NTNU – Trondheim
Norwegian University of
Science and Technology

Neural Net Controller for a Snake Robot

Eirik Skjeggestad Dale

Master of Science in Computer Science
Submission date: June 2014
Supervisor: Keith Downing, IDI

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

Abstract

This project describes a system using an Evolutionary Algorithm to evolve an artificial Neural Network used to control serpentine robots. Snake robots have the potential to move much easier than conventional robots with wheels in cluttered environments. Since many areas of the world, on earth or on other planets, consist of cluttered or difficult environments, this provide ample motivation to develop snake robots. The project evolves a system taking the environment and target destination into account through its evolutionary algorithm. This allows the system to keep evolving artificial Neural Networks while the environment changes, and allows the controller to adapt to changes in the environment and target destination.

Contents

List of Figures

Chapter 1

Introduction

When we're talking about the field of robotics, one of the key components is the controller for the robots. The focus has recently begun shifting away from the traditional wheeled robots, into biologically inspired robots. Research is being done on walker robots with different numbers of legs, and also serpentile robots are being worked on. The previous and reasonably well-established robot controllers for the wheeled robots is generally not applicable for the new robots, since it is now a much larger focus on the motorics and how the robot must manipulate its limbs to be able to move. This opens up a new and more advanced area of research, where both navigation and movement have to be taken into consideration when making the controllers.

This project will attempt to make a controller for serpentile robots using an Evolutionary Algorithm to evolve the network on its own, since this has shown promise in previous projects in the field (See subsection 2.4.1). We also introduce a simple environment into the system to see how the system is able to handle and adjust to changes in the terrain. Primarily we will be looking at changes in elevation, but impassable terrain is also possible using the scheme we implemented. The snake robot is mainly just trying to move as far as possible in any direction, but we will also try giving it specific target destinations.

In this introduction chapter we will first take a look at our motivation for using this approach as well as the real world motivations to make controllers for robots of this type. We will assess the problems that may arise and take a look at the relevant research questions before we briefly introduce the approach we chose for this project.

1.1 Motivation

There are several sources of motivation for this project. As far as the robots go, the traditional wheeled robots are good for use in flat areas where terrain can be broken down into passable and non-passable. However, when it comes to environments where the ground is not flat and there might be shifting objects, or things like logs blocking the route, these wheeled robots start to struggle, and the pathing becomes a lot more difficult, sometimes even impossible. The robots using limbs could then simply step over the log, or adjust its limb manipulation to account for the change in elevation. In recent years we have seen more and more research going into the robotics and controllers for these kinds of robots, and Serpentile or segmented robots are an interesting challenge with a lot of potential in navigating difficult terrain. Where vehicles like the mars rovers are prone to getting stuck, and require careful planning from the controllers, a snake robot could handle the terrain features online and that way get a much better rate of movement. Building the actual robot is difficult and requires a lot of work in itself, but in this project we will focus on the equally difficult task of creating a good controller. The snake robot has been chosen over other segmented or limbed robot types because the university has a project going with a snake robot [1]. This means that the project had access to reasonably detailed information and formulas on snake robots and general snake locomotion, and it was a good stepping stone for this project.

So we saw that for the robots, a big part of the reason we want to do this is that they can handle difficult terrain better, and unstable, changing terrain. It is not only the robots movement abilities that should reflect this, but also the controller should be adapting to changes in the environment. Coding a good all-around controller that will handle different environments statically or with traditional Artificial Intelligence methods is extremely difficult, borderline to impossible with today's technology. This can be bypassed using an Evolutionary Algorithm that evolves a network as the snake is moving. Since using the snake itself to test the different networks in the algorithm is massively impractical, we need to create a simulator that does this, based on the observed environment. By evolving the networks directly instead of training them off-line with training data, we allow for more adaptability to new environments, which can be very helpful in many scenarios. One example scenario is that when exploring a burning build, a part of the building collapses further, adding a new obstacle for our explorers, and a potential dangerous zone the robots might want to avoid.

All this put together would allow for robots to operate in a myriad of new environments, and could be hugely useful in many fields, for example exploration and routine operations in hazardous environments. The project also

provides more research into evolving relatively complex neural network controllers for robots, and especially segmented robots. Progress in this type of robots and controller software could open up a lot of possibilities for future expansions and possible new technologies, with more focus on biologically inspired robotics. There are currently several studies into this area as seen in subsection 2.4.1, but a wider range of studies and more research focused on biologically inspired robots is necessary before they will be able to become as usual as wheeled robots are today. We also need more research into practical tasks they can perform to create a commercial marked to further fund the research and development into biologically inspired robotics and biologically inspired Artificial Intelligence.

1.2 Problems and Research Questions

Now that we have taken a look at the motivation for the project it is time to look at the problems the project attempts to tackle, and the research questions that rises from it. This is closely linked to the motivations, but there are slight differences and also problems that have little to do the motivation. Since the goal of the project is to create a system evolving Artificial Neural Networks used to control serpentile robots, it stands to reason that the main Research goal of the project is if this is at all possible. A follow up question could then be how much performance we can get out of such a system, and what the computation time will be like.

The problem here is getting the right balance in the trade-off between having a more realistic and that way more computationally expensive simulator and having the program actually find a new network in a tolerable amount of time. One can have very heavy algorithms running and calculating mostly all the physics that would naturally happen in such a system, but doing this for many systems at each iterations, like we do in an evolutionary algorithm, is simply too expensive with the current technology and available computation power. Even if this is not part of this project, another idea could be for the snake robot itself to run the evolutionary algorithm, making the system completely independent, but this would massively limit the computation power and is therefore not part of this project.

In the other end of the scale we have the system executing very fast, but this will typically not be a very good system either and it runs the risk of the simulator being inaccurate, or not getting close to a good, let alone optimal solution. This would be because the evolutionary algorithm is not allowed to run for long enough, with a big enough gene pool, to avoid local maxima. It might also simply not have enough time for the genotypes to mutate enough for a good solution to be found. There is also the concern that a system

focused on fast execution will have trouble dealing with obstacles, and the more variable the environment is, the more trouble it will have finding a good path and a good movement network for the serpentile robot.

Essentially we need to find a good trade-off between complexity and computation speed that fits well for our project. Since we want the snake to be able to move in more than one of the possible snake movement patterns (section 2.2), this system will be leaning more towards complexity than speed. In chapter 4 we see that we were actually able to generate more than one movement pattern, specifically during the fourth test conducted in that chapter.

Another problem the system needs to tackle is how to introduce the environment into the system. The main concerns are the same, with the trade-off between complexity and speed. In a real life scenario where the robot is in a closed environment it would have to gather information about the changing environment itself, and either upload the information to be processed remotely, or process it itself, if uploading is difficult. The latter is very time consuming, and in this case it would be possible to also add more complexity to the evolutionary algorithm, since the robot will already have to have a certain amount of processing power to be able to convert the data it gathers to a useful data model.

In the other case, where the data has to be uploaded, the model either has to get the next controller fast, or it has very little computation power itself. In at least the first case we can assume speed is essential, and that the robot is dependent on continued movement and getting the new version of the controller quickly. If this were the case we could make abstractions so that it would be simple enough for our situation, and giving us a fast enough system that will be able to deliver the new controller to the robot in time for it to maintain high functionality.

This problem gives rise to our secondary Research question for this project: identifying if it is possible to incorporate an environmental model into the system, how advanced the model have to be to achieve decent results, and exploring different ways to do this. If simple versions of relatively general environments that can be integrated into the locomotion simulation can give good results this would be very beneficial, because a separate environment simulator would slow the system down quite a lot. On the other hand, if the simple version means the robot does not get a good enough controller for the environment, the performance will suffer.

1.3 The approach

In this section we will briefly explain the approach we have taken to answer the Research questions stated in the previous section. The approach we took was, as previously mentioned, using evolutionary computation to evolve an artificial Neural Network to control the snake robot. The Evolutionary Algorithm we used and the Topology of the evolved Artificial Neural Network will be looked at in more detail in section 3.1 and section 3.2, respectively. There was always the possibility to create the Evolutionary algorithm using a preexisting physics simulation software, but the arguments for and against coding the simulator ourselves, and the reasoning behind doing just that over using simulation software can be found in section 2.1.

While a small goal for the project was to actually test an evolved controller on the snake robot this became too ambitious, and the entire project was done in the simulated environment. An example Neural Network made by the evolutionary algorithm is analyzed in detail in section 4.2. We will also look at the results, including the distance moved by the robots, in chapter 4. The distance moved is for the largest part of the project used directly as the fitness score for each genotype, without regards for direction. From there we will, in section 4.3, discuss further work that can be done as future projects in this field, with insights to the problems we encountered in this project and suggestions on how they can be solved by anyone else working on this type of project. Finally we will summarize the project and conclude with what we found out in light of the Research questions we started with, all this in chapter 5.

The main goal throughout the project remained to develop a system giving a controller for a more dynamic system that would be capable of adapting to changes in the environment. This means showing that this is both possible and a good way to go about changing environments, as opposed to manually having to update the controller whenever the environment changes to be able to encompass the new information about the environment.

Chapter 2

Background

This chapter will encompass the theory and background research behind the project. We start the chapter off by looking at the simulation software available to the project, as well as the development methodology we decided to use for this project. We will then be exploring the dynamics of snake locomotion in biological snakes, albeit at a simple level, before we move on to taking a look at the physical models used when constructing serpentile robots. The abstractions and formulas used in the actual system are also shown in section 2.3. It is worth noting that we so far in the project have been considering flat surface locomotion, and will not add to the difficulty of this chapter by considering cluttered environments. However, we will start by briefly looking at the considerations made when selecting the simulation software for this project.

2.1 Simulation Software and Methodology

When we started this project we had the choice of several programs that can be used to simulate robots in a physical environment. The main two programs we considered during the project were V-Rep [9] and Webots [10]. The third option, and ultimately the one we decided to go with, is programing the simulation ourselves. The two programs mentioned previously is reasonably similar, they both support Python, which is the language the Evolutionary algorithm is written in, but they differ on the point that Webots is commercial merchandise, and V-Rep is open source. This means we would have had to pay for webots, and v-rep does not have the same level of support as webots. If it was only between these two, it's reasonable to assume we could have started with v-rep, and if it weren't accurate enough, we could switch to Webots, and try that instead.

The big downside to using a preexisting environment though, and the reason we chose to implement a simulator ourselves, is that if we hope to implement an actual evolving network on an actual snake robot, programming some sort of physical simulation will be necessary. By doing this from the very start we can also have an easier time understanding results and problems we encounter later. The basic philosophy here is to work harder in the beginning so we can have more potential at the end of the project. This also gives us the freedom to change how complex the simulator is, allowing for testing of complex versus quick execution. Having full control over the simulator also allows us to add environmental features at a lower level, and this is a strong motivation for coding our own simulator, since it makes the secondary research goal much easier to approach. All in all the arguments for choosing an existing simulation software were that it would make the entire project easier, and it was more likely for a good neural network controller to be developed, at least for a simple environment. The arguments for a self-coded simulator were that we had full control over it and could test different approaches and levels of complexity with our system, which coincided nicely with our research goals. This was the reason we chose to create our own simulator, although it should be mentioned that this proved a difficult task and a source of many errors and bugs throughout the project, which is discussed more in chapter 4.

For the development methodology, I chose to use a spiral development scheme. The project is focused largely on experimentation and exploring what the genetic algorithm can get us. Planning and adding more functionality at each iteration in the spiral seems appropriate, and the since we don't really need any user interaction and feedback on any user interfaces, we did not want to use an agile development method. This decision was also aided by the fact that this is a one-man project, and the brainstorming sessions in agile development methods focusing so much on the group coming up with ideas together. Even with a group an agile development method would not be ideal since there is no big system that needs to be delivered to a customer. Simpler, older development schemes like the waterfall scheme and V-scheme is simply not good models, since it was expected from the start that we would have to go back and look at what we had done before quite a lot whenever new bugs where discovered, and introducing and trying new things could require changing big parts of the system. This led to the spiral development scheme being an easy pick for this project.

In short Spiral Development goes through a series of steps:

- Analysis: Determining objectives, alternatives and constraints for this iteration.
- Evaluation: Evaluating the objectives and alternatives from the previous step, and assessing risk.

- Development: Develop the planned system for this iteration.
- Planning: Plan the next iteration.

This is a good scheme that allows for most of what the project needs. The project to begin with is largely focused on research rather than following a development scheme designed to produce a commercial product. Having an advanced development model designed to get commercial system out is simply not a good idea. The spiral development scheme was followed during this project, but only loosely, with the main focus on figuring out bugs in the simulator and getting it to work properly. Only big things like introducing environments into the system or giving it a specific destination to move towards was actually done using the spiral scheme.

2.2 Biological snake locomotion

Snake locomotion is a complex process that depends on several factors, and there are several types, each specialized for a certain environment. When we talk about snake locomotion there are four main types of locomotion snakes are capable of ([7]). We will go through each of the different types of locomotion and explain how they work and how they fit into this project, as well as in which scenarios each type of locomotion is used by real snakes. The goal for the entire project was of course to be able to generate neural networks that can display all of these types of locomotion based on the environment. As we will see in chapter 4, we were not able to generate more than two of these behaviours, which is good to keep in mind as we go through them and explain where each of them is used in the real world.

2.2.1 Lateral Undulation

Lateral undulation is the fastest and most common type of serpentine locomotion, and is also the type most people will think of when they hear snake locomotion. The basic idea of lateral undulation is that the snake moves in a continuous wave that are propagated backwards along the snake body from head to tail. The sides of the snake push against irregularities in the surface the snake is moving on, and it uses the resistance from these irregularities to push the snake forward. Every point of the body touches the same point on the surface, and there is never any static contact between the ground and any point of the body. It also uses the friction in the scales of the snake, making sure it is easier to move forward than to the sides. This type of locomotion is found both in water and on land, but we will focus on the land part for this project. It is worth noting that the weight distribution of the snake during lateral undulation is not uniformly distributed along the

ground, but distributed in a way that the peaks of the body wave curves are slightly lifted from the ground.

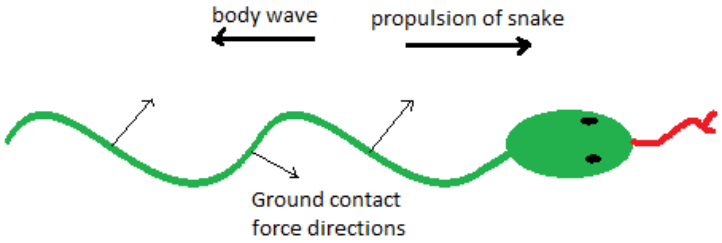


Figure 2.1: This figure shows the movement pattern during Lateral Undulation locomotion as well as where the force is generated from.

Although it has no impact on the start of the project, where we will only be working on a flat surface with friction, we will also have to keep in mind that on slippery or icy surfaces this type of locomotion has severely reduced effectiveness, because of the role friction plays in generating a forward pushing force. This is the main type of movement we hope to emulate, as well as sidewinding which we will explain later in this section. The movement pattern and pushing points can be seen in Figure 2.1.

2.2.2 Concertina Locomotion

Concertina Locomotion is often employed in narrow spaces where the snake is unable to use lateral undulation to move forward. The locomotion is done by first extending the front part of the body forward while the back part of the body is anchored, either by hooking its scales against small irregularities or using the narrow environment. Once the head and front part of the body is fully extended, they take on the part of anchor so that the back part of the body can be drawn up, and the motion is repeated. This means that it is quite slow going, and it is not efficient in terms of energy consumption, but it is often necessary when traversing narrow spaces. The motion can be seen in Figure 2.2. The principle behind the concertina locomotion is relatively simple; it relies on the difference between the large static friction of the anchor points, and low kinetic friction forces in the extending parts of the snake body.

For the model we will be using this is not immediately interesting. With no way to use the hooks as anchor points, since the robots have no hooks, there is no way to make use of the principle we just mentioned to move the snake. However, if we had added cluttered environments to the project, the robot could use the blocking objects of narrow spaces for anchorage, and this type

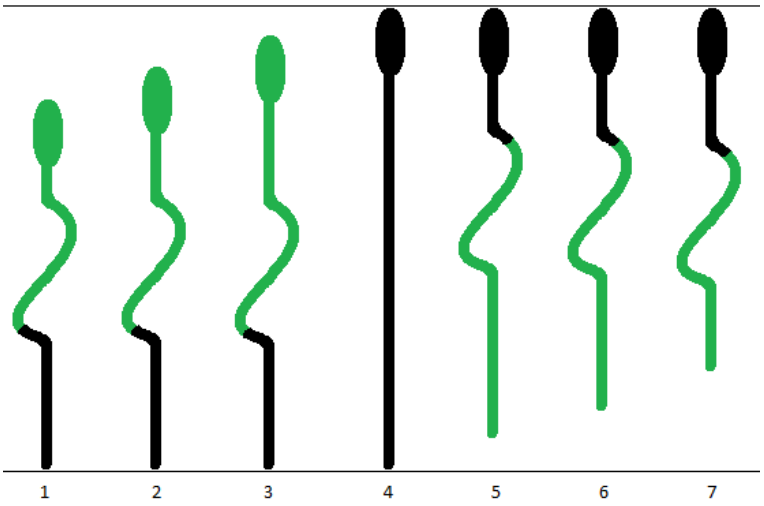


Figure 2.2: The basic principle of Concertina Locomotion. The black areas of the snake is the parts that is resting and acting as an anchor for the rest of the snake, being completely stretched out as in image 4 is not a requirement, but it does allow for the longest amount of movement at each cycle.

of locomotion instantly becomes more interesting. It is therefore important to keep this type of locomotion in mind when we develop the environment model for the simulation.

