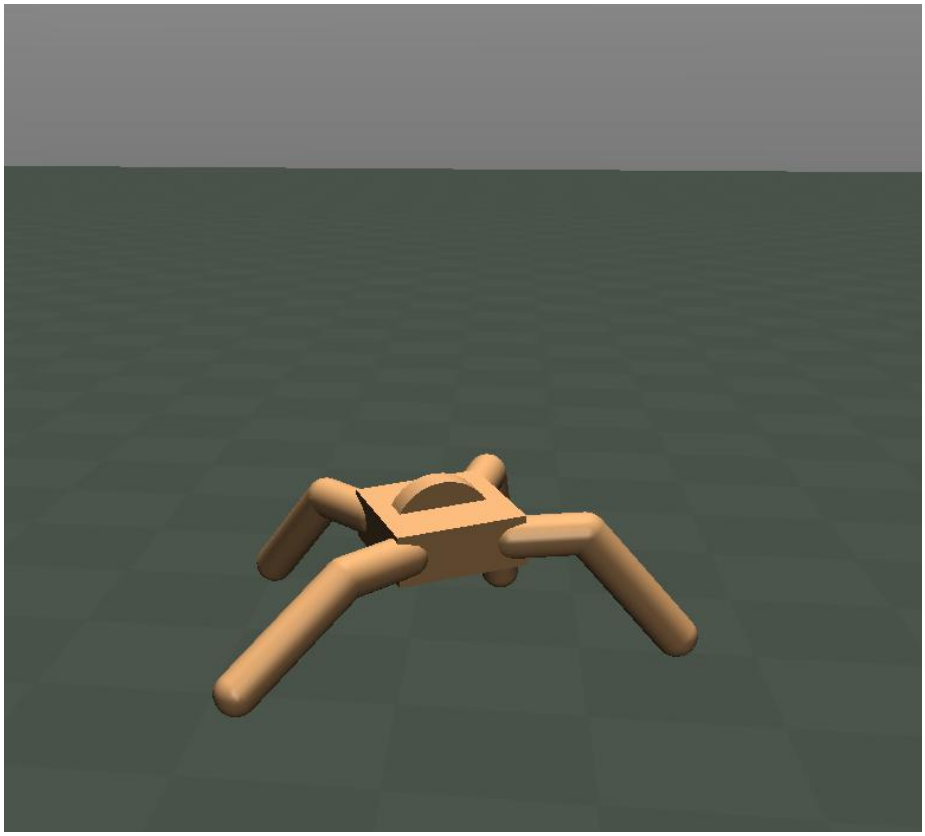
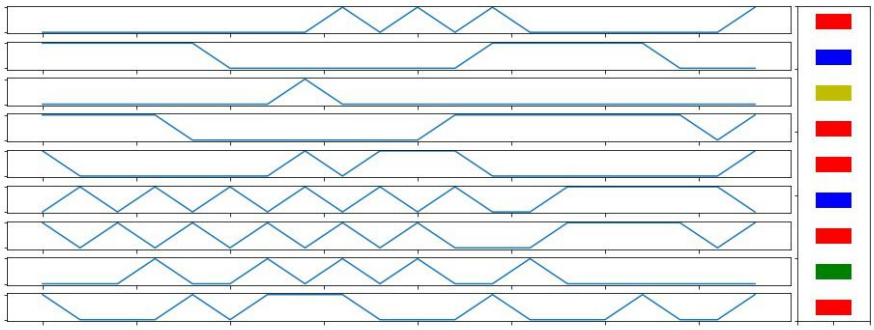
Fabrizio AMBROGI

Evolving a Spiking Neural Network controller for low gravity environments





Master Thesis in Artificial Intelligence

UNIVERSITY OF AMSTERDAM

MSc Artificial Intelligence
Master Thesis

Evolution of a Spiking Neural Network controller for low-gravity environments

by
Fabrizio AMBROGI
11403640

October 26, 2018

36 Credits
February 2018 - September 2018

Supervisor/Examiner:
dr. Arnoud VISSER

Assessor:
dr. Herke VAN HOOF

Abstract

This thesis addresses the development of a lightweight and quickly trainable controller based on bio-inspired and time-dependent spiking neural networks, tasked to enact an effective gait for a quadruped rover with a reaction wheel that operates in low-gravity environments. Such model would be an interesting addition to the methods presently used in robot controllers, especially in space exploration were the availability of computational power is limited. With the solution proposed in this work, the controller can be optimized in a day on a single computer and outperform conventional perceptron based architectures in simulations.

The use of Spiking Neural Networks makes the common back-propagation optimization techniques unavailable, requiring a new approach. Evolutionary methods then come into play. With this architecture the controller has a very powerful tool to memorize frequencies and reproduce complex periodical movements. The use of evolutionary algorithms makes their optimization effective and competitive.

Optimizing the spiking controller with Genetic Algorithms and Differential Evolution, the architecture proposed obtains good fitness scores in several known benchmarks in under a day of simulations. Furthermore, it learns effective gaits for the low-gravity environments with results that vastly surpass the ones obtained by similarly evolved perceptron based controllers. All the simulations and optimization done show that the spiking controller has a superior ability to environments with strong statistical confidence (Welch's t-Test $p \ll 0.01$). On the Mars and Moon environments it learns a periodical gaits that makes the rover advance and never touch the ground with its main body. On the Ceres environments it learns to optimize the fitness function to a local maximum far superior to the ones obtained by other architectures, although unfortunately not with a beautiful gait.

Contents

Notes

List of Abbreviations

low-g	-	Low gravity
NN	-	Neural Network
MLP	-	Multi-Layered Perceptron
CNN	-	Convolutional Neural Network
RNN	-	Recurrent Neural Network
SNN	-	Spiking Neural Network
GA	-	Genetic Algorithms
NES	-	Natural Evolution Strategies
DE	-	Differential Evolution
ReLU	-	Rectified Linear Unit
tanh	-	Hyperbolic tangent

Animation folder

Since in this thesis are included numerous animations and videos that may be not available on the medium in which is read (printed on paper or with some pdf readers), a public folder is made on the Cloud to find the single images of the short animations and the full videos of the simulations.

`tinyurl.com/evolvingSNNanimations`

1 Introduction

With the renewed interest in the space race, national agencies and private corporations are developing new rovers. Novel designs are being proposed to tackle new challenges: human bases on the Moon and Mars and the explorations of other celestial bodies. The new rovers will need higher autonomy, since the distance to their destinations as well as their number will quickly rise. Providing them a better Artificial Intelligence, dependability and autonomy should follow.

Between the most attractive destinations are the asteroids and little moons, full of rare metals. These celestial bodies are very light compared to Earth, with small gravitational forces. The lower gravity causes this body to have irregular surfaces and the subsequent absence of an atmosphere leaves them scattered with impact craters. These terrain irregularities and the low gravity in itself make the movement difficult for the rovers [8], especially wheeled ones.

A possible solution is to add vernier thrusters to the rovers, to make them maintain balance and take short flight when needed. However, this requires a fuel tank and that creates further balancing problems (with the movement of liquid in the container) and a disposable object, since the distance would make it impossible to come back to refuel, making the whole apparatus useless once the tank is empty.

Another option, explored by NASA with the LEMUR project [8], is to use legged rovers, that can crawl, walk or jump based on the need. The idea of jumping rovers for low gravity environments has very recently been made a reality by JAXA, with the succesful Hayabusa2 mission [18]. The introduction of this third degree of freedom, leaving the contact with surface, requires an instrument to orienter the robot whilst on flight, so that it could land safely. For this reason a reaction wheel has been added to the rover. Moreover, the presence of legs permits an additional instrument for self orientation, exploiting the conservation of angular momentum. This addition has costs too, a higher amount of energy consumed and the complexity increases with the addition of more controllable motors, with partially redundant functions.

1.1 Definition of the problem

The goal of this thesis is to develop a controller for a robot that acts in low gravity. Moving in such environments is not easy and there are no examples on Earth to take inspiration from. Developing the optimal gait by hand is a very complex deal, already starting with the inertia of the rover, and coping with the long jumps it will perform in low-g environments so that it may land gracefully. An artificial intelligence is then built to learn from experience in simulations and evolve into an effective controller for the shape and gravity under inspection.

1.1.1 Robot description

The robot is modeled after the body of an arachnid. It is a quadruped, with three joints per leg, and a reaction wheel in the central body. The legs are positioned on the longer side of the robot and very close to the corners. The reaction wheel is at the center of the body. The shoulder presents two joints in immediate succession, with the first going back and forth (Z axis at rest) and the second up and down (Y axis at rest). These approximate a universal joint. The elbow is a single joint that goes, again, up and down (Y axis at rest). All of the joints have limited rotation capability, simulating the impossibility of "clipping" a limb through the others or the body. The size of each body part can be seen in Table 1.

Part	Height	Length	Width
Body	1	1	0.4
Arm	0.32	0.8	0.32
Forearm	0.32	0.8	0.32
Wheel	0.8	0.8	0.32

Table 1: Robot parts sizes

Figure 1: The quadrupedal robot used in the experiments in low-g environments.

1.1.2 Active Mass Balance Auto-Control

The presence of limbs enables more than just the ambulation in uneven terrains. They can also be used to orient the robot while in flight, using the law of conservation of angular momentum [29]. The addition of the reaction wheel makes it so the change in orientation does not need a reconfiguration of the limbs' position. This should be particularly useful in the case under examination, since springing into a jump might modify the rotation of the rover in such a way that the landing would be uncontrollable. The use of the wheel and the limbs to adjust the relative position before landing is important and should give an edge over an object that cannot balance during flight.

1.2 Proposed approach

In this work evolutionary computing will be used to train a spiking neural network controller to drive the robot. The implementation of this spiking architecture is tied to the choice of evolutionary computing over other training techniques.
The intention of this thesis is to demonstrate that Spiking Neural Network controllers have an edge for frequency based behaviours, like encoding a gait [3]. The spiking component guided by the internal parameters is built to react to inputs in a very nonlinear manner and the combinations of spike frequencies can create highly complex periodical gaits, or modify them smoothly whenever the environment should require it. It has been theoretically proven that the function approximation power of a spiking neural network is superior to the perceptron architectures [24], but their parameters are more difficult to optimize, especially with back-propagation.

The use of evolutionary methods to optimize neural networks has grown in the last years, with a rediscovery of their potential. They need more examples to optimize a controller when compared to back propagation [32], but this is made up for since they do not use complex and convoluted derivatives to mutate the parameters. This reduces the computational cost, making the trade-off with the higher simulation count worthy. Theoretically, these methods can optimize any fitness function, with arbitrary long time breaks between an action and the reward. This last point is still considered a hard challenge for Q-learning models [25].

1.2.1 Novelties

From the knowledge of the author, this is the first application of Spiking Neural Networks to the MuJoCo benchmarks. This also seems to be the first time that the Differential Evolution method is applied to optimize Spiking Neural Networks. In this work, both are joined and expanded in a field specific test, tied with a present day task like space exploration.

Another important accomplishment of this work is to find a solution that can be trained and run on a personal computer. This absolves the function of simulating in-loco training by the rovers or planet control. The possibility of sending clusters of supercomputers on another planet is unlikely in the immediate future and the distance from Earth in some cases would make communication difficult. Therefore, sending a list of real observations could not be an easy or even affordable action (data transmission consumes energy). The ability of optimizing with a local computer the pre-trained behaviours from the Earth simulations, incorporating real data observed by the rovers on the mission, is an important feature.

An additional goal is to prove the quality of an often overlooked architecture, Spiking Neural Networks. These are usually synonymous with slow computations and complex hardware requirements. Yet, with the model proposed by Berland [2] and its implementation developed in this thesis, this architecture might gain a more positive image and spark new researches on these models.

2 Neural networks

Artificial neurons originated in the early second half of the 20th century as models of the biological brain. The original models were more interested in accuracy and simulation than having a practical use for controllers.

The attention later moved on the use of artificial neuron networks as computational tools and, with the introduction of Hopfield networks and perceptrons, the efficiency became the main focus. With the use of the latter, artificial neural networks become universal function approximators. Given a function a wide enough network can compute it with arbitrary precision. For years the only limit was training time, but with the late explosion in computational power and the use of parallel computing on GPUs, Neural Networks have become the main tool for artificial intelligence research.

2.1 Action-Value vs Policy Networks

There are two possible uses for neural networks when working on Reinforcement Learning problems. The first one is to have an Action-Value network, or Q-network, which learns from past experiences what returns an action will have, given the present state. This was used by Google's DeepMind to learn to play Atari games [25].

The strength of this approach is that it can be trained with a supervised learning framework, with stochastic gradient descent. Once a sufficient amount of experiences is collected and a formula for the Q-value of each action-state couple is defined the network will learn to output this reward given a state and an action as input. One perceived weakness of this approach is that, if actions are continuous, it is impossible to find the best one for the state without trying a huge number of random entries and only then choosing the one with the highest Q-value.

The second option is to have a Policy Network, which receives the state as input and returns an action as output. This can't be trained directly as the previous option, since there is no way of creating a dataset of states and optimal action to follow. Having a formula to make it would make the network itself superfluous. This was tackled by DeepMind by coupling a Q-network with a policy network [26], such that the Q-value could be trained with the experiences and the policy would be optimized to maximize the output of the former. These two operations would be alternated, with the creation of a new dataset in between by having the present policy network play the game.

Since in this thesis Evolutionary Algorithms will be used to learn complex continuous actions, the policy network is absolutely necessary, but the Q-network can be dropped, as evolutionary methods don't need a step-by-step reference to improve the parameters.

2.2 Architectures of Artificial Neural Networks

The most used architectures for Neural Networks are Multi-Layered Perceptrons (MLP), Convolutional Neural Networks (ConvNet or CNN) and Recurrent Neural Networks (RNN).

MLPs [42] are based on the concept of Perceptron, a function that takes an arbitrary number of inputs and emits a single output. More can be put in parallel in a Layer, so that the output will have a higher dimensionality. Furthermore it is possible to align multiple layers in succession, such that each one will have as input the output of the previous, and a high dimensionality can be used for the inner (hidden) layers to approximate more complex functions. This is one of the variants that will be tested in this thesis, as a comparison baseline for other, more complex, methods.

CNNs [20] use instead the concept of a filter, a small set of weights that computes over a small section of the input and moves around, giving a single output for each window it was applied to. They are typically used for Computer Vision and sometimes for Natural Language Processing. Since the policy networks will not be trained on the visual output of the simulation, but in vector form on raw numerical data (the rover will not see itself from outside, but instead have other sensors to define its pose) the use of ConvNets is not deemed necessary.

RNNs come in several forms. The simplest one feeds itself the previous output as an input for the next operation. A more complex and effective variant which has gained more popularity is the Long Short Term Memory network [15]. For sequential data, like a list of observations over time, RNNs are better suited then MLPs. For this reason a form of recursiveness will be used to strengthen the capability of the controller in two of the architectures tested.

All the aforementioned implementations are extensions of the perceptron, which is a kind of Artificial Neuron. Even if lately most experiments and researches were run on this neuron model, others options were proposed and have some useful feats that could make them more fit for this task.

2.3 Spiking Neural Networks

Artificial Spiking Neurons are an older and more realistic model of biological neurons. Compared to perceptrons, which are instantaneous functions with continuous output, spiking neurons have an "integrate and fire" behavior. They store the past inputs in a "leaky memory" and have self-adapting threshold that together determine the output.

The output is commonly interpreted as binary in nature, indicating the emission of a spike caused by the internal value overcoming the threshold. However it is strongly believed that the information is, in fact, not transmitted by the spikes themselves, but by their frequency [24] [1].

Several implementations exist, some more coherent and slow, while others more artificial and fast:

- **Hodgkin–Huxley model** This mathematical model was the first to be used to represent the action-potential of neurons. It uses four nonlinear differential equations with numerous parameters and variables, making it very unpractical for simulations, but very realistic and descriptive [16].

- **FitzHugh–Nagumo model** Already a simplification from the strongly realistic Hodgkin–Huxley model, this version uses two parameters (a, b) to determine the behavior of its two variables (voltage v and w) [10]. Its output is the voltage, a continuous measurement. The differential equations contain a cubic function, which complicates the computation and makes this the slowest method.

$$v' = v - \frac{v^3}{3} - w + I$$

$$w' = v + a - b \cdot w$$

- **Simple Model** Introduced by Izhikevich to further simplify the previous model [17]. It has three constants that are usually based on human neuron volt and ampere measures, but can be modified (or evolved) to accommodate inputs in other orders of magnitude. It also has four variables that are originally used to make the neuron analogous to one of the kinds found in the human brain. This model was built for simulations, instead of computational real-time use, but with some adjustments it can be made to work. The output can be continuous as the voltage v or binary with a spike every time the former goes over the threshold t and causes a reset. Its differential equation contain a quadratic function and an "if-then" statement. These make it faster than the previous model, but still slow for normal CPUs.

$$v' = k_1 \cdot v^2 + k_2 \cdot v + k_3 - u + I$$

$$u' = a(b \cdot v - u)$$

$$\text{If } v > t \begin{cases} v = c \\ u = u + d \end{cases}$$

- **Controller Model** This is an extremely simplified model that directly describes the memory m and the threshold t and uses only linear operations and if statements for their differential equations [2]. Although the direct

value m (or a transformation of it) could be used as the output, it was planned for it to be binary, as indicator of the presence of a spike when m is greater than t.

$$m = m + I$$

$$\text{if m} \geq \text{t} \begin{cases} t = t + b \cdot m \\ m = 0 \end{cases} \qquad \text{if m} < \text{t} \begin{cases} m = a \cdot m \\ t = t + b \cdot m \end{cases}$$

$$t = t + (c - t) \cdot \frac{b}{2}$$

Since in this work realism is not an objective, the controller model was chosen, especially as the only option that, although still slower, can be considered competitive on speed with the perceptron neurons.

2.3.1 Special Architectures and Attentions for Spiking NNs

Since SNNs are more closely related to the biological counterpart and have a unique time component, some complex but powerful architectures can be used. Some structures are not fully connected, some do not process the input instantaneously and others have blocks of neurons divided into strata.

Spike shape When a neuron spikes, the current that is transfered to its outputs can be either suddenly discharged or follow a more natural peaking distribution over time. Time dependent spikes are more realistic and offer a strong modeling power, with input neuron that can spike with specific patterns to have positive or negative inference in the output current to their shared output, this will permit incredible modeling power with a small ensemble of neurons.

Although, the advantage of a sudden discharge is the lack of computational cost and memory needed to save and update the spikes in each link, a difference which truncates the training time drastically. The digital output has been shown to be sufficient to convey considerable amount of informations anyway [48]. For this reasons punctual inputs were used in this thesis.

