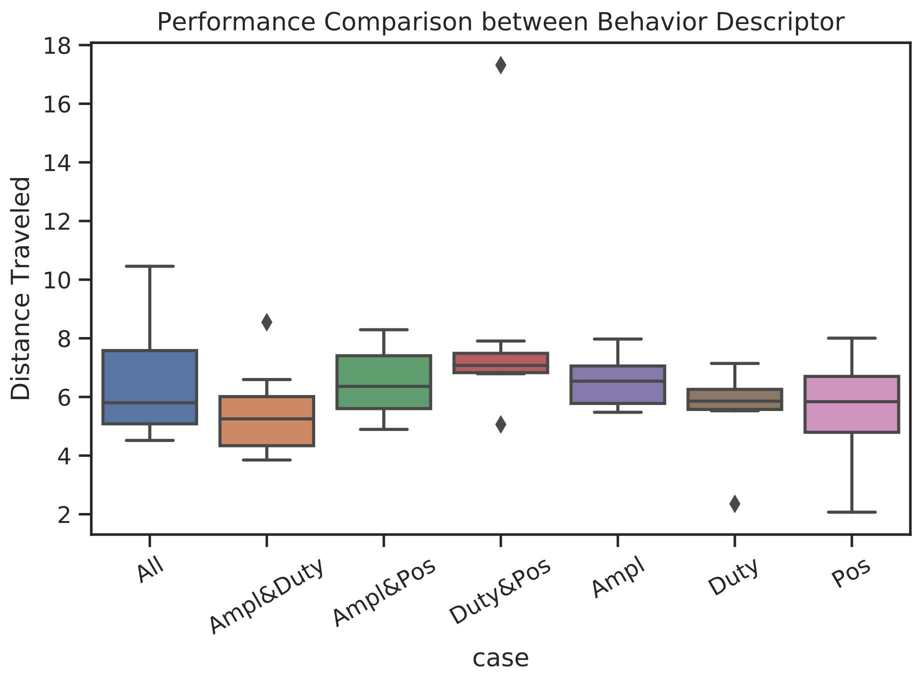


Figure 4.2: Distance traveled comparison of behavior descriptor comparision

While the fitness result above showed a statistically significant difference in terms of uncorrected p-values between the combinations, the performance evaluation does not show a clear distinction in the boxplot in figure 4.2. Note that the outlier in Duty&Pos, which walked over 16 meters, is an example of a controller exploiting the simulator by gliding around the surface. This particular genome caused the Duty&Pos case to be evaluated highest in mean in the table 4.1.

The pairwise MWU test given by the table A.5 reveals that Duty paired with Pos performs significantly better ($p < 0.05$) compared to Ampl&Duty and Duty alone. Although most of the differences are not significantly different, it generally follows the fitness comparison from the figure 4.1.

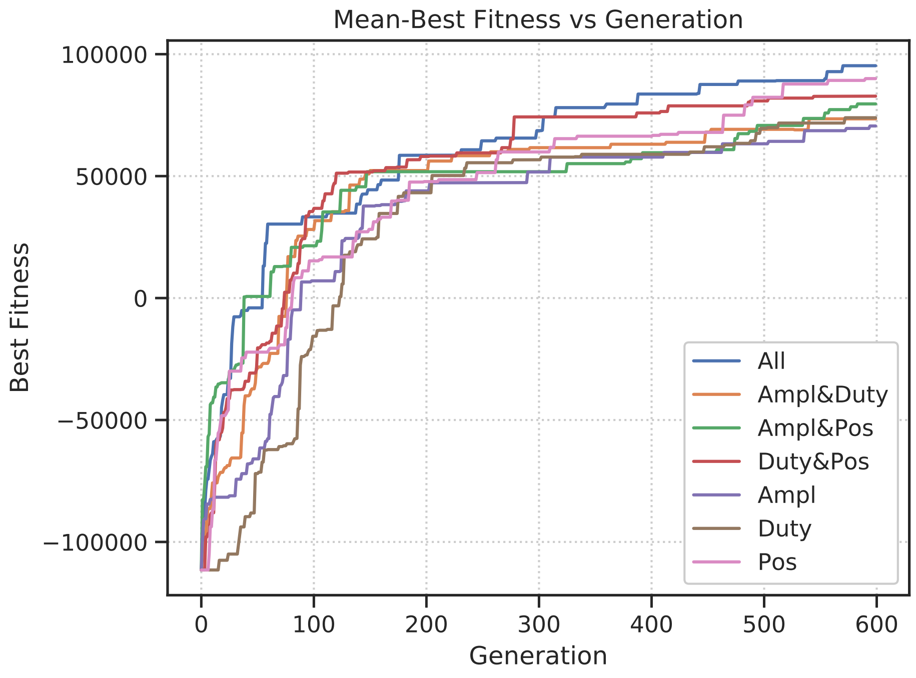


Figure 4.3: Mean best fitness for each behavior descriptor during evolution

One of the interesting things to check while comparing the behavior descriptors is to see how they compare in terms of convergence throughout the evolution. Figure 4.3 shows the mean best fitness of each case plotted against generation progression. There are some subtle differences can be spotted - i.e., Ampl and Duty lag behind at the early stage of the evolution. However overall progress looks very similar, starting to converge around the 200th generation.

4.2.3 Analysis

While the result of the experiment showed not much of significant differences in terms of performance, it revealed some properties of the behavior descriptors.

One of the interesting discoveries in this experiment would be the property of the last position descriptor. Although it comes with high variations, the last position descriptor itself was capable of creating a controller with high fitness. The reason behind this can perhaps be explained by the fact that the position information itself is embedded into the fitness function(see the second term of equation 3.5 in chapter 3). Besides, the novelty metrics on the last position will give a higher score as it is further away from the

	MOEA	**Novelty Only**	**Fitness Only**
Distance Travelled	$\mu = 8.66$ $\sigma = 1.56$	$\mu = 9.70$ $\sigma = 1.09$	$\mu = 11.37$ $\sigma = 2.38$

Table 4.2: EA comparison performance results, showing the mean and the standard deviation

majority of trivial solutions around the start point - the controllers end up not moving at all. As the MOEA tries to maximize both fitness and the novelty metrics, the result will favor higher fitness although not given explicitly - therefore higher deviance than other cases.

The conclusion drawn from this experiment is that while it is not statistically significant in every case, including all three behavior descriptor would be best to create a controller that performs best among all the possible combination of the descriptors.

4.3 Fitness vs Novelty vs MOEA

The purpose of this experiment is to determine if any of the EA schemes fit the best to evolve a gait controller for a morphology changing robot such as DyRET. The three different EA is implemented and tested by evolving eight runs.

4.3.1 Setup

In principle, these experiments also test evolving flexible controllers. While it will not be tested rigorously with the interpolation performance, the controllers will be evaluated for the three default leg configurations.

Fitness only experiment is done with the default fitness function from section 3.4.2. Other parameters are the same as it was introduced in the general setup section 4.1. The novelty search experiment uses only novelty metrics and the novelty archive as its evaluation mechanism through the generation. The parameters specific to novelty search are $k = 15$ for kNN classifier, and $\rho_{min} = 0.8$, increased from the previous behavior descriptor experiment. These runs are then compared with MOEA experiment, where the controllers are evolved using both fitness function and novelty metrics as their multi-objective to maximize with the same parameter setup.

4.3.2 Results

Table 4.2 summarizes the result on each run. The best genomes from novelty only run could walk for 9.70 m on average with standard deviation of 1.09. This is slightly more than the best mean performance of MOEA at 8.66 m. The fitness only run was able to generate the best walking genomes. On average, the best genomes could walk for 11.37 m, with the best one as far as 14 meters. Despite the differences in the mean of the best performances in each case, the MWU has shown to be not significant. See table A.2 for the result of the pairwise MWU test between the data. Figure 4.4 shows the distribution of the resulting data with a boxplot. Besides the performance of each EA schemes, it would be interesting to look at how the search process of each EA unfolded since the differences between each EA are based on how they optimize in their search.