One way to introduce Concertina Locomotion into the flat surface model is to add to the robot the inability to slide backwards, i.e. that the friction backwards is very high, which might make the Concertina Locomotion model a viable and easy way to move the snake. However, it would still not be as effective as good Lateral Undulation, so we will not be adding that in this project.

2.2.3 Rectilinear Crawling

Rectilinear Crawling is a very slow form of locomotion often used by heavy-bodied snakes like pythons, boas and snakes in the final stages of hunting to avoid alerting its prey, and can be seen in Figure 2.3. The locomotion relies on using the scales as anchorage, and two opposing muscle-groups, namely the Costcutaneous inferior and superior muscle, present on every rib, connecting the rib to the sin. The snake starts the motion by having the costcutaneous superior lifting the snakes belly from the ground and placing it ahead of its former position. The scales are then used to anchor the snake while the costcutaneous inferior pulls backwards, propelling the snake forwards. Since alternate parts of the body stretches and pulls at the

same time, this looks like a continuous motion when done by real snakes, and it is almost soundless.

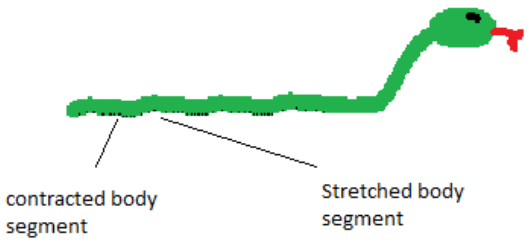


Figure 2.3: The amount the belly is being lifted is being exaggerated to show how rectilinear crawl works. After each segment is stretched it anchors and pulls the rest of the body forward.

Since both the motivation and actual mechanics for this type of movement is quite contradictory to the current technology used in snake robots, we will not be considering rectilinear crawl for this project. The robot is not going to be stalking any prey, and it cannot easily change the length of its joints. Since it also faces the same anchorage-problems as Concertina Locomotion, we will not be implementing any features supporting this type of movement for this project.

2.2.4 Sidewinding

Sidewinding is a form of locomotion with dynamics very similar to those of Lateral Undulation, despite appearances. It is usually used by snakes living in deserts, but has also been observed in muddy areas. The reason for this is that it is very effective in areas with slippery terrain, ice or loose sand, where we argued that Lateral Undulation would have decreased effect. This type of movement is also similar to Concertina Locomotion, in that it uses parts of its body as anchor for the rest to follow or move ahead of the rest of the snake. The snake starts by lifting and throwing the head sideways, ahead of the body, before using the head and neck as anchorage for the rest of the body to follow the snake into the new line of movement, giving a very characteristic trail from this type of movement, as seen in Figure 2.4. The snake moves at about 45° with respect to its heading.

This is one of basic types of snake movements, and by changing both the simulation friction constants and the fitness function it was one of the main hopes for this project that we would be able to generate this type of movement. This was still a secondary goal, and Lateral Undulation was the pri-

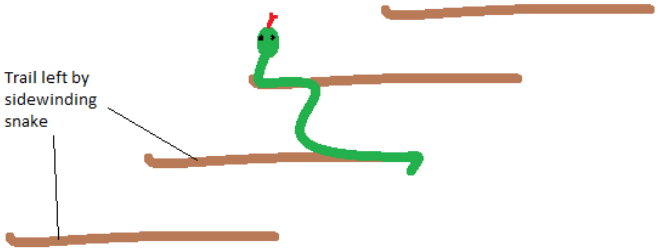


Figure 2.4: The characteristic trail from a sidewinding snake comes from the fact that anchoring and moving allows the snake to move in almost a straight line at each step, before jumping to the next one.

mary snake locomotion we tried to achieve. Lifting and throwing its head especially is a behavior not easily modeled by a simulator not specifically targeted towards this type of movement, and it was therefore uncertain if any evolved neural net will use sidewinding. After the tests done using a target destination however, we saw signs that the snake was actually sidewinding.

2.3 Physical Model for the robot snake

In this section we will briefly discuss the physical model underlying the simulation. The simulation is used to evolve the Neural Networks that will control the robot snakes. We will start with the complex model, as it is necessary before we can start discussing the simplified model to understand how it is simplified. The literature where this model can be found is [1].

2.3.1 Complex model

The robot consists of N links, each with length l, connected by motorized joints. All the links have the same mass m and moment of inertia $J = \frac{1}{3}ml^2$. We assume that the center of mass (CM) of each link is at the middle of the link, i.e. that the weight is uniformly distributed in each link. All the parameters used is shown and explained briefly in Table 2.1. In Figure 2.5 we see them represented in an illustration of the model.

The model will from here on assume that the snake is moving on a horizontal, flat surface and has $N+2$ degrees of freedom, one for each link and the planar position of the robot. The robot has no explicitly defined orientation since

Symbol	Description	Vector
N	The number of links	
l	Length of each link	
m	Mass of each link	
J	Moment of inertia of each link	
θ_i	Angle between link i and the global x axis	$\theta \in R^N$
ϕ_i	Angle of joint i	$\phi \in R^{N-1}$
(x_i, y_i)	Global coordinates of the CM of link i	$X, Y \in R^N$
(p_x, p_y)	Global coordinates of the CM of robot	$p \in R^2$
u_i	Actuator torque on link i from $i+1$	$u \in R^{N-1}$
u_{i-1}	Actuator torque on link i from $i-1$	$u \in R^{N-1}$
$(f_{R,x,i}, f_{R,x,i})$	Ground friction force on link i	$f_{R,x}, f_{R,y} \in R^N$
$(h_{x,i}, h_{x,i})$	Joint constraint force on link i from $i+1$	$h_x, h_y \in R^{N-1}$
$-(h_{x,i-1}, h_{y,i-1})$	Joint constraint force on link i from $i-1$	$h_x, h_y \in R^{N-1}$

Table 2.1: The parameters used in the complex snake robot model

there is an independent link angle associated with each link, θ. The model then defines the heading of the robot to be the average link angle. The robot then achieves forward propulsion on a flat surface by continually changing its body shape to induce ground friction forces, propelling it forward, as we also saw in section 2.2. The propulsive force from a single link is given as:

$$F_{prop} = -\sum_{i=1}^{N}((c_t cos^2\theta_i + c_t sin^2\theta_i)\dot{x}_i + (c_t - c_n)sin\theta_i cos\theta_i \dot{y}_i)$$

Necessarily the sum of these forces is the force with which the entire snake propels forward. Also note that $\dot{x}_i$ and $\dot{y}_i$ here is the linear velocity of $link, i$ in the global x and y directions. There are several more formulas for this model, but this is one of the key formulas. Here we will look at the most important formulas and their impact, but the complete model is available in [1]. The book also introduces into the model a couple of friction models. We will try all of these models out when we're implementing this, to see which gives the best results, or if they give different types of movements. Briefly, the friction models give us a value for the f_R we saw in Table 2.1. This is also where the values for c_t and c_n are found. As an example of a friction model, and the first model we will implement in this project, is the Coulomb friction model, given by the formula:

$$f_{R,i}^{link,i} = -mg \begin{bmatrix} \mu_t & 0 \\ 0 & \mu_n \end{bmatrix} sgn(v_i^{link,i})$$

$$f_{R,i} = R_{link,i}^{global} f_{R,i}^{link,i} \ , \ R_{link,i}^{global} = \begin{bmatrix} cos\theta_i & -sin\theta_i \\ sin\theta_i & cos\theta_i \end{bmatrix}$$

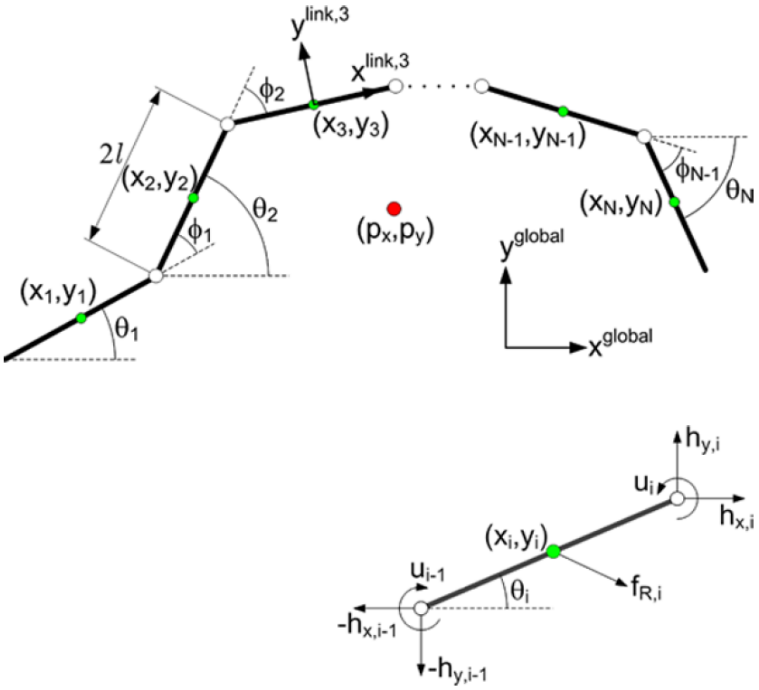


Figure 2.5: (Figure used with permission of Pål Liljebäck) This figure, from the book "Snake Robots: Modeling, Mechatronics and Control" shows how the different parameters from Table 2.1 correlate to the actual snake robot.

$sgn()$ produces a vector containing the sign of each individual element of its argument, and $R_{link,i}^{global}$ is called the rotation matrix. $v_i^{link,i}$ is the link velocity in the local link frame, shown in Figure 2.5, and finally g is the gravitational acceleration constant.

The most important part of the model is still how the snake actually gains momentum, i.e. the acceleration. Since we will be breaking down the model into discrete time steps, we can simplify and avoid some of the more complex formulas, and add the acceleration to the current speed at each time step. The formula for the global acceleration along the global directions is given as:

$$\ddot{p} = \begin{bmatrix} \ddot{p}_x \\ \ddot{p}_y \end{bmatrix} = \frac{1}{Nm} \begin{bmatrix} e^T f_{R,x} \\ e^T f_{R,x} \end{bmatrix}$$

Where e^T is simply used as a summation vector on the form $e = [1, ..., 1]^T$.

Another important part of this model is being able to make reasonable fitness score for it after running the simulation. For this we will at first be using the position, both before and after, and the take the difference and use that as the fitness value. This can be found through the global position for the

center of mass of the snake:

$$p = \begin{bmatrix} p_x \\ p_y \end{bmatrix} = \begin{bmatrix} \frac{1}{Nm} \sum_{i=1}^{N} mx_i \\ \frac{1}{Nm} \sum_{i=1}^{N} my_i \end{bmatrix}$$

2.3.2 Simplified model

Even though the complex model is what we have spent most of our time working on during this project, and it is the model we will be using later, it is still worth taking a brief look at the simplified model. The basic simplification is done by describing the snake robot not as rotational link displacements, but as linear link displacements transversal to the forward direction of the motion, and the simplification can be seen in Figure 2.6. The parameters are still the same as it was for the complex model. Even though the rotational motion of the links during the body shape changes is disregarded in this model, it will still capture the effect of the rotational link motion.

Since the different expressions was explained in the previous section, we will therefore only briefly list the formulas used in the simplified version, starting with the formula to be used in the fitness function of our evolutionary algorithm, namely the snakes Center of Mass position.

$$\begin{aligned} p_t &= \tfrac{1}{N} e^T t, \\ p_n &= \tfrac{1}{N} e^T n. \end{aligned}$$

Where t and n is vectors with the positions of the individual links, and e is the same as in the previous section. Also note that the relationships between the t-n frame position and global frame positions is given as:

$$\begin{aligned} p_t &= p_x cos\theta + p_y sin\theta, \\ p_n &= -p_x sin\theta + p_y cos\theta. \end{aligned}$$

We also have the relationship between global frame velocity and t-n velocity which is given by:

$$\begin{aligned} \dot{p}_x &= v_t cos\theta - v_n sin\theta, \\ \dot{p}_y &= v_t sin\theta + v_n cos\theta. \end{aligned}$$

The friction model has also been simplified, and as long as we keep in mind the c_n and c_p values of the Coulomb friction model, we can simplify away the rest of the friction models. The last equation we need to complete the simplified model now is a way to find the change in speed within the t-n frame. This is in the simplified model done as:

$$\dot{v}_t = -\tfrac{c_t}{m} v_t + \tfrac{2c_p}{Nm} v_n \bar{e}^T \phi - \tfrac{c_p}{Nm} \phi^T A \bar{D} \dot{\phi},$$

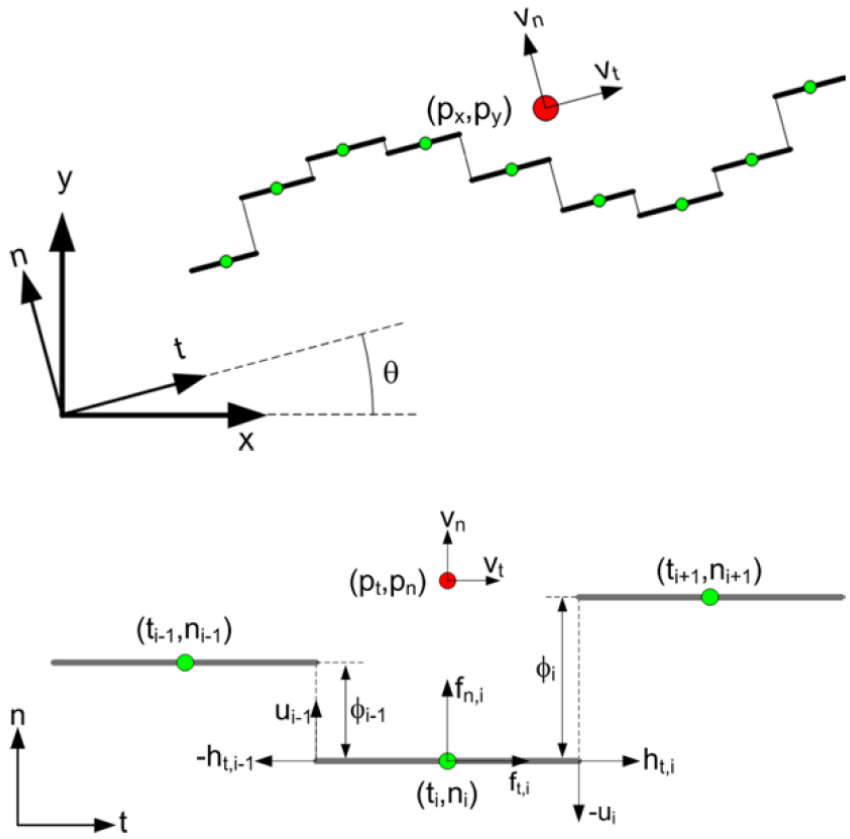


Figure 2.6: (Figure used with permission of Pål Liljebäck) This figure, from the book "Snake Robots: Modeling, Mechatronics and Control" shows how the different parameters from Table 2.1 correlate to the actual snake robot, using the simplified physical model.

$$\dot{v}_n = -\frac{c_n}{m} v_n + \frac{2c_p}{Nm} v_t \bar{e}^T \phi.$$

Where ϕ is the vector of all the link angles, $\dot{\phi}$ is the vector of angle changes and A, $\bar{D}$ and $\bar{e}$ support matrix' defined as:

$$A = \begin{bmatrix} 1 & 1 & & \\ & \cdot & \cdot & \\ & & \cdot & \cdot \\ & & & 1 \quad 1 \end{bmatrix} \in R^{(N-1)\times N},$$

$$D = \begin{bmatrix} 1 & -1 & & \\ & \cdot & \cdot & \\ & & \cdot & \cdot \\ & & & 1 \quad -1 \end{bmatrix} \in R^{(N-1)\times N},$$

$$e = [1, ..., 1]^T \in R^N \ , \bar{e} = [1, ..., 1]^T \in R^{N-1},$$

$$\bar{D} = D^T(DD^T)^{-1} \in R^{N\times(N-1)}.$$

With all of these formulas in place the simplified version of the model is complete. As previously mentioned this was the model used in the early version of the system, to get the evolutionary algorithm working before moving on to the actual simulator and the complex model described above.