Ring architecture This model imagines the neurons disposed in the shape of a ring, with layers of variable size following each other and the last right before the first one in the circular shape. Each neuron in a layer is connected to at least another in the following layers; the connection are generated randomly, with probability of connection dropping with distance. A layer of neurons in the circle is fed with input data and another is taken as output.

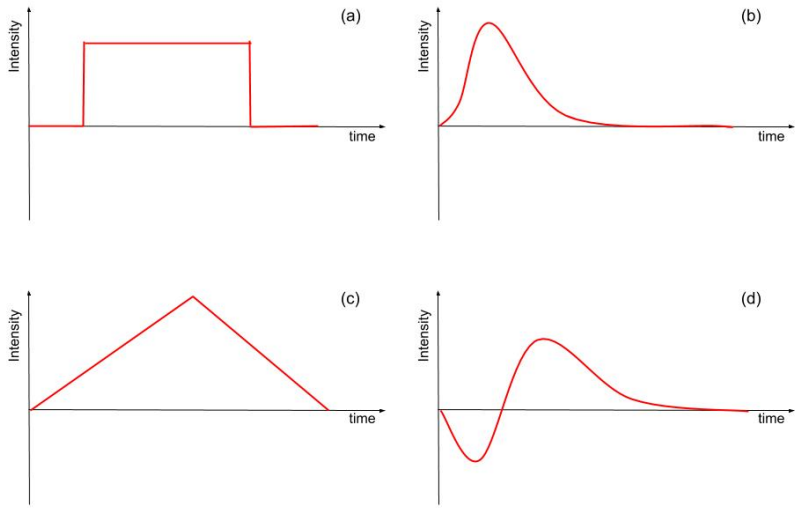


Figure 2: Examples of time dependent spikes. (a) is a step function with a delay. (b) is the Poisson density distribution. (c) is a triangle function. (d) shows a spike with an inhibitory component, followed by a positive activation.

Although this model is very powerful, and prone to periodic behaviours for its built-in feedback loop, its specific design is difficult. The stochasticity of the connection introduces new hyperparameters to tune, which makes it a two step optimization process. Even if the random links were to be changed with a hand picked design, the manual tuning would have to take place beforehand.

Information Transfer Time The fact that in a ring configuration a neuron might easily be connected with neurons in different layers brings up a problem. If the input spikes and this current is enough to make the potential of two neurons in consecutive layers greater than their threshold, should they spike at the same time?

This is an important decision to make, because any extra spike might generate a chain reaction. Once again it is good to keep in mind the original model of the biological neurons in animal brains. In these structures, the rules are set by physics and biology, so computation is almost instantaneous, but current transfer takes time based on the distance to travel. A longer connection will delay the current arrival from a spike. So it has been decided to compute layer by layer given all the input currents from the previous ones summed.

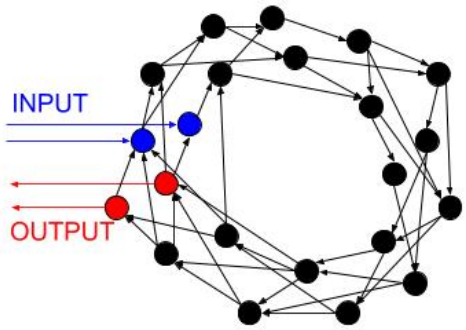


Figure 3: Depiction of a very narrow ring of spiking neurons with adjacent input and output neurons.

Stratified Networks The last structure hereby described is simple, fast and maintains some of the important features of the ring structure. The Stratified Spiking Neural Network [40] is divided in three blocks: a single layered input strata, a context strata of an arbitrary number of consecutive layers, and an output strata with one or more parallel layers, hence the name of stratified.

The input strata receives preprocessed data taken from the environment, the number of neurons is in a 1-1 relation to the observed measure. These neurons are fully connected to the first layer in the context strata. This loses the 1-1 relationship and can instead be of arbitrary size in (each of) its layer(s). In this case the strata is fully connected to its previous output, making it recurrent. Furthermore it is fully connected with all layers in the output strata. The output strata is divided in one or more parallel layers, each of them receives input from the context layer and by its own previous state. The separation of the output in more parallel layers can be useful in case of a system that controls independent peripherals that should not communicate directly with each other. The spikes of this strata define the output of the whole network. This output is also fed back to the context layer to use for the next iteration.

All of these recurrent connections make this structure very good at working with temporal sequences and commands, with the characteristic introducing past actions together with new observations in the context strata. For these reasons and its superior efficiency in computation this is the chosen architecture for the Spiking Neural Network controller.

Function of the output spikes The desired output is a continuous value, a function that translates a discrete series of binary values into the desired

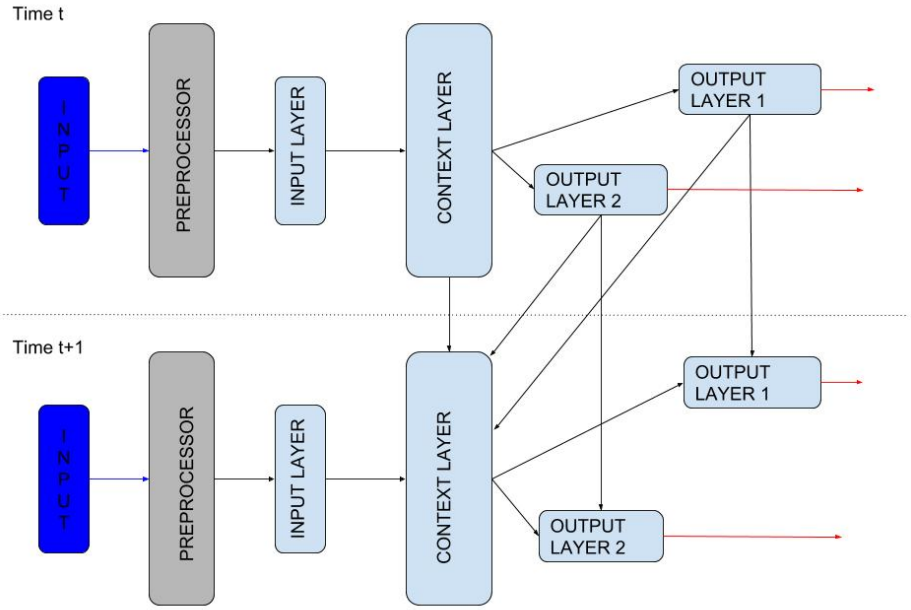


Figure 4: Representation of a Stratified Spiking Neural Network. The preprocessor is made of perceptrons, while all light cerulean layers are composed of Spiking Neurons.

number. Different techniques are possible, taking the frequencies in the past N time-steps is the simplest but it is limited, as, when there is no input, there will be non output spikes, so all actuators will start in full reverse while instead the natural idea is that no spikes means no movement. Some have proposed to apply a Poisson process to the spikes [40], which is in some ways a more refined function of the frequency, but with some problems: no negative number is a natural output of a Poisson process, and the function is not symmetrical, so the distribution of outputs between positive and negative would not be either.

Both versions can be fixed by having a second set of spikes that determines the direction, while the first one only defines the intensity. This solution still leaves an issue, as the direction would be a binary value that can immediately then switch the direction of the motors, hypothetically even from 1 to -1 in one timestep, which in a real situation would damage them. A more gradual change of pace is strongly preferred to keep the simulation realistic.

The solution used in this work is to apply a simple function with continuous dominion and periodical, smooth and symmetric output limited between 1 and -1,

a sinusoidal. This function takes the frequency as input and outputs the direction and intensity of the command in a single value. With an input frequency of 0, the output will be equally null. Once the process is started, the passage between a positive and negative output (the engine going forward or backward) can happen by a rise or descent in the frequency. For these reasons it was selected with the addition of a parameter that is multiplied by the frequency, to define how many peaks are available without having to change the length of the spike series on which the frequency is measured on.

3 Evolutionary methods

Evolutionary Algorithms are optimization techniques inspired by natural evolution, which has worked slowly but relentlessly to make life happy to live in every corner of the world and, now, even beyond, with human space exploration. These methods are proven to work in diverse environments, with few parameters and, as such, they are both adaptable and simple. This makes them well suited for the problem faced in this work.

When approaching the optimization of a neural network via evolutionary methods there are two possible modi operandi. The one which more easily comes to mind is having the genome contain every parameter of the network in a 1-1 relationship, so evolving a single gene will change only the value weight or bias it encodes. This approach was recently used by OpenAI [32] and UberAI [39].

Otherwise one could evolve a function that computes the parameters of the network based on some intrinsic variables, an indirect encoding. The most known implementation of this technique is HyperNEAT [37]. The strength of this method is the use of symmetries and patterns to quickly encode the same response for specular parts of the robot, but its weakness is the need to create an encoder function for each layer and each parameter of the neuron.

In this work the first option will be explored, with the use of three different methods described in the following section.

3.1 Direct encoding

The advantage of this implementation is the power of versatility. By tuning every single parameter independently it will approach the optimal performance given the network shape and type. This might require a considerable amount of time as while one parameter gets optimized another might move from the peak, but with the right algorithm and the inclusion of mutation adaptability this risk is strongly limited.

3.1.1 Genetic Algorithms

This is the simplest version of an evolutionary method that will be implemented in this thesis. Already from its formulation it looks very similar to a parallel random walk, yet in previous studies it appeared to be superior to random search in all tests with policy networks [39].

Every generation the best elements are kept for the new population, while the remaining spots are filled by randomly sampling individuals and mutating them. Mutation is of fixed scale and is normally distributed around the zero vector. Sampling can be either uniform across the population or weighted by fitness or rank. In both cases the sample will also include replacement, the same

individual can be picked more than once and mutated differently, before being introduced into the new population.

Parameters of the algorithm:

- **Population size** refers to the number of individuals that are "alive" at the same time. A higher number means more combinations are tested every generation. Since this method ignores derivatives completely, a certain number of individuals are needed to effectively explore the landscape.
- **Number of Elites** is the number of individuals that is passed to the next generations unchanged, without mutations, given their superior fitness score. In the original paper this is only one single individual, the best fitting. The higher it is, the more conservative the algorithm becomes, exploring frequently around the same superior individuals.
- **Mutation scale** is simply the standard deviation of the random noise which is added to a parent to create a new individual.
- **Sample type** indicates which kind of function will determine the parents of the new generation. Typically this is a stochastic function that samples from the previous population with either uniform distribution or weighted by their fitness. In case of a weighted sample the most common approaches for each probability are fitness and rank based: $w_i = fitness(i)/\sum_{j=1}^{N} fitness(j)$ vs $w_i = rank(i)/\sum_{j=1}^{N} rank(j)$ respectively. The former is more efficient at finding high value areas when there is high variance in population fitness, the latter is more robust at identifying the best even if they are very close in fitness. In both cases it is possible to apply a function to the value of interest to make the distribution more skewed or smooth. In the previous study a uniform sample was used, in this thesis the fitness proportional was considered, albeit smoothed to still allow for exploratory behaviour.

```
input : N (population size), K (number of elites), σ (mutation scale)
Population = N random individuals
for g in 1:G do
    newPopulation = Elite(Population, K)
    while size(newPopulation) < N do
        X = sample(Population)
        ϵ = sampleGaussian(0, σ)
        Mutant = X + ϵ
        newPopulation.add(Mutant)
    end
    Population = newPopulation
end
```

Algorithm 1: Genetic Algorithm pseudocode

It is technically possible (and quite easy) to introduce a kind of crossover, but the study of its effect will be over the scope and practical possibilities of this thesis. The algorithm will then be left as proposed by the original paper [39], which focused on simplicity as its strength.

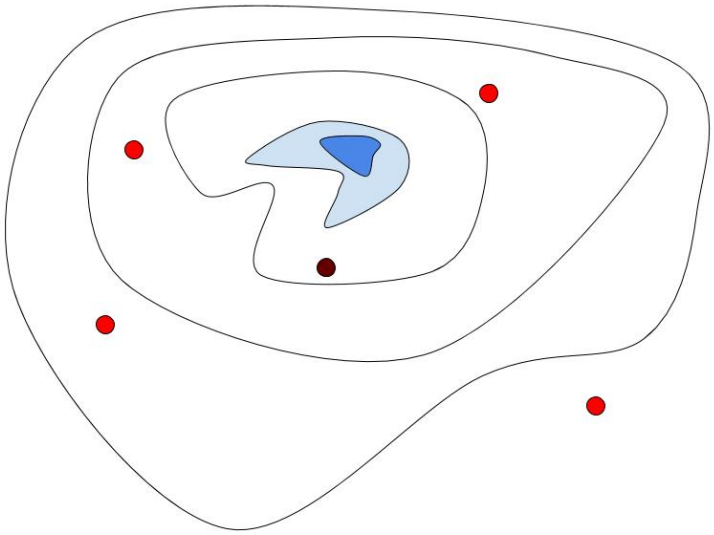


Figure 5: Animated representation of the Genetic Algorithm in action. The fitness landscape grows towards the center (darker blue). The best individual of the current generation is the darker red point, while the lighter are the ones not sampled to be parents.

3.1.2 Natural Evolutionary Strategies

This is a gradient approximation method, which optimizes the parameters with respect to the fitness function. The gradient is estimated by adding noise to the parameters before testing them, and this creates a Gaussian smoothing over the fitness function, making it sensible to apply a form of gradient ascent [32].

Practically, random mutations are applied to the original genome, the new population is tested and all those mutations are added to the original with magnitude proportional to their fitness gain. A mutation that has caused a better fitness will be summed, while one that has made the individual worse

will be subtracted. Trying to describe this version in more classical evolutionary terms, a single generation has actually two steps. Firstly the original individual creates a population of children with Gaussian mutation, then these create a single individual with a weighted average crossover.

Parameters of the algorithm

- **Mutants set size** is the number of individuals that are created in the intermediate generation, the higher the number of samples the more accurate the gradient estimation is going to be.
- **Mutation scale** determines the standard deviation of the distribution of the noise that is added to the single individual to create the population of mutants.
- **Learning rate** is the scaling factor that exacerbates or softens the mutation of the new individual, given the gradients found with the set of mutants.

input : N (mutants set size), γ (learning rate), σ (mutation scale)
X = 1 random individual
for *g in 1:G* **do**
  for *i in 1:N* **do**
    $\epsilon_i = \sigma$*sampleGaussian(**0**, I)
    Mutant = X + ϵ_i
    F_i = fitness(Mutant)
    j = N + i
    $\epsilon_j = -\epsilon_i$
    Mutant = X + ϵ_j
    F_j = fitness(Mutant)
  end
  $X = X + \gamma \cdot \frac{1}{2N\sigma} \sum_{i=1}^{2N} \epsilon_i \cdot F_i$
end

Algorithm 2: Natural Evolution Strategies pseudocode

This algorithm lacks any kind of self-adaptability, as the fitness grows so does the mutation, while in a noisy landscape, like Neural Networks parameter spaces, a form of mutation moderation can help. This problem is muffled by the forced symmetry in mutations which makes the final movement proportional to the difference in fitness between the two mutants. Furthermore, from the original OpenAI paper [32], this algorithm does not work well on simple Neural Networks and, instead, requires some advanced tricks to work properly, like virtual batch normalization, which makes the network more sensitive to little changes, this does not exist in SNNs and was not implemented in the perceptron based models, so it was decided to not use this method in the final implementation.

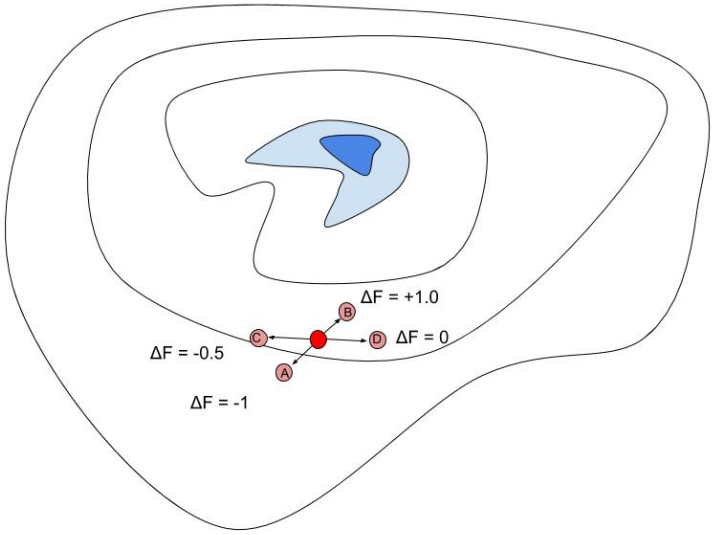


Figure 6: Animated toy representation of the Natural Evolution Strategy in action. The multiple light red points are the symmetrical couples of sampled individuals.

3.1.3 Differential Evolution

Differential Evolution [38] is a parallel direct search method that can tackle discontinuous and non differentiable fitness functions. Its unique mutation type, called differential mutation, is calculated as the difference between two genomes in the population, then scaled by a global parameter before adding it as perturbation to a third genome; this can be the one with best fitness or randomly chosen each time. This algorithm uses a uniform crossover between the new genome created with the differential mutation and another random genome from the population. The selection is of tournament type, with groups of size 2 composed of the mutant genome after the crossover and the one in the population it crossed over with.

Parameters of the algorithm

- **Population size** is the number of individuals alive at the same time. It is advisable to have a higher population size when the problem dimensionality

is big, this also helps with having better variation when applying the mutation mechanism.

- **Number of couples** determines how many pairs of parents are sampled from the population and subtracted to create the same number of differential mutations. When increased it becomes more likely to explore differences in all parameters between them. Generally, the higher the problem dimensionality, the higher this number should be, but never exceeding half the population size since, by rule, the sample is done without replacement.
- **Scaling factor** increases or decreases the entity of the differential mutation, to further decide the exploration of the method. No exact rules clip this parameter, but generally it is selected between 0 and 2.
- **Crossover probability** determines which percentage on average of the individual parameters are exchanged with the mutant ones before comparing it with the original. Its value is bound between 0 and 1. A higher value makes it so the individual from the previous generation is compared with a very different element, a lower value means that only some parameters will change and the exploration is focused on lower dimensional hyperplanes.

These parameters have been tested for different problem dimensionality and number of evaluations available (comparable to time constrain) in [28]. Those results have been treated as a base for the parameters used in this work.

