Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

Evolving Locomotion for a Humanoid Robot

Bachelor Thesis
im Arbeitsbereich Knowledge Technology, WTM
Prof. Dr. S. Wermter

Department Informatik
MIN-Fakultät
Universität Hamburg

vorgelegt von
Heye Vöcking
am
04.03.2013

Gutachter: Prof. Dr. S. Wermter
Dr. S. Magg

Heye Vöcking
Matrikelnummer: 6139373
Wiesendamm 135
22303 Hamburg

Acknowledgments

Thanks to Sven Magg and Marc Bestmann for their DarwinFramework which formed the basis for the software developed in this work. Furthermore, I want to thank Tayfun Alpay, Ta Dejsuwannachai, Molly Brady-Martin, and Adam Whitticker for their help and advice during the revision. And thanks to Reinhard Zierke for helping with technical issues regarding the Webots license server etc. I also want to thank the whole Knowledge Technology group to be appreciative of the discomfort with the simultaneous occupation of up to 16 computers at the same time for experiments. And last but not least, I would like to thank everybody I have talked to about this work for the help, the solace, and the fruitful discussions that were sometimes necessary to get the project going again when it was stuck.

Abstract

The purpose of this bachelor work was the evolution of artificial neural networks to develop locomotion for the DARwIn-OP robot. The DARwIn-OP, henceforth referred to as Darwin, is a 45cm tall humanoid robot which is used, amongst others, in the RoboCup for robot soccer.

The main problem in robot soccer is creating a robust and fast locomotion. Since a humanoid robot is a very complex system, it is difficult to handcraft a robust walking algorithm. Furthermore, it needs to be adjusted by hand if the floor or the weight distribution of the robot itself is changed.

One approach to automatically developing a walking algorithm is based on biological evolution, by which a gradual improvement of individual solutions can be achieved over many generations. Its parallel nature and pragmatic approach to solve problems makes artificial evolution a well suited solution for this task. But evolution too has certain difficulties which must be overcome. For example, tens of thousands of experiments need to be performed in order to find a good solution in a complex search space.

In this work, a system was developed, which exploits the concurrency offered by evolution and performs the experiments in the Webots simulator on several computers in parallel, thereby finding solutions in a reasonable amount of time. It used an accurate replica model of the Darwin to evaluate the solutions, which makes the transfer of a suitable solution to the real Darwin robot realistic.

This work focuses on the oscillating pattern generation within the artificial neural network (ANN) and by external sources, as well as the impact of neurons in the hidden layer of the ANN. The experiments have shown that an ANN is able to generate a pattern without the use of a central pattern generator. Furthermore the results indicate that at least four neurons in the hidden layer have to be present for a locomotion to evolve.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Due to the fact that humanoid robots are built to operate in an environment adapted to the human body, the research field of humanoid robotics is quite large and complex. One of the most important problems is stable locomotion, which enables the robot to move from one point to another without falling over or getting stuck. A number of different techniques for locomotion exist. Wheels are very popular because they are fairly easy to use on plain ground. But they are not very versatile for other kinds of undergrounds or environments. Stairs for example form obstacles for wheels in many man-made environments. Wheels haven't been evolved in nature; instead evolution has come up with different kinds of locomotive means, such as the establishment of wings, fins, and legs. In legs, for example, evolution has adapted a bipedal walking mechanism, which, considering the success of the human race on the planet, is quite an efficient type of locomotion.

Since this work focused on artificial bipedal locomotion, it is appropriate to examine human locomotion at first: A rhythm-generating system is located within the spinal cord and the brain which generates a pattern used as basis. This system is influenced by input from 'higher levels' of the brain and receives sensory feedback from the muscles, joints, and skin of the legs [30, 6]. The generated pattern produces a cyclic movement of the legs, as described in Figure 1.1. How the sensory information is transmitted and processed is explained in Section 3.3.

Following the human example robots in humanoid soccer leagues are forced to use two legs for locomotion. This faces researchers with the challenge of developing algorithms for bipedal locomotion, which is the most important function in humanoid robot soccer. If the robot can only move very slowly and unstably, and has a tendency to fall over or an inability to change its position at all, the work done by the other modules is meaningless because the robot is not able to fulfill the required tasks.