2.4 Literature study

The literature study done early in this project can be divided into two parts, where the first part depicted the physical model of robot snakes. The other part was a structured literature review on the subject of using neural networks evolved with a genetic algorithm to control robots in general, and more importantly, control biologically inspired robots.

2.4.1 Structured literature review

We will start off with the structured literature review, which we wanted to do to make sure we covered the existing papers on this subject well, and we attempted to find anything we could use or build on for our project. In this section we will briefly go through the papers we found useful for this project, and evaluate it within the bounds of a structured literature review ([8]). During the structured literature review many articles were found and discarded either because of the quality of the paper or because the contents were not relevant to the project. The papers mentioned in this section are the papers that actually contributed to our project. What we discovered during the review was that many papers skirted the lines of what we wanted to try, meaning that there was many instances of showing a neural net that can control a robot, but most of these where tuned off-line and then uploaded. The articles were found in the IEEE Xplore digital library, and in Table 2.2 we see the search criteria used, as well as number of hits and how many of these we actually considered. Note that the third search was a generalization of the second one, and only one more paper was considered since the others was done during the second search. The Generalization was done mostly to check that nothing was missed that could be useful, and this was the only one that would have been missed.

Search Criteria	**Number of hits**	**Articles after title consideration**
Neural + Network + Robot + Genetic algorithm	561	5
Neural + Network + Robot + Controller	1811	14
Neural + Network + Robot	6707	1

Table 2.2: The different search criteria and results

Many of the papers that was looked at after they had passed the title consideration where discarded after a brief glance at the paper and reading

the abstract, either because they were too similar to other papers or simply wasn't helpful for our project in the way we had hoped after seeing the title. These papers will not be commented on further or mentioned in the text or reference list, as they were not an inspiration to any aspect of this project.

Cellular Neural Network Trainer and Template Optimization for advanced Robot Locomotion, Based on Genetic Algorithm

The first paper ([2]) we looked at appeared as perhaps the most promising one. After a quick Internet search we also discovered a video showing the result of the controller. The paper attempts to describe a learning algorithm for advanced robot locomotion, by evolving a Cellular Neural Network with a genetic algorithm. The research goal for the paper is exploring whether or not this is a valid approach that can achieve reasonable results.

Evaluating the quality of the paper, I started by noticing a lack of falsifying methods and resulting data backing up quite bold claims, like how this method lets a disjointed walker robot move with the "highest performance". Since there is mostly textual claims that this was actually the case, and nothing on how the highest possible performance was actually found, means that any evaluation of the claims would have to come from a complete reconstruction on the part of the reader, which is not a good way to write a paper. Another huge area that was missing from this was any information on simulation software used in this project and therefore any ability for the reader to detect or evaluate if there might be faults in the simulation giving the good results the writer claimed to get.

Another piece that was missing was information on the topology of the network. Some of it was listed, and there I do believe if you actually have seen this specific network before you will be able to remake it from just these data, but for a new reader without much pre-existing knowledge on Cellular Neural Networks this was quite confusing, and it took much more time than necessary to learn what they were actually talking about. The physical model is not shown either, or how they handle input and output, which makes reconstructing their system difficult.

All in all this looked very promising to begin with, but what we found in the paper had to be taken more as inspiration than proven statements. Since the projects are so similar, this was influential especially early in the project, and helped give us a way to evaluate fitness. The further works section also held some interesting points. As of the quality of the paper, I gave it a score of 2/10 after reading it.

Transfer of Human Skills to Neural Net Robot Controllers

This paper ([3]) attempts to create a neural net controller for robots that attempts to transfer Human Manipulation skills into the robot. It is worth noting that this paper is from 1991 and is therefore a little bit outdated, but the approach and results still made it interesting for this project. The results in the paper are validated using Lipschitz's condition, and the results are documented well.

The paper shows a neural network and its topology, with the weights resulting from the training data, and it describes well how they approached it, how the network works and what the data was.

However, since this paper is reasonably old, it didn't contain the inspiration I hoped after reading the abstract, but it shows a network that managed to transfer biological behavior into a robot, and the network was modified to fit our project, and used as a basis for our project, earning the paper a mention in this report. The quality of the paper overall reasonable, so the score it got was 6/10.

Mobile Robot Navigation Using a Neural Net

The next paper ([4]) explores navigation using a neural network, and whether or not this is a possible approach for robots in unknown and changing environments. It postulates that this approach should be simple to implement and fast in response, before going on to create a test of this using an actual robot with a context-sensitive neural net controller.

The question being explored in this question is something to consider in one of the biggest points in this projects Further works section (section 4.3), namely how to navigate to a certain target. With that in mind, it is worth noting that it is not of immediate use, that the primary goal of this project is the locomotion network.

The paper presents the training data and neural network topology in an elegant and simple way, making it understandable, without taking up too much place, or flooding the reader with more formulas and numbers than necessary. It has chosen a graphical approach to explaining the system and tests done, and the figures are easily understandable. So long as the results isn't fabricated they provide ample documentation of the success the method had, and it seems they have put quite some time and resources into testing.

The neural net was developed off-line with specific training data, and it uses completely blocking objects in its development. This mean it is not directly

applicable, as we work more with difficult terrain, instead of impassable objects, and we also would ideally want to evolve this network aside our locomotion network. The paper helped us modify the network from the previous paper, and also lent us a basis scheme for including impassable objects into our system. The language was good, and the paper was enjoyable reading, so the paper got a scoring of 9/10 during the review process.

SONCS: Self-Organizing Neural-Net-Controller System for Autonomous Underwater Robots

Another interesting research paper was the one about SONCS ([6]). The paper presents an adaptive control system for autonomous underwater robots through using a neural network controller. The results are validated through a small underwater robot and free-swimming tank tests. It is worth noting the variety of underwater robots is wider that the land-based robots and the dynamics of actually repositioning and moving the robot is simpler and well understood. However, this paper deals with angles input to a neural net in a similar way as we inspire to, and although it is a one-joint system that doesn't move by changing this angle, but with a propeller, we can still learn something from how this has been done.

Instead of an evolutionary algorithm, the adaptability of this system depends on the back-propagation algorithm, and it uses a multi-layered neural network, called a forward model network.

The paper explains its methods, data and controller setup well and thoroughly, making it easy to understand their approach, so even though the network and development of the network is different from the one we will end up using, we can easily extract any useful information from the paper. Mostly this paper was used for writing this report, and it was mainly used as background for adaptive systems in changing environments. During the review process I gave the paper a score of 8/10.

Reaction-Diffusion CNN Algorithms to Generate and Control Artificial Locomotion

[5] takes a look at using a specialized type of Cellular Neural Nets to implement artificial, bio-inspired locomotion. It focuses mainly on hexapod robots, but attempts to keep the methods more generally applicable, so this is still useful. This paper was used as a reference for the first paper we looked at, and we can here see more directly the network and methods used, described in a more thorough way. It is however, a little older, and is not directly applicable. The network generates a pattern of states the legs

should follow. This might not be the best way to go about it, but we can still take the network as an inspiration for use in our project, while changing inputs and outputs up. Another interesting point this paper brings up is developing hardware specifically designed to run the neural network, and as we will discuss further later one of the big problems with evolving a neural net is computation power. The hardware discussed in this paper is a bit outdated, but served as inspiration and a starting point.

The paper in itself is well written. It backs up claims and it's methods and results with a good amount of formulas, tests and their results, as well as figures and diagrams for both software and hardware solutions discussed here. If we consider the aims and approach of this paper along with newer technology, this is quite helpful for our project, and as it was also a good paper, I gave it a score of 8/10 in my literature review. Even though the number was already used through the first paper, the Reaction-Diffusion paper deserves a mention here since a lot of the testing numbers and numbers for the network was found here.

2.4.2 Snake Robots: Modeling, Mechatronics and Control

The second part of the literature study, and perhaps the most crucial one, was done on the book "Snake Robots: Modeling, Mechatronics and Control" ([1]). Even though this book focuses on the physical robot, it provides several formulas, as seen in section 2.3, and was used as a big inspiration for the actual simulator made during this project.

The book starts off talking a bit about the anatomy of biological snakes, and the way they move. It also takes a brief look at the motivation for making and using snake robots and shows examples of applications based on previous research, specifically from the NTNU and SINTEF cybernetics department. It focuses on the physical robot implementation, and is divided into two parts, one for locomotion on a flat surface and one for locomotion in cluttered environments. We focused on the Locomotion on flat surfaces in this project since this alone proved quite difficult, with the cluttered environment discussed more in section 4.3.

At this point the book once again splits into two parts, describing first a complex model and then a simplified model. Both models are implementable, and the complex model is the one used as a basis for the development of a mechanical snake robot to be used for motion across a flat surface, and the book goes through the different mechanical systems needed for this. It also proposes a ground friction model, which is very important for actual snake movements, and follows this up with a chapter of analyzing the snake robot locomotion.

The simplified model basically scales the dimensions of the complex model down to one. It focuses only on the change of joint positions in one direction and quantifies the joints as levels of a graph, assuming the snakes always move in the same direction, along the horizontal plane. This is fine for testing the simulators friction models, and it was the main model being used for the first part of the implementation phase, before we let the snake go in any direction later in the project. The friction model can be seen in [1]. It sums up part one by discussing some guidance strategies and path-following controls for the snake, before moving on to the second part.

The second part of the book is, as previously mentioned, about snake locomotion in a cluttered environment. This was one of the main sources of motivation for serpentile robots mentioned in the introduction chapter, and this chapter was used as a big source of inspiration for section 4.3, even though it was not implemented during this project, as it quite simply did not get past very simple environments.

The book always backs up each of the functions and claims with references to other articles, or provides mathematical proofs to support its claims. It is enjoyable and easy to read, and it gives a good foundation for any project centered on snake locomotion and snake robots.

Chapter 3

Methodology

In this chapter we will describe the methodology of the project. This includes a thorough description of the system as it is at the end of the project. The system has been divided into three parts, each of which will be handled in their own section. The Evolutionary algorithm is the first part of the system, and the key parts of the algorithm will be explained and described in section 3.1. The second part of the system is the actual Artificial Neural Network that was evolved by the Evolutionary algorithm, and the topology of the network will be explained. Originally we wanted to test out more than just one Neural Network topology, but the network shown in section 3.2 displayed reasonably good behaviour, and the focus was therefore more on the simulator, which proved difficult to get working. The last part of the system is the simulator, and it is also the largest part discussed in this chapter, since it is the most important one. The basic outline of the simulator will be described before moving on to the changes made throughout the project, in terms of introducing new elements into it, like target location and environmental factors.

After we have looked at the individual parts, we will describe how the different parts of the system fits together into the complete system, and look at why it was done this way, in relation to the research goals for this project. The programming language used throughout the system is Python. This was mainly to keep the code as clean as possible, and since we were going to handle a lot of numbers that might be difficult to troubleshoot, this is quite important. The argument that python is slow is not as relevant since we are not implementing the system directly on the snake, and we therefore have more computation power. The graphics engine is programmed in Tkinter, the standard GUI interface for python.

3.1 The Evolutionary Algorithm

This section will explain the Evolutionary Algorithm (EA) we developed for the project. We will systematically go through the different key aspects of the evolutionary algorithm, with a brief explanation of what an evolutionary algorithm is to start things off. It is necessary to mention that if the EA is to be transferred to an actual robot snake, it will need to be reworked completely, as it is currently not focused at all on being speedy and computationally light. Even though this is one of the major research questions, the focus on performance and complexity have all been centered on the simulator for this project, so the Evolutionary Algorithm is not made for direct integration into a robots controller system. It is also written in python, while C is most likely going to be the language necessary for the snake. Most of the techniques used and implemented in this chapter can be found in [11], which was the biggest source of knowledge for the EA.

This section will first go over an overview of the EA and then briefly go over the theoretical background for creating Evolutionary algorithms, before we take a look at the representation of the genotypes and the phenotypes, as well as the conversion between the two. Lastly we will go into more detail as of how the different parts and mechanisms of the EA work and which choices were made at the different levels of the EA.

3.1.1 Overview and theoretical background for the Evolutionary Algorithm

Genetic Computation is a relatively new and interesting field in Artificial Intelligence. It's inspired from the Darwinian Evolutionary Theory, and it works much the same as genetic mutation in the biological world. We will be relating the EA to biological evolution throughout this section, as it is a good example and makes the EA easy to understand. The workings of the EA can be seen in Figure 3.1. As we can see from the figure, an Evolutionary algorithm work much the same as evolution in the wild, where the well adapted individuals survive in their respective environments and faulty mutations will have a lower chance of survival. In an Evolutionary Algorithm, this makes the simulation, or fitness scoring, a very key component, and we will take a closer look at ours in section 3.3.

The basic idea is that we will be keeping a set of neural net controllers for our gene pool, where each controller will initially be generated as random values. Each of this Neural networks will represent one genotype. A subset of the genotypes from the previous generation will then be paired up and mixed with the other, before this in term is mutated into a genotype for the next generation. This is just like when biological animals mate in the

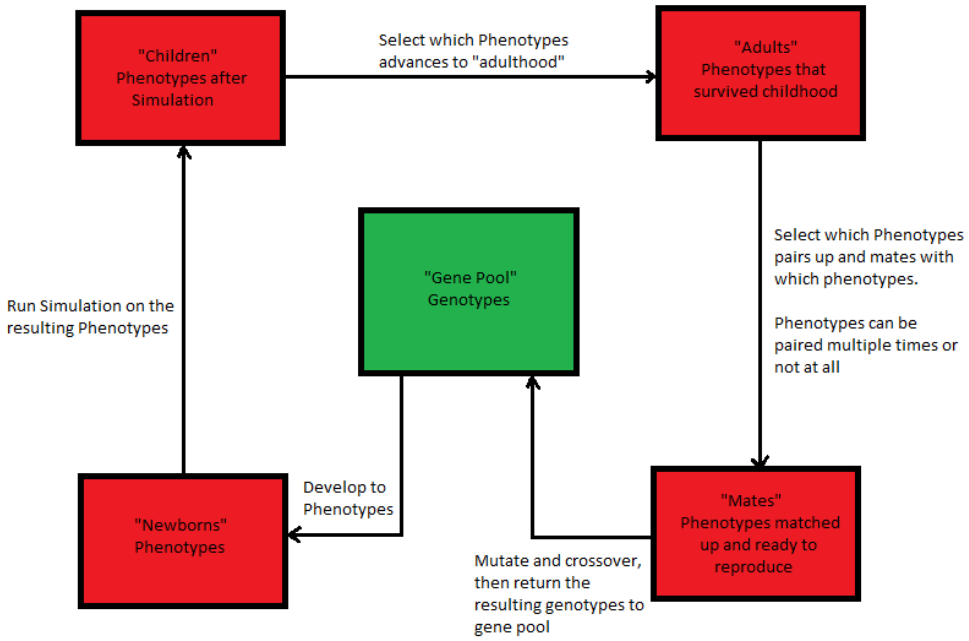


Figure 3.1: This is a schematic for the Evolutionary algorithm, showing the circle completed at each iteration. Some types of EAs allow the older-generation genotypes to compete with the younger ones for getting back into the gene pool, others do not.

real world, and produce a new set of genotypes, from which new phenotypes or offspring is produced. Each of the new offspring is then thrown into the world, or in the Evolutionary algorithms case, the simulator, an equally harsh and unforgiving place. If the genotype represents a superior mutation, both the offspring and the phenotype in the simulation should have a higher-than-average chance to survive and score well on the fitness test, meaning this is simply put survival of the fittest. As in the biological world, the phenotypes that do not do well have a chance to be dropped from the pool of individuals that gets to go into next round of the EA iteration, this being the programs way of letting poor mutations "die". The animals that actually grow up, and the genotypes not dropped from our gene pool, have a chance to mate. In nature, the strongest animals has a higher chance to mate, and this is also the case in our EA, and we will look closer at the mechanisms for this in subsection 3.1.3. When finally two phenotypes have been paired up, the genotype they carry with them is mutated and crossed over into a new genotype and the circle of life starts again.

During this project we did a lot of testing as to how big the gene pool should be, and how many generations we needed. This is one of the easiest ways to influence the computation speed as it is essentially a multiplication of how many times the simulator will run. Since we have the computation power to run relatively demanding programs, we tested different values to see how it affected the results, and the final numbers we arrived at was a gene-pool of 200 genotypes, running for 500 generations. During these generations most of the runs saw a steady increase in fitness, and most of the runs got up to a high fitness score, as shown in chapter 4. With a smaller population or fewer generations we saw the number of runs being successful drop, and with more generations or bigger gene-pool there was little pay-off, so we ended up testing most of the system using a population of 200 running for 500 generations.