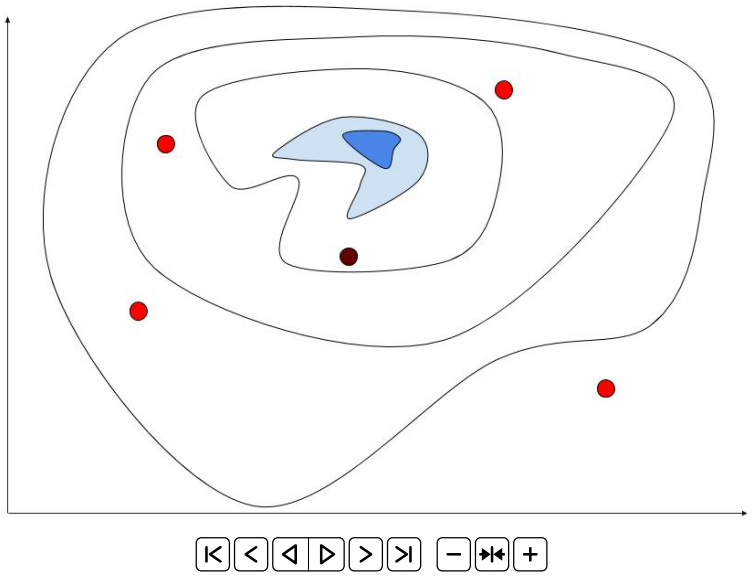


Figure 7: Animated representation of Differential Evolution. The best individual is in dark red. The new one at its various mutation steps is in light red.

```
input : N (population size), CP (number of parents),
        F (scaling factor), Cr (crossover probability)
Population = N random individuals
for g in 1:G do
    Base = Elite(Population, 1)
    for X in Population do
        Parent_1, Parent_2, ... Parent_2CP = sample(Population, 2CP)
        delta = sum_{i=1}^{CP} Parent_i - Parent_{CP+i}
        Mutant = Base + F · delta
        for x_i in X, m_i in Mutant do
            if Uniform(0,1) < Cr then
                m_i = x_i
            end
        end
        if fitness(Mutant) > fitness then
            newPopulation.add(Mutant)
        else
            newPopulation.add(X)
        end
    end
    Population = newPopulation
end
```

Algorithm 3: Differential Evolution base algorithm pseudocode

An important factor in evolutionary algorithms is the ability to self-regulate the mutation scale, so that it might explore or exploit when needed, possibly also for each gene independently. Differential evolution grants this by design, since its mutation is strongly related to variance in population. The more scattered the individuals, the more exploratory the differential mutation, with higher absolute values. If instead the population starts to cluster around some local maxima this mutation will be small, and permit an exploitative mutation of the landscape around this established optima, while still having the ability to explore when the "parents" randomly selected are from two different clusters.

This is the only evolutionary method used in this thesis with this self-adaptability to the explored landscape. For this reason it was decided as the main implementation, together with positive results in the preliminary grid search which showed it was comparable or superior in performance to both other implementations on Spiking Neural Networks optimization.

3.2 Importance of the fitness function definition

The choice of the fitness function is of the utmost importance when developing evolutionary algorithms, because every opportunity to raise the fitness is taken, sometimes even with unpredictable behaviors as consequences.

A typical and intuitive example is to set a fitness function to train a vehicle. Fitness goes up with distance from the starting point and is coupled with a efficiency incentive that lowers the fitness for each action, a cost function. If this cost function is badly weighted then the optimal solution is to stay still, not performing any actions so that the fitness will remain null instead of going in the negative with the first, uncoordinated and useless, movements. This is a case of local maximum.

Another problem is the one of exploitative behavior, in which the agent develops a strategy which is parallel to the one imagined still getting a high reward. This is observed, for example, in [41], where an agent had to run on a circuit and was awarded by the each movement along the circuit and penalized each time it went out of it. The circuit had an 8 shape and the best solutions avoided the curves, too complex to learn and slow to ride on, and only learned to go back and forth on the central straight lines.

From an evolutionary point of view these are very logical adaptations, and the fault resides in the incomplete or badly balanced fitness function. In the former example the problem can be avoided with a two-step training which first rewards distance covered and in a second iteration introduces the cost function, when the population of agents is already apt at covering distances and efficiency becomes the target. The latter occurrence was solved by introducing a condition to the fitness function which rewarded movements on the circuit only if that part of the circuit was not already being traversed before [41]. This forced the agents to leave the straight line and, eventually, to try the bended sides of the circuit.

In this thesis another example of exploitation was also observed. The fitness function of the rover rewards moving fast in a direction and penalizes the use of energy and strong hits to the structure while landing. Furthermore, there was a stopping condition if the body touched the ground. This was implemented by setting a lower limit on the height of the central body, such that the reaction wheel would not touch the ground. What was not considered was that the height was calculated from the bottom of the robot mainframe and not from the center. The robot quickly learned to flip on its back and use the reaction wheel to run forward at speeds that were not possible using legged locomotion . This exploit was simply solved by raising the lower limit of the robot height and that behavior was made impossible.

3.2.1 Novelty Search

One of the solutions that has recently been used to avoid the stagnation of the population around local optima of the fitness function, and to contrast lazy exploitative behavior in favor of more useful solutions, is Novelty Search [23]. In fact it is also an important part of the NEAT approach, with the formation of species. This method rewards how different a new solution is with respect to the previous by adding to the fitness function, so that a group of variations of an atypical individual, distant from the ones around a local optimum could survive in a competitive environment.

The impact of this novelty can then be an absolute or a relative factor. It could be decided that the "weirdest" individual has to survive all the time, or that each individual fitness will be summed to the novelty after scaling it with some arbitrary weight, making it possible to have small groups of aberrant individuals. While this subpopulation of solutions explores its neighborhood, it is kept alive by this added novelty value. This extra reward reduces with density. Therefore, after enough exploration, the subpopulation will have to find a new optimum, so that the falling novelty reward will be compensated by a higher pure fitness to make these individuals survive. Otherwise the overpopulation will likely kill the whole subgroup in one go, as soon as the density is too much to make up for the lack of fitness.

Once again the question is not easily solved, as the definition of Novelty is not set in stone, especially with noisy systems like Neural Networks or structures like graphs or wavelets. Even once a distance is defined between individuals, the problem persists in the weight this should have against the fitness. For example, in the case of a direct encoding genome, should the novelty factor be in the genome or in the behavior? When using an indirect encoding, should the difference in the graph structure be prioritized, or the resulting parameters of the network or once again the behavior?

It appears very difficult to quantify differences in behavior, so the better option appears to be measuring the distance between parameters vectors.

Possible distances between vectors

- **Euclidean:** the most classical form of distance, but it is not optimal for this strategy, since when parameters become big the measure follows

$$\sqrt{\sum_i (x_i - y_i)^2}$$

- **Chi squared:** a weighted version of the Euclidean distance. It manages to maintain the distances less affected by the parameter scale

$$\sqrt{\sum_i \frac{(x_i - y_i)^2}{x_i + y_i}}$$

- **Cosine:** the weighting factor is brought to the extreme, as this measure can only take values between -1 (diametrically opposed) and 1 (same orientation). The absence of a measure of distance between magnitudes is its strength, but neural networks might not work the same with proportional weights, if they contain non-linearities different from ReLU

$$\frac{arcos(\frac{X^T Y}{||X|| \cdot ||Y||})}{\pi}$$

For a population of N individuals the number of distances are N(N-1), so this operation is in the order of O(N^2). Since it has to be run for every generation and all the direct encoding require a relatively big population size to challenge the problem dimensionality, this method could end up burning many resources for an unknown return.

Even if the computational cost was affordable, a weighted function should be applied to this measure of distance before summing it to the fitness of the individual. In regard of this function, the two main possibilities are using the lowest distance from another individual or the average of all of them, maybe scaled so that they end up between 0 and 1. Furthermore, another scaling factor has to be decided before applying this Novelty value to the fitness.

Taking in consideration all these open questions and discussions, it was decided to not include a Novelty Search algorithm between the techniques used for this work. This decision is taken without casting doubts on the efficacy on this method, but it would have required a long preliminary phase of experiments to define the best implementation.

4 Experiments

In the following chapter all the practical steps of the implementation are discussed. From the network size, to the hyperparameter choice and, finally, the practical results in both benchmarks and low-g environments.

4.1 Architectures

Multi-Layered Perceptron

The input is a concatenation of the past N observation, there are three hidden layers with dimensionality directly proportional to the input size, and the output layer is directly controlling the joints. All activation functions are hyperbolic tangents. For the experiments in this thesis four observations were used as input, following what was done already in [32] and [25].

Recurrent Neural Network

This version of a Recurrent Neural Network is used as a midway between the MLP and the SNN architectures, as it uses the Stratified Network shape but with perceptrons. It is very important to test this intermediate solution, to better understand which component gives a greater edge to the Spiking Neural Networks over the standard Multi-Layered Perceptron. There is an MLP working as an input preprocessor. Input and output of this network have the same size and a single hidden layer double that size. The activation functions of the hidden layer are ReLU and the ones of the output are hyperbolic tangents. The preprocessed input gets fed to the first recurrent layer, twice as big, which is connected bidirectionally to the output layer controlling the joints directly. The recurrent layers activations are hyperbolic tangents.

Spiking Neural Network

The Spiking Neural Network architecture is almost the same of the Recurrent Neural Network, but with Controller Model neurons instead of perceptrons.

An MLP preprocessor with input and output layer of the same size and a single hidden layer twice as big. The activations are ReLU for the hidden layer and tanh for the output one. The preprocessor is connected to the first of three layers of spiking neurons, the second and third of which are bidirectionally connected and recurrently connected, as seen in the Stratified Network paragraph. The activation function was used as surrogate of the spiking mechanism, so there is not one applied to the output of the spiking layers.

The output of the last spiking layer is stored in spike trains and the frequency of spike in the last N steps is multiplied by an evolved parameter and processed by a sine function. This value is used as an input for the robot joint controller.

4.2 Software and Hardware

Hardware All experiments were run on a single home owned Personal Computer, the specifications are as follow:

- **CPU** Intel 8700
- **RAM** 16 Gigabyte DDR4
- **GPU** Nvidia GTX 1080

Since Evolutionary Algorithms don't use backpropagation all the computations are done by the CPU and the GPU is only used by MuJoCo during the rendered simulations.

Software MuJoCo is used to simulate the environment and the robot movements, with the help of the OpenAI gym framework. All the code is written in Python 3.6 and uses the *pymp* package to run the training simulations on more cores in parallel. Everything is run on 8 parallel threads.

4.3 Hyperparameter Optimization

To find optimal parameters for the evolutionary methods a grid search was used, evaluating the best fitness obtained by the algorithm training the same MLP architecture population on the Ant-v2 Gym environment.

Since it was observed in previous tests how most of the growth in fitness happened during the first 50 generations, this was selected as the number of iterations for each combination of parameters. The 12 parameter tuples that produced the best individuals were then plotted and evaluated. Handmade pattern recognition was used to find the best values for each value, singularly or in combinations. Or at least to identify the worst values and avoid them.

Following are shown the various sets explored with the grid search: Tables 2, 3 and 4. In Appendix A the plots can be found and reasoning behind the choice of the optimal parameters.

Genetic Algorithms

Sigma	0.01, 0.02, **0.05**, 0.1
Number of Elites	**1**, 3, 5, 10
Parent selection	**Rank based**, Uniform
Parent ranking	**Fitness proportional**, Ledger ranking

Table 2: All values in the grid search to tune the Genetic Algorithm on the Ant-v2 environment. In bold are highlighted the chosen ones. When Parent selection was set to Uniform, Parent ranking was skipped as it would have been uninfluential.

Natural Evolution Strategy

Sigma	0.01, 0.02, **0.05**, 0.1
Number of Samples	30, 50, **100**, 250
Learning Rate	0.005, **0.01**, 0.05, 0.1, 0.5

Table 3: Values tried for each parameter in the grid search to tune the Natural Evolutionary Strategy to the Ant-v2 environment. In bold are highlighted the chosen ones.

Differential Evolution

Scaling Factor	**0.1**, 0.5, 1, 2
Crossover Probability	0.1, 0.4, **0.9**
Couples of Parents	1, 2, **5**, 10, 25
Mutant Base	Random, **Best individual**

Table 4: Values tried for each parameter in the grid search on the Differential Evolution. In bold are highlighted the chosen ones.

4.4 Benchmarks

To test the effectiveness of the method it is run on benchmarks present in the OpenAI gym framework; specifically, on three taken from the MuJoCo environment and based on periodic movements, or gait. For these tests a population of 100 spiking neural networks is created and evolved with three different evolutionary algorithms. Since the Natural Evolutionary Strategy technique has a single main individual, the best element of the starting population is selected as the first generation individual.

All populations are trained for 4 epochs of 100 generations. New individuals are assigned a fitness which is the average of 15 trials in the simulation. Every 100 generations the whole population is tested with 30 simulations and assigned the average result as new fitness. In the case of the Natural Evolution Strategy the samples are evaluated with an average of 15 trials, but, since there is a single individual per generation, the computational cost of knowing its fitness more accurately is definitely inferior. For this reason, every generation the newly created individual is tested on 30 simulations.

After 400 generations, the best individual evolved with each method is tested for 100 simulations on the environment and its median fitness is reported. The sets of 100 results are tested in pairs with a t-Test to see whether the difference in the controllers fitness is statistically significant.

4.4.1 MuJoCo Environments

The goal of all these environments is to find the optimal trade-off between going fast and using as little energy as possible.

- **Swimmer** This environment is one of the simplest. Only two joints to control, no gravity and no stopping condition. The reward is given based on speed, and the cost is relative to the magnitude of the actions taken.
- **Half Cheetah** With six joints to control and no stopping condition, this bidimensional environment is harder and requires more complex periodical movements to maximize its fitness. The reward is given by the speed of the cheetah and the cost is proportional to the entity of the movement.
- **Ant** This simulation permits movement in all three dimensions and has stop conditions, it is a hard environment to master. With eight joints to control and the condition to not touch the ground or jump too high, the controller has to learn a steady but fast gait to optimize the returns. The reward is given by speed and the cost by the magnitude of the combined actions taken. Furthermore, a generous reward is given for each timestep the simulation runs. This is to help with enforcing the avoidance of the stopping condition.

4.4.2 Results

Environment	GA	NES	DE
Swimmer-v2	51.8	30.6	**57.4** [8]
HalfCheetah-v2	**180.6** [9]	-4.9	47.4
Ant-v2	**1213.8** [10]	1160.5	1094.4

Table 5: Results on the benchmark environments of the best SNN individual mutated after 400 generations of each Evolutionary Method. The best performance is highlighted in bold, in all cases the difference in fitness between individuals is significant (Welch's t-Test $p < 0.01$). The best results are shown in 8, 9 and 10. Bigger animations are shown in Appendix D.

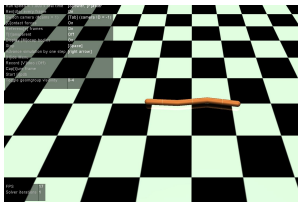


Figure 8: Swimmer benchmark. SNN evolved with Differential Evolution.

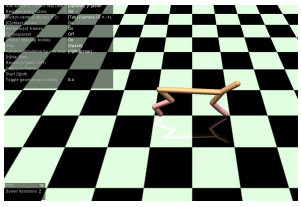


Figure 9: Half Cheetah benchmark. SNN evolved with Genetic Algorithms.

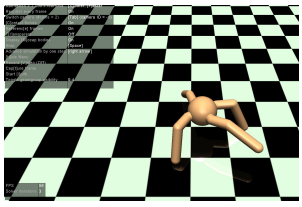


Figure 10: Ant benchmark. SNN evolved with Genetic Algorithms.

Swimmer

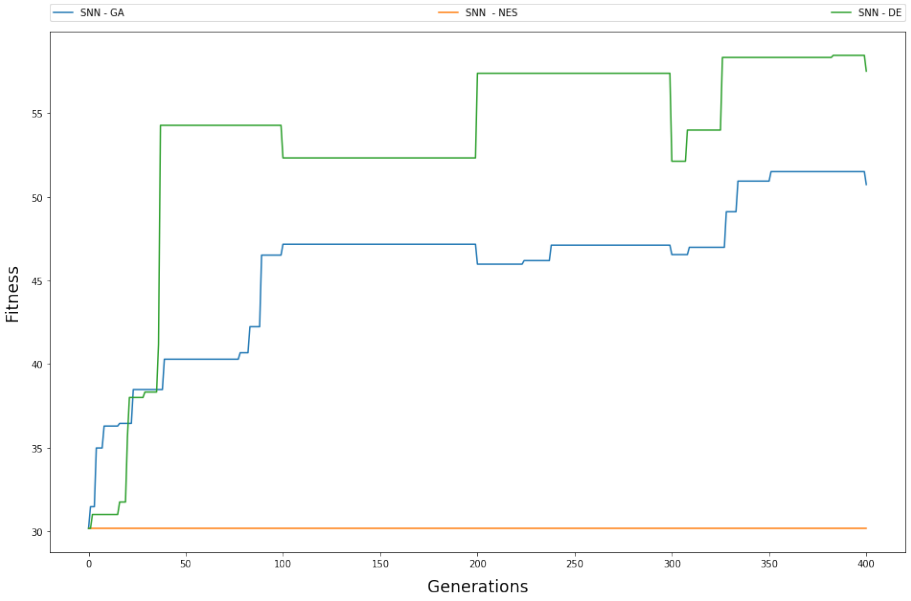


Figure 11: Best individual fitness on the Swimmer-v2 benchmark over generations.

Figure 11 shows the results of the best individual of each population over generations. The most notable detail is the lack of progress by the element evolved by the Natural Evolution Strategy. This might be due to the starting individual which was merely a local maximum and the mutation step of the algorithm was not enough to escape from it. About the individuals produced by the other methods, Differential Evolution seems to have an edge, but the distribution has long plateaus in which the best fitness does not change except at the end of an epoch. After 300 generations the situation unlocks and there is a raise in fitness that holds the last more consistent test. In spite of this higher variance it holds the best result after the first 50 generations. The Genetic Algorithm has very little drops during the more consistent fitness tests at epoch end, and presents a similar plateau between generations 100 and 350, before leaving it and growing again.

In the end the edge is still of the population optimized with the Differential Evolution method, even if the big gaps are a sign of higher inconsistency in behaviour.

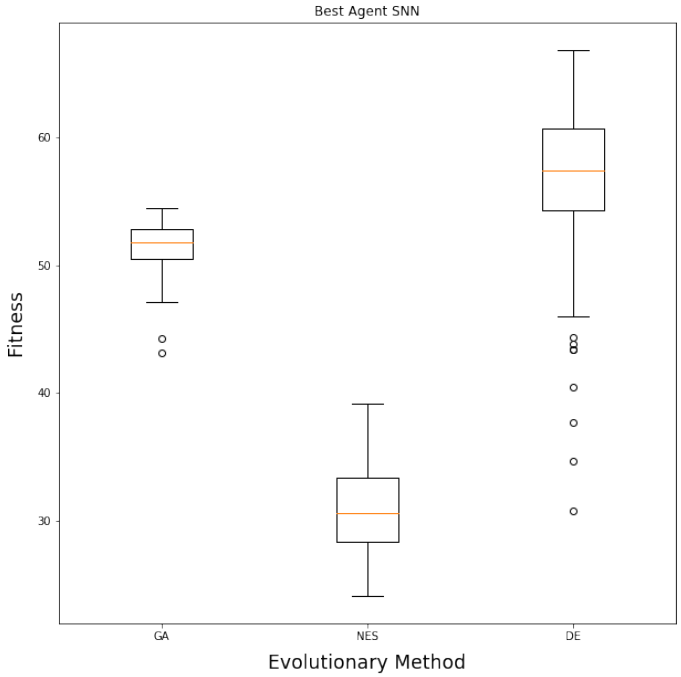


Figure 12: Comparison between the best SNN individuals fitness result distribution on the Swimmer-v2 benchmark, trained for 400 generations with each evolutionary method. Single results of 100 simulations.