In summary, we have a complex humanoid robot and need a solution that is:

Stable The robot does not get stuck and uneven or sloped ground does not affect the locomotion to the extent of making the robot fall.

Robust The locomotion stays stable when facing unexpected incidents, like obstacles on the ground or pushing from other robots.

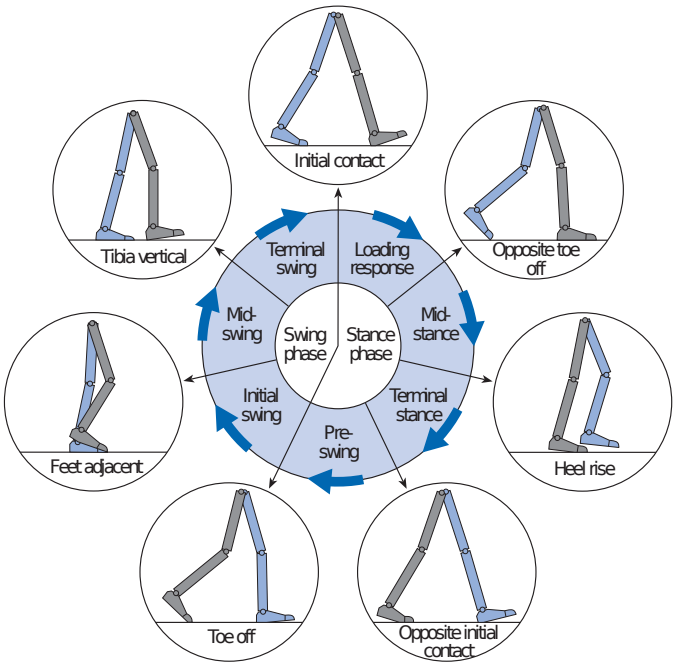

Figure 1.1: Position of the legs during a single gait cycle by the right leg (gray). Figure taken from [30].

Fast While being careful to satisfy the requirements mentioned above the speed of the locomotion should still be reasonable.

One way to achieve this is to handcraft a solution, but this takes a lot of manual work. At this point handcrafted solutions have reached their limit because they aren't easily adaptable to changing morphology or environments.

1.1 Motivation

In order to overcome the handcrafting advantage of artificial was taken intelligence which can be used to find solutions, even when no target solution (e.g. training data) is available. The approach chosen in this work is artificial evolution. The advantage of evolution is that the problem itself doesn't need to be solved by hand, rather one only needs a so-called fitness function, which rates the quality of a solution. In addition, the execution of the required experiments can be parallelized very well.

At first a domain and a search space which can be explored by an algorithm have to be defined in order to automate the search for a walking gait. As others have shown, see Section 2, parameters of an ANN can span up the search space which can then be explored by an evolutionary algorithm. In this work that knowledge was used to develop an automated system for evolving locomotion by using artificial evolution and further researched, how suitable this approach is for the simulated model of a real world robot.

Training of ANNs is basically the approximation of a function which gets input data, the sensors in our case and provides output data, the position of the servos. This is basically the same approach as in human locomotion where the nerve cells process the input from the receptors to decide how the muscles responsible for the walking should contract.

There are different approaches to design and train ANNs. In this case, the ANN was designed without information on how to solve the problem; it was simply evaluated on how well it managed to fulfill the given task. The training was performed with artificial evolution. As mentioned before it requires no training data and the definition of the fitness function, with the exception of the difficulties explained in Section 3.2.3, is basically straightforward. Furthermore, artificial evolution can find solutions that one has never thought of[1]. Sometimes these are very strange, but some might even have an advantage over handcrafted solutions.

This thesis will discuss the current state of the art technology in Chapter 2, explain the approach taken and the tools used in Chapter 3. The experiments performed and their results will be presented in Chapter 4. In Chapter 5 a conclusion of the results will be drawn. Finally an outlook for future work will be given in Chapter 6.1.

[1] Orgel's Second Rule: "Evolution is cleverer than you are."

Chapter 2

Related Work

Bipedal walking is a difficult task and has therefore been an active research field for over 45 years now, starting with Miomir Vukobratovićs Zero-Moment Point method [29], commonly abbreviated as ZMP, which was introduced in January 1968 at "The Third All-Union Congress of Theoretical and Applied Mechanics" in Moscow. About 20 years later another method called limit cycle walking was established following McGeer's initial studies on passive dynamic walking [21]. In this section the current state of the art technology as well as conclusions of other scientific publications will be discussed.