3.1.2 Genotype and Phenotype Representation

Normally one could think the Genotype and Phenotype representations would be separated into two different sections, but as they are very similar in this project there is little point in doing that and this section will encompass both of them. When we are discussing genotypes and phenotypes we also need to think about how much of the Artificial Neural Network it should actually evolve. It is possible to have the EA evolve even the network topology, or we can program in a topology and have the EA evolve the weights. By evolving weights with values near 0 the EA can still to some degree reduce the connectiveness of the network, but it cannot add more edges this way. For this project we decided to evolve only the edges, for two reasons. The first reason is how letting the EA evolve the network can easily put us in the dark about how the network works. It would also spread the results further out and give a wider variety of results, but it would also most likely require a higher population and more generations to run properly and get consistent results. Evolved software and hardware have the pro that they often are very effective. The con is that it can be very hard to explain why it works so well. With the topology in place we will both avoid too large networks and they will be easier to analyze and understand the network, which is an important part of this project. Secondly, if we want to actually implement this on the robot snake we need to both be able to understand the network, and for the EA to be quite fast at simulating and creating the solution, which means just evolving the weights will be a lot easier than to be creating both the weights and the topology of the network. The concern about having to evolve a bigger population over more generations also adds to this; since it would drastically increase the execution time the snake would require to consistently achieve high performance.

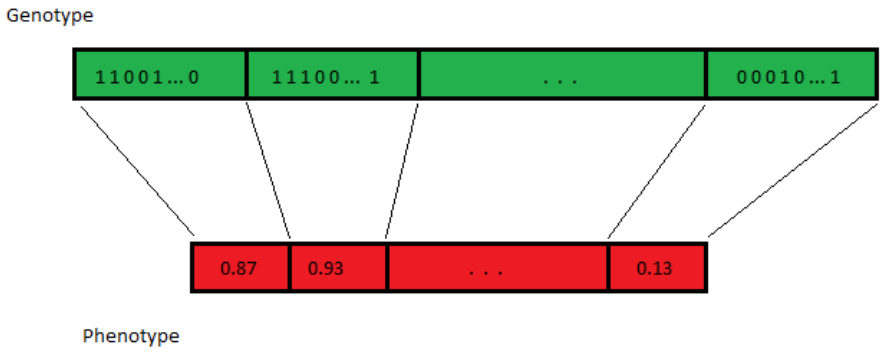


Figure 3.2: The Representation of the Phenotype and Genotype, a set of bits in the genotypes is made into a float value in the Phenotype when the genotypes transform. Note that the numbers in this figure is picked at random and there is no correlation between any of these numbers, nor are they accurate.

Now we will move on to the actual representation of the Genotype and Phenotype. Since we wanted to be able to try out networks with different topologies, the Genotype and Phenotype representation will be a vector of weights. This way we can dynamically program the EA to handle variable length Genotypes and Phenotypes, and it really suits the purpose of this project. The Difference between the Phenotype and the Genotype is that the Genotype will be a vector of binary numbers, while the Phenotype will be actual float values. This of course means that the Genotype will store many more numbers than the Phenotype, and how many bits in the Genotype is dedicated to each number in the Phenotypes decides how accurate the network will be able to tune it's weights, and also how long it will take to evolve it, since longer genotypes means the EA becomes more computationally heavy. The main reason why we choose to have the genotypes as a string of binary numbers is so that it is easier to mutate it in ways that ensure that the mutations are similar to mutations in the wild, and that it can actually achieve big differences outside of what we ourselves could think of, as is the main philosophy of an Evolutionary algorithm. The mutations are then done simply by changing one or more binary numbers to the opposite value, regardless of where the binary numbers are in the genotype, and in that way ensuring that anything can happen. To begin with we have dedicated twenty bits for each number, but it's reasonable to go as far down as eight, or even down to six, should the solution for the actual snake require it. The Representations of the genotype and phenotype can be seen in Figure 3.2.

3.1.3 Evolutionary Algorithm specifics

There are a few more things that needs to be specified when we are considering the EA. The fitness test and simulation is one of the most important, but at this stage it is purely the formulas that was shown in section 2.3, at the time of writing the simplified model is being used. This leaves some key traits that we will go through briefly in this section. To do this we will follow Figure 3.1. The general outline of the Evolutionary algorithm can be seen in Figure 3.3, illustrating how the different parts of the algorithm comes together in the implementation.

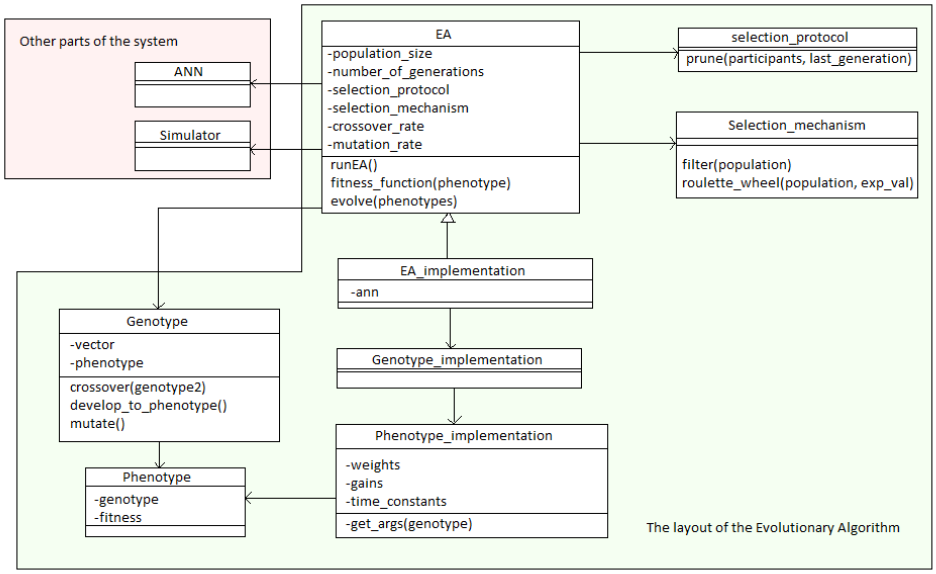


Figure 3.3: This figure shows a high level layout of the evolutionary algorithm to show where the different pieces explained in this section fits into the actual implementation, as well as showing how the Evolutionary connects with the remaining parts of the system.

The first thing that happens when a genotype is converted to a phenotype is relatively simple, and don't need to be explained in much detail. The bit values in the genotype is converted into a number in the tens system, before it is being moved into the range that the weights are allowed to be in. This then completes the conversion of a genotype into a phenotype, and we can move onto the simulation in order to get the fitness value for the phenotype. The simulation process is explained in more detail in section 3.3, and won't be covered here. After the simulation we use the Center of Mass that the simulated snake had at the end of the simulation to compute the fitness value:

$$fitness\_score = \sqrt{(cm[0])^2 + (cm[1])^2}$$

This gives us the fitness when the snake is only required to move as far as possible in any direction. To get the fitness when it has a target point to move towards, the formula to compute the fitness becomes:

$$target = [target\_x\_value, target\_y\_value]$$
$$starting\_distance = \sqrt{(starting\_cm[0] - target[0])^2 + (starting\_cm[1] - target[1])^2}$$
$$fitness\_score = starting\_distance - \sqrt{(cm[0] - target[0])^2 + (cm[1] - target[1])^2}$$

This fitness score is then given to the relevant Phenotype. Even though the fitness function is relatively simple, it is also functioning quite well, since the simulation only runs for a set amount of time it evaluates both speed and if impassable objects were to be placed into the system, its ability to handle these. Now that the phenotypes have received fitness values we are ready to begin sorting them. The selection protocol comes into play next. In the example we used earlier, the selection protocol determines which animals reach adulthood. In the project it simply selects a predetermined number of phenotypes from the full pool, and each phenotype can be selected multiple times or not at all. The Phenotypes of the old generation is also allowed to be selected in our implementation, so the selection protocol follows a generational mixing scheme. Other schemes were available, but since the system quite quickly reached decent fitness (see chapter 4) we decided to go with generational mixing over a full generational replacement scheme or overproduction scheme.

After this we are ready to cho0se who gets to mate with whom, in the words of our analogy. The selection mechanism is what is responsible for doing this. There are several relatively advanced selection mechanisms available, and several were tested. Common amongst the more advanced selection mechanisms is that they are computationally heavy, especially for populations as large as ours, so we decided to go with Sigma Scaling, a scheme that modifies the selection pressure by using the populations fitness variance as a scaling factor to determine which phenotypes is chosen. This scheme gave good results, and is also very fast, especially compared to more advanced methods that almost gives the same results as sigma scaling. This mechanism also scales with the fitness value, giving the better phenotypes better chances to reproduce, but by also taking standard deviation into account, gives it a more stable performance than simple fitness proportionate selection, without requiring much more in terms of computation.

Finally it is time for the phenotypes to reproduce and give us the next generation of genotypes. This is done with two methods, crossover and mutation. The Mutation is very important in order to be able to move your

gene pool out of local minimum in the fitness landscape, and it is classically done by flipping one or more bits in the genotype. In this EA we first give each genotype a chance to be mutated, set by default to 0.8, since there are so many numbers in this system, and such a large population. Only rarely did the high mutation rate lose the best performing Network due to mutating all versions of it. If it is chosen to be mutated it changes a randomized number of bits inside of one number, as it would have been transformed into a float for the phenotype, also chosen at random. This is to ensure that mutations are more stable and have more impact, since we saw in the early stages of testing that the weights didn't change enough with mutation. The Crossover is also being done with respect to the float number representation, meaning that the combined, new genotype gets a pseudo-random amount of bits from one parent and the rest from the other one, but still keeping the actual weights intact, because it does not split the gene outside the float number representations. This means splits can occur at bit twenty, forty and so forth, since the first twenty bits is one weight, the next twenty is one weight and so forth, and is in effect so that crossing two genotypes does not risk ruining both.

3.2 The Neural Network

The essential thing that this project seeks to produce is an extensive study around controllers for serpentine robots, and since we have chosen to use a Neural Network, this is a very important part of our system. After the literature study several possibilities have been presented. While ideally we would have been able to test mostly all of these during the project, this is not the case, and only a few changes have been made to the network during the run of the project. This was because the network described in this section worked rather well, and the trouble implementing the simulator (see section 3.3) the testing of different topologies was put on hold. A decently working neural network is in place though, or to be more specific, one neural network for each joint of the snake is in place. So far the majority of the effort has been spent on the model, simulation and EA, but if the project had gone further the next big point of focus would have been the network and its topology, testing different things and seeing what works best. This chapter will explain the workings of the current network, and show the topology we are using at the moment, as well as how this fits in with the rest of the system.

One of the key points when we're considering the controller is the fact that we are not using one single neural network, like we would if the robot we are making it for where a wheeled robot. Every single joint of the robot will require its own network, any other way would have meant that it would have been very difficult to create a good network topology to encapsulate the state of the entire state, and the network doing that would have been very large, depending on the size of the snake and how well it is required to perform. All in all it is simpler and less computation heavy to distribute the neural network so it is one for each joint. Having multiple networks raise the issue of what the input should be, where the most reasonable options are the current state of all the joints, the last output for all the joints, and the same, but for a neighborhood of joints instead of all. During this project all was tested, but the one that gave the best results was using the current state of all the joints as input. Necessarily using all the joints would be better than a neighborhood, but it is worth mentioning that if the snake is large enough it might still be best to use a neighborhood, although this is based on reasoning rather than tests done during this project. The input scheme can be seen in Figure 3.4. The difference in performance between using the previous outputs as input and using the state of the joints as input can be seen in Figure 3.5.

When we are talking about the topology of the neural network, how the inputs are interpreted and handled is a very crucial aspect to consider. Since there is one network at each joint, it should not be too complex, but it cannot

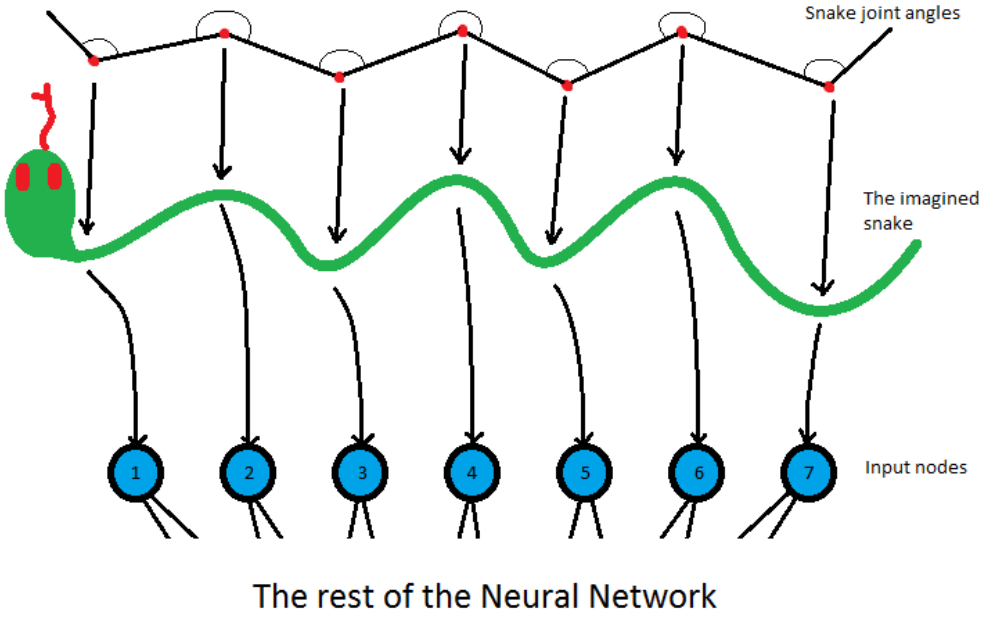
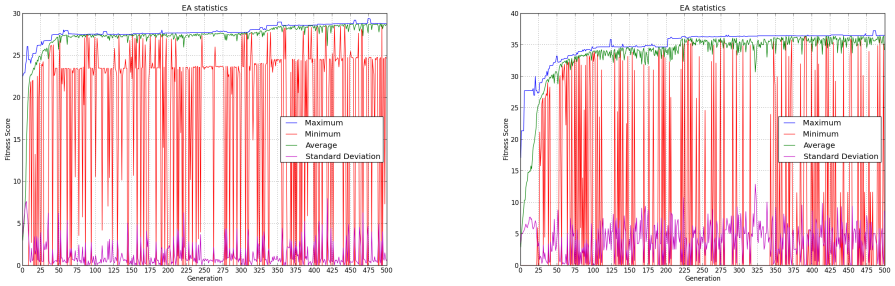


Figure 3.4: This figure shows where the Neural Network gets its input from. Each joint has an angle relative to the other joints, kept in radians in our system, and this is the value that is given as input for the network to use the next time step of the simulation.

be so simple that the snake is unable to bend its joints from the get-go and generate movements according to the principles we saw in section 2.2. Since we don't have to worry about direction of movement using only the simplified physical model, I decided to use only seven input nodes for the network, the previous link angles as input. This means that the snake we did most of the testing on had eleven joints, but even with only this number the behaviour we got out of the system was quite good. The reasoning behind this exact number was that fewer joints might make it difficult to develop the desired behavior, since it does not have the required number of links to bend to perform the snake locomotion. More joints would make the network more complex than necessary, and increase the computation time more than is necessary. This was especially relevant since we were coding the simulator ourselves and therefore had to test the system over and over to root out as many bugs as possible from the simulator. The subject of which inputs to choose and how to handle them, along with the fitness scoring is especially important for further work done on projects similar to this one, where more advanced environments are introduced.



(a) Using previous output

(b) Using current state

Figure 3.5: The following figure shows the general difference between using the previous state as input, done in figure [b], and using the previous iteration output as input, done in figure [a]. As we see the highest fitness is steadily around 5 points higher using the current state as input

The Hidden layer of nodes has two nodes in this system, regardless of how many input nodes there are. If the number of joints increases the hidden layer might need more nodes to be able to handle the inputs from all the nodes. If there was more time, we would have done testing on having more hidden nodes to handle an increased number of input nodes, but since time is limited and seven joints gets the desired behaviour out of the system, this was not a priority. Each node in the input layer sends its input data to both the hidden layer neurons, as seen in Figure 3.6. This is to allow both the hidden neurons to encompass all the information available and pass it on to the output layer of the network. To keep the motion smooth the hidden layer keeps an internal state that is used to compute output, and deteriorates over time. During the project a test was run where the internal state was removed and the results of the simulation plummeted, as each join would swing the other way a lot faster and the motion never got big enough to carry the snake any real distance. This test is not included in chapter 4, because it after a few runs became clear it never got the snake moving much and that the internal state would have to be reinstated.