As already intuited from the plots in 11, the box-plots in Figure 12 show an individual with a high variance in performance from the population optimized with Differential Evolution, while the one created by the Genetic Algorithm has the highest consistency. Nevertheless in more than half of the simulations the DE individual scored better than the best fitness the GA element reached.

The best element obtained from the Natural Evolution Strategy is actually the best individual produced by the random generation of the initial population. It is still useful to show, as it better remarks the notable improvement of both other evolutionary methods over the starting individuals.

Half Cheetah

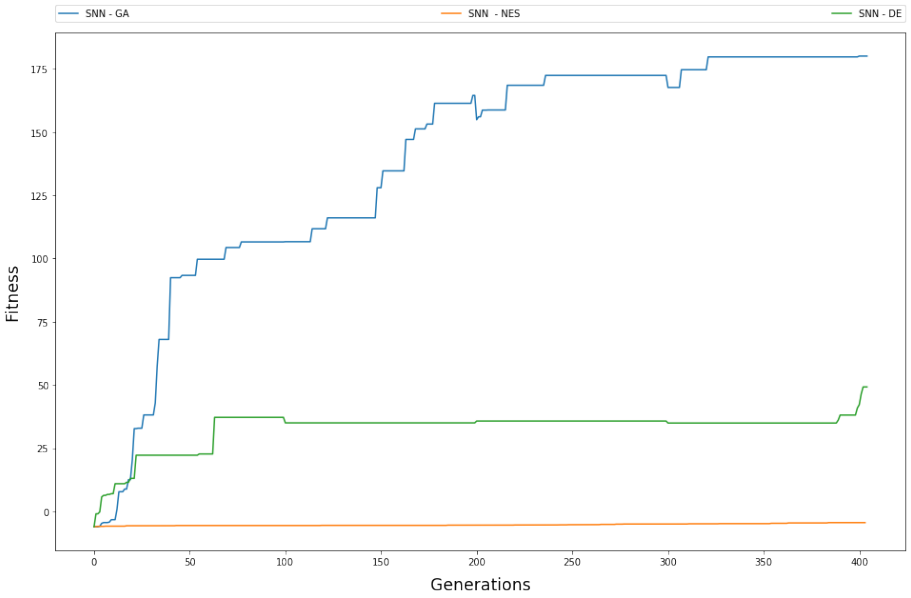


Figure 13: Best individual fitness on the HalfCheetah-v2 benchmark over generations.

From Figure 13 there is little doubt that the three methods have different effectiveness, with Genetic Algorithms taking the lead in a very convincing manner, followed by the Differential Evolution at less than a third of the former in final fitness and at last by Natural Evolution Strategy, without any serious gain from the starting best individual.

The Genetic Algorithm population produces a fast fitness growth in the first 50 generations, settles on a slower one up until about the 225th and then seems to reach a plateau with rare exploitations jumps. It appears as the Differential Evolution reaches a very lengthy plateau at a way lower fitness level, and abandons it just a few generations before the end of the last. In both cases it is interesting to notice that for the stronger consistency simulations at the end of the epochs the drops are very low, or even little gains. This shows that both methods produced an individual with very consistent behaviour, and consequent fitness results.

Natural Evolutionary Strategies fail this task with abysmal gains in fitness in the course of 400 generations. Possibly for a too small learning rate or mutation scale, alternatively because the best individual in the starting population was a lonely peak kind of local maximum.

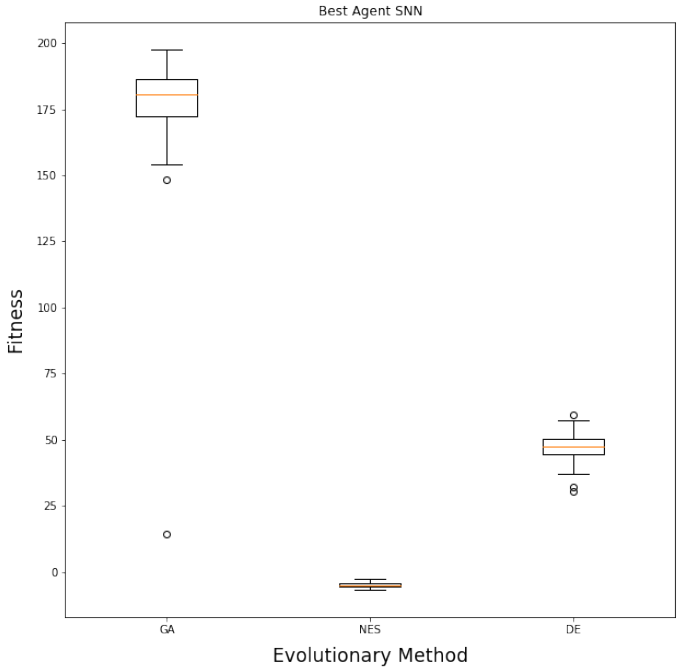


Figure 14: Comparison between the best SNN individuals fitness result distribution on the HalfCheetah-v2 benchmark, trained for 400 generations with each evolutionary method. Single results of 100 simulations.

In Figure 14 is the box-plot of the 100 simulations results for the best individual produced by each evolutionary method. As seen from Figure 13, the best individual is produced by Genetic Algorithms, with all simulations but a single low outlier scoring a fitness over 150 and an average of 180. The Differential Evolution individual follows with a more consistent series of simulation scores distributed symmetrically around almost 50. The individual evolved with the Natural Evolution Strategy is the lowest at scoring, with all simulations very close to -5. The fact that it does not even reach a positive result in all 100 simulations means that the low score is not caused by an unstable behaviour, but a consistently bad one.

In this case, even if the Differential Evolution has a higher consistency in results than Genetic Algorithms, the greater variation is explained by the higher score and there is no doubt that the latter method outputted the best individual.

Ant

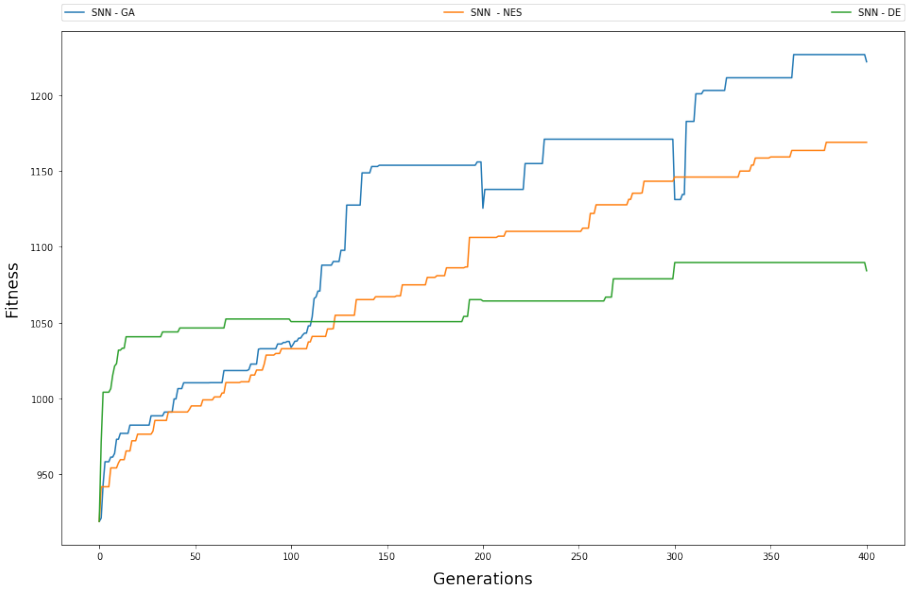


Figure 15: Best individual fitness on the Ant-v2 benchmark over generations.

In Figure 15 it is possible to observe the best fitness over time by evolutionary method. Differential Evolution is way faster in gaining fitness for the first 50 generations, after those its growth slows down substantially until the fitness obtained with the other methods gets better after the first 100 generations. The other populations keep gaining fitness at higher pace, with both apparently far from any asymptotic maximum.

The faster convergence can be explained by the self-adaptability of the differential mutation. This one, as explained in chapter 3, has a magnitude that variates with the variance in population. In fact the exploration/exploitation trade off was part of the reason why it is used in this thesis. This switches too early and converges around a good local optimum.

For the other two algorithm there is a fixed magnitude mutation step which is Gaussian based and explores only through the samplings from the tails of the distribution. Although this might make the movement to the optima too conservative or too jittery, in this case it appears to have the right sigma and it proceed in surpassing the differentially evolved individuals.

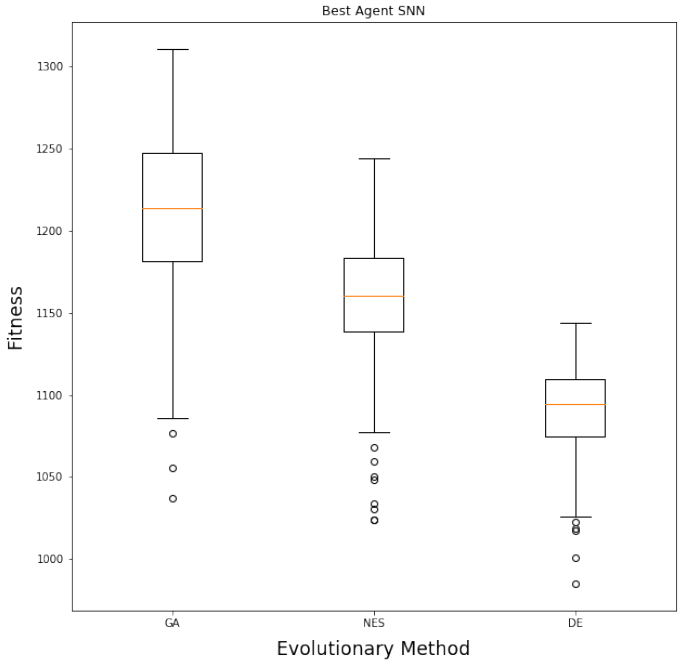


Figure 16: Comparison between the best SNN individuals fitness result distribution on the Ant-v2 benchmark, trained for 400 generations with each evolutionary method. Single results of 100 simulations.

In Figure 16 is the box-plot of the 100 simulations results for the best individual produced by each evolutionary method. The best scoring is the one evolved with Genetic Algorithms. The Natural Evolution Strategies individual follows with its third quartile slightly above the first of the former. The individual evolved with Differential Evolution is the lowest at scoring, but is also the most consistent, with a lower variance in the results.

4.5 Low-g Environments

For these experiments, each environment is created with the same rover shape but different gravity and different fitness functions. In lower gravity bodies, the cost of impact and energy used are decreased to represent the lesser effect of the mass.

Each simulation is run for 1000 steps and it is prematurely interrupted if the central body of the rover or its reaction wheel touch the ground, independently from the force of impact. For each individual, 15 simulations are run and its personal fitness is the average result of these. Every 100 iterations of the evolutionary algorithm, a stronger test is done on the whole population, which is tested on 30 simulations and a fitness equal to the new simulations results average is assigned to each individual. This is done for two reasons: avoiding stagnation around a lucky result and an eventual early stopping of the evolutionary algorithms, if this more confident fitness does not grow in the course of 100 generations.

For every architecture (MLP, RNN, SNN) a population is created with 100 individuals and evolved with two algorithms (Genetic Algorithm and Differential Evolution) for 500 generations. At the end of this training, the best individual of the population is tested for 100 simulations and its median fitness is shown as the final result. The whole set of 100 results in the simulation is compared with a t-Test to see if they belong to the same population or have a statistically significant difference.

4.5.1 Mars

Mars is the next destination for humanity. ESA, NASA, ISRO and SpaceX all have plans for a (manned or unmanned) trip to the red planet, and the recent discovery of an underground water lake [27] might push even more on the gas. Several rovers have already explored its surface, but all of them are designed with wheels and low elevation, this has caused problems in the past. For example, in 2005, Opportunity remained stuck in the sand with almost all wheels and risked to be permanently immobilized, before managing to get free after 7 weeks. A lucky escape, since the rover is still in function in 2018.

The rough terrains of mars could be explored by a legged rover with greater ease than with the current design, more prone to the sandy plains. On the other hand, with a gravity of $3.7 m/s^2$ gravity of Mars is still quite strong, and legged locomotion would consume a lot of energy. For this reason it is unlikely to observe the rover presented in this thesis move gracefully or jump. More likely it will behave similarly to ones evolved on the Ant environment, but with slower pace, so to not become unstable and tumble.

Results

	MLP 4obs	RNN	SNN
GA	-148.2	-72.8	573.5
DE	-141.8	-119.7	**574.5**

Table 6: Fitness obtained in the simulated environment of Mars after 500 generations of Differential Evolution by the three architectures. In bold the best score.

From Table 6 and Figure 17 it is apparent how the spiking controllers outperform the other architectures. Between the two spiking controllers there is very little difference in median, but the one optimized by Differential Evolution has lower variance. The evolutionary methods made a difference in optimizing the Recurrent Neural Networks, where the one evolved with Genetic Algorithms is superior in both median and consistency.

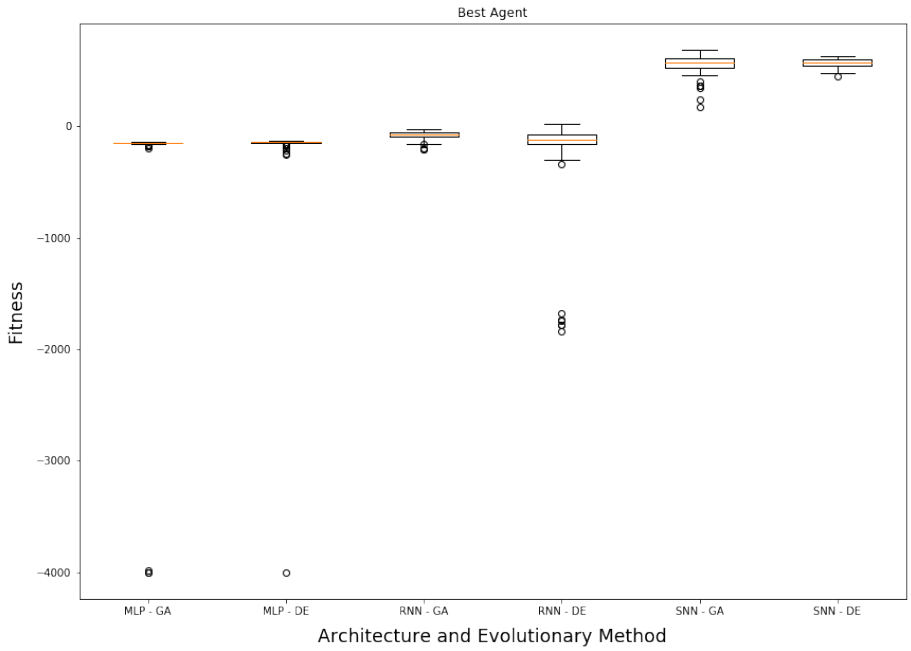


Figure 17: Distribution of results on the Mars gravity environment for each of the best individuals, by architecture and evolutionary method.

The Multi-Layered Perceptron controllers suffer of a strange effect. Their behaviour is surprisingly consistent for the vast majority of the simulations, except a few exception in which the final score is an abysmal -4000. Probably a

small percentage of the random starts causes a negative chain effect of actions that makes the controller perform worse and worse. It is important to notice how, even excluding these random initializations, the results of the MLP controllers are still the worst median-wise.

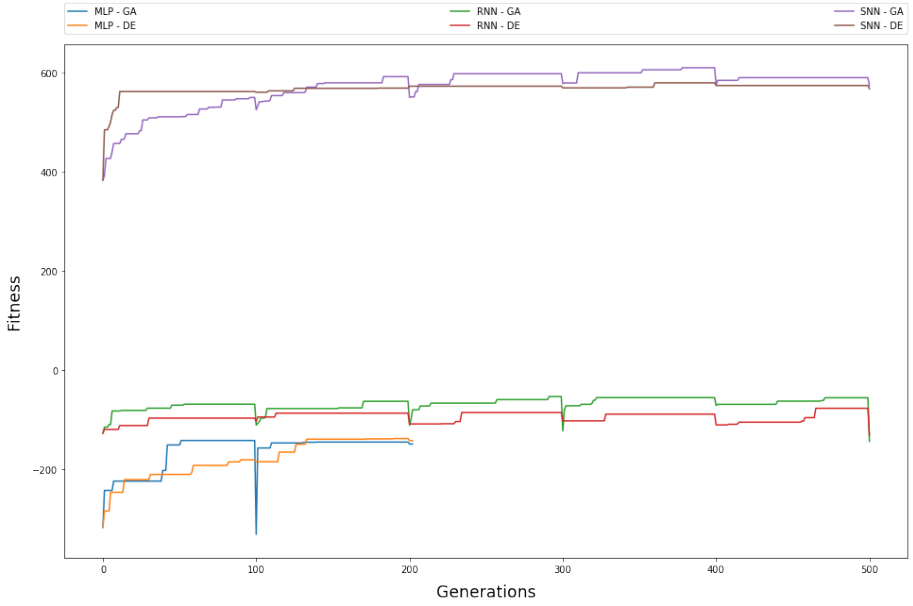


Figure 18: Fitness score of the best individual of each population over generations.

Figure 18 shows the trend of the best fitness obtained by each combination of architecture and evolutionary algorithm. The separation between Spiking Neural Networks and the others looks even starker than from Figure 17. What can be observed for all implementations is that after the first epoch the gain in fitness score is very little and all populations reach an asymptote. The recurrent controller does not actually gain any fitness for the whole five epochs. While both the MLP and the SNN do. Interesting how for the former the Genetic Algorithms have a faster convergence to the local maximum, while for the latter the Differential Evolution has an extremely fast gain in first 25 generations and then slows down substantially.

Especially for the MLP and SNN controllers, Differential Evolution appears to create individuals which are more consistent in their fitness scoring, as shown by the smaller drops in best fitness when tested more consistently every 100 iterations.