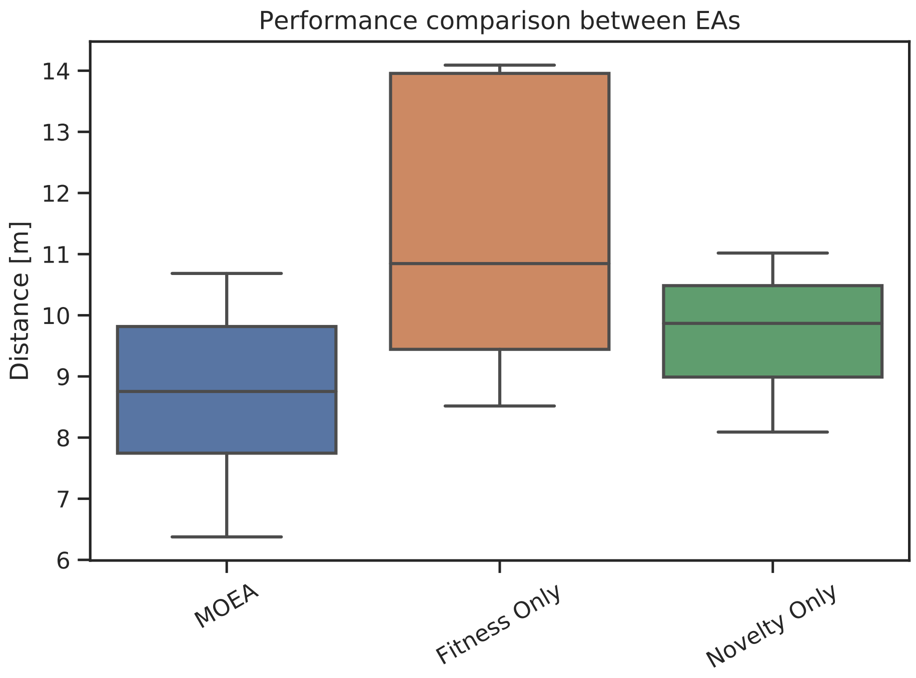
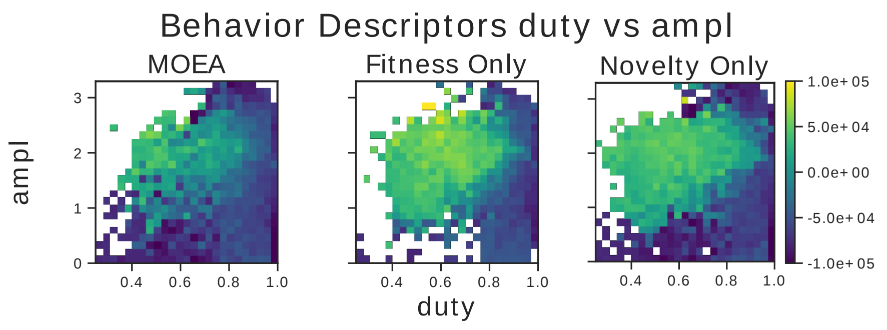


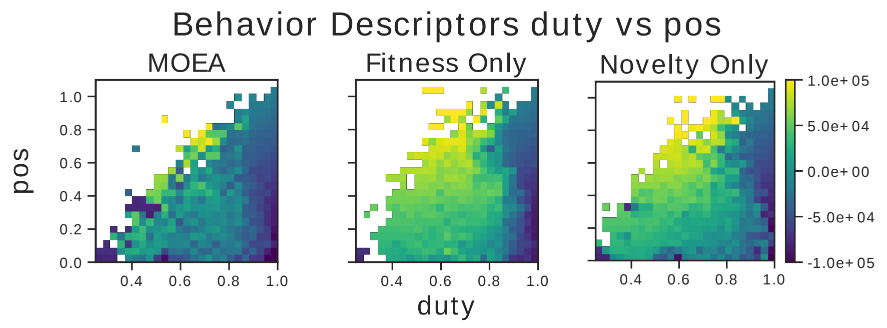
Figure 4.4: Comparing distance travelled by each EAs

Figure 4.5 summarizes the behavioral diversity of each EA found during the evolution of all eight runs. The behavior descriptors are expressed as a scalar value on each axis. This follows the description given in chapter 3 section 3.4.3, besides the last position descriptor calculated with the absolute value instead of 2D coordinates. The color values in the pixels indicate the average fitness of the genomes that fit in the bins. As the behavior descriptors are initially expressed in vectors, this presentation does have its pitfall where the same scalar value could potentially be different behavior. However, this plot can still show the degree of searching, and it is possible to observe where reasonable solutions are located in the behavioral search space.

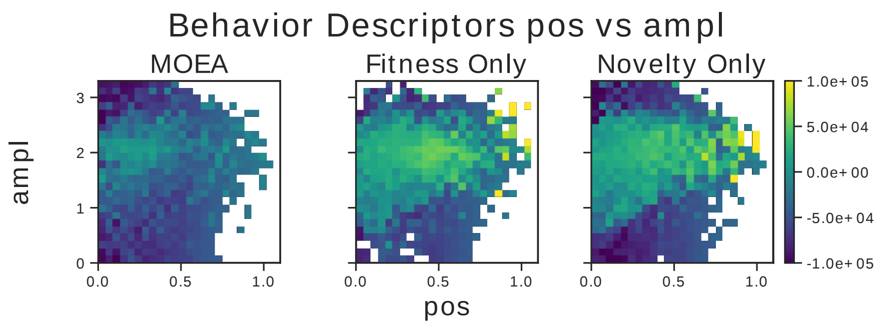
While the meaningful differences were not found in terms of the performance of the best genomes, it is possible to see the slight differences in the found solutions in the behavioral search space. The figure 4.5a especially shows the differences in illuminated search space. The novelty search only case managed to try out the most different solutions and fitness only case tried out least compared to the other two. This is an expected result as the



(a) Duty factor vs Max-min amplitude



(b) Duty factor vs Last position



(c) Last position vs Max-min amplitude

Figure 4.5: All solution for each EA scheme are plotted in 2D histogram with the average fitness in each bin as its scalar value in color. Each behavior descriptors are converted into scalar value by taking norm of the behavior descriptor vector.

novelty search only encourages novel behaviors where fitness-based EA will only search around what has been proven to work. However, if the last position behavior descriptor is involved, they show a somewhat similar pattern besides MOEA when compared against duty factor at the figure 4.5b.

The supplementary video material[3] attached to this thesis shows the gaits evolved by each EA. See 00:05~00:25 for MOEA, 00:45~01:05 for Novelty search only, and 1:05~1:25 for fitness only evaluation. Novelty search only and fitness only shows gaits on its shortest leg length, the one that has been used in learning, while MOEA shows an unseen leg configuration in learning. Although they show variety of gaits walking in different directions, the gaits evolved by the fitness only appear to be more monotonous compared to the gaits evolved by MOEA and Novelty Search only. However the best controllers - in terms of walking distance for given 10 seconds were both from the fitness only evaluation, walking up to 14 meters each.

4.3.3 Analysis

The first thing to be mentioned is that increasing the k value to 15 from 5 and the ρ_{min} to 0.8 from 0.3 compared to the previous experiment seems to result in better performance although statistically not significant - comparing 'All' behavior descriptor run against MOEA run($p \approx 0.052$).

One of the most intriguing aspects of the result in this experiment is that the fitness only evolution actually managed to produce the best controllers although it is not statistically significant compared to other EAs. Fitness-based EA has been criticized in the context of evolutionary robotics when evolving gait controllers due to the tendency of being stuck around local optima of the search space[22]. This is especially true when a high degree of freedom is concerned, such as this experiment.

In the previous experiment at section 4.2, it was pointed out that the last position descriptor is, in fact, redundant with the bonus term in the fitness function, and that the last position descriptor was alone capable of creating high fitness, although not necessarily the highest performing one. With this in mind, a suspicion arose that the second term of the fitness function(eq 3.5) which gives reward for the last position as the absolute distance from the origin is responsible for the fitness only evolution being competent against behavioral diversity EA.

To clear out the suspicion, an additional experiment is conducted, this time without the last position reward in the fitness function. Also, novelty search is tested again since the implemented novelty search takes use of the fitness in the process of limiting the population size of the novelty archive(See section 3.4.3 in the chapter 3).

Figure 4.6 shows the result of the additional experiment without the last position reward in the fitness function in the fitness only run, and the novelty search only run. The boxplot clearly indicates that the fitness only run without the position reward performs significantly worse than other experiments. As for the novelty search only run without the position reward, it performed similarly with other EAs.