The hidden layer then sends its information on to the output layer. The output layer consists of two nodes, the left node helps the snake bend its joint to the left, and the right output node lets the snake bend its joint to the right. In this project this is done by a simple subtraction, so that the biggest value wins. This was chosen over having a single output node that would have to go negative to bend the other way, so that each node have a smaller scale over which they need to activate, and can be more responsive that way. This means that the difference between the two nodes decides which way the snake bends, and how much, which in turn means that if

the two output nodes has the same value, the snake will not move at all. These output nodes receive information from all the hidden nodes, since the information from both ahead and behind the current link on the snake is important to get into both the output nodes, so it can move smoothly. This also allows the output nodes to utilize all the information available, which is important, especially when we're using so few joints in this project. In addition, the output nodes has self-edges and edges between each other to further help the snake move correctly and let information be passed between the two nodes. Like the hidden layer nodes the output layer also keeps and internal state.

Each of the Neurons in the hidden layer and the output layer is has a sigmoidal activation function. The Neurons keeps an internal state that together with an evolved gain value. This means the input is computed based on the following two formulas:

$$State+ = (-State + inputs * weights)$$
$$Output = \frac{1}{1+e^{-gain*State}}$$

So we see that the state is added to the previous state, giving the network a bit slower reaction time, but smoother motions, which is ideal for repetitive tasks such as movement. Note that the state was never allowed to drop below -125 since this cause the program to crash since the output value became so small it left the float number range and caused an error.

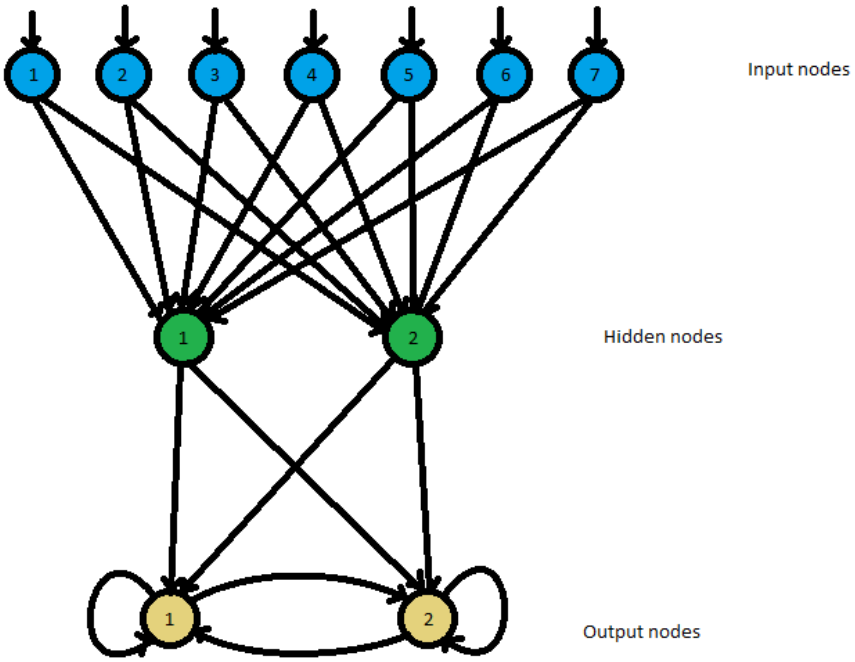


Figure 3.6: The current topology of the network. There are no weights currently displayed as the simulation has not been tested thoroughly enough yet.

The topology of the Neural Network used in this project can be seen in Figure 3.6. One thing worth noting is that we have allowed the neural net to have a lot of edges. This means that a lot of the information is getting utilized in the system. This might also have to be decreased if it is actually tested on a robot, since more edges means more computations to run the evolution on the networks. As we previously stated, the EA can effectively remove edges by giving them low weight and very little influence on the system. A logical step when we are looking to prune the Neural Network would be to run the simulation more times and see if any edges in the network is given consistently low weights, and then consider cutting away those edges.

3.3 The simulator

The simulator is the most important part of this system, and it also proved to be by far the most difficult to implement. Even at the end of the project it is not working completely correctly, and the current problems will be explained in subsection 3.3.3. Problems aside, the simulator does fulfill its purpose even though it still needs work to be bug free, since it does produce networks that actually makes the snake move forward, using similar movement patterns as the snake locomotions in section 2.2. This section will go through the simulator as it was at the different stages of the project, since this is the actual part of the system that was focused on and heavily updated at the different points of the project. We will start by the earliest version of the simulator, when only the simple version was implemented, before moving on to the more complex version where the simulator started using two dimensions. Lastly we will take a look at the final system and the test versions where we tried introducing basic environments and movement target into the system.

3.3.1 The early version of the Simulator

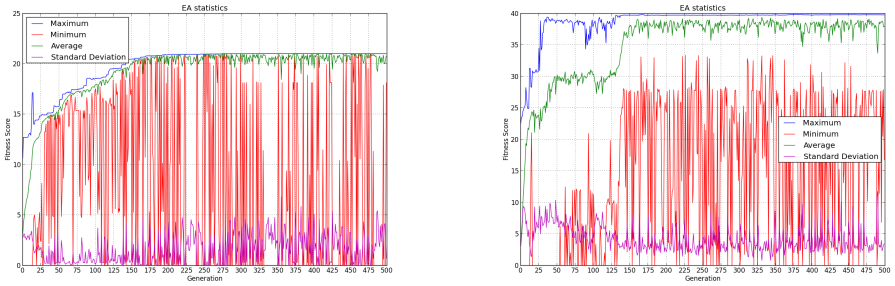
The earliest version of the simulator was implemented mostly to allow the Neural Network and Evolutionary algorithm to be tested properly with a simple simulator, so focus could be on rooting out bugs in these two. At this point there was no graphical representation implemented, so the results were interpreted from reading the position of the points and different points of the simulation, as well as the returned fitness score. The model this version of the simulator used is shown in subsection 2.3.2. The most important of the equations for this simulator is the ones that describe the changes in position and velocity for the snake:

$$\begin{aligned} \dot{p_x} &= v_t cos\theta - v_n sin\theta, \\ \dot{p_y} &= v_t sin\theta + v_n cos\theta. \end{aligned}$$

Describes the position of the center of mass, and the change in velocity is given by the formula:

$$\begin{aligned} \dot{v_t} &= -\tfrac{c_t}{m} v_t + \tfrac{2c_p}{Nm} v_n \bar{e}^T \phi - \tfrac{c_p}{Nm} \phi^T A \bar{D} \dot{\phi}, \\ \dot{v_n} &= -\tfrac{c_n}{m} v_n + \tfrac{2c_p}{Nm} v_t \bar{e}^T \phi. \end{aligned}$$

Using these formulas, the support matrix' and variables defined in subsection 2.3.2, we used the change of speed to compute the position of the different links of the snake, as well as the position of the center of mass. This resulted in a simple model that quite easily ran, but even if it was quite easy to implement the computation cost was quite high, and the re-



(a) The simple simulator

(b) The final simulator

Figure 3.7: The following figures shows the general difference using the simple simulator, as done in figure [a] and the final simulator as done in figure [b]. As is expected, the performance is quite alot higher in figure [b]

sults were not as good as in the later systems, both in terms of actual fitness (see Figure 3.7) and in terms of the movements of the snake, that was reasonable slow. This version of the simulator also only managed to achieve one of the types of snake movement described in section 2.2, but this was quite expected and reasonable, since the system in reality is working along a single dimension with no room to play on the breath of the space, like is a requirement for sidewinding. It also was not able to rectilinear crawl, since it cannot shorten and extend its joints up from the ground or into itself, like a real snake.

To sum this up, the simple model was a tool to help test the other parts of the system so that the simulator alone could have the focus from there on out, and did its job in that regard. The fitness score from this simulator was most of the time between 15 and 25, while the fitness for the final simulator was at a wider range, usually between 20 and 40. The final simulator could also handle movement towards any target and some terrain features, which the simple simulator simply could not handle.

3.3.2 The two dimensional Simulator

After we finished the first phase of the project and got the Evolutionary algorithm and Artificial Neural Network working properly, we moved on to improve the simulator beyond a single dimensional system. In the end this improved the results of the system, and this was also where the bugs started getting out of control in the simulator, and it took a long time to get it working decently and producing reasonable Neural Networks. In this section we will focus more on how the final system ended up being, and we will describe and explain the different bugs we faced in the next section

(subsection 3.3.3). Moving to a two dimensional simulator required the entire system to be scrapped and made fresh, since the formulas used for the one dimensional system simply did not hold up. In this section we will start by looking at the general flow of work and information through the two dimensional simulator. We will then move on to the most important parts of any simulator, the base formulas that allow the simulator to run and bring the simulated snakes to life. We will also explain some limitations that were put in place to avoid certain unwanted behaviors. This might sound like it would defeat the purpose of an evolutionary algorithm, and to some extent that is true, but these limitations put into the program where made to reflect the physical limitations of snakes and serpentile robots, as for example that the body can only bend so much before the snake hits itself or the bones can't bend further.

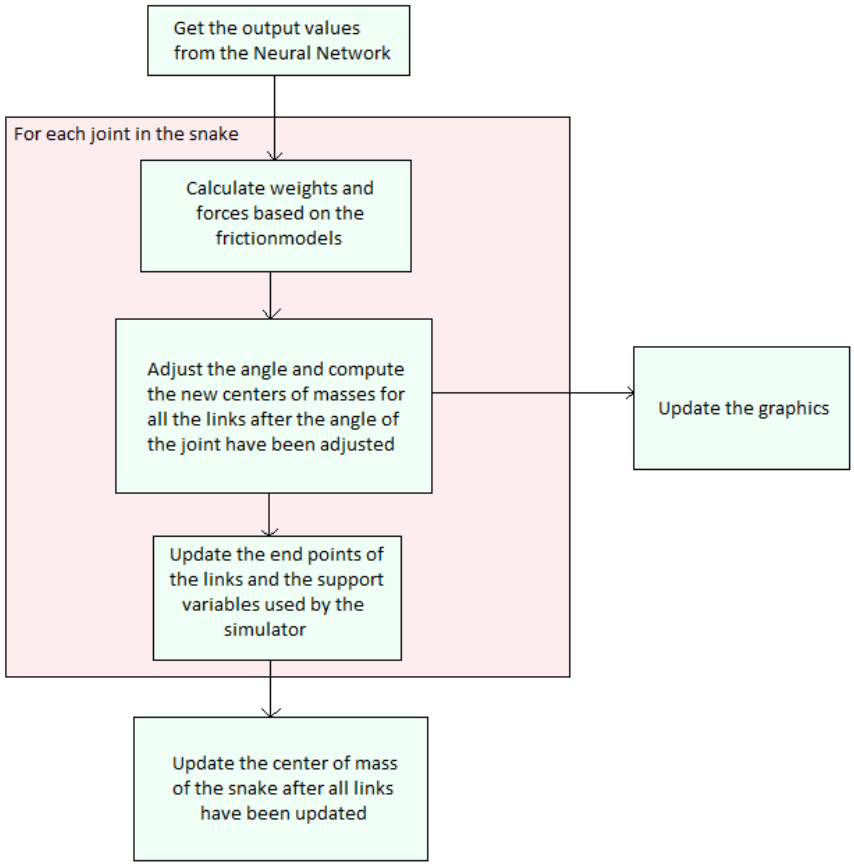


Figure 3.8: This figure shows the flow of the problem and the order in which things are done in the simulator.

First we will take a look at the general information flow and work order of

the two dimensional simulator. This is shown in the graph in Figure 3.8. The simulator starts by receiving the output values from the Artificial Neural Network, before it moves on to updating the snake using these outputs. The model used for this is given in subsection 2.3.1. The force on the snake from each individual link is given by the formula:

$$F_{prop} = -\sum_{i=1}^{N}((c_t cos^2\theta_i + c_t sin^2\theta_i)\dot{x}_i + (c_t - c_n)sin\theta_i cos\theta_i \dot{y}_i)$$

The last piece the simulator needs to readjust the position of the snake for the output now is a good friction model, and the one given by this model is given as:

$$f_{R,i}^{link,i} = -mg \begin{bmatrix} \mu_t & 0 \\ 0 & \mu_n \end{bmatrix} sgn(v_i^{link,i})$$

$$f_{R,i} = R_{link,i}^{global} f_{R,i}^{link,i} \ , \ R_{link,i}^{global} = \begin{bmatrix} cos\theta_i & -sin\theta_i \\ sin\theta_i & cos\theta_i \end{bmatrix}$$

Using these formulas the simulator can quite easily find the new positions for each link and the angles between them. The limitations that we mentioned in the section introduction also kick in here. The angle of each individual joint cannot exceed $\frac{\pi}{2}$. This means that it requires two joints for the snake to reach parallel with itself, which is a reasonable limitation to make, it could not be a larger angle than that because of the low amount of links, and larger than this led to reduced results and some weird behaviour where the snake kept standing relatively still and just clumping its links up.

At this time we also had in place a simple graphical representation of the system, and both discovering bugs and confirming locomotion patterns. A screenshot from the graphical representation can be seen in Figure 3.9. With the graphics in place and the resulting networks described in chapter 4 the simulator does make networks in thread with the first research goal: an Artificial Neural Network that moves the snake in a snake locomotion pattern. This is a quite good result, considering that there at this point there was still relatively heavy bugs in the simulator.

3.3.3 The final version of the simulator

The difference between the final version of the simulator and the two dimensional is not as large as we had hoped it to be, it is mainly bugs being fixed, but also some testing into having a target destination to move towards, and different levels of elevation as a very basic type of environmental factors. In this section we will first take a look at the bugs we found in the previous version, and the bugs that remained in the final version. The reason bugs remain at the end of the project is simply that it is difficult to evaluate

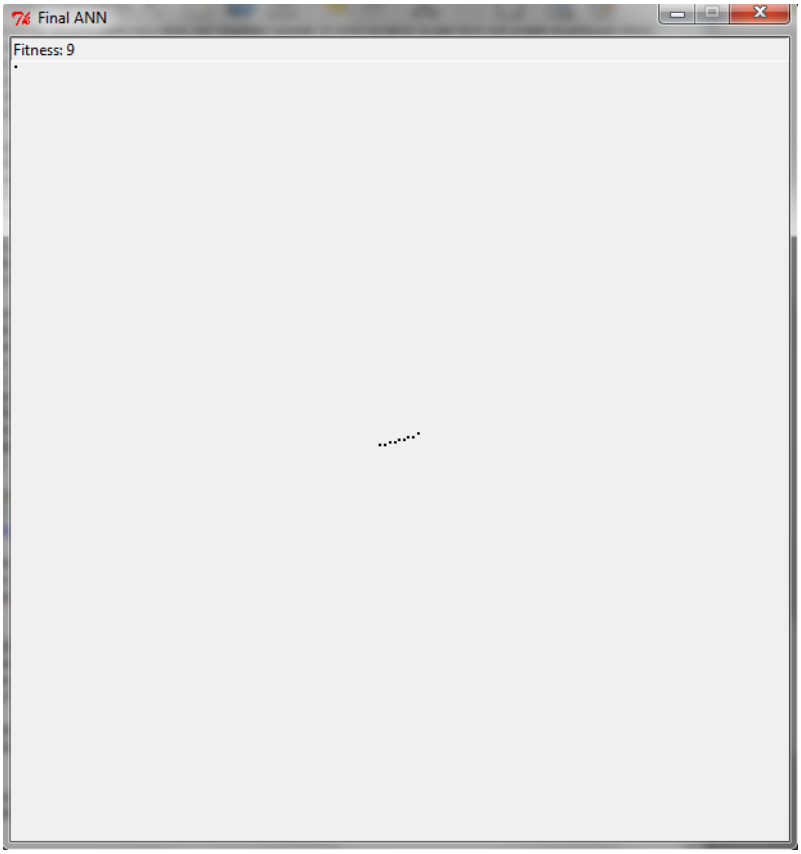


Figure 3.9: This figure shows the the graphical representation of the simulation. Each dot of the snake represents an end point of a link, since this makes the movements smoother than using the links center of mass.

where the bugs are in the code, since they don't cause crashes and there is a big amount of numbers and equations to evaluate. After this we will look at how the target direction was implemented and finally how we got environments into the simulator.