2.1 The State of the Art in Locomotion

Research in the area of limit cycle walking was done amongst others by Garcia et al. [9] and Goswami [11]. The limit cycle walking is an open loop concept, where no sensor or feedback data is provided and solely a pattern generator is used to establish a locomotion. It simply follows the pattern, which allows a faster and more natural looking locomotion, but it allows unstable states, so if the robot stops walking while being in an unstable state it would fall down. Since no feedback is incorporated the concept needs an unchanging reliable environment. While many teams in robot soccer are using this method, it does not include the feedback needed for a stable walk. In soccer games the robot may be pushed by other robots which, would require a countering action that cannot be accomplished when no external information is fed into the walking system. In order to react to unexpected changes a closed loop concept was used in this work, using the provided accelerometer and gyroscope.

Work on setups using a central pattern generator (CPG) with and without additional input from the robot's sensors has been done by a number of researchers, for example by Collins and Richmon [3], Kimura et al.[17, 18], as well as Ijspeert on a simulated salamander [15]. Experiments using a CPG have been performed for this work, but the results of these were sobering.

Bipedal locomotion is a field of interest in the commercial as well as scientific sector. Publications for real world robots by research groups of companies include

Hondas Asimo [14] and Sonys SDR-4X [19], and on the other hand projects in scientific institutions, like the KHR-3 [31] and the HRP-2 [16]. All these use the ZMP-based control. The upper body can be thought of as a pendulum weight that should be kept stable, which is accomplished using the ZMP method. In contrast to limit cycle walking ZMP does not allow any unstable state, therefore the robot could stop the walk at any time without falling down. The position of the ZMP indicates whether the robot is stable or will fall over. It needs a planar ground and sufficient friction to work. Furthermore, foot contact sensors are necessary for the ZMP method. The Darwin model used does not offer foot contact sensors, so the ZMP concept could not be used in this work. This makes the development of a stable locomotion more complex. The approach in this work was to overcome the missing foot contact sensors by using ANNs and the sensor data from servos, accelerometer and gyroscope. Unstable states during locomotion were allowed during evaluations of solutions in this work.

Besides ZMP and limit cycle walking there have also been some approaches using ANNs as controllers. These are usually trained on simulated robots instead of real robots. This is due to the high number of evaluations during the learning phase. Experiments in simulated environments have been performed on numerous setups: On constructed robots without an upper part of the body [26, 25, 27], with quadruped [28, 22, 12] or even more legged robots [1]. Where Reil [26], Paul [25] and Solomon [27] used the rigid body dynamics simulation software developer's kit of MathEngine. Tellez [28] used Webots (the same simulator as in this work), which calculates the physics with the Open Dynamics Engine (ODE). The ODE is also responsible for the physics calculation in the Simulators used by McHale [22], Heinen [12], and Asif [1].

As Paul [25] has shown, it is possible to generate a gait without a hidden layer, when input from additional sources is available. They used (for left and right respectively): waist orientation, difference between waist and foot sagittal position, waist height, and foot contact sensor. The robot used was a two legged construct without an upper body part. In contrast to the simulated model, there is only limited sensor information available in the work done for this thesis, because the model of a real platform was used, which only provides the angle positions of the servos and the gyroscope and accelerometer values in x, y, and z direction. Additionally, the Darwin is a humanoid robot, hence not only the legs, but also the whole body is simulated, which, due to the different morphology and higher complexity of the platform, requires a completely different walking gait.

Ouannes et al. discussed in their work [24] that it is possible to evolve locomotion for a humanoid robot. The robot they used is a theoretical model of which no real world version exists. So it is unknown whether the robot used by Ouannes would be able to walk in reality or could even be practically realized to match the simulated model. Furthermore, when using a fictive robot, it is possible to adapt its morphology to the needs of the neural network. None of this was possible in this work, because the simulated model is bound to its real world counterpart.

An anticipated transfer of an evolved solution to a real robot was the goal of the work done by Glette et al. [10]. They evolved locomotion as well as morphology of

the robot, which is then produced to match the evolved model. Since the Darwin model used in this work was already given, it was not possible to adapt it to the needs of the evolution and the ANN.