[3]https://youtu.be/a9M2IFrUjG8

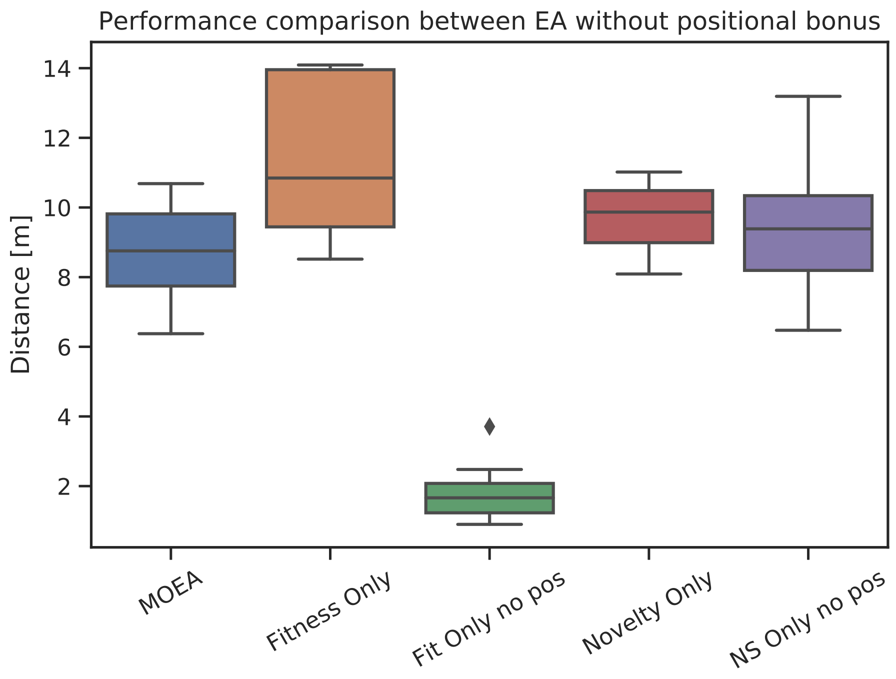


Figure 4.6: Performance comparison between EAs including runs without the last position reward in the fitness.

While it may seem evident that adding the position reward is enough to make fitness-based EA as competent as the other EAs, in this case, there are a couple of things to be considered. The first thing is the constraints for the controller scheme, which are designed to favor specific gaits. The default stable gait of DyRET is enforced as the null points for all of the joints, with the specific direction associated (see section 3.3 for the details). These constraints limit the search landscape, thereby making it easy for EA to discover suitable solutions. In addition to the limited search landscape, it can also be argued that the designed fitness function puts not enough pressure without the positional reward, as other terms are more focused on the stability of the robot and the moving velocity.

4.4 Static Controller and Flexible Controller

In this experiment, static and flexible controllers are evolved by MOEA scheme and compared. It reproduces the previous work by Risi[39] and will confirm that the same method can be applied to the DyRET platform.

4.4.1 Setup

Besides the general setup described in the previous section 4.1, the difference between the static controller setup and the flexible controller setup is the input for the CPPN. Instead of implementing new CPPN scheme for static controller, the input for the leg configuration, L is zeroed for the static controller experiment. By setting L as zero, the CPPN is a FFNN with eight inputs for the two 4D coordinates and three outputs.

Both the static controllers and the flexible controllers are evaluated in the three different leg lengths - fully compressed, in half, and fully extended, as noted in the section 4.1. The controllers are evolved by MOEA with all three behavior descriptors. The k value for kNN classifier is set for 15, and ρ_{min}, the threshold for adding genome to the novelty archive is set for 0.8.

4.4.2 Results

	Static	**Flexible**
Training Performance	$\mu = 8.47$	$\mu = 8.66$
	$\sigma = 1.72$	$\sigma = 1.56$
Interpolation Performance	$\mu = 8.86$	$\mu = 8.02$
	$\sigma = 1.90$	$\sigma = 2.03$
Average number of solutions out of 13 runs	12.875	12.875

Table 4.3: Static & flexible controller performance evaluation results. Training performance and interpolation performance denote the furthest distance travelled in each of eight evaluations. The average solutions represent average number of solutions that didn't fall during the evaluation time of 10 seconds.

Both experiments were capable of generating walking gaits. Table 4.3 summarizes the results. For the trained three leg configurations, The best static controllers in each evolution could walk for 8.47 meters on average, with a standard deviation of 1.72, while the best flexible controllers could walk for 8.66 meters on average with a standard deviation of 1.56. All of the best flexible controllers were able to walk in the three trained configuration while one of the best static controllers failed in one leg configuration.

While the best flexible controllers have a slightly higher average, the MWU test has shown that the difference between the two groups is not significant($p \approx 0.79$). See figure 4.7 for boxplot indicating distribution of the data.

Perhaps the more interesting results from these evolved controllers would be how they perform on the interpolated leg configurations, which were not seen in their training. Surprisingly, both controllers could walk as well as they did on the trained leg configurations. The mean of best static controllers could walk 8.86 meters slightly more than evaluated on the trained three leg configurations with a standard deviation of 1.90. As for the best flexible controllers, the mean decreased a little at 8.02 meters with a standard deviation of 2.03, while the best controller could walk a little further at around 11 meters compared to 10.5 meters when evaluated on the trained leg configurations. As figure 4.8 indicates, the difference between the two groups are not significant at $p \approx 0.37$.

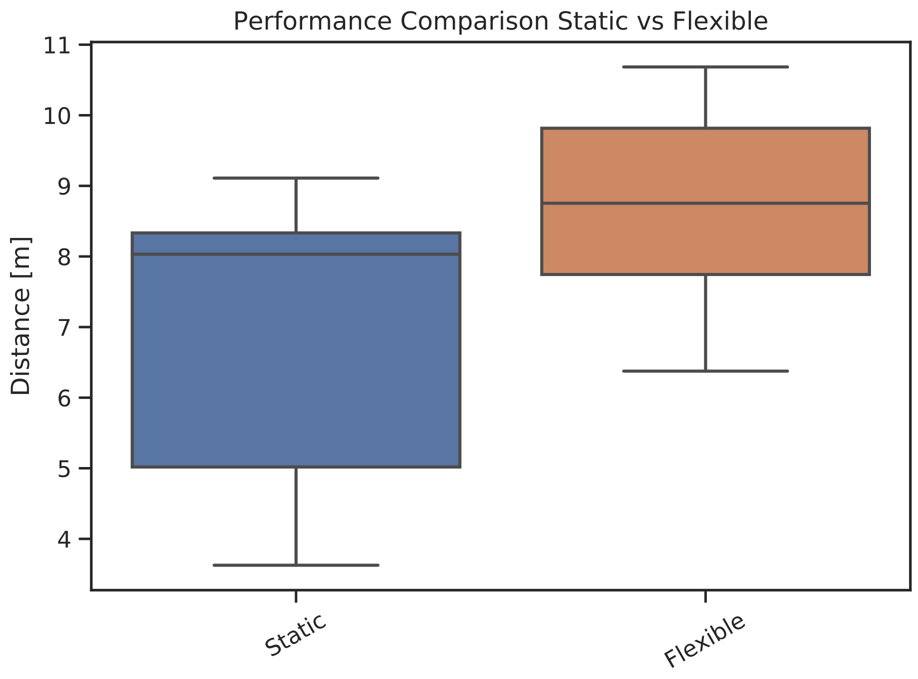


Figure 4.7: Performance evaluation on the trained 3 leg configurations.

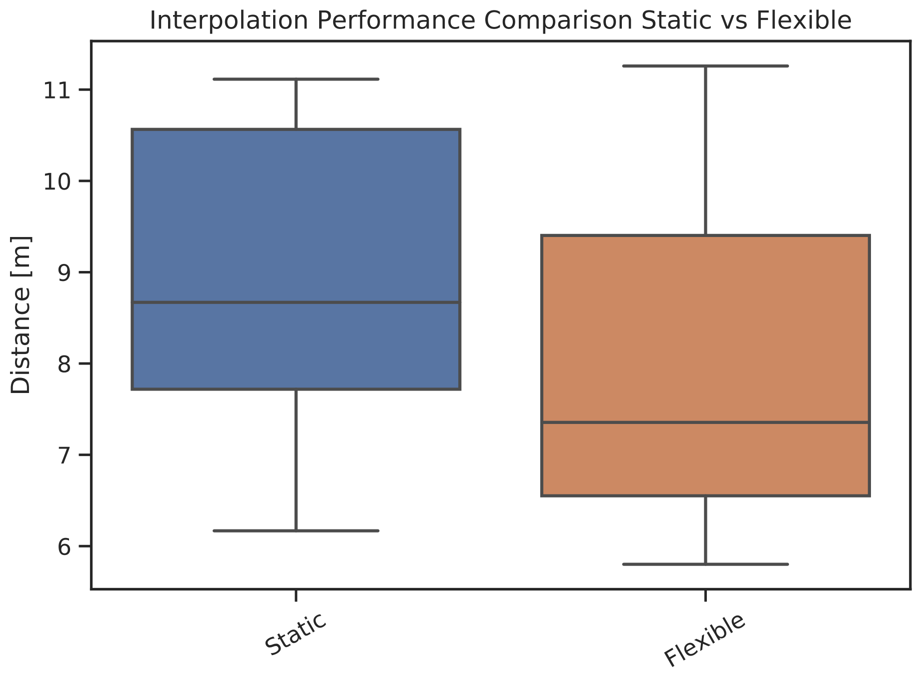


Figure 4.8: Performance evaluation for 13 interpolated leg configuration.