The first and biggest bug we found was a simple bug with big consequences. The first week of working on this simulator the output values and desired change of angles was done correctly, but after updating the centers of mass and endpoints for each links, it was completely off. The reason for this was that the sine and cosine in the friction model described in subsection 3.3.2 was swapped. This should've been caught faster than it were, but it too several rounds of doing the math on paper before the mistake was spotted. As soon as this bug was fixed the snake started moving in the desired locomotion patterns, but it was not the end of the problems. The next bug we found was first discovered when the graphics was put in place. As stated

before it was difficult to find bugs before this since there was a large amount of numbers to evaluate at every generation. This bug basically was that the links in the snake physically started drifting apart after 5-10 iterations in the simulator. This really boosted the change in the snake's center of mass, which was used as the fitness scoring, and therefore happened every time, since evolutionary algorithms is really good at abusing bugs like this to get higher fitness scores. It was easily spotted when the graphics was in place, the snake drifting apart can be seen in Figure 3.10. The bug was again in how the link positions were updated, and the formulas were fixed. In addition a failsafe was implemented, that never allowed the end-points of the links to be separated. After this fix the snake stays gathers and moves in a correct movement pattern, and these were all the bugs that were fixed during this project. There still remain bugs in the final simulator, but it is difficult to say what it is. The effects is that the snake always moves correctly for 15-20 time steps, and then starts to jump back and forth between two states. This is most likely the failsafe from the previous bug kicking in and resetting the state because the points of the snake would otherwise start drifting.

Now that we're done with the bugs we will take a look at the parts of the final simulator that actually worked correctly. The first new part that we implemented was relatively simple, the target location for the snake robot. This was done by simply changing the fitness score to be computed as $starting\_distance\_to\_target/(1+distance\_to\_target\_location)$. The snake then moved towards the target location. The snake never quite reached the target points if it was far away like it typically would be in a real scenario, due to the last bug described, but it started moving in the right direction, and if the point was close enough it actually reached the point and got max fitness. In Figure 3.11 we see two different target points and the snake moving towards it.

Lastly we will look at the environmental change were we tried introducing different levels of elevation. This is briefly explained as having variables at each cell in a discreet cell map for the field, where each cell holds a value for how high it is in the terrain. The difference in the cell the snake wants to move to and the cell it is moving from is then added to the friction value, so it is harder for the snake to move uphill than it is to slide downhill. An illustration of the cell map scheme and the graphics from a run using that cell map is shown inFigure 3.12. In terms of performance, understandably using the cell map scheme decreases the distance the snake moves a little bit, since it now has to change direction during the run.

All in all the final simulator worked reasonable well, and produced working Neural Networks, as analyzed further in section 4.2, even if there were relatively serious bugs even in the final version.

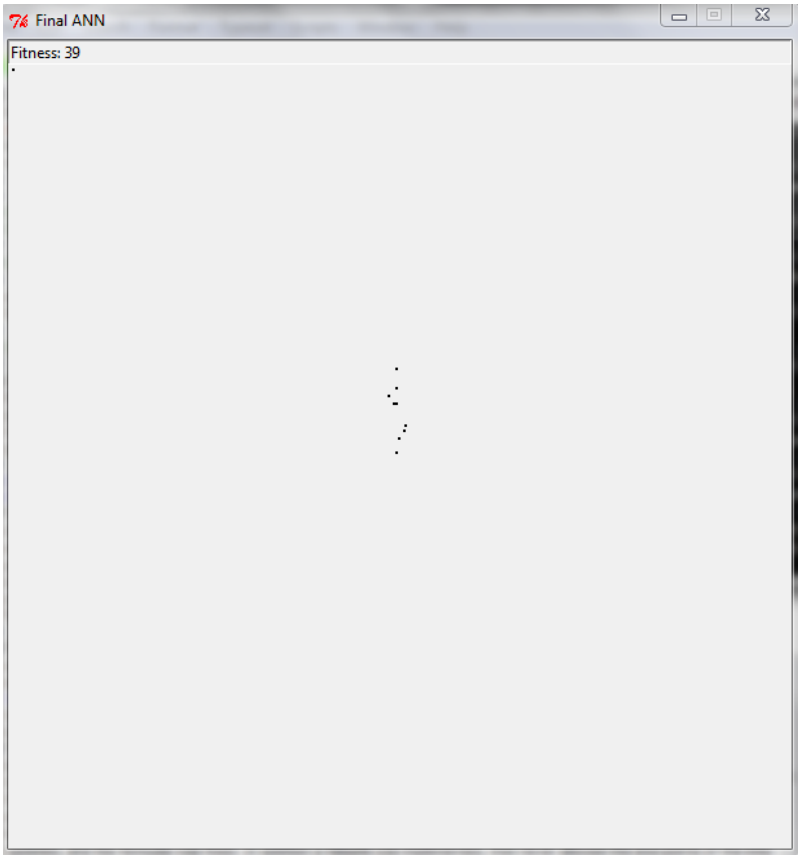
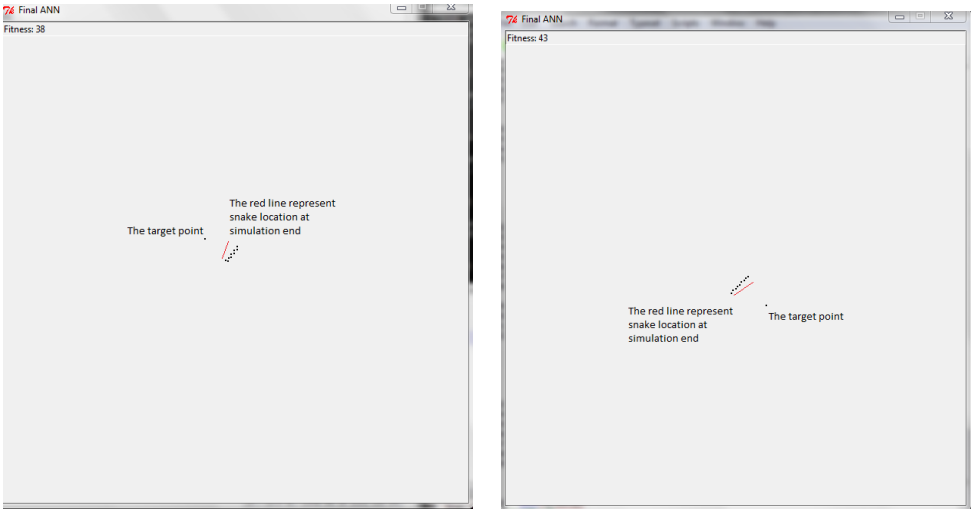


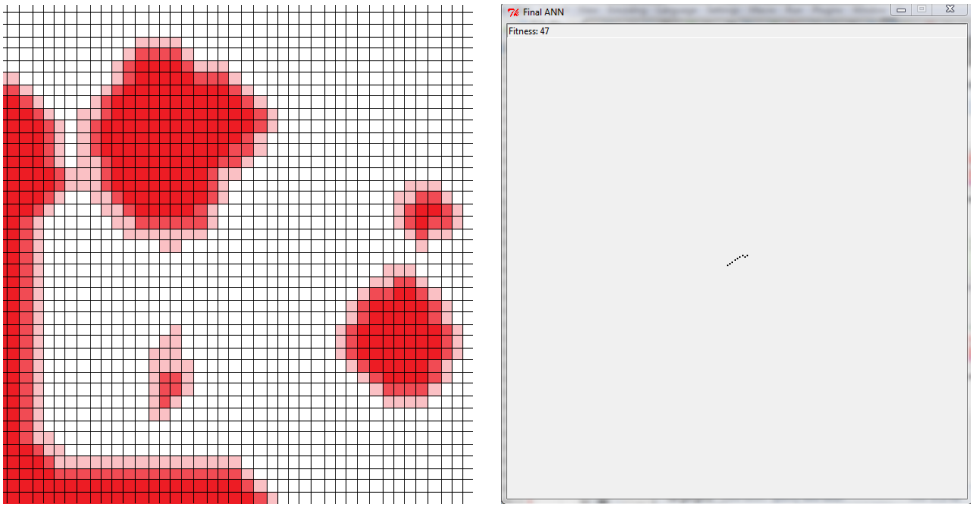
Figure 3.10: This figure shows the bug where the snake started drifting apart after a few generations. Notice from the fitness score that even though the dots is still relatively close to the middle, the furthest out dots drag the center of mass out of it and gives this run a relatively good fitness score.



(a) First target location

(b) Second target location

Figure 3.11: The following figures shows the graphics from two runs with different target locations. The red lines and the text have been added after the screenshot was taken.



(a) The cell map scheme

(b) The graphics from a run with the cell map

Figure 3.12: The following figures shows a graphical illustration of the matrix containing the cell map, where the redder the cell is, the higher it is in the environment. The number of cells is not accurate to the number of cells in the actual cell map, which is 200x200, but the colors follow the areas in the actual cell map correctly. The screenshot is from a run with that specific scheme in place

3.4 The system put together

Finally in this chapter we will take a look at the system at a whole, and how the different parts work together. This does not require a huge amount of explanation, but it's still necessary to understand the complete system. The basic flow and diagram of how the systems works together can be seen in Figure 3.13.

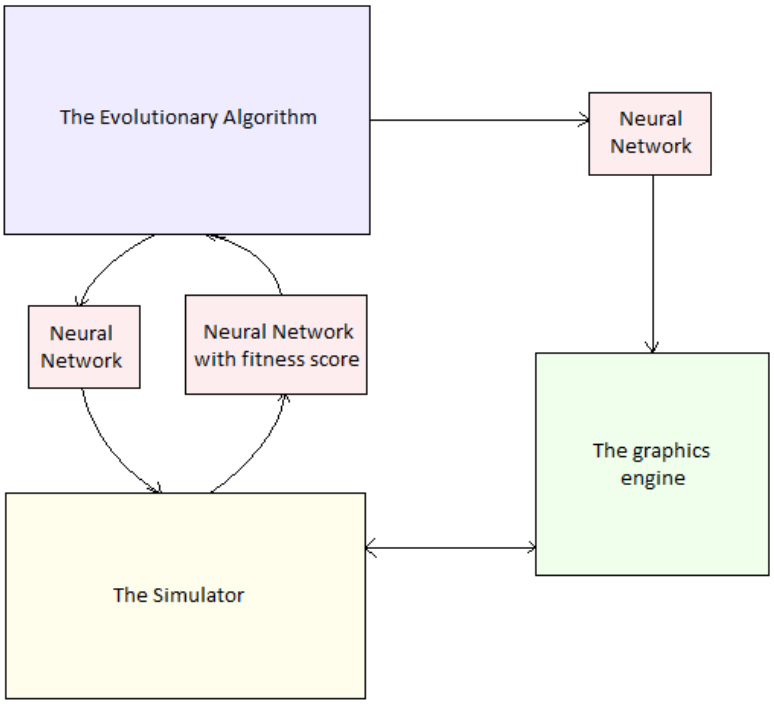


Figure 3.13: This figure shows how the complete system works together, where the arrows represents flow of information. The role of each subsystem have been explained in the previous sections.

The first part of the system to run is the EA. As was explained in section 3.1 the evolutionary algorithm starts evolving an set of Artificial Neural network with the topology we looked at it section 3.2. These Artificial Neural networks is then sent to the simulator where the networks take control of a snake, and is evaluated based on how close they came to the target point, or how far away from the starting point they got, based on which simulator is used. Finally the Evolutionary Algorithm sends the best scoring network to the graphics generator that runs the simulation again and shows the different

stages as dots on the screen. Even though the system had bugs, especially in the simulator, it still managed to evolve useful networks that showed the desired behaviour to fulfill both of the research goals we mentioned in the introduction.

Chapter 4

Results and Discussion

In this chapter we will go through the results achieved in the system and discuss them. While there as mentioned was some hick-ups the system still managed to evolve a working network, as we will see in more detail when we actually analyze a network in section 4.2. Before that we will take a look at the general results from the series of test runs we did with the system at the different points in time, with selected fitness graphs showing a good or average run from each of the different levels. It is important to keep in mind that the bugs discussed in subsection 3.3.3, the results were even out some, since the best neural networks also got stuck between two states and didn't get the chance to build up a high fitness score difference, compared to the worse neural networks. This means that the fitness score is closer than it should be, and that the general growth rate of each graph is really slow, but it became clear from the behaviour of the snake seen in the graphics for the simulation that some simulations were clearly better than the rest, and they did also achieve a clear but small fitness score advantage.

After we have looked at the runs and analyzed the network we will have a discussion about where the project is at the time it is finished, comparing it to the projects ambitions and the research goals, as well as some insights into what can be done to improve on the current system, and our views on where to go next in terms of further development. Since this is a field with little research this last section will be quite extensive, since there is a lot of potential in projects like this and it could be a valuable field for future research and technology.

4.1 Tests and results

In this section we will show the results from the actual tests completed in this project. Since there were different versions of the systems and several things we tried out, we did a total of 70 tests for this project. We did five tests in total, with at least ten runs on each test, and we think this is alright since some of the tests in reality were quite similar. The first two tests were done on the simple system described in subsection 3.3.1. Since we at this stage were focusing mostly on the Neural Network and the Evolutionary Algorithm, this test focused on testing which of the inputs were better for the Neural Network, current state or previous iterations output. The third test was on the two dimensional simulator, to see how it measured up to the simple one. The last two tests were done on the final simulator, to find out how the system worked when introducing target destination and environments into it. In this section we will start by seeing the results from the tests in Table 4.1, before we start discussing and analyzing these numbers in more detail and showing example graphs from the different tests. Before we start we should note that the units is not to scale with anything and is relative to the length of the snake. In the simulation each link of the snake was 4 units long. Since the snake has 7 joints this means the total length of the snake is $8 * 4 = 32$. This is mentioned so that it is easier to relate the actual results to something.

Test	**Number of runs**	**Highest fitness**	**Average fitness**	**Standard deviation**
1	10	20.025	18.391	2.604
2	15	26.852	23.927	3.842
3	15	33.866	31.970	3.407
4	15	13.417	11.613	2.739
5	15	23.718	21.463	2.901

Table 4.1: This table shows the results from the different tests. The values is the average over all the runs, meaning that the highest fitness denotes the average fitness score of the best scoring networks from all the runs in that test, and so on.

As a general note before we move on to look in more detail at the individual tests, we see some common traits amongst the tests. These will become even clearer once we start seeing the actual fitness graphs. The main thing we see that is not very usual when we are looking at Evolutionary algorithms is how close the average fitness is to the highest fitness, and how low the standard deviation is across all the runs. This is because of the high number

of population and generations we used. Many of the runs the system rapidly reached the highest fitness value and after that progress was slow, but about every fifth run needed this high numbers to get to a decent fitness value. This however means that the runs that doesn't improve much beyond a certain point has a lot of time to evolve the entire population, and this results in a very low standard deviation, and the average score is very close to the highest score.

4.1.1 The first test

As previously stated, the first test was done early in the project, and it used the simple simulator, while mainly focusing on the network and the Evolutionary Algorithm. Seeing the values from Table 4.1 we see that this test got the lowest score of the three first tests where it was just the simulator being tested. This was understandable, seeing as the simulator was the simple one and wasn't being the focus of the testing. Based on the values printed out at each step and also the final network produced by the system, the simulator still worked and produced reasonable networks, but the simplicity of the simulator didn't allow it to reach very high fitness values. We see from the table that the most of the fitness scores lay between 15 and 20, with a very low standard deviation. Since the simulation was so simple, more of the runs reached high fitness and this meant that it usually had more time to get the average score up, and this resulted in a very low standard deviation.

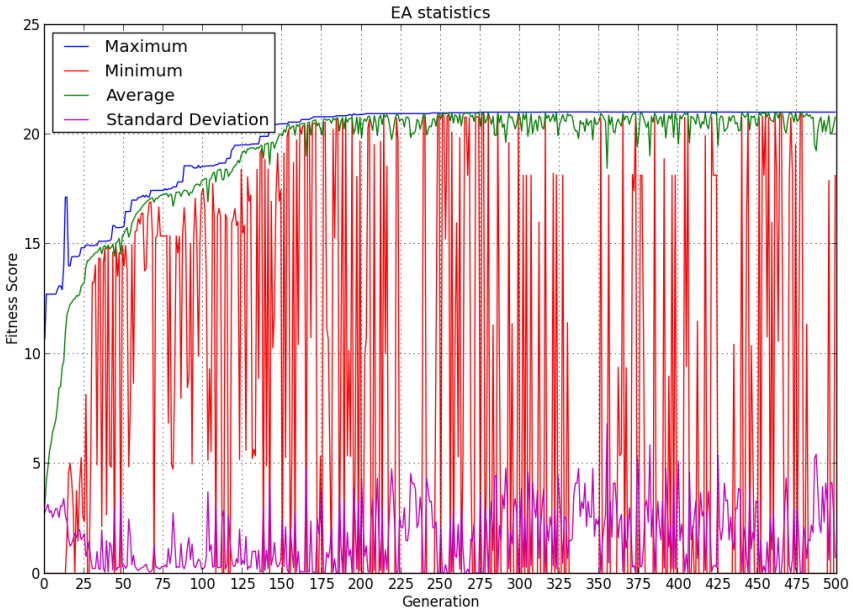


Figure 4.1: In this figure we see an average graph for the first test that was conducted during this project. We can see the average score following the highest fitness value closely, since the highest value doesn't keep increasing for long. This also means the standard deviation is very low here.