In summary, the morphology of the platform used in this work is more complex, because it is already predefined by the real robot. Therefore, the body has to offer more space for the battery slot, the controller board etc. This also affects the weight distribution. But not only the internal space requirements and the predefined weight distribution add to the platform's complexity; the joints also have to match the servo requirements just like their real world counterparts do. This constricts the free moving space considerably. All these limitations combined, make the task of learning to walk a lot harder. In order to find out how a compensation for the missing sensors and the given morphology can be achieved on a realistic platform, the importance of pattern generation and number of neurons in the hidden layer are analyzed in this thesis.

Chapter 3

Approach

In this chapter the tools, techniques, and the process of executing the experiments is explained, as well as their parameters.

3.1 Evolution

3.1.1 Natural Evolution

Charles Darwin first described the theory of evolution in 1859 in his work The Origin of Species [4]. It basically says that a population is adapting to the environment by altering the traits of individuals over generations. The adaptation is driven by the need for reproduction, so the traits of the individuals that manage to reproduce themselves in high numbers are more likely to survive over many generations.

Charles Darwin was not able to describe how the characteristics were passed from parent to child, but a few years later in 1866 Gregor Mendel discovered laws to describe this inheritance [23].

In fact these characteristics are encoded in a long string of deoxyribonucleic acid (DNA), also called the gene, which can be found in every nucleus of every cell. In 1953 Watson and Crick discovered how this genetic information is stored, and since then the process of inheritance can be explained in very fine detail. When this genetic information is passed from a parent to a child some of it is mutated, allowing the child to have slightly different characteristics. If these are advantageous in the competition with other individuals in a population, the child's traits are more probable to be passed on to even more children. Therefore, the weak individuals have a higher probability of dying early and producing less offspring because they are not able to compete with the fitter ones. This leads to a population with individuals that are very well adapted to the environment.

Now inspired by Charles Darwin's principle of Natural Selection [4], computer scientists have developed an algorithm that optimizes a solution for a given problem. This is accomplished by artificially selecting the best performing individuals in a population, which are then reproduced and mutated over many generations.

In order to use evolution the problem has to be transformed into a certain way, so it can be solved using a genetic algorithm in this case artificial evolution.

3.2 Evolutionary Algorithm

According to Eiben and Smith [7] the six most important components for an evolutionary algorithm are the population, the definition of the individuals, the fitness function, the selection mechanism, the mutation operators, and the survivor selection. These will be explained in detail in the following sections.

Artificial evolution is based on a genetic algorithm, which needs some kind of gene where the information of its owner is stored. The requirements for this gene are that every value that is mutable can be modified without breaking the code and a small mutation changes the execution by a very small amount. The main reason for the latter is that if a small change in the genetic code would cause an unpredictable big change in the behavior, it would be impossible to gradually improve the previous solution.

The benefits of using artificial evolution are that it tends to find solutions we have never thought of. Furthermore, it is reusable for different problems, be it a changed morphology, a different environment, or simply a slightly altered task. But there are also certain drawbacks: A large number of experiments have to be performed which requires a lot of processing time, and the fitness function has to be designed very carefully to provide a smooth increase of fitness towards a good solution, also called a local minimum.

3.2.1 Population

The population is the set[1] that contains all current solutions of the evolution, all currently alive individuals. An individual is immutable, it is the population that changes when weak individuals are removed and replaced by the offspring of the fitter individuals. For this work, the population had a fixed size during the whole evolution. It was formed by individuals arranged in a two dimensional grid with a size of 7×8 and therefore hosting 56 individuals.

3.2.2 Individual

An individual carried a gene that held the structure of the ANN that was controlling the servos of the robot, so that the ANN was able to move them according to the network output. In this work a gene (also called the genotype) was a one-to-one mapping to an artificial neural network (the phenotype), hence an individual was defined only by its network.

[1]The term *set* is not stringently equivalent to the mathematical term, because in very rare cases two individuals can carry the same gene due to a coincidence of no loci being mutated or two mutations creating the same gene which can happen in a discrete system.