Both of the evolved static controllers and the flexible controllers managed to walk for a given 10 seconds without falling, 7 best controllers out of 8

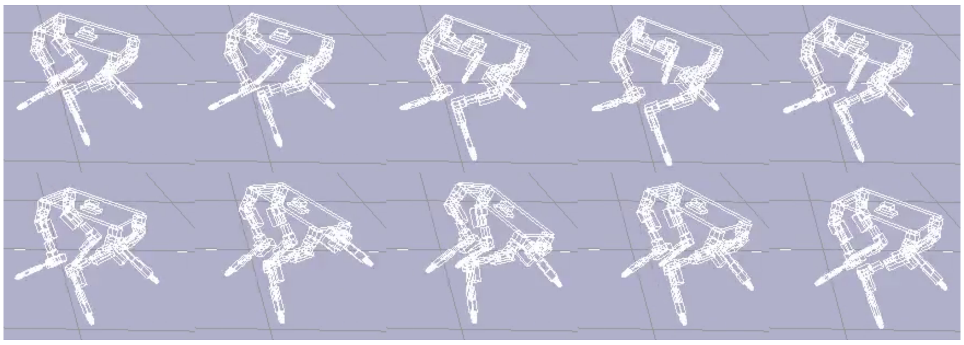


Figure 4.9: Walking gait sequence. The first frame from the top left to the last frame on the bottom right.

runs. Even the ones that did not manage to walk all leg configurations solved 12 out of 13.

The supplementary video material[4] compares the static and the flexible controllers in the leg configuration that has not been seen during the learning. One of the static controller fails for this specific leg configuration. The biggest difference between the gaits of the static controller and the flexible controller is that the static controller are more diverse in a way that they move forward. While most of the flexible controllers are trot-like as seen on figure 4.9, the gaits of static controller varies and more dynamic - one can spot a gait walking side ways and one of the gait shows characteristic of a galloping gait. Also, the gaits of the static controllers tend to be less symmetrical in terms of their movement.

4.4.3 Analysis

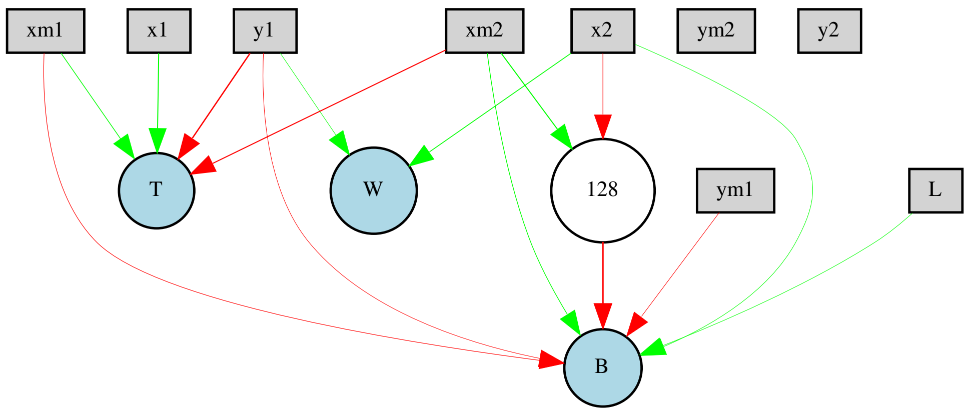
This experiment has shown a somewhat conflicting result than the previous work done by Risi[39], which has demonstrated that evolving a flexible controller using the same methodology as this experiment performed better than the static controllers in the interpolated performance.

One of the first things that could be looked into is how the evolved controllers makes use of the leg length configuration input L and how the output of the controllers are affected by the change of the L. Examining each of evolved flexible controller revealed that three genomes used its L value to modulate the output of each joint, out of eight best from each run in the flexible experiment.

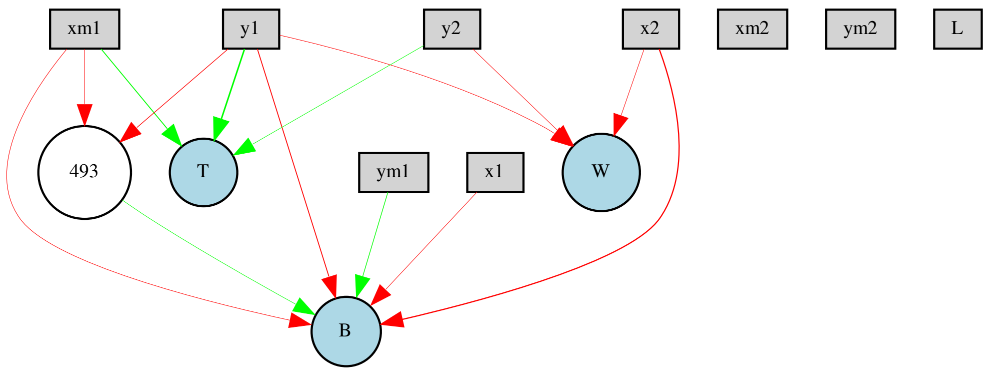
Figure 4.10 shows the evolved CPPNs from the flexible controller experiment. The first subfigure 4.10a is one of the three controllers that actually evolved to modulate the L for different leg configurations. The input L is directly connected to the bias B, which results in the modulated output of each joint based on the leg length. In this case, the weight associated with this connection, L to B, is positive weight indicating higher biases as

[4]https://youtu.be/a9M2IFrUjG8

L increases. The higher biases for each neuron will result in higher output value for each joint. This makes sense as the longer legs would need larger swing for better stability. On the contrary, figure 4.10b shows a CPPN which does not utilize *L* at all. The consequence is that the CTRNN queried by the CPPN is precisely the same for different leg configuration - regarding weights, time constants, and biases. Resulting movements created by these controllers are the same. Both of the CPPNs can generate controllers that are capable of walking for the given evaluation time and 13 different interpolated leg lengths.



(a) Evolved CPPN utilizing *L* input.



(b) Evolved CPPN without utilzing *L* input.

Figure 4.10: Example of evolved CPPNs. The gray boxes are input nodes and blue circles are output nodes. The color of arrows indicates red for negative weight and green for positive weight.

One thing to note here is that it is perfectly normal for NEAT to mutate the connection gene and disables it during evolution. At the initialization phase, all the genomes are fully-connected from the input nodes to the output nodes with randomized weights. Considering this, perhaps looking into the trend of L's usage would be in order.

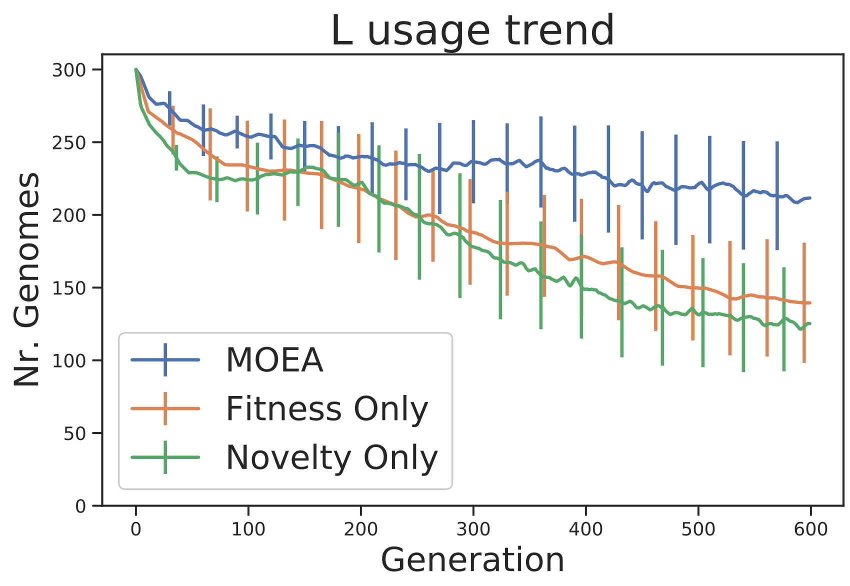


Figure 4.11: Number of genomes with valid connection from L input node. Vertical lines indicates standard deviation.

The figure 4.11 illustrates the trend of L usage for three different EA scheme plotted along with the generation with the mean number of CPPNs that has a valid connection to the L input node with the standard deviation noted with vertical errorbars along with the mean. Each of the EA is evolved for the flexible controller, and have been compared in detail in the section 4.3.