In Figure 4.1 we see an average fitness graph from the first test we ran during this project. Most runs got about the same type of graph we see in

the example one, reaching the high fitness as fast or even a little bit faster than the graph we chose to show. Since we did not have the graphics in place at this point it is very hard to analyze any more from this test, aside from the network analyzed in section 4.2, which is relatively similar to the ones evolved by this simulator.

4.1.2 The second test

The second test was done on the two dimensional version of the simulator. After serious round of bug fixes the fitness score finally managed to beat out the fitness of the first test, but not by as much as we would have liked. Analyzing the movement pattern we observed with the graphic interface we could quite easily see the reason for this. As we see in Figure 4.2, the snake changes direction every run and does not keep heading in a straight line, so since we are using the distance from the starting point as fitness score instead of the actual moved distance of the snake. This means the snake actually moved longer than the fitness score, but it still got better fitness scores than the first test.

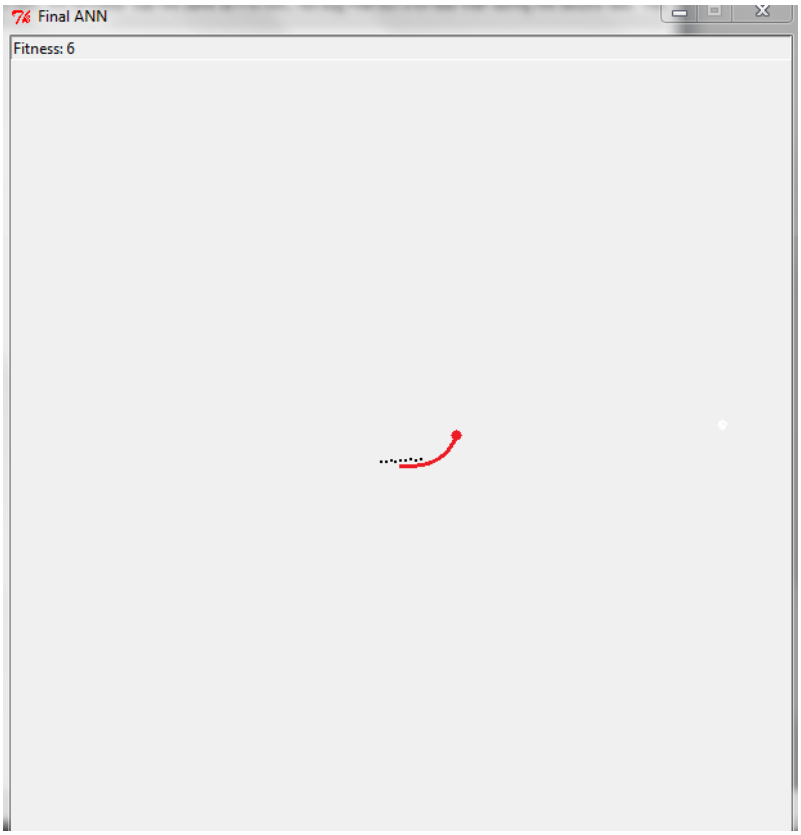


Figure 4.2: This figure shows the graphics during the second run. The red line represents a typical path for the snake to take, and the red dot is the ending point for the snakes movement.

Now on to discussing the actual values achieved during the test. This was the simulator test where we got the highest standard deviation. During this test the simulator was still pretty bugged, and many of the networks ran into

these bugs, and the average fitness could not keep up with the networks that actually worked well. The average score still follows the highest relatively well, but relative to the other tests, it's quite much farther behind. At this point the snake has also moved almost a whole snake length in the 15 time steps the simulator runs for, which is already a reasonable good result. An example fitness graph from these tests is shown in Figure 4.3.

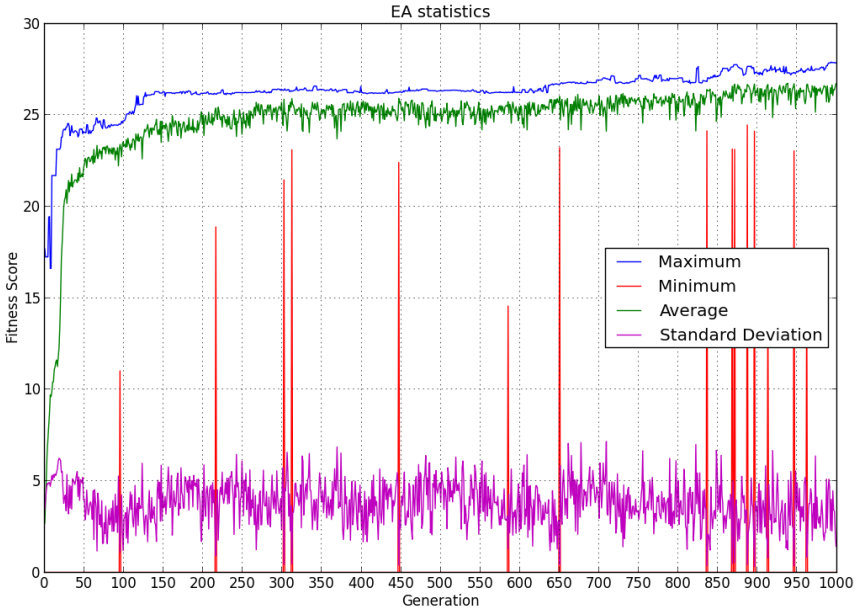


Figure 4.3: In this figure we see an average graph for the second test that was conducted during this project.

In Figure 4.3 we see that the average fitness value is further away from the highest fitness value, and the worst fitness score almost always stays at 0 instead of jumping a lot like in Figure 4.1, because of the bugs that were found and fixed for the third version of the simulator. This also means that the Standard Deviation is steadily higher than it was during the first test. Evolutionary Algorithms is quick to abuse bugs if it increases the fitness value, but once those obvious bugs are found and fixed, the evolutionary algorithms actually handle bugs in the other end really well, since they networks that fall prey to these will stay at very low fitness and therefore not have much of a chance to be picked for the next generation.

4.1.3 The third test

Moving on to the third test, the simulator was the same as it is now, not bug free but a lot less than during the second test. This is also reflect in the fitness scoring, which is quite clearly a lot higher than the previous two rounds of tests. This is actually the first time the Highest Fitness score on average throughout the test was able to move further than the snake is long, as is illustrated in Figure 4.4, where we also see the typical final orientation of the snake. It seems to always rotate slightly and not head in a straight line anywhere, as discussed in subsection 3.3.3 and subsection 4.1.2, and it means it could have moved a little bit further.

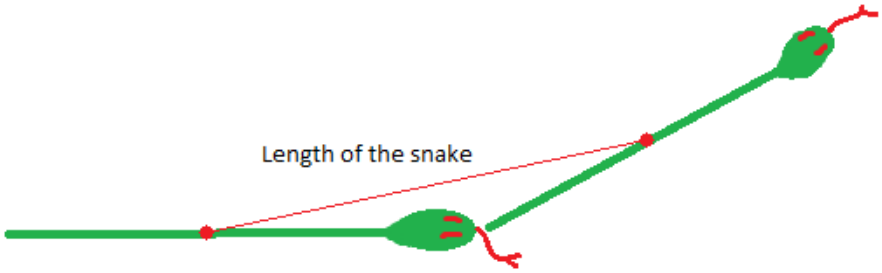


Figure 4.4: This figure shows an illustration of how far the snake is moving during the third test, and the orientation the snake had at the end of the simulation. It is an illustration to make it clearer than the dots from the graphics engine, where the distance can also be difficult to see since the field is so big.

As mentioned the numbers for the third test was quite a lot higher, and it was clear from the graphics that the snake was moving better than in the previous tests. The average fitness score is also closer to the highest one, and the standard deviation is down, which is very good since the highest score is considerably higher than the previous test. Both a sample fitness graph and a zoom-in from the graphics to show the snake moving during the simulation can be seen in Figure 4.5.

As we see in the figure the highest fitness score is now quite a lot higher than the previous tests, and it also improves more across the entire run of

(a) Zoom-in of the simulation

(b) Sample fitness graph

Figure 4.5: The following figures shows the the snake during the simulation [a] and a sample fitness graph [b] from another run during test three.

the simulation, since it is more room for improving the network when it does not bug out as fast, and actually is able to move for the entire simulation. The graphics is a sample showing the snakes state during one of the tests, and this gives an idea of how it moves even if this is just a still frame from the simulation and not an animation of it.

4.1.4 The fourth test

Now that we are done with the regular tests of the simulator it is time to move on to the most interesting tests, where we actually test the system in different situations and to accomplish different tasks. The first we look at is when we give the system a specific target to move towards. Noticeably this has by far the lowest fitness scores, and if looked at individually, the most erratic. The test was done with three different target locations, and this clearly affected the results. It could have been divided into three tests, but the difference in highest fitness value is instead shown Figure 4.6. The fact that it was so different stems mainly from the fact that the snake tried to turn in most cases and this cost it a lot of time.

Figure 4.6: This figure shows the difference in highest fitness value for the three different targets used during this test.

This sort of behavior actually brings us back to our research goals. So far the snake have mainly displayed only one of the desired four kinds of snake locomotion described in section 2.2, namely the lateral undulation, since this is the fastest on solid ground with no terrain. With the target point directly above or below it, we also saw the snake do something similar

as sidewinding, where it moved in bigger motions and did not even try to realign and face the target before moving. Since the main research goal of this project was to develop a system that made Neural Networks controlling a snake in correct locomotion patterns, this is a huge victory, bugs in the simulator aside. That we're seeing more than one type of behaviour shows that it is possible for evolutionary algorithms to evolve robust networks that handle different forms of locomotion.

Happy results aside, the actual fitness values was quite low, and relative to the other tests the standard deviation is very high. An example fitness graph is shown in **??** and we can look back to Figure 3.11 for the graphical example.

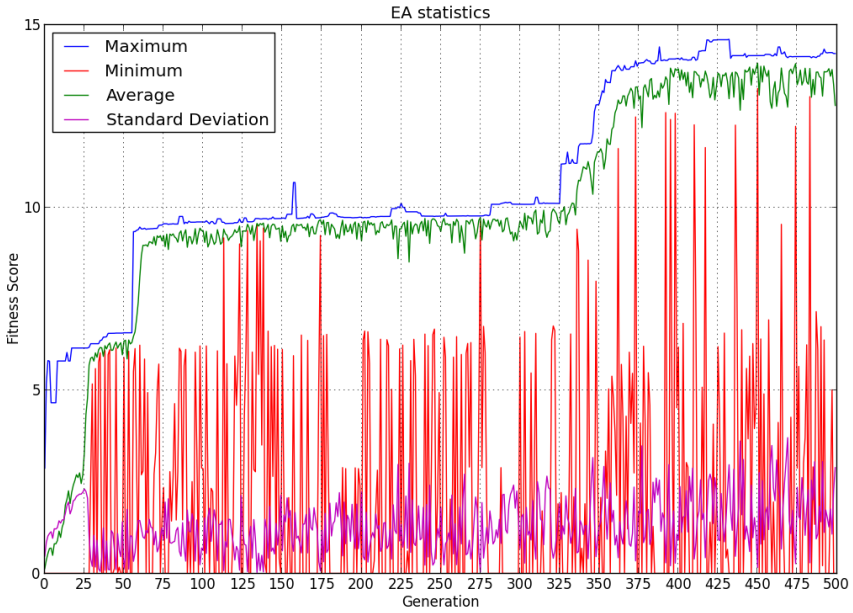


Figure 4.7: This figure shows the fitness graph from a run during the fourth test of this system.

Things to note from this figure is that we see during this test that it actually took a while for the fitness to reach the higher levels, and displays why we chose such large numbers in our population and such a large number of generations.

4.1.5 The fifth test

Now that we are done with the direction test, it is time to move on to the last test done in this project, where we test out the environmental scheme we explained in subsection 3.3.3. After we saw the fitness scores drop quite drastically for the simulator with targeting, the same is expected here, because when you add more friction to uphill movement, the snake should ideally choose other routes, and this requires it to realign, just as in the previous test. The environment scheme that was used can be seen in Figure 3.12. With this cell map scheme only the lateral undulation locomotion pattern was observed, but one can postulate that it would be able to do sidewinding if the hills were placed differently. The direction the snake moved in still means that the snake avoided the high grounds and moved through the low ground, which is what we desired, but the low standard deviation might mean the cell map was not as good as it might have been. By this I mean that the upper right low ground might have been too easy to find, and most of the networks were able to find it too easy. The fitness scores is still down as is expected when the snake have to turn while it moves, or get the penalty of the uphill movements.

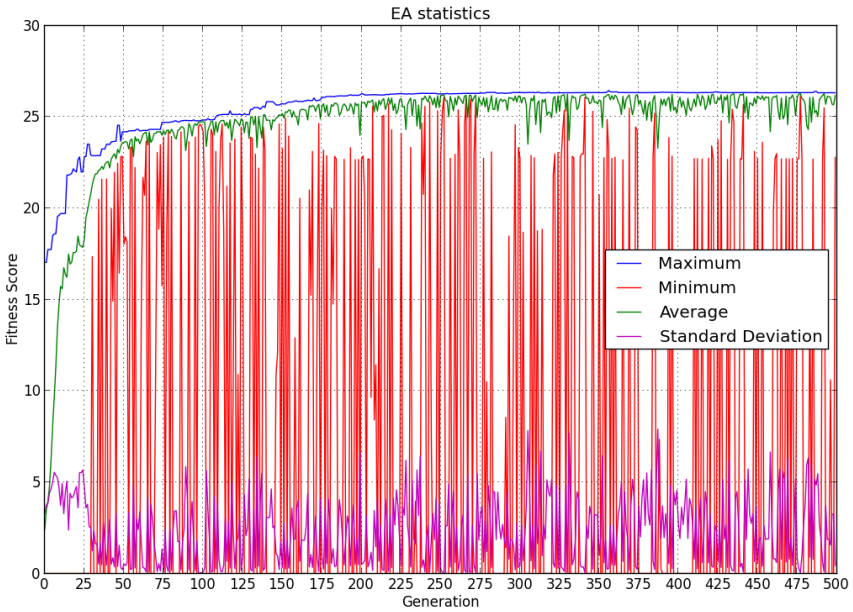


Figure 4.8: This figure shows the fitness graph from a run during the fifth and last test of this system.

As we see in Figure 4.8 this is a sample fitness graph from the fifth and

last test. We previously mentioned that the low Standard deviation might hint to that the cell map was too easy for the evolutionary algorithm to be tested sufficiently, and that the simulator might not have been tested well enough. One point to note here is that if the cell map was harder to figure out, we might have had to change the input for the Neural Network to include information about the friction of the neighboring cells, but this is purely speculations and possible solutions to problems we did not really encounter with this test.

If we look at the numbers in the sample fitness graph we notice that as expected the fitness scores has dropped quite significantly from the third test that was the pure simulator, but not as much as for the simulation with a target destination. As we have been over for some time now, the highest fitness value hits its high ground values relatively fast, faster than we expected before this test, and related to that we also see that the standard deviation is not as high as we expected before the test. This concludes the tests done during this project.

4.2 Analysis of a Neural Network

In this section we will analyze one of the networks from the third test. This was the test done on the final simulator with no additional factors added in. We will be doing the analysis bottom up, meaning we will start analyzing the output nodes, and evaluating which values from the hidden nodes triggers the output nodes. From there we will do the same with the output neurons, where we evaluate which outputs trigger the hidden neurons. The network we will be analyzing is shown in Figure 4.9. Before we begin our analysis we make a note that this is just the network for one joint. To completely analyze the resulting controller we would need to do this analysis seven times, but analyzing one network is already a big task, and the other networks in the controller would also most likely be very similar to the one we're analyzing.