The gene was represented by a long string of loci that stored all attributes of the ANN. Each of these loci held information about one parameter of the ANN. The information stored included the current value, a minimum and maximum for the range of the parameter, and a Boolean field indicating whether this parameter could be mutated or not. The different parameters will be explained in Section 3.3.1.

3.2.3 Fitness Function

The fitness function is the core of every EA. It defines the quality of a solution, so it provides an absolute order over all genotypes. This necessitates a careful design and in this case a lot of preliminary work including the criteria for the termination of an evaluation.

Four different fitness functions have been used in the experiments, namely F1, F2, F3, and F4. They differ in multiplication factors for the rewards and in their termination criteria. The fitness values of two different fitness functions are therefore not comparable with each other. The termination criteria for an evaluation were:

1. The robot's y position got below 17cm (for F1-F3), and 27cm (for F4).
2. The robot was killed by the circle-of-death, as will be explained in Section 3.2.3. (Only included in F3 and F4)
3. The time limit of 20 seconds (640 frames) was reached.

Criteria 1. and 2. were penalized by subtracting 0.74cm from the traveled distance.

The fitness function consists of five individually weighted parts, the general form is:

$$fitness = \omega_1\alpha + \omega_2\beta + \omega_3\delta + \omega_4\Delta + \omega_5\tau$$

rewards for servo movements (α and β), rewards for distance traveled (δ and Δ), and the running time of the evaluation (τ).

A calculation of the ANN was done every 32ms, each of these calculations is called a *frame*. During the execution data was collected, including the coordinates of the robot and the servo angles measured every frame. The data was then analyzed. In the following formulas N describes the number of frames, $P \in \{0, 1\}$ describes the penalty: 1 if penalized, 0 otherwise.

The treatment of the servo movements was quite complex. Basically if there was continuous movement, β was 1. Depending on the efficiency of this movement, α was 1.5 (small movement) and 0.0 (big movement). The details of the formulas are listed under Appendix B.1.

Δ is the integrated distance:

$$\Delta = \sum_{i=1}^{N} \sqrt{(x_{i-1} - x_i)^2 + (z_{i-1} - z_i)^2}$$

δ is the absolute distance after penalty subtraction:

$$\delta = \sqrt{(x_0 - x_N)^2 + (z_0 - z_N)^2} - P * 0.74$$

τ is the logarithmic runtime:

$$\tau = \ln(N)$$

For F1:

$$fitness = 2.0 * \alpha + 2.0 * \beta + 3.0 * \delta + 15.0 * \Delta + 1.4 * \tau$$

For F2 - F4:

$$fitness = 2.0 * \alpha + 4.0 * \beta + 3.5 * \delta + 19.25 * \Delta + 1.4 * \tau$$

Circle-of-death

The circle-of-death (COD) was introduced to encourage the development of locomotion. It expands around the starting point with a velocity of $\frac{1}{20}$m/s. As soon as the robots position was within this circles range, the experiment was terminated. This criteria was checked every frame, after the simulation had been running more than one second.

3.2.4 Tournament Selection

Different kinds of selection operators can be found in literature, which are applied to remove less fit individuals from the population and fill their spots with offspring created by the more fit individuals. The selection mechanism chosen in this work is called tournament selection, because it does not operate on all individuals in the population, but only on a small portion of it, four in this case. These four individuals competed against each other in a tournament. While the evolution was running, individuals were waiting for their evaluation. As soon as anywhere on the population grid all individuals in a 2×2 field had their fitness assigned, a tournament was performed. The two winners replaced the two losers with mutated offspring of theirs. In Figure 3.1 a visualization of a 7×8 population can be seen, while a tournament was performed within the red square.

The tournament selection was chosen over, for example ranking selection due to the fact that tournament selection is independent of the rest of the population. This decreased the waiting times for evaluated individuals, which arose if some of the parallel evaluated experiments were delayed. Furthermore, premature convergence was avoided, because a leading individual could only spread its offspring to a small area in the population and it took longer for this solution type to spread through the whole population.