Although the experiments are designed to evolve a flexible controller - evaluating three different leg length configuration to promote usage of L value, the overall trend observed from figure 4.11 is that there is an increasing number of genomes that are mutated to ignore the L input. While MOEA run had the highest number of genomes that used L input at around 220 in the end, the other two EAs were fairly low at around 150 genomes. It should be also considered here that having a valid L input node connection does not always mean that the output is modulated by the given leg length.

Another thing that should be considered is the direction of the robot is not considered to be a decisive factor during the evaluation. Although the fitness function does encourage to move the robot along y-axis in positive direction, this factor is much smaller than the traveled distance bonus in the second term. The consequence is that as long as the stable and moving gait can be generated, the controller does not need to adjust itself to move in the same direction for different leg length configurations. In other words, static controllers move along different trajectories for different leg lengths due to the difference in the location of center of mass and dynamics. Hence a static controller, which does not alternate its direction, can be favored by selection process. This analysis also provides explanation for the more diverse and dynamic gaits of the static controllers shown in the supplementary video[5].

Furthermore, an additional contributing factor to the unexpected performance of the static controller, is the difference between DyRET and the simulated quadruped from [39] in terms of the physical properties; DyRET does not almost has a payload. Besides the minimal mechanics for the movements, DyRET is practically a skeleton. Yet, the simulated quadruped from Risi has a huge torso with a density of 1.0 kilograms per cubic meter. This factor makes it harder for DyRET to fall in comparison.

This extended analysis of the evolved CPPNs have shown that while there are solutions that use L node - modulating the output based on its morphology, it is not always necessary to find a gait controller that can be used on different leg length configurations. Nevertheless, as it was analyzed in the previous experiment in the section 4.3, the current implementation of evolving flexible HyperNEAT controller relies heavily on the constraints on the controller scheme. The limited search space introduced by the constraints made it possible for the static controller evolution to find good solutions that does not need modulation for different morphology for DyRET. However, it should be noted that the gaits of the static controllers shown in the supplementary video are less conventional for a typical quadruped walking gait - such as non-symmetrical movement patterns.

[5]https://youtu.be/a9M2IFrUjG8

Chapter 5

Conclusion

5.1 Discussion

5.1.1 Summarizing the Results

In the previous chapter, it was provided with the results and analysis of the implementation of evolving a flexible gait controller, which bases on neuroevolution approach suggested by Risi[39]. The implementation was simulated on the DyRET platform, which can adjust its own leg lengths.

The first experiment compared the effect of behavior descriptors in terms of the fitness and the performance, measured by the traveled distance for the given 10 seconds. Combining all behavior descriptors together could give the highest fitness while the last position descriptor was able to produce controllers with high fitnesses with high deviation. However, comparing the performance did not show a statistically significant difference between the combinations of the descriptors while the best performing controller was evolved by combining all. Although it was expected that combining all descriptors worked the best, the last position descriptor's property came as a surprise. The analysis was that the last position descriptor was implicitly embedded in the fitness function, giving a bonus to the deviated last location. This resulted in amplifying the effect of the MOEA counting on both fitness and novelty metrics.

Different EA evaluation schemes were compared in the second experiment. Flexible gait controllers were evolved by fitness only evaluation, Novelty Search only evaluation, and the MOEA, combining both. Although it was expected that the fitness only evaluation would perform worst, the mean best performance of the result has shown otherwise by the fitness only evaluation generating the best performing controllers. Yet, the differences between the mean best performance of each EA evaluation scheme were not statistically significant. Suspecting that the positional bonus term in the fitness function may have generated enough pressure in the search process, additional experiments were conducted without the positional bonus in the fitness function. The results have shown that the positional bonus term indeed was responsible for the unexpected result. The given analysis pointed out that the heavy constraints applied in the controller scheme have limited the search space, hence contributing to a better chance of finding a suitable solution by the fitness only evaluation.

The last experiment compared the static controller and the flexible controller by reproducing the previous work done by Risi[39]. Both controllers were evolved by MOEA. The static controller was evolved without using the L input in CPPN, which was designed to utilize the leg length configuration. While it was expected to see better performance on the flexible controller, the result on the interpolated leg length evaluation has shown that there was no statistical difference between them besides the quality of gaits shown by the supplementary video. Upon analyzing the evolved CPPNs, it was revealed that there is a trend of an increasing number of CPPNs that are mutated to ignore the L, leg configuration input. Although the solutions exist that alter its movement based on the leg length configuration, for given constraints and the simulation environment, they were not favored compared to the static controllers. Hence the static controller evolution performed as well as the flexible controllers.

5.1.2 Back to the Research Questions

The experiments have demonstrated some unexpected results and revealed some interesting property of the implementation, regarding the behavior descriptors, the EA schemes, and usage trend of L, leg configuration input node.

Addressing back to the proposed research questions in chapter 1;

- **Is it possible to evolve a neural network-based flexible gait controller for DyRET?** Although it may be arguable to call it a flexible controller, in a sense that the controller adapts to the various morphology, the implemented EA was able to find solutions that are stable and able to move in various leg lengths, despite the varying performance based on the EA scheme. Besides, it has been observed that solutions exist that are adaptive to its leg lengths, modulating the joint movement based on the leg configuration given by an input to the CPPN, which encodes the CTRNN.
- **Which selection pressure works best? - diversity vs. fitness vs. both** The second experiment compared diversity-based EA, Novelty Search and fitness only evaluation, and MOEA combining both of them. While the result was that there is no statistically significant difference between them, the fitness only evaluation had the highest mean best performance.
- **Which behavior descriptor works best for evolving a gait controller for DyRET?** The first experiment concluded that combining all three behavior descriptor - max-min amplitude, duty factor, and last position, works best when the fitness function is concerned.

In terms of the performance, combining all had the highest mean although not statistically significant.

5.1.3 Limitations

Despite the fact that the implementation successfully evolved gait controller, be it static or flexible; there are a couple of points that should be mentioned.

First of all, the usage of heavy constraints is a limiting factor in terms of searching for novel behaviors. Although these constraints accelerate the search process of finding suitable gaits, there exist many other gaits that could not have been expressed due to the constraints. The introduced constraints force the search process to circulate around the default pose of DyRET and limit free joint movement, especially when the lower joints are concerned. Also, the heavy constraint usage may have led to some of the unexpected results, such as the high performance of the static controller for the interpolated leg configurations, and the fitness only evaluation. Without the constraints, however, the implementation would have required a lot more empirical testing to find a parameter setup that would be able to find a walking gait.

The open-loop system applied to the control scheme doesn't bring out the full potential of CTRNN and HyperNEAT. The implemented controller has an open-loop control system, where the input is a constant sinusoid signal. This can be seen as a simple CPG like system, where a lower level small neural network that controls muscle contraction is driven by a higher-order agent. This leaves an opportunity to evolve an agent that could control the evolved controller later on. However, sensory feedback is an important aspect of CPG, as mentioned in chapter 2, section 2.2.2. In addition, It has been shown that a simple neural network itself can be evolved to be a standalone agent, such as maze pathfinder robot experiment in [22], [27], a cockroach-like AI agent evolved by [1], and as gait controller with a closed feedback loop system in [39]. This suggests that implementing a control scheme with a sensory feedback could have revealed more diverse behaviors and results which perhaps can adapt to different surfaces or landscapes.

The two points mentioned above reveals another limitation, that the combination of the open-loop control scheme and the uniform leg configuration, i.e., same leg length for each leg limits the full potential of DyRET's platform which can change its leg length segment during operation.

The last to be mentioned is that the overall implementation and the experiment design did not consider the application on the physical robot. One of the most interesting perspectives of implementing Risi's work[39] into the existing robot platform DyRET would be actually seeing how it is going to work on a physical robot. Nonetheless, during the implementation and experiment, this aspect was not considered. Bridging the reality gap from the current simulation implementation to deploying it on the real robot would be another challenge.

5.2 Conclusion

In this thesis, a flexible neural gait controller is evolved for the robot platform DyRET. The implementation is based on Risi's work[39], which utilizes HyperNEAT with a special CPPN configuration that takes the input for a leg length configuration. Although it is only realized in a simulated environment, the result has shown that stable gaits can be generated on the leg length configurations that have not been seen during the evolution. Also, the result has revealed that the implemented method is able to discover gait controllers that can perform well on the untrained leg configurations while, in fact, being static controller - i.e., does not modulate for different leg lengths. The result also demonstrated that heavy usages of constraints could lead to unexpected results, e.g., objective optimization can work as good as quality diversity.