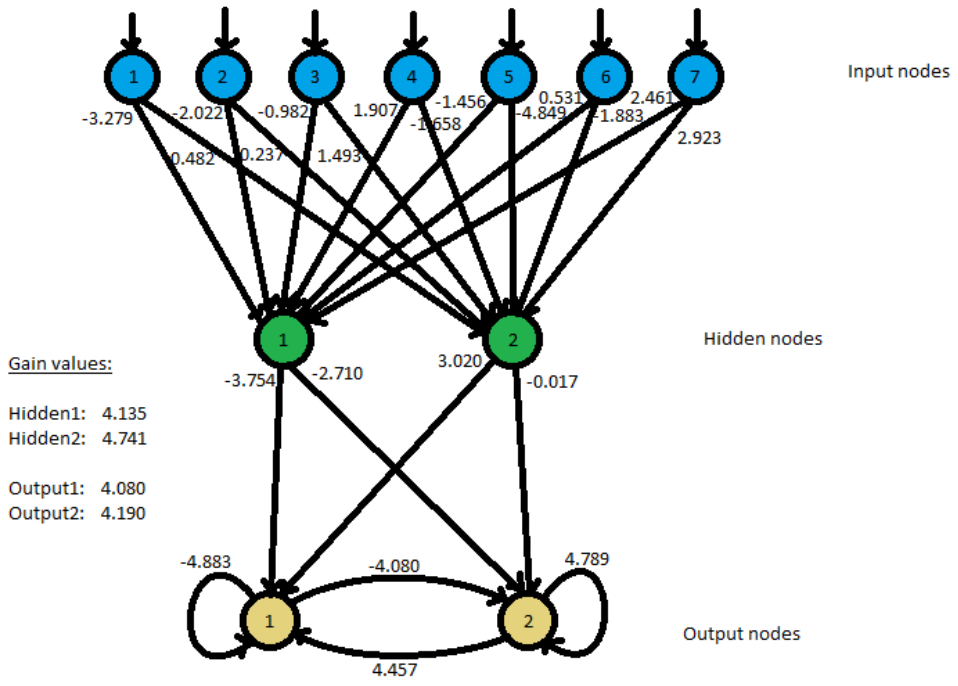


Figure 4.9: This figure shows the network from one of the runs from test number three. The weights are next to the edges, and the gains value is listed to the left of the network.

Starting on the analysis, we first look at the output nodes. The first and most obvious difference we see between the two is that the first output Neuron has a highly negative self-edge, as well as a highly negative edge to the second output neuron. The second output neuron is the other way

around, with strong positive edges to itself and the first output neuron. Before we start looking at what this means we will also include the hidden neurons to get the full picture of it. The first hidden neuron has strong negative edges to both the output neurons. The second hidden neuron has a strong positive influence on the first output neuron, while the edge to the second output neuron is practically nulled out. Keeping in mind that the output of the output neurons will never be negative, but always between 0 and 1. The snake is moved through $first\_output - second\_output$, so we are essentially interested in when one node has a larger value than the other here. This is relatively simple here, since the only scenario where the second output neuron will actually get largest is where only the first hidden output neuron fires. For as long as this happens, the state of the second output neuron is the one that will deteriorate the slowest, Since its self-edge is so large. If both or only the second hidden neurons fire the first output neuron will be bigger than the second one. If none of the hidden neurons fire, the self-edge for the second output neuron will mean the inner state here approaches 0. What this means in the big picture, is that if the second neuron fires the first output neuron will be much bigger than the second, and the joint will bend to the right. If the first hidden neuron fires strongest or both neurons does not fire, the second output neuron will be largest and the joint will bend to the left. This helps explain the behaviour of the snake, and why it has a tendency to turn before it starts moving, since the second neuron will be largest at the start, and the snake will keep bending left for a few time steps, before the first neuron takes over and they start alternating.

Moving on to the hidden neurons, and the last part of this analysis, we notice that there is a big number of input neurons, and that looking at them individually would take a long time and probably not be helpful. Instead we are going to target patterns that we see from the inputs. In Figure 4.9 the weight values is a little bit hard to see since there is so many edges, but if there is not a clearly visible minus sign in front of the number it is positive, since this is the most important thing to make clear in this analysis. We see a clear pattern forming, where the first hidden neuron receives negative input from input nodes 1, 2, 3 and 5, and positive from the remaining. The fourth and fifth input node contributes with relatively small values; so that they are reversed in the pattern does not have the biggest impact on the system. The second hidden neuron receives negative input from input nodes 4, 5 and 6 with positive contributions from the rest. The pattern that forms this way is that the first hidden neuron is firing when the back part of the snake gives larger inputs and the second hidden neuron fires when the front part of the snake gives larger input. Since the beginning of the simulation is quite interesting, we notice that both hidden neurons will receive highly negative input. At this point all the angles is set to π, since the snake is

fully extended. This means the first hidden neuron gets inputs totaling to -8.922, and the second neuron gets inputs totaling -10.226. When both are receiving this much negative input both their firing rates becomes very low.

Putting the analysis from both the layers together, we can get some sense out of the observed behaviour where the snake turned at the start and the numbers of the network. Because both the hidden layer neurons is firing very little at this stage, the second output neuron will dominate the first one and the snake will start bending to the left, which is what we also saw in the simulation. Another point we have yet to touch on is the gains. Since the gain values is so similar it has not been a big part of this analysis, but we should still mention that because of the output formula, $Output = 1/(1 + e^{-gain*State})$, the high gain values means that the state values is amplified a lot, and since the state is so negative in the beginning, this makes the second output neuron dominate the first one even more. This particular join follows the pattern that it will bend right when the back part of the snake has biggest angles, since the current angles are used as inputs to the network. The snake will bend left when the snake is extended or the front part of the snake has biggest angles. The fitness for this network was 27.154, which was far from the highest of the test, and this is also reflected in the network. While it does work to some extent this particular network will have trouble changing from side to side, since the output throughout the simulation will be relatively low, because of the lack of positive edges to the output nodes. It also goes to show that the network topology and setup still needs a lot of work, and should still have high priority second to fixing the simulator.

4.3 Further work

Since this project is very research focused its potential is not close to be reached yet, and therefore we chose to include a whole section discussing further works, and possible directions to evolve the project or make similar projects in the future. There are several things that could be improved on the current project, and there are several new, interesting areas that we want to include in this section to give a starting point for future research. The first priority is to get the simulator working completely and free from bugs, since we will need an accurate testing environment to be able to test more advances scenarios for our system, or the possible sources of errors will become too many, and you cannot quite trust the results you get while the simulator is running with bugs. It would be a shame if the networks worked well in the simulator but could not actually handle the advanced environments when put into a real scenario with an actual snake.

There is one main area of expanding that opens several more options, and it has already been touched on in the introduction. This is of course getting the simulation and controller over onto an actual snake robot. This would take reprogramming the entire system into a quicker language, and tuning it for the smaller computation power of a robot. The issue of the computation power also raises another issue with the real world application of this kind of online tuning of the controller: getting the environment information. Necessarily the simulator will need some form of knowledge about the layout of the environment. In the project so far we have been assuming that we already know this, but for the system to work completely independently it cannot make such assumptions.

One possibility is for the robot to get this through two or even only one camera. If it uses only one camera it will of course need to move a measurable amount to get more than one reference point on the environment, in order to be able to do the 3d reconstruction and get useful information about the environment. Unfortunately the algorithms to reconstruct a 3d model of the environment from 2d images are quite computationally heavy and require a lot of image processing to detect the edges in the images. Detecting either edges or the position of a set of reference points is important to be able to reconstruct the environment in 3d, and this means each snake robot would require more powerful integrated computers.

Another possibility is for the system to get the images off-line, from a different observer or robot itself, before computing the 3d model on a more powerful, external computer. From there the model would be uploaded to the robot with regular intervals so it can keep adjusting its controller to changes in the environments. This scheme would make use of distributed computing as well, in the sense that the evolution of the network is still done

by the robot, but the model is made by an external source.

Getting a more advanced environment into the simulation is definitely a big part of what remains. Applications for real robot snakes would lie outside flat surfaces and tidy environment, so even though tests conducted in such spaces is still very useful to assess the movement patterns of the robot; it is still not the major goal when talking about tests on real robots. Getting a more advanced environment into the simulator like it is now will be difficult, and redoing it with this purpose in mind might be the fastest and best way forward. At this point in the further works section we have already started crossing the artificial intelligence part of the project with areas from image processing, and it is well worth keeping this in mind as one goes forth and does more research into the area. With more complex algorithms in image processing, combined with artificial intelligence we could be able to gather larger amounts of information from images of the snake's surroundings, and this again could greatly improve upon systems evolving controllers for the robots.

The last that will be mentioned in this chapter, as we do not want to get to speculative and keep the suggestions within what we see as promising areas that can actually be improved upon relatively soon, is the navigation of the snake. During this project this was done relatively simply by changing the fitness function, but with more advanced environments this will no longer be a scheme that holds up. One possibility is making more points, either from a remote operator manually setting rally points for the snake to follow, or for the system to figure it out itself based on the observed environment. The snake would then use these points as the single point we have been using so far, so it basically goes from one point to another. Another idea for the navigation system is to actually have a different controller that controls the navigation, maybe combining the locomotion network with a separate navigation Neural Network.

Chapter 5

Conclusion

In this chapter we will take a look at where the project stands at its end through a brief summary of this paper. We will also make some concluding remarks and see how we measure up to goals stated in the introduction. This includes evaluating how well the project is able to answer the research questions we also introduced at the beginning of the project.

5.1 Summary of the project

We will use this section to briefly summarize this project. The ambitions for the project was relatively high, but they were scaled back during the run of the project, as getting the simulator working took longer than expected. We still managed to complete a system that works to some extent. The project started out with a large amount of theoretical work, where papers ranging from real snake locomotion to detailed programming of evolutionary algorithms and Neural Networks were covered. The research phase revealed that similar projects had been done before, but without the focus on more advanced target destination and environmental factors. This opens up a new path for research in the field to take, and although not a lot was tested during this project, it still gives us a scheme to do this by, and one that worked relatively well during the tests.

During the background research phase the biggest slice of time was spent on reading, understanding and extracting a usable model from the book "Snake Robots: Modeling, Mechatronics and Controls". The book explains it well and thoroughly, but the subject is advanced and the models quite complex, so it was quite time consuming setting up the actual model from it. The essentials of the models have been extracted and explained in section 2.3, and this can be a useful starting point for project seeking to duplicate and

evolve the results achieved in this project. As we discussed in section 4.3, there is definitely room for improvements on this project, and there is a lot of potential for similar projects to get good results and enable a big amount of new technology and research.

After the theoretical study we chose what software to use, which is simply Python, and a self-programmed simulator. For the graphical representation we used Tkinter, the standard graphical user interface used in Python. The choice was relatively easy to make, as even though the basics goals of the project is to explore the possibility in a simulated environment, I wanted to take a more ambitious approach. Python helps us do this through being an easy to use programming language that will not require much advanced coding, and is easy to bug fix, as well as being named after a snake, even though that was not taken into consideration when choosing Python. This would allow more freedom to make more complex environments and tasks for the snake, and having full control of the simulator also makes it easier to find bugs. It was still a project goal to test the networks on an actual robot snake, but this never happened. The fact that it was written in Python did make this less likely to happen, but I still think it was the correct choice, so we were at least able to get a simulator running.

After we were done with this we created an implementable model, so we would have a basis for the simulator. Then we chose a development methodology, spiral development, so we could start focusing on developing the actual system. From here we started implementing the early version of the simulator, as well as the Evolutionary algorithm and the artificial Neural Network. While it has not been the focus of this paper, this iteration of the development was by far the longest, since it was so much to develop, program and get working for this. The evolutionary algorithm was relatively simple to get functional, but it was still time consuming considering it is a relatively big and complex system, so it takes time to develop. The neural network was also made as part of the coding for the Evolutionary Algorithm, and since it got the snake moving in correct patterns it was not changed much after the first iteration.

What was changed a lot after the first iteration was the simulator. After the second iteration, when we finally got the graphics in place it became clear the second version of the simulator had bugs that the evolutionary algorithm abused really well, and this initiated the great bug fixing phase of the project, that lasted longer than expected. Since the system is still not completely bug free this phase was never actually completed, but we decided that we had to move on and do the planned tests with the system we had in place.

At the last phase of the project we did testing and writing the thesis. The tests gave us mostly good results, and these are explained and analyzed in

more detail in chapter 4. In section 4.3 we take a look at ways to take this project if it is resumed, or similar projects is started, and with that the project was finished, and only the last section remains.

5.2 Conclusion for the project

In this section we will conclude the project that we have worked on for a year now. We had great ambitions for the project at the start of the year, but these were scaled back when we realized the scale of the system and also when we ran into problems with bugs. We have already covered this really well, so instead of rehashing that, we will move back to the goals for the project that we stated in the introduction, before we take a look at the research questions and how well they were answered.

The first goal for the project was to make a controller for serpentile robots using an evolutionary algorithm to evolve it, and this was technically completed, but not as well as we would have liked. With the simulator still containing bugs and the Neural Network not as tested and perfected as we would like, we would not yet be comfortable presenting this solution as a fully functional controller for the robot snake. At best the networks are working well enough to be tested on actual snakes, but not good enough for much more than that. The analysis of the network confirms that the snake should display proper snake locomotion, but it would most likely not be optimal. The other goal set for our system was to introduce a changing environment into the simulation, and a target destination for the snake to be implemented, and we had greater success with this goal. The network evolved is still not the best, but the introduction of the cell map scheme and the target destination tests were relatively good. The environment still does not change during the test, but there is no logical reason the cell map could not be changed halfway through the simulation and the evolutionary algorithm would be able to handle it. Both the cell map and the directional schemes is discussed in more detail in both chapter 4 and section 4.3, and both of them definitely have more potential if worked further on, but this project still managed to include these as desired.

All in all the project goals were not met as much as we had wanted, and although that is a bit disappointing, it still shows that this type of systems have a lot of promise, and there is a lot of possible research to be done here in the future.

Moving on to the research questions posed in section 1.2 we set the level of these a little bit lower than the ambitions of the project. The main research question was if it was at all possible to evolve a Neural Network controller that can work as well as specialized controllers. Considering the ability of

the evolved networks to abuse bugs in the simulator, I believe we can answer this as yes, you can do this. It definitely requires more testing for us to be completely sure of this, but it is a good beginning and a definite step in the right direction.

We also had a secondary research question for the project, which was concerning the environmental scheme. We wanted to explore how terrain features and targeted destination could be implemented into the system, and if it would be possible to do it online for the robot. The fourth and fifth test conducted in chapter 4, shows how this was solved. With simple schemes like this it should be possible to evolve it fast enough to do it online and you can definitely have more advanced schemes if you can do the simulation off line. This is discussed in more detail in section 4.3, as well as different possibilities to make the pathing more advanced to get to the target destination. This answers the research question we started with relatively well, that this is both possible and a very good way to go about producing systems to handle these kinds of changing environments.

In terms of the research questions the secondary one got the most satisfying answer, but both were answered to some extent. As a concluding remark, this project was a moderate success. The results were not groundbreaking, but it shows that the field has a lot of potential to create very well performing systems, even if the one we have here is only partly working.

Bibliography

[1] Pål Liljebäck, Kristin Y. Pettersen, Øivind Stavdahl, Jan Tommy Gravdahl. *Snake Robots: Modeling, Mechatronics and Control*. 2011. Springer-Verlag London 2013.

[2] Alireza Fasih, Jean Chamberlian Chedjou, Kyandoghere Kyamakya. *Cellular Neural Network Trainer and Template Optimization for advanced Robot Locomotion, Based on Genetic Algorithm*. December 2008. University of Klagenfurt, Austria. Available at: <http://ieeexplore.ieee.org/l>

[3] Haruhiko Asada, Sheng Liu. *Transfer of Human Skills to Neural Net Robot Controllers*. 1991. Massachusetts Institute of Technology. Available at: <http://ieeexplore.ieee.org/l>

[4] Prabir K. Pal, Asim Kar. *Mobile Robot Navigation Using a Neural Net*. May 1995. Bhabha Atomic Research Centre. Available at: <http://ieeexplore.ieee.org/l>

[5] Paolo Arena, Luigi Fortuna, Marco Branciforte. *The Mechanism of Locomotion In Snakes*. February 1999. Available at: <http://ieeexplore.ieee.org/l>

[6] Teruo Fujii, Tamaki Ura. *Soncs: Self-Organizing Neural-Net-Controller System for Autonomous Underwater Robots*. 1991. University of Tokyo, Japan. Available at: <http://ieeexplore.ieee.org/l>

[7] J. Gray. *The Mechanism of Locomotion In Snakes*. May 1946. University of Cambridge. Available at: <http://jeb.biologists.org/content/23/2/101.full.pdf+html>

[8] Anders Kofod-Petersen. *How to do a Structured Literature Review in computer Science*. April 16, 2012. NTNU. Available at: <http://www.idi.ntnu.no/emner/it3708/lectures/notes/SLR_HowTo.pdf>

[9] Coppelia Robotics. *V-rep: Virtual Robot Experimentation Platform*: <http://www.coppeliarobotics.com/>

[10] Cyberbotics. *Webots: fast prototyping and simulation of mobile robots*: <http://www.cyberbotics.com/overview>

[11] Dario Floreano, Claudio Mattiussi. *Bio-Inspired Artificial Intelligence: Theories, Methods and Technologies* .2008. The MIT press, Cambridge MA, U.S.A.

[12] Dale Purves, George J. Augustine, David Fitzpatrick, William C. Hall, Anthony-Samuel LaMantia, Leonard E. White. *Neuroscience fifth edition* .2012. Sinauer Associates, Inc. Sunderland, Massachusetts, U.S.A.