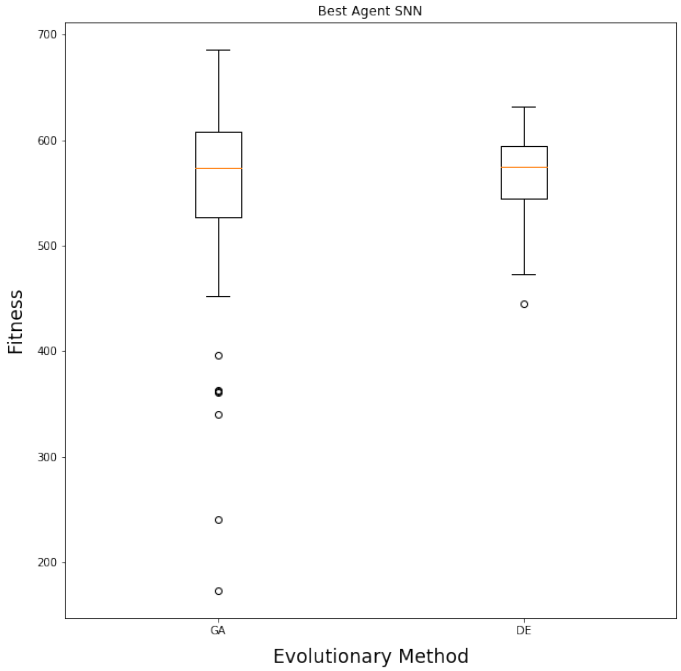


Figure 19: .

SNN	Mean	Median	Std Dev
GA	557.4	573.5	86.3
DE	**567.5**	**574.4**	**37.2**

Table 7: Statistics of the performance by the best individuals of the differently evolved SNN population on the Mars environment.

Table 7 contains the statistics of the distribution of results in the final test for the best spiking individual evolved with both Differential Evolution and Genetic Algorithms. The results are shown in two boxplots in Figure 19. What can be observed from these is how the individual evolved by Differential Evolution is more consistent in score, with less outliers in the lower side, but also a shorter tail in the better results.

It is important to notice how the distributions of results of the spiking controllers evolved with Genetic Algorithms and Differential Evolution are not statistically different from each other (Welch's t-Test $p > 0.05$).

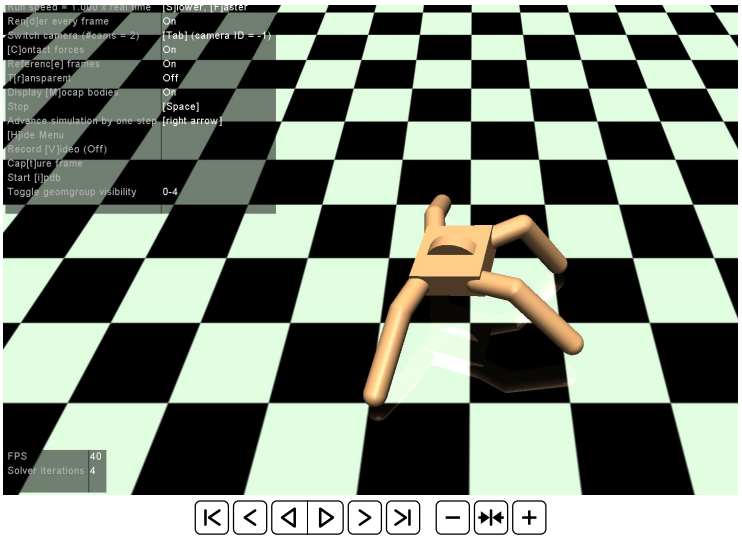


Figure 20: The controller evolved with Genetic Algorithms on the Mars environment

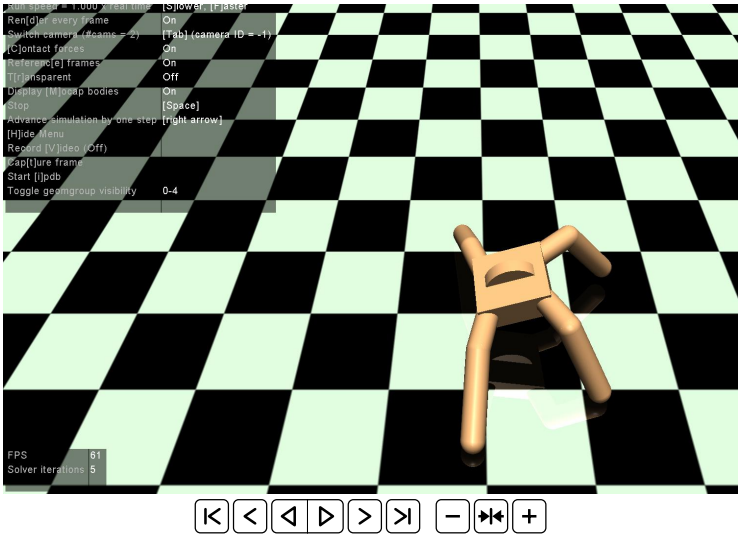


Figure 21: The controller evolved with Differential Evolution on the Mars environment.

4.5.2 Moon

Our own satellite has naturally been the first celestial body to be explored, first by humans and then by rovers. Since today only three rovers have landed on the lunar surface, the Soviet Lunokhods in the 70s and the Chinese Yutu in 2013.

Both designs belong to the typical six-wheeled rover. The Soviet rovers achieved impressive results with their design, with the Lunokhod2 holding the record for the longest drive on a celestial body up until Opportunity passed it 40 years later. On the other hand, the Chinese probe was stuck in sand after just 42 days, but actually kept working and sending data for a total of 31 months, showing how much more potential was in the mission.

With a legged design the more energy consumed and complex controller are heavily counterbalanced by the unlikeliness of getting immobilized by sand. Lunar gravity, set around $1.68 m/s^2$, should make jumping a reasonable choice of movement, as some of the astronauts actually tried to do, even with their big spacesuits on. It is expected for the controller to do small jumps, amortize the landing and follow with a moment of stabilization.

Results

	MLP 4obs	RNN	SNN
GA	-216.1	-95.8	499.3
DE	-186.7	-183.9	**578.8**

Table 8: Fitness obtained in the simulated environment of the Moon after 500 generations of Differential Evolution by the three architectures. In bold the best individual, the spiking architecture evolved by Differential Evolution.

From Table 8 and Figure 22 it can be seen that the separation between the Spiking Neural Network controllers and the perceptron based architectures is clear and significant, being the only ones that manage to obtain consistently positive results and with lower variance.

The other architectures score mainly between 0 and -500 with some heavy tail on the lower side. A small edge goes to the recurrent networks, especially to the one evolved with Genetic Algorithms, but the high variance and frequent outliers under a fitness of -500 make them incomparable to the spiking counterparts.

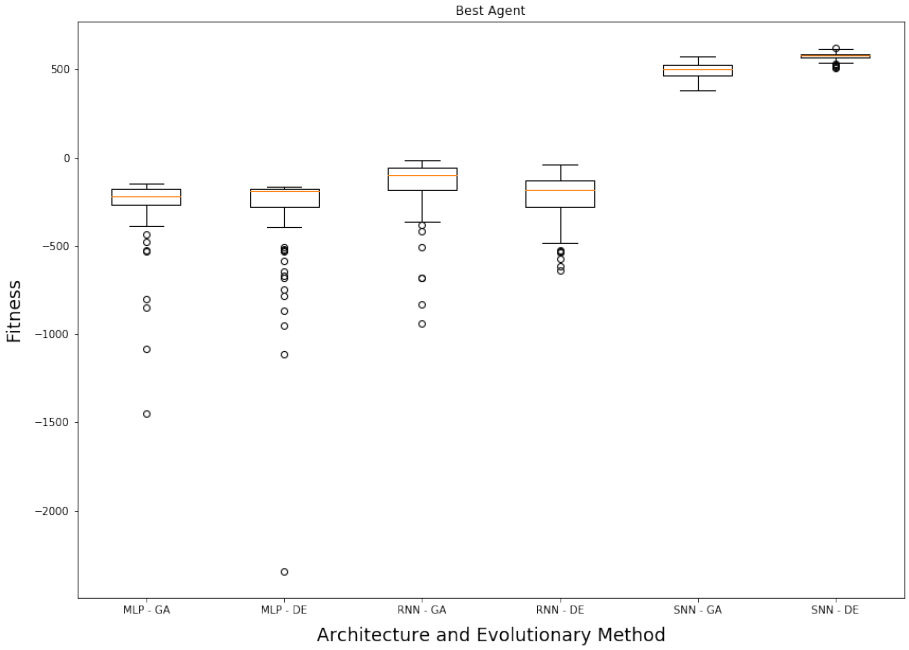


Figure 22: Comparison between the best individuals of each block of training. The box-plots represent the distribution of the fitness results for the 100 simulations run for each of these optimal agents.

The absolute worst result is from the Multi-Layered Perceptron controller evolved with Differential Evolution, scoring just above -2500 in fitness. Such outlier is a peculiar sight, but further samples have shown this happen with the same frequency of around 1%. The possible explanation is that one of the random initializations for the simulation, which happens roughly once every 100 tests, causes the controller to act in a very unrewarding manner, by creating a chain reaction of bad states and worse responses.

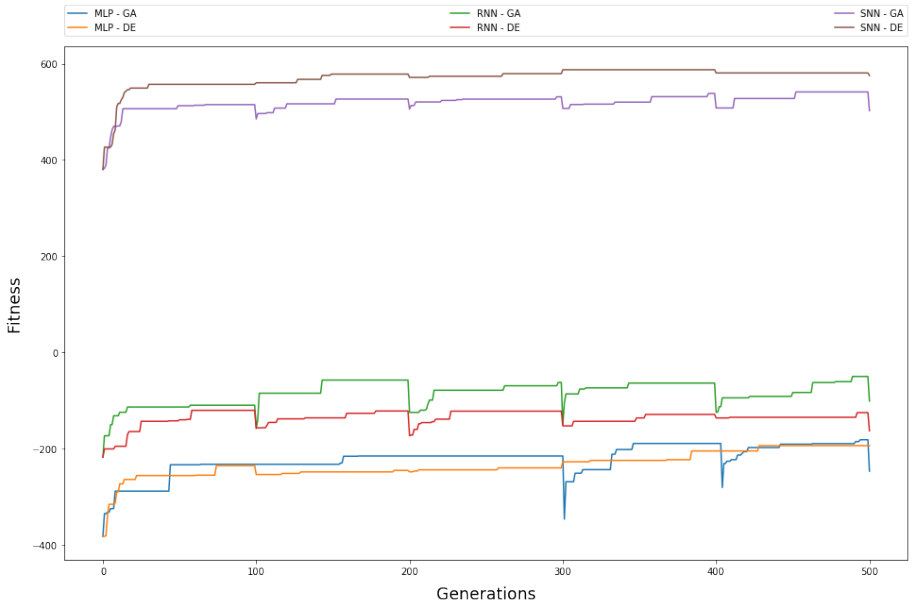


Figure 23: Fitness of the best individual on the Moon Environment over time, with all three architectures and two evolutionary methods each. The division is clear and once again spiking neural networks are substantially superior.

In Figure 23 are shown the average scores of the best individual of each population over the generations. In this case the three architectures appear well separated from the start to the end. Most of the gain happens in the first 100 or even 50 generations for all versions and a long plateau with little gains happen after the first epoch.

For all the architectures the Differential Evolution seems to make the best individual more consistent, as observed from the less drastic drops in fitness score at the end of the epochs. For the MLP-based population it even creates a better individual than Genetic Algorithms. Although the absolute result is not thrilling, one of the goals of this thesis was to show the validity of differential evolution as a Neural Network optimization mechanism, and the ability to produce this controller competitively to the already accepted Genetic Algorithms method is a good result.

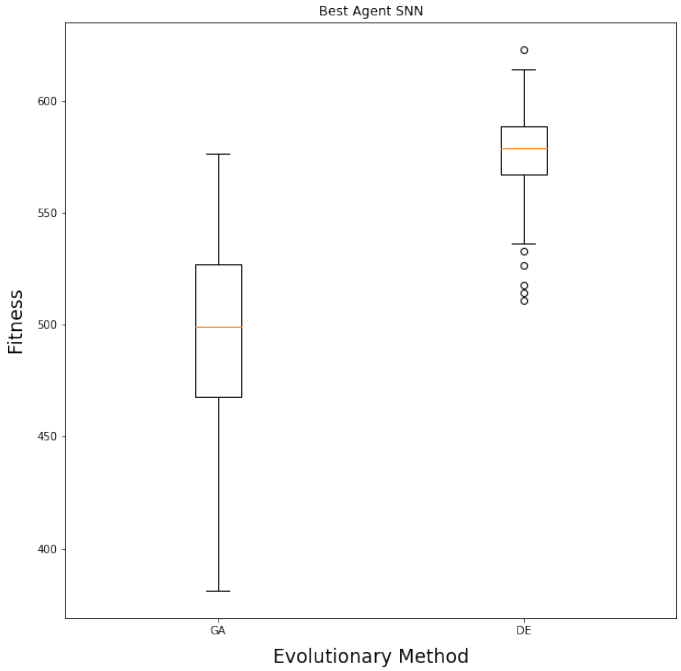


Figure 24: Comparison between the best SNN individuals fitness result on the Moon environment distribution, by evolutionary method.

As seen in Figure 24, and confirmed in Table 9, the individual optimized with Differential Evolution is better in both average and deviation. Both controllers have fairly symmetrical and Gaussian shaped distribution in scores. This is a good sign and indicates that regardless of the initialization of the simulation the rover behaves in a single way and ends up with a good fitness score. Once again, this is more true for the individual optimized with the Differential Evolution method.

	Mean	Median	Std Dev
GA	492.9	499.3	42.5
DE	**575.6**	**578.8**	**20.8**

Table 9: Statistics of the performance by the best individuals of the differently evolved SNN population.

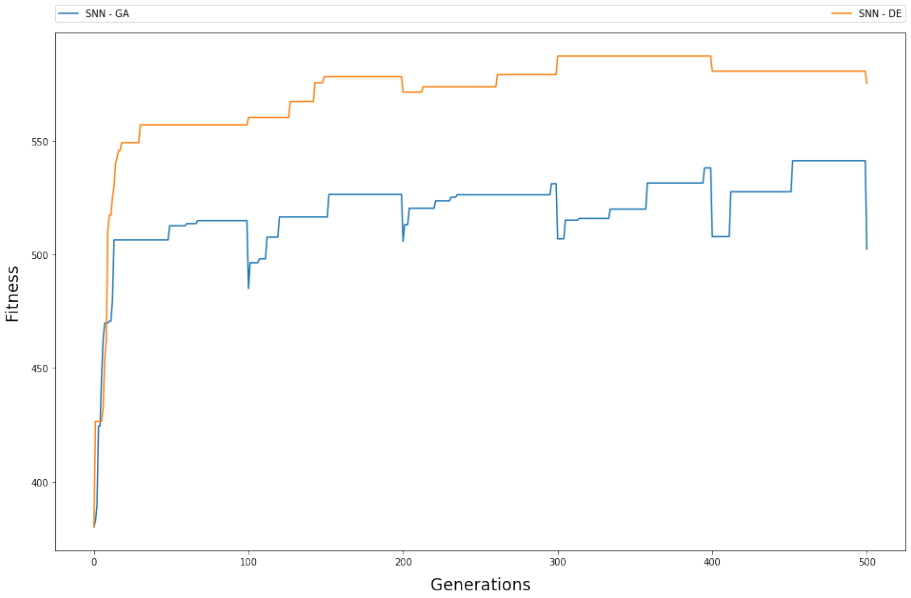


Figure 25: Detail of the Spiking Neural Network controller evolutions on the Moon environment over generations.

For the SNN controller Differential Evolution seems to have worked better as its score reach a plateau fixed at around 50 more than the individual produced by Genetic Algorithms, as seen in Figure 24. Furthermore, the drops in score at the end of the epochs are significantly more contained for this population (Figure 25).

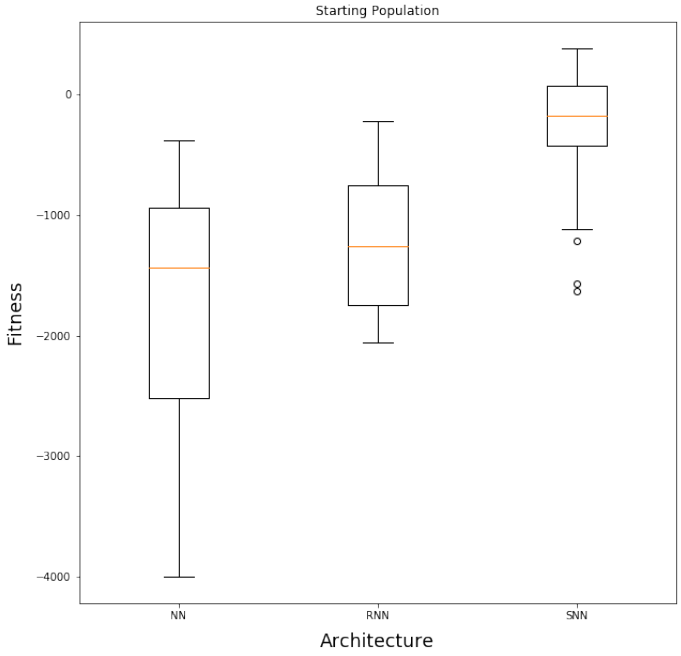


Figure 26: A comparison of the distribution of the individuals average fitness in the three randomly initialized populations. From the start it appears clear that the Spiking Neural Network structure is better, followed by the Recurrent and finally by the MLP.

To investigate the pure effect of the architecture, without the influence of the evolutionary methods, the starting populations are shown in Figure 26. That can be interpreted as a small random search, of 100 random individuals. Also in this case the distribution of average fitnesses are definitely different between architectures, with the Spiking Neural Networks having an edge over the others wit ha few individuals already scoring in positives.

This should dismiss the chance that the architecture and the optimization method are concurring in making the spiking controller a better solution, and instead prove how this model has an innate greater power in working with gait based behaviours.

4.5.3 Ceres

Ceres is the biggest and most massive celestial body in the Mars-Jupiter asteroid belt, comprising one third of the total mass. It is large enough to be considered a dwarf planet as its gravity, at 0.28 m/s^2, is sufficient to give it a round shape. It is composed by a rocky core surrounded by a 100km high water ice layer. This huge quantity of water is the main reason of interest for this body. There have been even hypothesis that life could have been possible in an underground ocean, although this is nowadays considered unlikely. An expedition on this dwarf planet would offer an answer and prepare the ground for possible future water extraction and rocket fuel production.

Since the gravity is so low it is expected that the rover proposed in this thesis should work better than a wheeled one and that the reaction wheel will help much with the long jumps resulting from even minimal energy applied by the legs.

Results

	MLP 4obs	RNN	SNN
GA	-182.9	-89.1	458.1
DE	-295.7	-129.9	**550.0**

Table 10: Fitness obtained in the simulated environment of Ceres after 500 generations of evolution by the three architectures. In bold the best result, obtained by the Spiking Neural Network Controller optimized with Differential Evolution.