Compared to the deep/reinforcement learning, NE has gained less attention in the field of AI recently. Unfortunately, the motivation that evolving a simple neural network like how brain would have evolved could perhaps reveal adaptive behavior did not get tested in this work. However, the result suggests that within the context of ER, for a simple low-level gait controller, NE approach can be viable. The implementation also suggests that a neural network-based gait controller can be realized on the DyRET platform.

5.3 Future Work

Although the results demonstrated some success, it has been only touching the surface of the problem - flexible, self-adapting gait controller for a given morphology. Within the framework of HyperNEAT/CPPN, there are vast possibilities for the current work to be improved and developed further. A few suggestion are provided that could be realized to improve the current work and other research possibilities.

Promoting *L* input usage. It has been observed that the number of genomes that utilizes *L* input decreases as the evolutionary process goes further. The given analysis was that the heavy constraints result in the search process favoring gaits that are simple - i.e., outputting the same movement for different leg configuration, hence stable controllers that are stable in different leg lengths. In order to evolve true flexible controllers that modulate on different morphologies, it should be possible to introduce constraints into the evaluation process - e.g., giving a penalty for those who don't use the *L* input node. One of the things that can be address in this suggested work is that whether the flexible morphology controller is necessary for DyRET platform. By enforcing the usage of *L* input, it would be possible to further analyze whether the observed trend of diminishing *L* usage was a natural result.

Closed-loop control system. In the discussion, it was pointed out that closed-loop controller with sensory feedback could lead to the more dynamic behavior of the robot, along with the fact that the current control system - similar to CPG, can indeed benefit from sensory feedback, as well. One could imagine a simple feedback loop from the foot sensor to the input node of the corresponding node with a dedicated weight which could be evolved by CPPN.

Evolving an adaptive agent that operates the evolved gait controller. The current implementation only provides a low-level gait controller in a form of a CPG. As there is a pool of capable controllers available now, one can further design an agent network that can actually drive the evolved gait controller. This future project could be used to test ideas on the adaptive agents that can interact with the environment and further evolve the controller.

Realize in the physical robot. Since we have seen that some solutions can perform well, the implementation can be extended further so that the evaluation can happen on the physical robot itself, as DyRET is designed for. In this case, incremental evolution should be considered as starting from a random controller would take a too long time to get to actual stable gaits and possible complications on the hardware.

Bibliography

[1] Randall D. Beer. *Intelligence as adaptive behavior : an experiment in computational neuroethology*. Academic Press, 1990, p. 213. ISBN: 9780120847303.

[2] Randall D. Beer. 'The Dynamics of Brain–Body–Environment Systems: A Status Report'. In: *Handbook of Cognitive Science* (Jan. 2008), pp. 99–120. DOI: 10.1016/B978-0-08-046616-3.00006-2. URL: https://www.sciencedirect.com/science/article/pii/B9780080466163000062.

[3] Randall D. Beer, Hillel J. Chiel and John C. Gallagher. 'Evolution and Analysis of Model CPGs for Walking: II. General Principles and Individual Variability'. In: *Journal of Computational Neuroscience* 7.2 (1999), pp. 119–147. ISSN: 09295313. DOI: 10.1023/A:1008920021246. URL: http://link.springer.com/10.1023/A:1008920021246.

[4] Randall D. Beer and John C. Gallagher. 'Evolving Dynamical Neural Networks for Adaptive Behavior'. In: *Adaptive Behavior* 1.1 (June 1992), pp. 91–122. ISSN: 1059-7123. DOI: 10.1177/105971239200100105. URL: http://journals.sagepub.com/doi/10.1177/105971239200100105.

[5] Graham T. Brown. 'The Intrinsic Factors in the Act of Progression in the Mammal'. In: *Proceedings of the Royal Society B: Biological Sciences* 84.572 (Dec. 1911), pp. 308–319. ISSN: 0962-8452. DOI: 10.1098/rspb.1911.0077. URL: http://rspb.royalsocietypublishing.org/cgi/doi/10.1098/rspb.1911.0077.

[6] Hillel J. Chiel, Randall D. Beer and John C. Gallagher. 'Evolution and Analysis of Model CPGs for Walking: I. Dynamical Modules'. In: *Journal of Computational Neuroscience* 7.2 (1999), pp. 99–118. ISSN: 09295313. DOI: 10.1023/A:1008923704408. URL: http://link.springer.com/10.1023/A:1008923704408.

[7] Jeff Clune et al. 'Evolving coordinated quadruped gaits with the HyperNEAT generative encoding'. In: *2009 IEEE Congress on Evolutionary Computation*. IEEE, May 2009, pp. 2764–2771. ISBN: 978-1-4244-2958-5. DOI: 10.1109/CEC.2009.4983289. URL: http://ieeexplore.ieee.org/document/4983289/.

[8] Alessandro Crespi et al. 'Salamandra Robotica II: An Amphibious Robot to Study Salamander-Like Swimming and Walking Gaits'. In: *IEEE Transactions on Robotics* 29.2 (Apr. 2013), pp. 308–320. ISSN: 1552-3098. DOI: 10.1109/TRO.2012.2234311. URL: http://ieeexplore.ieee.org/document/6416074/.

[9] David B. D'Ambrosio, Jason Gauci and Kenneth O. Stanley. 'Hyperneat: The first five years'. In: *Studies in Computational Intelligence* 557 (2014), pp. 159–185. ISSN: 1860949X. DOI: 10.1007/978-3-642-55337-0_5. URL: http://link.springer.com/10.1007/978-3-642-55337-0%7B%5C_%7D5.

[10] Kalyanmoy Deb et al. 'A fast and elitist multiobjective genetic algorithm: NSGA-II'. In: *IEEE Transactions on Evolutionary Computation* 6.2 (Apr. 2002), pp. 182–197. ISSN: 1089778X. DOI: 10.1109/4235.996017.

[11] Agoston E. Eiben and James E. Smith. *Natural Computing Series Introduction to Evolutionary Computing*. 2015. ISBN: 978-3-662-44873-1. DOI: 10.1007/978-3-662-44874-8. URL: www.springer.com/series/.

[12] Matthew Hausknecht et al. 'A neuroevolution approach to general atari game playing'. In: *IEEE Transactions on Computational Intelligence and AI in Games* 6.4 (Dec. 2014), pp. 355–366. ISSN: 1943068X. DOI: 10.1109/TCIAIG.2013.2294713. URL: http://ieeexplore.ieee.org/document/6756960/.

[13] Patrick Hénaff et al. 'Real time implementation of CTRNN and BPTT algorithm to learn on-line biped robot balance: Experiments on the standing posture'. In: *Control Engineering Practice* 19.1 (Jan. 2011), pp. 89–99. ISSN: 0967-0661. DOI: 10.1016/J.CONENGPRAC.2010.10.002. URL: https://www.sciencedirect.com/science/article/pii/S096706611000211X.

[14] Sture Holm. 'A Simple Sequentially Rejective Multiple Test Procedure'. In: *Scandinavian Journal of Statistics* 6.2 (1979), pp. 65–70. ISSN: 03036898, 14679469. URL: http://www.jstor.org/stable/4615733.

[15] Kurt Hornik. 'Approximation capabilities of multilayer feedforward networks'. In: *Neural Networks* 4.2 (Jan. 1991), pp. 251–257. ISSN: 0893-6080. DOI: 10.1016/0893-6080(91)90009-T. URL: https://www.sciencedirect.com/science/article/pii/089360809190009T?via%7B%5C%%7D3Dihub.

[16] Jemin Hwangbo et al. 'Learning agile and dynamic motor skills for legged robots'. In: *Science Robotics* 4.26 (2019), eaau5872. DOI: 10.1126/scirobotics.aau5872.

[17] A. J. Ijspeert et al. 'From Swimming to Walking with a Salamander Robot Driven by a Spinal Cord Model'. In: *Science* 315.5817 (Mar. 2007), pp. 1416–1420. ISSN: 0036-8075. DOI: 10.1126/science.1138353. URL: http://www.sciencemag.org/cgi/doi/10.1126/science.1138353.

[18] Milan Jelisavcic, Diederik M. Roijers and A.E. Eiben. 'Analysing the Relative Importance of Robot Brains and Bodies'. In: *The 2018 Conference on Artificial Life*. Cambridge, MA: MIT Press, 2018, pp. 327–334. DOI: 10.1162/isal_a_00063. URL: https://www.mitpressjournals.org/doi/abs/10.1162/isal%7B%5C_%7Da%7B%5C_%7D00063.

[19] Milan Jelisavcic et al. 'Analysis of Lamarckian evolution in morphologically evolving robots'. In: *Proceedings of the 14th European Conference on Artificial Life ECAL 2017*. Cambridge, MA: MIT Press, Sept. 2017, pp. 214–221. ISBN: 978-0-262-34633-7. DOI: 10.7551/ecal_a_038. URL: https://www.mitpressjournals.org/doi/abs/10.1162/isal%7B%5C_%7Da%7B%5C_%7D038.

[20] Mrinal Kalakrishnan et al. 'Fast, robust quadruped locomotion over challenging terrain'. In: *2010 IEEE International Conference on Robotics and Automation*. IEEE, May 2010, pp. 2665–2670. ISBN: 978-1-4244-5038-1. DOI: 10.1109/ROBOT.2010.5509805. URL: http://ieeexplore.ieee.org/document/5509805/.

[21] Gongjin Lan et al. 'Directed Locomotion for Modular Robots with Evolvable Morphologies'. In: Springer, Cham, Sept. 2018, pp. 476–487. DOI: 10.1007/978-3-319-99253-2_38. URL: http://link.springer.com/10.1007/978-3-319-99253-2%7B%5C_%7D38.

[22] Joel Lehman and Kenneth O. Stanley. 'Abandoning objectives: Evolution through the search for novelty alone'. In: *Evolutionary Computation* 19.2 (June 2011), pp. 189–222. ISSN: 10636560. DOI: 10.1162/EVCO_a_00025. URL: http://www.mitpressjournals.org/doi/10.1162/EVCO%7B%5C_%7Da%7B%5C_%7D00025.

[23] Joel Lehman and Kenneth O. Stanley. 'Evolving a diversity of creatures through novelty search and local competition'. In: *Genetic and Evolutionary Computation Conference, GECCO'11*. 2011, pp. 211–218. ISBN: 9781450305570. DOI: 10.1145/2001576.2001606.

[24] M. Anthony Lewis and George A. Bekey. 'Gait adaptation in a quadruped robot'. In: *Autonomous Robots* 12.3 (May 2002), pp. 301–312. ISSN: 09295593. DOI: 10.1023/A:1015221832567.

[25] Stephen Marsland. *Machine Learning*. Chapman and Hall/CRC, Oct. 2014. ISBN: 9781466583337. DOI: 10.1201/b17476. URL: https://www.taylorfrancis.com/books/9781466583337.

[26] Warren S. McCulloch and Walter Pitts. 'A logical calculus of the ideas immanent in nervous activity'. In: *The Bulletin of Mathematical Biophysics* 5.4 (Dec. 1943), pp. 115–133. ISSN: 0007-4985. DOI: 10.1007/BF02478259. URL: http://link.springer.com/10.1007/BF02478259.

[27] Jean Baptiste Mouret. 'Novelty-based multiobjectivization'. In: *Studies in Computational Intelligence*. Vol. 341. 2011, pp. 139–154. ISBN: 9783642182716. DOI: 10.1007/978-3-642-18272-3_10. URL: http://link.springer.com/10.1007/978-3-642-18272-3%7B%5C_%7D10.

[28] Jean-Baptiste Mouret and Jeff Clune. 'Illuminating search spaces by mapping elites'. In: (Apr. 2015). arXiv: 1504.04909. URL: https://arxiv.org/abs/1504.04909.

[29] Jørgen Nordmoen, Kai Olav Ellefsen and Kyrre Glette. 'Combining MAP-Elites and Incremental Evolution to Generate Gaits for a Mammalian Quadruped Robot'. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 10784 LNCS. 2018, pp. 719–733. ISBN: 9783319775371. DOI: 10.1007/978-3-319-77538-8_48. URL: https://doi.org/10.1007/978-3-319-77538-8%7B%5C_%7D48.

[30] Jørgen Nordmoen and Tønnes F. Nygaard. *DyRET - OpenAI Gym Environment*. Mar. 2020. URL: https://github.uio.no/jorgehn/gym-dyret.

[31] Jørgen Nordmoen et al. 'Dynamic mutation in MAP-Elites for robotic repertoire generation'. In: *ALIFE 2018 - 2018 Conference on Artificial Life: Beyond AI*. MIT Press, 2020, pp. 598–605. DOI: 10.1162/isal_a_00110.

[32] Jørgen Nordmoen et al. 'Evolved embodied phase coordination enables robust quadruped robot locomotion'. In: (2019). DOI: 10.1145/3321707.3321762. URL: https://doi.org/10.1145/3321707.3321762.

[33] Tønnes F. Nygaard, Jim Torresen and Kyrre Glette. 'Multi-objective evolution of fast and stable gaits on a physical quadruped robotic platform'. In: *2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016*. Institute of Electrical and Electronics Engineers Inc., Feb. 2017. ISBN: 9781509042401. DOI: 10.1109/SSCI.2016.7850167.

[34] Tønnes F. Nygaard et al. *Exploring Mechanically Self-Reconfiguring Robots for Autonomous Design*. Tech. rep. arXiv: 1805.02965v1. URL: http://robotikk.net/project/dyret/.

[35] Tønnes F. Nygaard et al. 'Real-World Evolution Adapts Robot Morphology and Control to Hardware Limitations'. In: (). DOI: 10.1145/3205455.3205567. arXiv: 1805.03388v1. URL: https://doi.org/10.1145/3205455.3205567.

[36] Tønnes F. Nygaard et al. 'Self-Modifying Morphology Experiments with DyRET: Dynamic Robot for Embodied Testing'. In: *undefined* (2018). arXiv: 1803.05629. URL: http://arxiv.org/abs/1803.05629.

[37] Dai Owaki et al. 'A minimal model describing hexapedal interlimb coordination: The tegotae-based approach'. In: *Frontiers in Neurorobotics* 11.JUN (2017), pp. 1–13. ISSN: 16625218. DOI: 10.3389/fnbot.2017.00029.

[38] Sebastian Risi. *An Enhanced Hypercube-Based Encoding for Evolving the Placement, Density and Connectivity of Neurons*. Tech. rep. 2012. URL: https://eplex.cs.ucf.edu/papers/risi%7B%5C_%7Dalife12.pdf.

[39] Sebastian Risi and Kenneth O. Stanley. 'Confronting the challenge of learning a flexible neural controller for a diversity of morphologies'. In: *Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference - GECCO '13*. New York, New York, USA: ACM Press, 2013, p. 255. ISBN: 9781450319638. DOI: 10.1145/2463372.2463397. URL: http://dl.acm.org/citation.cfm?doid=2463372.2463397.

[40] Sebastian Risi and Kenneth O. Stanley. 'Indirectly Encoding Neural Plasticity as a Pattern of Local Rules'. In: Springer, Berlin, Heidelberg, 2010, pp. 533–543. DOI: 10.1007/978-3-642-15193-4_50. URL: http://link.springer.com/10.1007/978-3-642-15193-4%7B%5C_%7D50.

[41] S Rossignol. 'Locomotion and its recovery after spinal injury.' In: *Current opinion in neurobiology* 10.6 (Dec. 2000), pp. 708–16. ISSN: 0959-4388. URL: http://www.ncbi.nlm.nih.gov/pubmed/11240279.

[42] David Silver et al. 'Mastering the game of Go with deep neural networks and tree search'. In: *Nature* 529.7587 (Jan. 2016), pp. 484–489. ISSN: 0028-0836. DOI: 10.1038/nature16961. URL: http://www.nature.com/articles/nature16961.

[43] Alexander Sproewitz et al. 'Learning to Move in Modular Robots using Central Pattern Generators and Online Optimization'. In: *The International Journal of Robotics Research* 27.3-4 (Mar. 2008), pp. 423–443. ISSN: 0278-3649. DOI: 10.1177/0278364907088401. URL: http://journals.sagepub.com/doi/10.1177/0278364907088401.

[44] Kenneth O. Stanley. 'Compositional pattern producing networks: A novel abstraction of development'. In: *Genetic Programming and Evolvable Machines* 8.2 (June 2007), pp. 131–162. ISSN: 1389-2576. DOI: 10.1007/s10710-007-9028-8. URL: http://link.springer.com/10.1007/s10710-007-9028-8.

[45] Kenneth O. Stanley, David B. D'Ambrosio and Jason Gauci. *A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks*. Apr. 2009. DOI: 10.1162/artl.2009.15.2.15202. URL: http://www.mitpressjournals.org/doi/10.1162/artl.2009.15.2.15202.

[46] Kenneth O. Stanley et al. ‘Designing neural networks through neuroevolution’. In: *Nature Machine Intelligence* 1.1 (Jan. 2019), pp. 24–35. ISSN: 2522-5839. DOI: 10.1038/s42256-018-0006-z. URL: http://www.nature.com/articles/s42256-018-0006-z.