From Table 10 the spiking controller is the only one that manges to turn the fitness function to a positive value. In the perceptron based architectures Genetic Algorithms manges to evolve a better individual, but in the end the best controller is produced by Differential Evolution applied to the Spiking Neural Network.

In Figure 27 the results of the best individuals evolved are shown side to side for comparison. For both average and deviance the spiking controllers appear significantly superior, while the Multi-Layered Perceptrons are the worse and the Recurrent Neural Networks lay in the middle, but with bad results. Specifically, the MLP individuals have most of their results between 0 and -1000, without a single positive value, but both have a group of outlier scores around -4000. This appear when the random start makes the controller jump at full strength in the wrong direction, and keeps flying towards worse and worse returns.

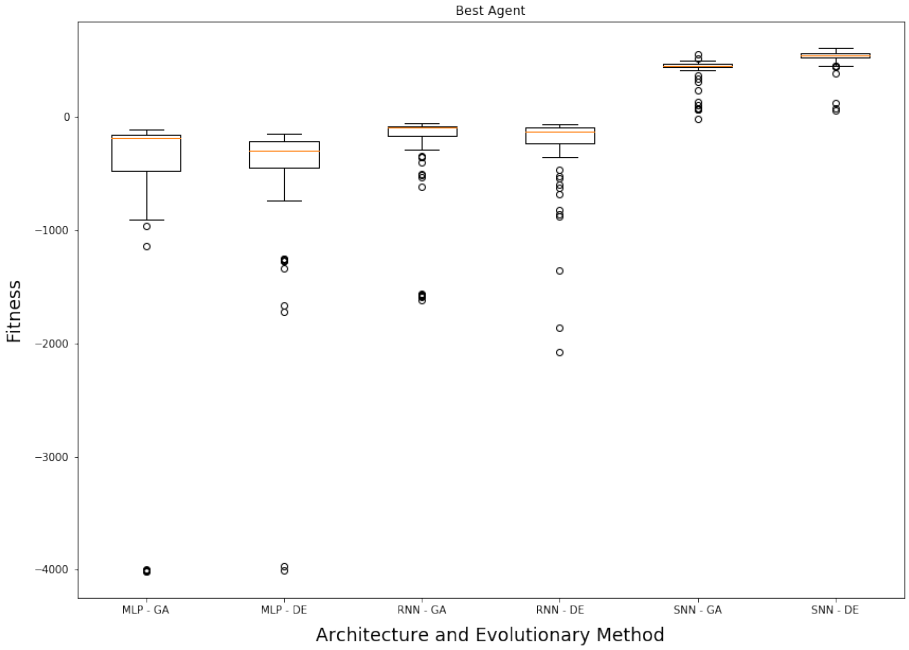


Figure 27: Comparison between the best individuals of each block of training. The boxplots represent the distribution of the fitness results for the 100 simulations run for each of these optimal agents.

In general the fitness distributions of all individual have significantly heavier tails in the lower score side. This probably means that the controller is not able to cope with some rarer random initialization of the simulation, and when it starts in those position it acts sub-optimally which creates a chain event of unseen or unsolved situations in which the controller cannot get a high fitness return. The only solution to this occurrence would be to optimize the networks on more simulations every step, but that would go against the premise of the thesis, of having good results with little computational power and time expenditure.

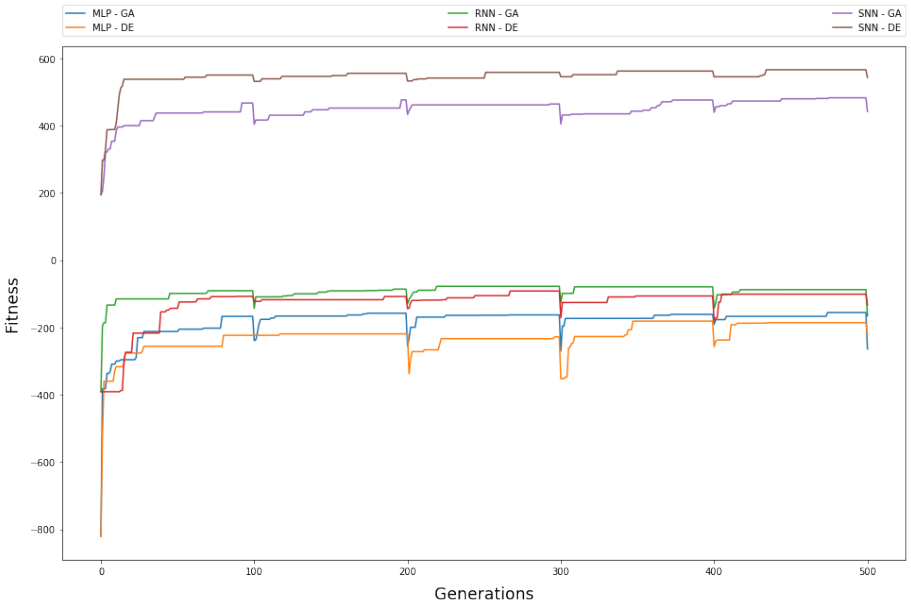


Figure 28: Fitness of the best individual on the Ceres environment over time, with all three architectures and two evolutionary methods each. The division is clear and once again spiking neural networks are substantially superior.

In Figure 28 are shown the trends of the best scoring individual by architecture and evolutionary method over generations. There is a clear distance in performance between the spiking controllers and the perceptron based, which appear all clumped up with a little edge for recurrent networks.

Most of the score growth happens in the first epoch and afterwards it is mostly little gains in fitness, which might be due to randomness of the less consistent fitness function used for the optimization.

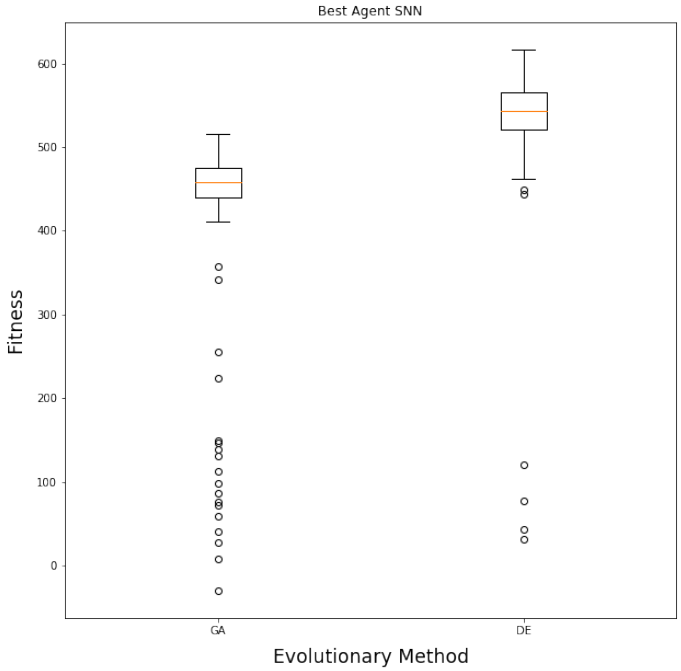


Figure 29: Comparison between the best SNN individuals fitness result distribution, by evolutionary method

	Mean	Median	Std Dev
GA	403.5	457.8	137.7
DE	**524.6**	**543.2**	**99.5**

Table 11: Statistics of the performance by the best individuals of the differently evolved SNN population.

As seen in Figure 29 and Table 11 not only Differential Evolution created a better individual, by mean and median (Welsh t-Test $p = 6.6e - 06$), but it has also more consistency in its behavior, with less variance in fitness results. It is important to notice how the median is higher than the mean in both cases, as the individuals tend to have only negative outliers.

It is interesting to notice that the individual evolved with Genetic Algorithms has many score outliers clustered around a fitness of 100. This seem to indicate a second, less likely behaviour followed by the controller that brings less efficient returns. It could be likely caused by a bad landing after a jump, as little difference in the starting position made the controller take flight in an unplanned way and unable to correctly compensate before landing roughly. Interestingly the one evolved by Differential Evolution appears to have the same problem but

with considerably inferior frequency. To what is due this consistency is very hard to find out, but the important information given by the data is that this controller is scoring better than the Genetic Algorithms one with and without considering these outliers.

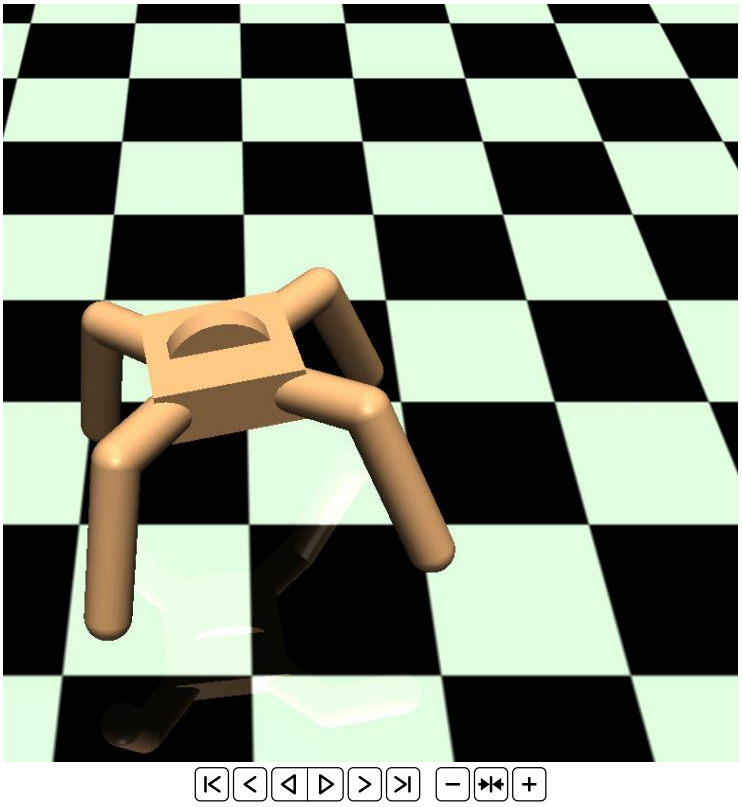


Figure 30: Animation of one of the uncommon instances in which the SNN evolved for the Ceres environments shows a proactive behaviour that exploits the low gravity. In this case the controller optimized with Genetic Algorithm moved similarly to a Golden Wheel Spider (Carparachne aureoflava).

This lower variability in behaviour is the reason that gives Differential Evolution an edge over the Genetic Algorithms. In each simulation, the controller acts similarly, so that a good score is actually indicative of a good controller and not caused by a lucky draw in the tail of the actual distribution of responses to the starting condition. Consequently, the evolution that rewards good fitness is actually promoting good controllers over mediocre ones.

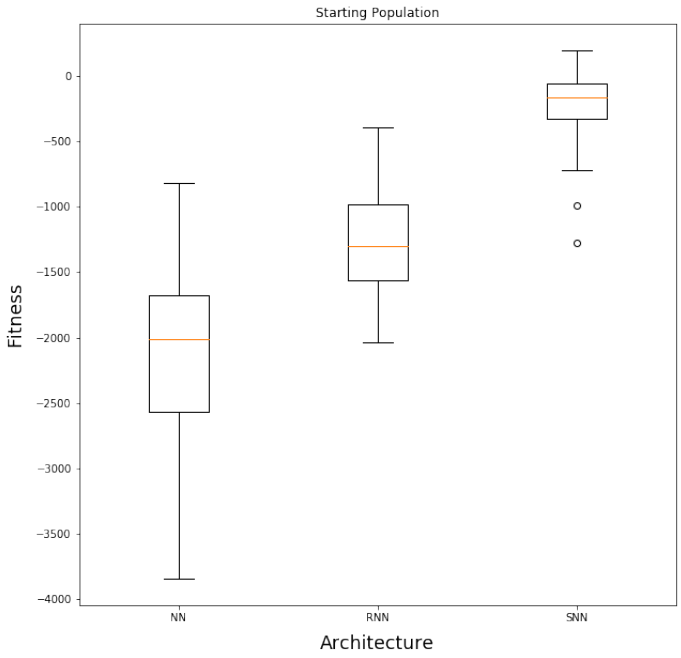


Figure 31: A comparison of the distribution of the individuals average fitness in the three randomly initialized populations. From the start it appears clear that the Spiking Neural Network structure is better, followed by the Recurrent and finally by the MLP.

To investigate the effect of random events in the evolution process which could have brought advantage to the SNNs, the distribution of fitness in the starting populations of each architecture is plotted in Figure 31. From this one it can be seen how already from the beginning the three sets of individuals are ordered clearly with the simpler architecture ranking last and the spiking one getting the best place. However, the distributions appear different but not so separate as the final individual end up being. This to show that the evolutionary methods are still having an important effect in the optimization over the random generation.

Implementing a different fitness function Upon visual investigation of the best individuals performances in the simulation, it was observed how the spiking controllers scored higher by exploiting the survival reward. The rover would slide lazily on the surface, using the minimum amount of energy possible. The other two architectures did not learn this exploit and were instead more proactive. As a result, they often crashed or moved in the wrong direction. To try and obtain a more interesting behaviour for the Spiking Neural Network controllers (along the line of 30), a different fitness function was defined and the three architectures were optimized again on it. Results are in Appendix B.

5 Discussion

5.1 Related work

This thesis does not exist in a vacuum, and many previous works have touched the same arguments. Without their mathematical models and results, this thesis would have been a much harder challenge. Two are the main topics of inspiration:

5.1.1 Optimizing Spiking Neural Networks

Berland in his thesis uses an Evolutionary Algorithm to optimize a Spiking Neural Network to command a simulated robot in a food chase, beating regular MLP and converging faster than other Spiking Neuron models [17]. In the conclusion of his work he leaves open the question of a more complex application of its neuron model. This thesis is, in a way, an answer to that question, and a positive one.

Other special Spiking Neural Networks were used as controller of robots and optimized with evolutionary algorithms. In [9] a robot structure similar to the one used by Berland receives a SNN controller based on Reservoir Computing ([34]) and evolved with satisfactory results. In [30] a special kind of Spiking Neuron is created to simulate the biological ones dedicated to the olfactory system, which is then used as the main part of a robot that acts in a simulated 3D environment to find the source of a chemical compound, by following its concentration in the air. In [13] the Spiking controller is used for a real robot in a real environment that learns to actuate its two wheels to avoid obstacles.

A version of backpropagation to train Spiking Neural Networks has been proposed in the past [21]. This implementation is used to optimize the connection weights between neurons and time delays between firing and spike emission. Since in this thesis (and the previous examples) the values to optimize include internal parameters and the spikes are treated as punctual digital output, this method would not be useful.

5.1.2 Training methods for controllers

The success of Google's DeepMind on MuJoCo benchmarks [26] [14] and GO [36] are based on Reinforcement Learning. This technique is applied to Neural Networks with the Actor-Critic model [19], in which a Critic entity is trained to evaluate the value of a couple State-Action (for example with direct supervised learning over observation of a random behaviour entity). At the same time, an Actor entity is optimized so that given an input it will output the action that the Critic considers best (for example through backpropagation over the Actor network, to maximize the output of the Critic). This method has shown great results in Perceptron based architectures, and does not necessarily require backpropagation to optimize the agents, but the use of Evolutionary Algorithms makes the training of the Critic superfluous, as stated in Chapter 2.

Several studies have been performed on alternative policy gradient techniques, in which the Actor is a stochastic entity that gets monotonically optimized. One of these methods, Trust Region Policy Optimization, has been applied to some of the MuJoCo benchmarks with good results [35]. Another method is to treat the set of Neural Networks as a Deep Dynamical Model and solving it so to optimize a robot movement [44].

Hinted in Chapter 3, indirect encoding is a promising tool to optimize Neural Networks in spatially related tasks [37]. HyperNEAT has been used with success in a similar task to evolve a controller for a quadrupedal robot [5], and it has been proven to beat other parameter optimization techniques [47]. In those experiments the architecture of the network was very simple, with a single hidden layer of the same size of the input and output, and the same evolved set of weights was used between the layers. The same process has been used to optimizing a controller in a simulate environment, before complete the evolution on a physical robot, saving much time with this bootstrap technique [22]. HyperNEAT, as any indirect encoding method, brings also several questions to the table, as it is based on spacial relationships which coordinates should the input values have [6]. Once this dependency to the geometry is taken in consideration and solved, it can actually become a strong feat of the method, that can generate versatile graphs capable of encoding the right controller to many diverse robots with the same body shape, but different proportions [31].

A study was done to test the effectiveness of hyperNEAT, proving that it does not scale well [41]. Specifically, its internal encoder graph quickly loses the ability to improve the final parameters in an efficient way, and it simply becomes larger and heavier. In the same study, another indirect encoding method was applied, based on wavelets, with results that surpass the hyperNEAT ones. This maintains the same geometric complication of finding the right coordinates for the input and output values, and even more for the hidden layers.

5.2 Results wrap up

5.2.1 Spiking Neural Networks controllers

The spiking architecture adapted to all the benchmarks, and at least one of the evolutionary methods gave good results. This shows potential for compact Spiking Neural Network controllers in simulated environments (like video games) or even in real life robotics, especially considering the limited hardware used for these experiments.

In addition, SNNs have a natural edge over perceptron based architectures on gait controllers, from their tendency to work with periodic behaviours and set internal frequencies to follow. Besides, the comparison between MLPs and RNNs has shown that recursiveness can slightly improve results, while most of the edge is given by the different model of neurons.

5.2.2 Differential Evolution for SNNs

Although the benchmarks showed mixed results, Differential Evolution was the better algorithm to optimize the spiking architecture in the main application of this thesis. Populations that were evolved with Differential Evolution are consistently superior to the ones optimized with Genetic Algorithms in all the low-g environment (Welch's t-test $p < 0.01$ in all cases).

The individuals created by Differential Evolution have lower variance in behaviour in all the celestial bodies, as well as in two of the benchmarks, with respect to Genetic algorithms. For environments in which mistakes are very costly, like robotics, some performance can be reasonably traded off for more consistency; even if the final goal is still to have both at the same time.

5.2.3 Low-g Environment results

On all simulations of the varying celestial bodies gravities, the Spiking architecture evolved with Differential Evolution created the best controller. On Mars the developed behaviour is more natural looking, with small steps dragging the rover on the surface. On the Moon the controller evolved a gait of short jumps to advance.

The results on Ceres are less impressive: SNNs evolved an overly cautious behaviour which slowly slides on the surface, helped by the abysmal gravity. At the same time Perceptrons evolved into a jumping rover that looked nicer, but scored worse on the fitness, for the excess of energy used and the likelihood of landing improperly and ending the simulation, with MLP showing a less spasmodic behaviour than RNNs that actually lowered its chance of landing safely.