[47] Kenneth O. Stanley et al. *Evolving Neural Networks Through Augmenting Topologies*. 2002. DOI: 10.1016/j.envres.2018.03.034. arXiv: 1407.0576. URL: http://nn.cs.utexas.edu/keyword?stanley:ec02.

[48] Steven Borowiec. *Google’s AlphaGo AI defeats human in first game of Go contest*. 2016. URL: https://www.theguardian.com/technology/2016/mar/09/google-deepmind-alphago-ai-defeats-human-lee-sedol-first-game-go-contest.

[49] Felipe Petroski Such et al. ‘Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning’. In: (Dec. 2017). arXiv: 1712.06567. URL: http://arxiv.org/abs/1712.06567.

[50] Daniel Wolpert. *Daniel Wolpert: The real reason for brains | TED Talk*. 2011. URL: https://www.ted.com/talks/daniel%7B%5C_%7Dwolpert%7B%5C_%7Dthe%7B%5C_%7Dreal%7B%5C_%7Dreason%7B%5C_%7Dfor%7B%5C_%7Dbrains?language=en.

[51] Jason Yosinski and Jeff Clune. ‘Evolving robot gaits in hardware: the HyperNEAT generative encoding vs. parameter optimization’. In: *Proceedings of the European Conference on Artificial Life* (2011), pp. 890–897. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.375.5366%20http://creativemachines.cornell.edu/sites/default/files/yosinski11%7B%5C_%7Decal.pdf.

Appendix

Tables

Table A.1: NEAT/CPPN Parameters

NEAT	
fitness_criterion	max
fitness_threshold	0
no_fitness_termination	TRUE
pop_size	300
reset_on_extinction	FALSE
Node activation options	
activation_default	sigmoid
activation_mutate_rate	0.5
activation_options	gauss sin abs sigmoid
Node aggregation options	
aggregation_default	sum
aggregation_mutate_rate	0
aggregation_options	sum
Node bias options	
bias_init_mean	0
bias_init_stdev	1
bias_max_value	1
bias_min_value	-1
bias_mutate_power	0.5
bias_mutate_rate	0.6
bias_replace_rate	0.1
Genome compatibility options	
compatibility_disjoint_coefficient	1
compatibility_weight_coefficient	0.5
Connection add/remove rates	
conn_add_prob	0.2
conn_delete_prob	0.2

Connection enable options	
enabled_default	TRUE
enabled_mutate_rate	0.25
feed_forward	TRUE
initial_connection	full
Node add/remove rates	
node_add_prob	0.1
node_delete_prob	0.1
Connection weight options	
weight_init_mean	0
weight_init_stdev	1
weight_max_value	5
weight_min_value	-5
weight_mutate_power	0.5
weight_mutate_rate	0.6
weight_replace_rate	0.1
Species	
compatibility_threshold	3
Stagnation	
species_fitness_func	max
max_stagnation	180
species_elitism	90
Reproduction	
elitism	150
survival_threshold	0.2

Table A.2: Pairwise MWU test result comparing EAs. The highlighted row indicates $p < 0.05$ for uncorrected p-value.

A	B	U-val	Tail	p-unc	p-corr	p-adjust
MOEA	Fitness Only	9.0	two-sided	0.018129	0.054387	holm
MOEA	Novelty Only	16.0	two-sided	0.103562	0.207124	holm
Fitness Only	Novelty Only	43.0	two-sided	0.270149	0.270149	holm

A	B	U-val	Tail	p-unc	p-corr	p-adjust
MOEA	Fitness Only	13.0	two-sided	0.052030	0.312178	holm
MOEA	Fit Only no pos	64.0	two-sided	0.000939	0.009391	holm
MOEA	Novelty Only	18.0	two-sided	0.156254	0.781270	holm
MOEA	NS Only no pos	25.0	two-sided	0.494837	0.989673	holm
Fitness Only	Fit Only no pos	64.0	two-sided	0.000939	0.009391	holm
Fitness Only	Novelty Only	43.0	two-sided	0.270149	0.810446	holm
Fitness Only	NS Only no pos	46.0	two-sided	0.156254	0.781270	holm
Fit Only no pos	Novelty Only	0.0	two-sided	0.000939	0.009391	holm
Fit Only no pos	NS Only no pos	0.0	two-sided	0.000939	0.009391	holm
Novelty Only	NS Only no pos	35.0	two-sided	0.792896	0.989673	holm

Table A.3: Pairwise MWU test result EA comprison with no position runs. The highlighted row indicates $p < 0.05$ for uncorrected p-value.

A	B	U-val	Tail	p-unc	p-corr	p-adjust
All	Ampl&Duty	55.0	two-sided	0.018129	0.326322	holm
All	Ampl&Pos	56.0	two-sided	0.013587	0.258158	holm
All	Duty&Pos	52.0	two-sided	0.040569	0.608533	holm
All	Ampl	60.0	two-sided	0.003876	0.081397	holm
All	Duty	60.0	two-sided	0.003876	0.081397	holm
All	Pos	36.0	two-sided	0.713191	1.000000	holm
Ampl&Duty	Ampl&Pos	12.0	two-sided	0.040569	0.608533	holm
Ampl&Duty	Duty&Pos	11.0	two-sided	0.031324	0.501186	holm
Ampl&Duty	Ampl	25.0	two-sided	0.494837	1.000000	holm
Ampl&Duty	Duty	17.0	two-sided	0.127808	1.000000	holm
Ampl&Duty	Pos	16.0	two-sided	0.103562	1.000000	holm
Ampl&Pos	Duty&Pos	27.0	two-sided	0.636502	1.000000	holm
Ampl&Pos	Ampl	44.0	two-sided	0.227147	1.000000	holm
Ampl&Pos	Duty	54.0	two-sided	0.023949	0.407127	holm
Ampl&Pos	Pos	25.0	two-sided	0.494837	1.000000	holm
Duty&Pos	Ampl	50.0	two-sided	0.066082	0.859065	holm
Duty&Pos	Duty	46.0	two-sided	0.156254	1.000000	holm
Duty&Pos	Pos	28.0	two-sided	0.713191	1.000000	holm
Ampl	Duty	34.0	two-sided	0.874826	1.000000	holm
Ampl	Pos	17.0	two-sided	0.127808	1.000000	holm
Duty	Pos	23.0	two-sided	0.372029	1.000000	holm

Table A.4: Pairwise MWU test result for Behavior descriptor comparison - fitness. The highlighted row indicates $p < 0.05$ for uncorrected p-value.

A	B	U-val	Tail	p-unc	p-corr	p-adjust
All	Ampl&Duty	37.0	two-sided	0.325271	1.000000	holm
All	Ampl&Pos	29.0	two-sided	0.792896	1.000000	holm
All	Duty&Pos	21.0	two-sided	0.270149	1.000000	holm
All	Ampl	26.0	two-sided	0.563524	1.000000	holm
All	Duty	33.0	two-sided	0.958122	1.000000	holm
All	Pos	35.0	two-sided	0.792896	1.000000	holm
Ampl&Duty	Ampl&Pos	16.0	two-sided	0.183233	1.000000	holm
Ampl&Duty	Duty&Pos	10.0	two-sided	0.042844	0.856872	holm
Ampl&Duty	Ampl	12.0	two-sided	0.072849	1.000000	holm
Ampl&Duty	Duty	20.0	two-sided	0.385418	1.000000	holm
Ampl&Duty	Pos	25.0	two-sided	0.772337	1.000000	holm
Ampl&Pos	Duty&Pos	26.0	two-sided	0.563524	1.000000	holm
Ampl&Pos	Ampl	30.0	two-sided	0.874826	1.000000	holm
Ampl&Pos	Duty	40.0	two-sided	0.430897	1.000000	holm
Ampl&Pos	Pos	39.0	two-sided	0.494837	1.000000	holm
Duty&Pos	Ampl	43.0	two-sided	0.270149	1.000000	holm
Duty&Pos	Duty	54.0	two-sided	0.023949	0.502922	holm
Duty&Pos	Pos	47.0	two-sided	0.127808	1.000000	holm
Ampl	Duty	44.0	two-sided	0.227147	1.000000	holm
Ampl	Pos	41.0	two-sided	0.372029	1.000000	holm
Duty	Pos	33.0	two-sided	0.958122	1.000000	holm

Table A.5: Pairwise MWU test result for Behavior descriptor comparison - performance The highlighted row indicates $p < 0.05$ for uncorrected p-value.

Supplementary Video

The supplementary video is attached to DUO, see 'supplementary_video.mp4'. And it can also be played on YouTube via link : https://youtu.be/a9M2IFrUjG8