5.3 Possible expansions

5.3.1 Ad-hoc fitness functions

From the experiments on the Ceres environment it was observed that a good result in the numbers is not necessarily connected to a desired behaviour. Most of the fault resides in the fitness function that was defined. An environment with such a low gravity did not need such a high reward for survival, like on the other two, because the control needed to not touch the ground with the main body was minimal. Of course changing the fitness function would have made the comparison between environments almost impossible.

In Appendix B there is a test of a different fitness function for the Ceres gravity simulation, which is built to incentivize more movement. This test is not brought to its final conclusion and many different changes could be applied to the original reward function and their effect studied.

5.3.2 Less greedy optimization

For many experiments (Figures 18, 23 and 28) it appears as the evolutionary algorithms very quickly reached a plateau, which is almost surely a local maximum. Although this is not necessarily a bad maximum, as seen from the simulations, the chance of having more exploratory algorithms performing better in time is to be taken in consideration.

Even if it succeeded in the grid search (see Appendix A), the use of rank-based sampling for the Genetic Algorithms and using the best individual as base for the Differential Evolution might have made these optimization methods more exploitative than needed. In a future work the author will certainly try to use the alternative, less greedy methods and compare the results.

Another option, already stated at the end of Chapter 3, is to apply a Novelty Search approach, with all the study work behind it necessary to tune it to the problem in question.

5.3.3 Evolution of the rover body

As suggested in previous studies (for example [4]), the application of Evolutionary Algorithms can go over the optimization of the controller, and include the actual physical characteristic of the robot. Since different gravities would logically favour different bodies, the idea of optimizing in steps body and controller of a rover is quite enticing. In this occasion, the fact that the Spiking Neuron model used in this thesis has shown to be light and quickly trainable would make this ordeal easier to overcome.

5.3.4 Indirect Encoding

Introduced as a concept in Chapter 3, the use of indirect encoding might create more natural controllers that use the symmetries and patterns in the robot shape connecting them with the values outputted by the optimal controller. Future research might focus on testing and eventually developing these techniques for Spiking Neural Networks, without the need to use comparative architectures as in this thesis.

The HyperNEAT method (which has been tested throughly for regular Multi-Layered Perceptrons) or the more novel wavelet based one are not immediately translatable to SNNs for the presence of internal parameters to evolve in the spiking neurons, in addition to the weights between layers. The same goes for the recurrent connections. Some ideas to adapt them were imagined for this work, but in the end were considered too time-consuming to study and, consequently, too experimental to use in the tests.

5.3.5 Other Spiking Neurons architectures

The combination of Stratified SNNs with Controller Model spiking neurons had good results on the benchmarks. All other architectures and models demonstrated in preliminary tests to be slower, but not all combinations were tried and the use of time-based of spikes or a differently shaped stratified network might make the controller powerful enough to work with a significantly inferior number of neurons and obtain a similarly good result without requiring more time for optimization and computations.

References

[1] H. Amin and R. Fujii, *Spike train decoding scheme for a spiking neural network*, in 2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541), vol. 1, July 2004, pp. 477–482.

[2] A. Berland and S. Dumoilin, *A cheap spiking neural model for evolved controllers*. ttps://dspace.library.uu.nl/handle/1874/337623, 2016.

[3] S. M. Bohte, J. N. Kok, and H. L. Poutré, *Error-backpropagation in temporally encoded networks of spiking neurons*, Neurocomputing, 48 (2002), pp. 17 – 37.

[4] N. Cheney, R. MacCurdy, J. Clune, and H. Lipson, *Unshackling evolution: Evolving soft robots with multiple materials and a powerful generative encoding*, GECCO '13, 2013, pp. 167–174.

[5] J. Clune, B. E. Beckmann, C. Ofria, and R. Pennock, *Evolving coordinated quadruped gaits with the hyperneat generative encoding*, 2009 IEEE Congress on Evolutionary Computation, CEC 2009, (2009), pp. 2764–2771.

[6] J. Clune, C. Ofria, and R. T. Pennock, *The sensitivity of hyperneat to different geometric representations of a problem*, in GECCO, 2009.

[7] M. P. Deisenroth, D. Fox, and C. E. Rasmussen, *Gaussian processes for data-efficient learning in robotics and control*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 37 (2015), pp. 408–423.

[8] M. Z. Donahue, *A new generation of interplanetary rovers is crawling toward the stars*. https://www.smithsonianmag.com/science-nature/new-generation-interplanetary-rovers-crawling-toward-stars-180962625/, 2017.

[9] E. Eskandari, A. Ahmadi, S. Gomar, M. Ahmadi, and M. Saif, *Evolving spiking neural networks of artificial creatures using genetic algorithm*, in 2016 International Joint Conference on Neural Networks (IJCNN), July 2016, pp. 411–418.

[10] R. FitzHugh, *Impulses and physiological states in theoretical models of nerve membrane*, Biophysical Journal, 1 (1961), pp. 445 – 466.

[11] J. Gauci and K. O. Stanley, *A case study on the critical role of geometric regularity in machine learning*, in Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 2, AAAI'08, AAAI Press, 2008, pp. 628–633.

[12] ——, *Autonomous evolution of topographic regularities in artificial neural networks*, Neural Comput., 22 (2010), pp. 1860–1898.

[13] H. Hagras, A. Pounds-Cornish, M. Colley, V. Callaghan, and G. Clarke, *Evolving spiking neural network controllers for autonomous robots*, in IEEE International Conference on Robotics and Automation, vol. 5, April 2004, pp. 4620–4626 Vol.5.

[14] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, A. Eslami, M. Riedmiller, et al., *Emergence of locomotion behaviours in rich environments*, arXiv preprint arXiv:1707.02286, (2017).

[15] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, Neural Computation, 9 (1997), pp. 1735–1780.

[16] A. L. Hodgkin and A. F. Huxley, *A quantitative description of membrane current and its application to conduction and excitation in nerve*, Bulletin of Mathematical Biology, 52 (1990), pp. 25–71.

[17] E. M. Izhikevich, *Simple model of spiking neurons*, IEEE Transactions on Neural Networks, 14 (2003), pp. 1569–1572.

[18] JAXA. `http://www.hayabusa2.jaxa.jp/en/topics/20180922e/`, 2018.

[19] V. R. Konda and J. N. Tsitsiklis, *Actor-critic algorithms*, in Advances in neural information processing systems, 2000, pp. 1008–1014.

[20] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, 86 (1998), pp. 2278–2324.

[21] J. H. Lee, T. Delbruck, and M. Pfeiffer, *Training deep spiking neural networks using backpropagation*, Frontiers in Neuroscience, 10 (2016), p. 508.

[22] S. Lee, J. Yosinski, K. Glette, H. Lipson, and J. Clune, *Evolving gaits for physical robots with the hyperneat generative encoding: The benefits of simulation*, in Applications of Evolutionary Computation, A. I. Esparcia-Alcázar, ed., 2013, pp. 540–549.

[23] J. Lehman and K. O. Stanley, *Abandoning objectives: Evolution through the search for novelty alone*, Evolutionary computation, 19 (2011), pp. 189–223.

[24] W. Maass, *Networks of spiking neurons: The third generation of neural network models*, Neural Networks, 10 (1997), pp. 1659 – 1671.

[25] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, et al., *Playing atari with deep reinforcement learning*, arXiv preprint arXiv:1312.5602, (2013).

[26] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, et al., *Human-level control through deep reinforcement learning*, Nature, 518 (2015), p. 529.

[27] R. Orosei, S. E. Lauro, E. Pettinelli, A. Cicchetti, M. Coradini, B. Cosciotti, et al., *Radar evidence of subglacial liquid water on mars*, Science, (2018).

[28] M. E. H. Pedersen, *Good parameters for differential evolution*, Magnus Erik Hvass Pedersen, 49 (2010).

[29] M. T. Pope, S. Christensen, D. Christensen, A. Simeonov, G. Imahara, and G. Niemeyer, *Stickman: Towards a human scale acrobatic robot*, IEEE International Conference on Robotics and Automation (ICRA), (2018).

[30] P. Rhodes and T. Anderson, *Evolving a neural olfactorimotor system in virtual and real olfactory environments*, Frontiers in Neuroengineering, 5 (2012), p. 22.

[31] S. Risi and K. O. Stanley, *Confronting the challenge of learning a flexible neural controller for a diversity of morphologies*, in Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, GECCO '13, New York, NY, USA, 2013, ACM, pp. 255–262.

[32] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, *Evolution strategies as a scalable alternative to reinforcement learning*, arXiv preprint arXiv:1703.03864, (2017).

[33] J. D. Schaffer, *Evolving spiking neural networks: A novel growth algorithm corrects the teacher*, in 2015 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA), May 2015, pp. 1–8.

[34] B. Schrauwen, D. Verstraeten, and J. Van Campenhout, *An overview of reservoir computing: theory, applications and implementations*, in Proceedings of the 15th European Symposium on Artificial Neural Networks. p. 471-482 2007, 2007, pp. 471–482.

[35] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, *Trust region policy optimization*, in International Conference on Machine Learning, 2015, pp. 1889–1897.

[36] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, et al., *Mastering the game of go with deep neural networks and tree search*, nature, 529 (2016), p. 484.

[37] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, *A hypercube-based encoding for evolving large-scale neural networks*, Artificial Life, 15 (2009), pp. 185–212.

[38] R. Storn and K. Price, *Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces*, 23 (1995).

[39] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, *Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning*, CoRR, abs/1712.06567 (2017).

[40] D. Tanneberg, A. Paraschos, J. Peters, and E. Rueckert, *Deep spiking networks for model-based planning in humanoids*, in 2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids), Nov 2016, pp. 656–661.

[41] T. G. van den Berg and S. Whiteson, *Critical factors in the performance of hyperneat*, in Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, GECCO '13, ACM, 2013, pp. 759–766.

[42] C. Van Der Malsburg, *Frank rosenblatt: Principles of neurodynamics: Perceptrons and the theory of brain mechanisms*, in Brain Theory, G. Palm and A. Aertsen, eds., Berlin, Heidelberg, 1986, Springer Berlin Heidelberg, pp. 245–248.

[43] A. Vandesompele, F. Walter, and F. Röhrbein, *Neuro-evolution of spiking neural networks on spinnaker neuromorphic hardware*, in 2016 IEEE Symposium Series on Computational Intelligence (SSCI), Dec 2016, pp. 1–6.

[44] N. Wahlström, T. B. Schön, and M. P. Deisenroth, *From pixels to torques: Policy learning with deep dynamical models*, arXiv preprint arXiv:1502.02251, (2015).

[45] L. Wang, Y. Zeng, and T. Chen, *Back propagation neural network with adaptive differential evolution algorithm for time series forecasting*, Expert Systems with Applications, 42 (2015), pp. 855 – 863.

[46] S. Whiteson, *Evolutionary Computation for Reinforcement Learning*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 325–355.

[47] J. Yosinski, J. Clune, D. Hidalgo, S. Nguyen, J. C. Zagal, and H. Lipson, *Evolving robot gaits in hardware: the hyperneat generative encoding vs. parameter optimization*, in in Proceedings of the 20th European Conference on Artificial Life, 2011, pp. 890–897.

[48] D. Zambrano, R. Nusselder, H. S. Scholte, and S. M. Bohte, *Efficient computation in adaptive artificial spiking neural networks*, CoRR, abs/1710.04838 (2017).

[49] I. Zamora, N. G. Lopez, V. M. Vilches, and A. H. Cordero, *Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo*, arXiv preprint arXiv:1608.05742, (2016).

APPENDICES

A Grid Search in-depth results

In this appendix are presented more in depth the results of the grid search to find the hyperparameters for the evolutionary methods. Plots of the score over time obtained with the various methods are shown and a brief comment about each parameters is given to motivate the choices.

A.1 Genetic Algorithms

Sigma	0.01, 0.02, 0.05, 0.1
Number of Elites	1, 3, 5, 10
Parent selection	Rank based, Uniform
Parent ranking	Fitness proportional, Ledger ranking

Table 12: All values in the grid search to tune the Genetic Algorithm on the Ant-v2 environment. When Parent selection was set to Uniform, Parent ranking was skipped as it would have been uninfluential.

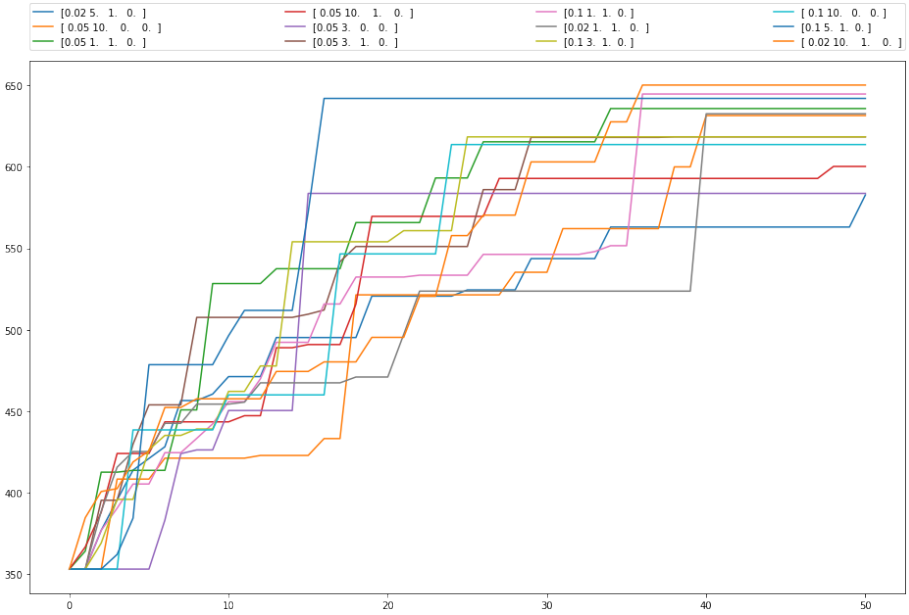


Figure 32: Plot of the 12 combinations of parameters for GA that made the best performing individuals

- Sigma is quite variegated, in the top 12 the most common values are 0.1 or 0.05. Since a slower convergence is better than an approximate one, especially with noisy parameter landscapes like in neural networks, the lower value was selected.
- Regarding the Parent Selection there is little doubt about the effectiveness of sampling with probability proportional to the individual fitness over a uniform sampling.
- In the opposite way, the use of ledger rank is almost surely the worse way of weighting the parents for the sampling
- As for the number of elites to keep each generation there does not seem to be a strong favourite, with 1 being the relatively better choice. Joining this with the original implementation by [39] a single elite individual will be kept at every iteration.

A.2 Natural Evolution Strategy

Sigma	0.01, 0.02, 0.05, 0.1
Number of Samples	30, 50, 100, 250
Learning Rate	0.005, 0.01, 0.05, 0.1, 0.5

Table 13: Values tried for each parameter in the grid search to tune the Natural Evolutionary Strategy to the Ant-v2 environment.

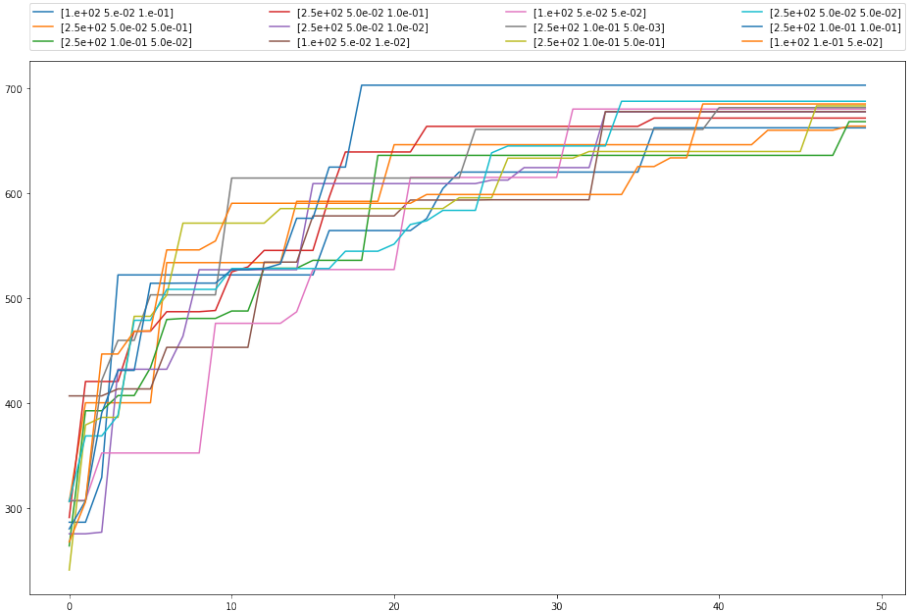


Figure 33: Plot of the 12 combinations of parameters for NES that made the best performing individuals

- Intuitively, the higher the Number of Samples for each generation, the better the result. This, however, seriously increments the computational time and makes the comparison somewhat unfair with respect to the other methods, which can try at most 100 new individuals every generation. For these reasons it was decided to set it to 100 symmetric samples per iteration in future implementations.
- The best Sigma is between 0.1 and 0.05. These values encourage exploration more than exploitation, and to better balance the two strategies, the lower value was chosen.
- From these tests the best Learning Rate seems to be 0.01, and generally lower than Sigma. This value will be used in the future implementations.

A.3 Differential Evolution

Scaling Factor	0.1, 0.5, 1, 2
Crossover Probability	0.1, 0.4, 0.9
Couples of Parents	1, 2, 5, 10, 25
Mutant Base	Random, Best individual

Table 14: Values tried for each parameter in the grid search on the Differential Evolution.

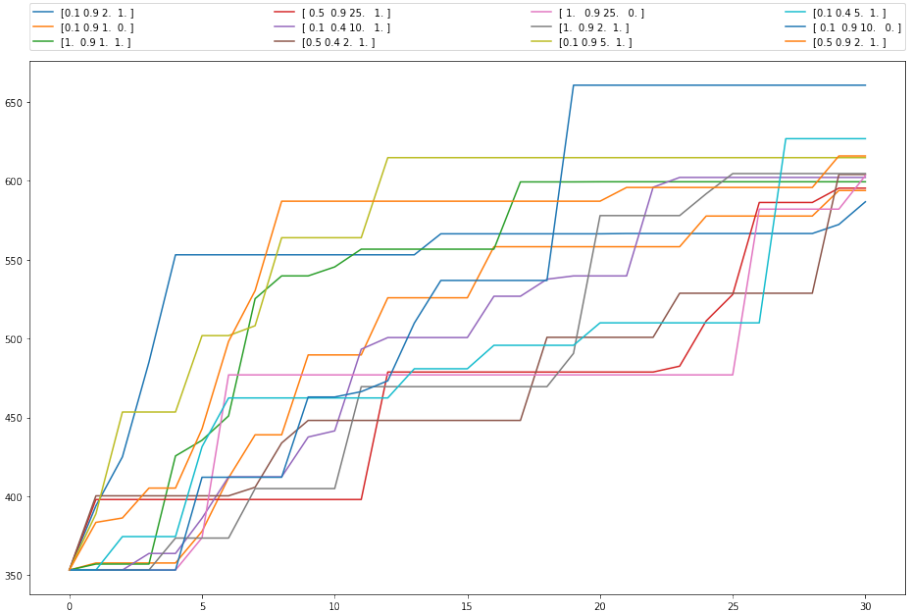


Figure 34: Plot of the 12 combinations of parameters for DE that made the best performing individuals

- The best value for the Crossover Probability seems to be 0.9, keeping on average 90% of the mutant genome.
- Half of the top 12 best individuals was evolved with a Scaling Factor of 0.1, it is then considered the optimal between the tested values.
- Having the best individual of the population as the base for the mutant genome seems better.
- Curiously the number of parent couples does not seem to influence much the results, as both the minimum and maximum values are present in

the top 12. Shrinking the podium, half of the top 4 used 5 pairs of individuals as parents, for this reason it was selected in future implementations.

B Different fitness function for Ceres

In this appendix are given the result of a brief exploratory experiment in which the fitness function is tuned to incentive more movement out of the controllers on the Ceres environment. Spiking controllers showed a good solution in terms of score, but not as desired behaviour, as they were able to survive with little adjustments of the engines so to never touch ground, but never advance either. The new fitness function severely reduces the cost of movement that was hypothesized was holding the SNN controllers from evolving a gait. Following are the plots and analysis of the result over a brief test of 100 generations.

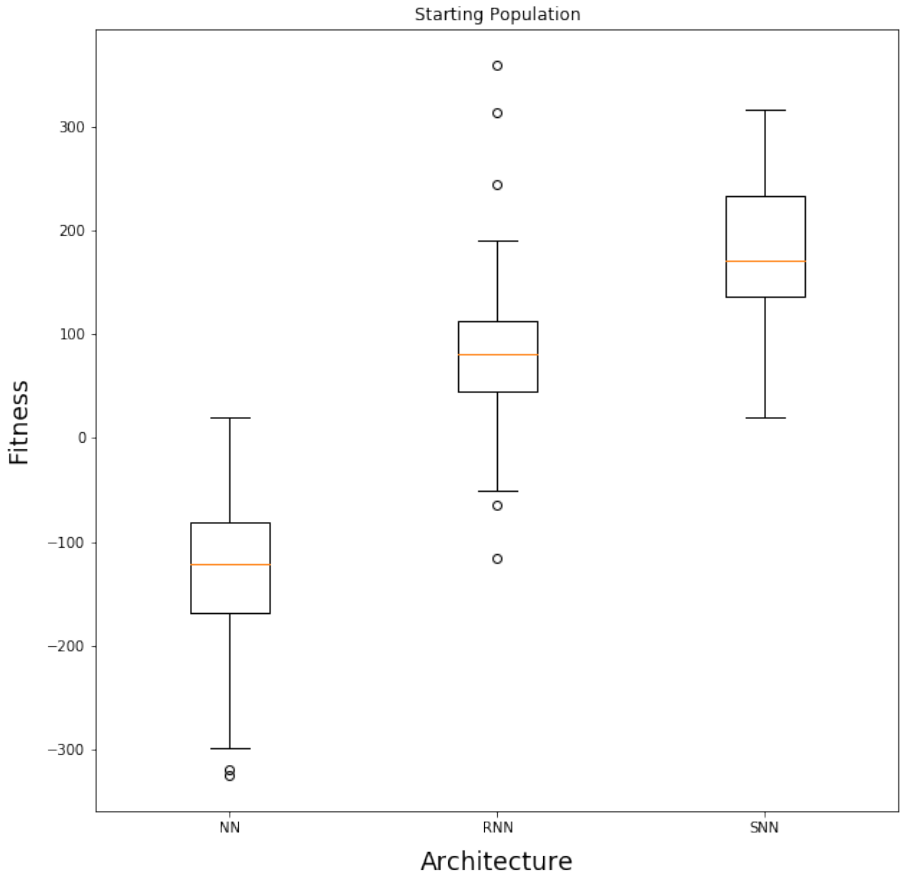


Figure 35: A comparison of the distribution of the individuals average fitness in the three randomly initialized populations. This are the results of the changed fitness function that reduces the cost of movement, to encourage a less passive approach.

In Figure 35 the distribution of the individuals results of the more proactive fitness function is shown. The best element belongs to the RNN population in this case, but they are all definitely closer and with less variance, with respect to Figure 31. This is explained by the fact that the two Perceptron based architectures are more prone to act at every step and this behavior is not as detrimental anymore, because the cost of performing an action has been lowered in the hope they will learn to move on this low gravity body.

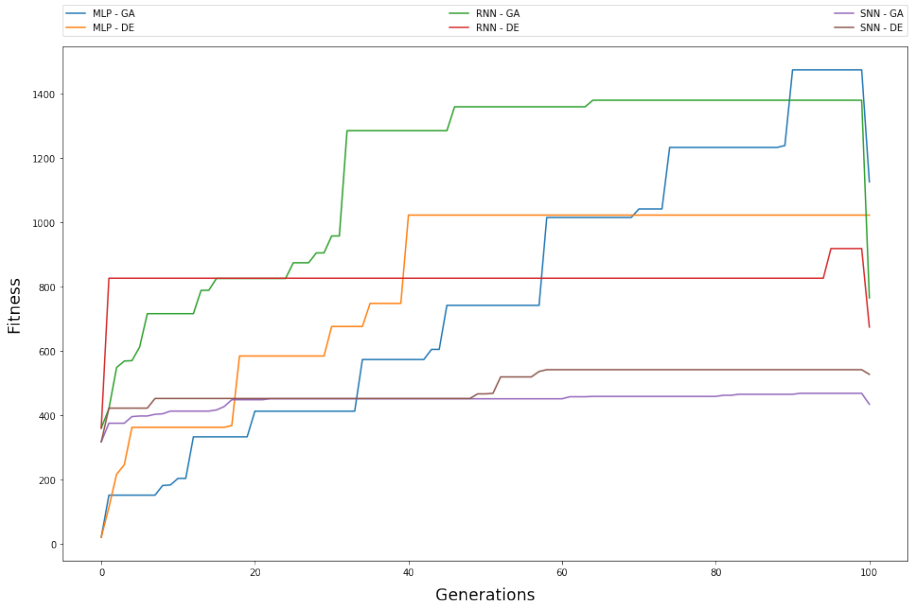


Figure 36: Evolutionary curves representing the score of the best individual of the population over 100 generations.

From Figure 36 the immediate reaction would be to consider the Recurrent architecture as the better one, yet the big drop following a plateau, reveals a strong instability in the individual results. A deeper analysis is done in Figure 37 and Table 15 that reveals the causes.

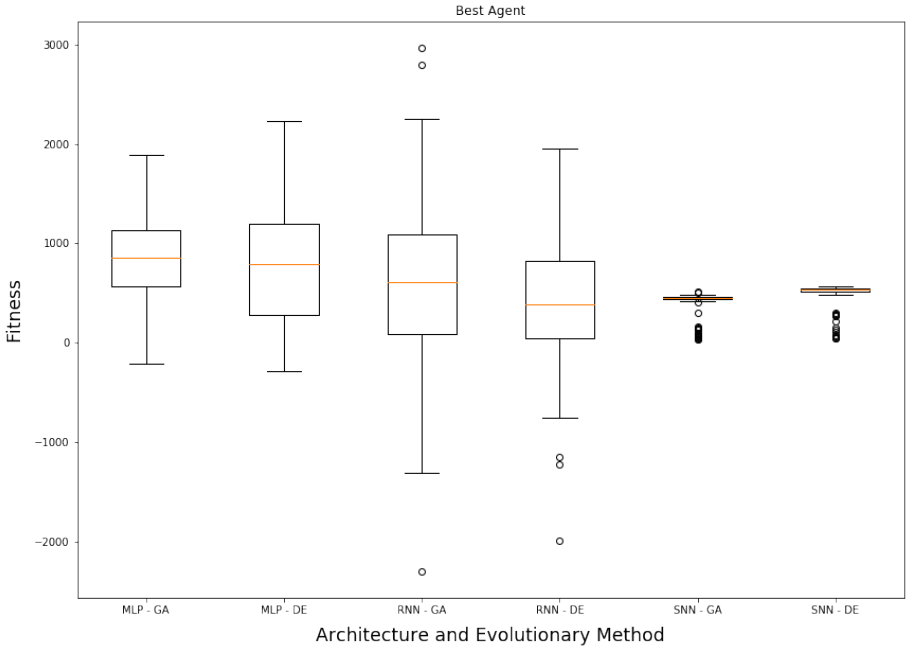


Figure 37: Comparison between the best RNN and SNN individuals fitness result distribution, trained for 100 test generations with each evolutionary method. Single results of 100 simulations.

As it can be seen in Figure 37 the results after 100 generations are not very promising for the Recurrent architecture. A very high variance makes it possible for an individual to score a successful result on a small sample of simulations, as during the training (see Figure 36), but the average score on a more sound series of test is balanced by negative results and ends up lower than the way more consistent Spiking Neural Network version. Furthermore from the training curves the RNN architecture seemed to have reahced a plateau at a higher score than the SNN, but this was only due to a sample size too small with regard to the variance.

	Mean	Median	Std Dev
MLP - GA	**862.1**	**858.4**	470.6
MLP - DE	764.2	792.0	581.9
RNN - GA	557.4	609.7	859.9
RNN - DE	393.0	384.3	662.2
SNN - GA	390.4	449.6	137.1
SNN - DE	499.0	534.0	**107.1**

Table 15: Statistics of the performance by the best individuals of all populations evolved with each algorithm. After 100 generations.

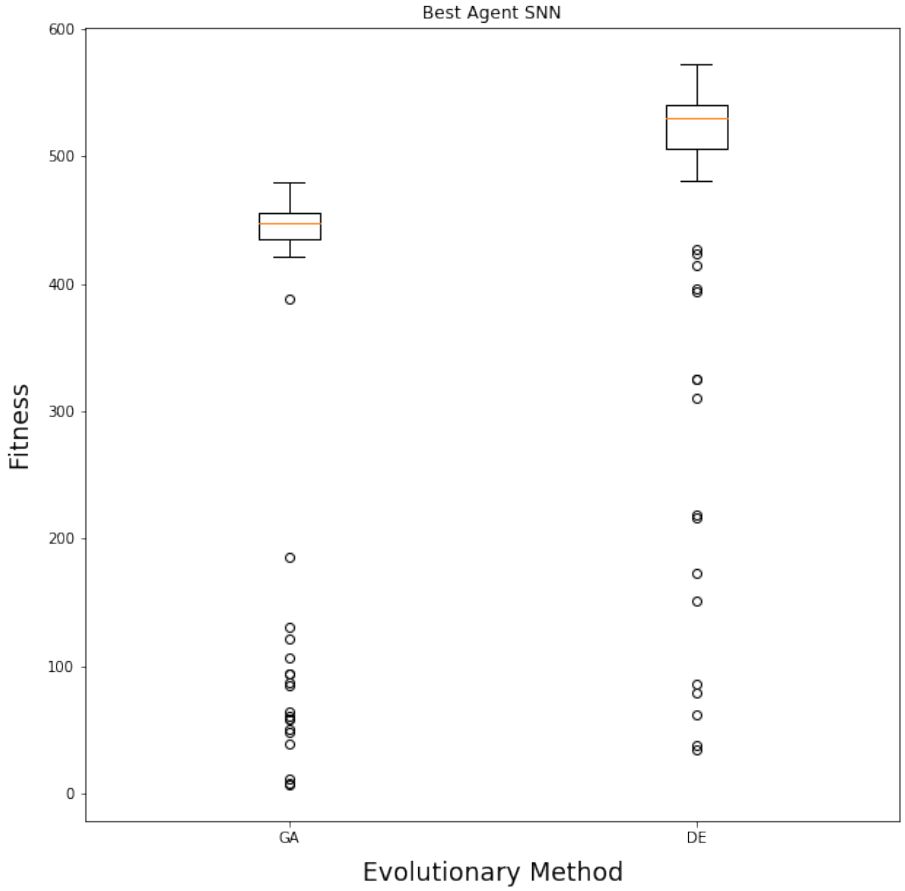


Figure 38: Comparison between the best SNN individuals fitness result distribution, by evolutionary method, 100 generations.

From Figure 38 it appears that all of the outliers in fitness results from the best individual evolved are on the lower side.
For the individual evolved with Genetic Algorithms these seem to cluster around a fitness that is about 25% of the mean one. This can be explained with the controller converging around two different behaviors based on the slight randomness of the starting position in the simulation.
The second behavior is less efficient and lowers the overall average more than the sparse outliers from the Differentially Evolved controller. From the data it also appears that this second behavior raises the variance in results. Were this behavior to be was erased from the controller by evolution, this would be more consistent although less performing on average then its alternative evolved with DE.

C Time analysis

From the experiments in the low-gravity environments it appeared that from the initial random population onward the Spiking Neurons architecture had a consistent edge over the others. For this reason the study of the time required by the calculations becomes almost superfluous.
A quick descriptive analysis is nonetheless added in this appendix, as one of the goals of this work was to show the approximation power of Spiking Neural Networks over perceptron based architectures.

	MLP 4obs	RNN	SNN
GA	6.06 (21833)	**1.2 (4314)**	4.47 (16080)
DE	11.91 (42859)	1.32 (4749)	4.51 (16237)

Table 16: Hours (seconds) required for the architecture to compute one epoch

Table 16 shows the time required to optimize with both methods the various architectures for an epoch. An epoch consists of 100 generations in which the fitness function is the average of 15 simulations plus an additional fitness check over the population with 30 simulations. The population size is 100 in all cases.

From the data presented it is possible to see how the Differential Evolution always takes more time then Genetic Algorithms. This is due to the greater amount of operations to perform to create a new individual. In the case of Genetic Algorithm there is a Gaussian random sampling of size equivalent to the number of parameters and the sum of these to the original values. For the Differential Evolution there's a sampling of K parents, followed by sum and subtraction of their parameters it is then K times the amount of sums with respect to the Genetic Algorithm counterpart. In addition, depending on the number of elites kept each generations the Genetic Algorithm saves that number of simulations to test the fitness. While, with Differential Evolution, a number of new individuals equal to population size is tested every generation. Since simulations are considerably heavier than the operations, this last point could be the most influential.

The comparison between RNN and SNN show how the use of Spiking Neurons instead of Perceptrons makes a big impact on computation time (around 350%). This is because the number of parameters raises and the differential equations to update the neuron state are not present in the Recurrent Neural Network.

Regarding the impressive time required by the Differential Evolution on the Multi-Layered Perceptron, since this is the model with highest amount of parameters, the higher computation count of this optimization algorithm makes a bigger impact. To the point of excluding its use in future experiments with very deep or wide Perceptron architectures.

D Benchmark videos

Better resolution of the videos of the best individual result on the benchmarks.

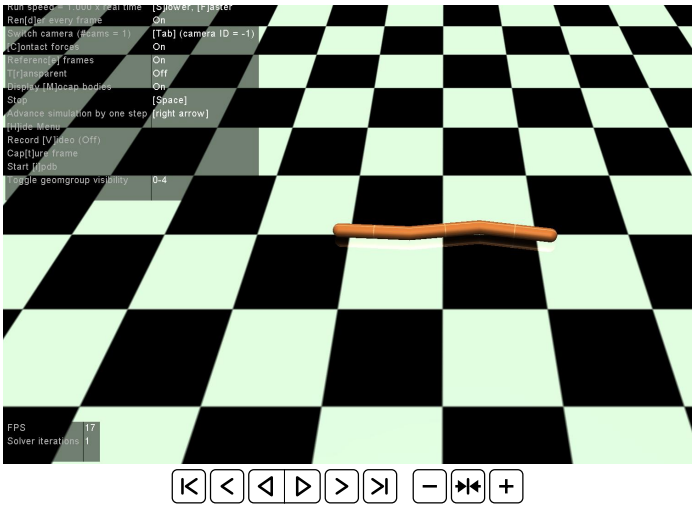


Figure 39: Swimmer-v2 Gym-MuJoCo benchmark evolved with Differential Evolution.

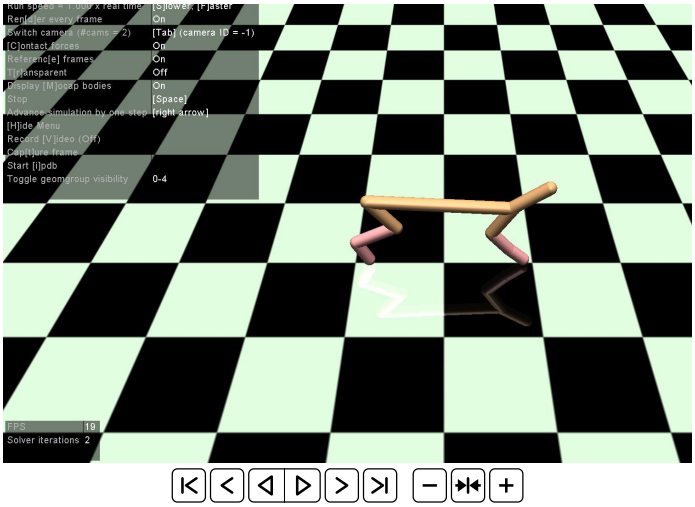


Figure 40: HalfCheetah-v2 Gym-MuJoCo benchmark evolved with Genetic Algorithms.

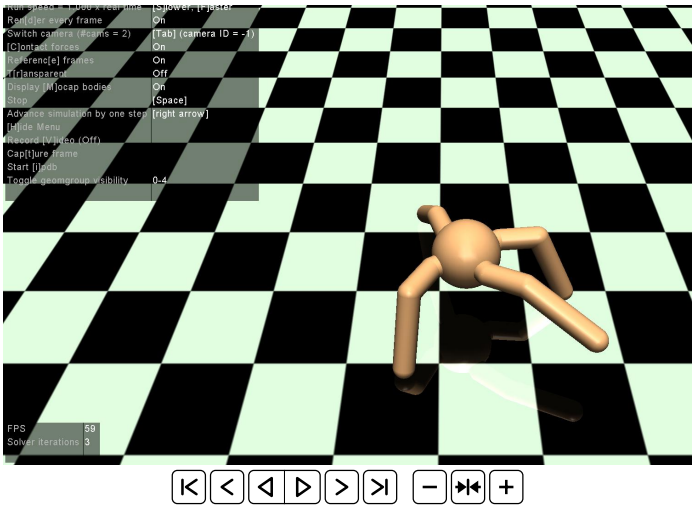


Figure 41: Ant-v2 Gym-MuJoCo benchmark evolved with Genetic Algorithms.