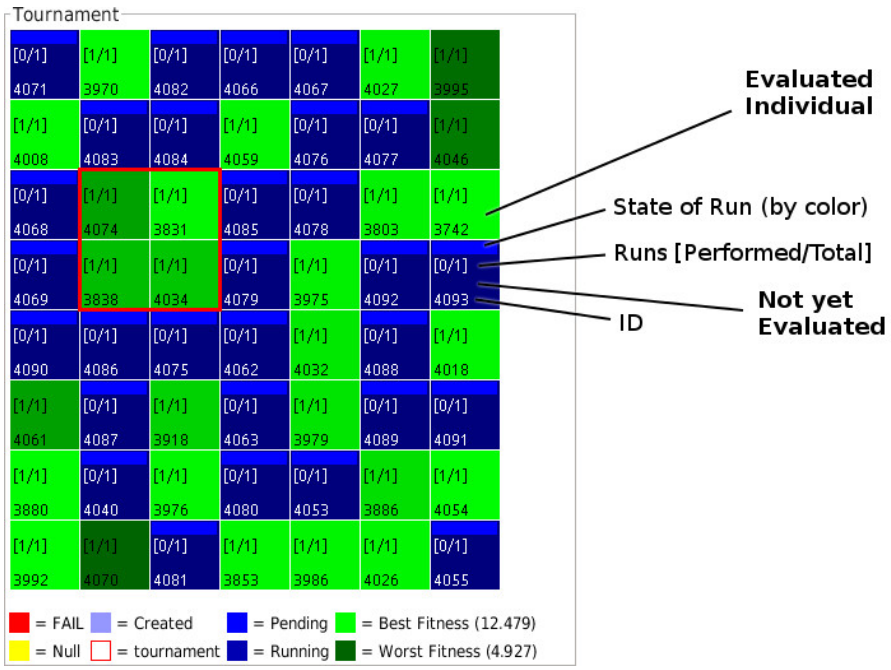

Figure 3.1: The population visualized in the graphical user interface (GUI) of the Client-Server Architecture while a tournament was performed within the red square.

3.2.5 Mutation

A mutation operator was applied to one genotype and produced a slightly different (mutated) version of it, which is considered a child or the offspring of the input genotype (its parent). A gene can also be thought of as one position in the search space, described by a vector with the length of all mutable parameters. The child gene $\vec{g_c}$ was created by copying the parent gene $\vec{g_p}$. Some of the entries of $\vec{g_c}$ were then modified by the mutation operator, as explained below. This modification is in fact a translation from the position of $\vec{g_p}$ to another position in the search space. Through mutation, the population "explored" the search space spanned by all mutable parameters.

For this work a mutator called *ConservativeGaussMutator* was developed. It modifies any mutable locus with a probability of 0.2 using the *ConservativeGaussMutator*. The grade of mutation depends on the allowed range of the loci value and is randomly chosen to be performed in three separated discrete cases with different probabilities. It adds with probability

P=0.6 a uniformly drawn number in the range from -1% to 1%.

P=0.2 a uniformly drawn number in the range from -4% to 4%.

P=0.2 a Gaussian drawn number.

The result was then shifted with a mirror effect at the range boundaries in case

it has lain outside the loci's range. The shifting process was applied as long as the new parameter value v was outside the range.

- If it is smaller than the minimum, $v = 2 * minimum - v$.
- If it is greater than the maximum, $v = 2 * maximum - v$.

Due to the probability of modifying the value of a locus by a Gaussian drawn number, it is basically possible to reach every location in the search space. Since most locations in the search space evaluate to a low fitness, large jumps destroy a good solution. In order to explore more conservatively the possibility of small steps was chosen to be higher than that of big jumps.

3.2.6 Termination of the Evolution

In most runs the evolution ran into a local minimum, usually a stable position where Darwin robot (3.4.2) walked one step and kept standing until being killed by the circle-of-death. A run was stopped when seeing no oscillating pattern after 1000 tournaments had been performed. In some cases the experiments were run longer for control sampling. When the algorithm evolved some kind of oscillating pattern this usually indicated the possibility of evolving locomotion. In these cases the fitness improvement lasted much longer than in the converging runs and therefore the algorithm was run longer. Since the cases of a walk were rare and in some cases improvements were made continuously even after an extensive amount of tournaments, no good point to stop the experiment could be evaluated.

3.3 Neural Networks

As the name states, a neural network is basically a number of neurons, connected to each other and thereby forming a network. The connections are established by axons, long arms that connect through a synapse to the dendrite of another neuron which transmits the information into the cell body. A neuron can be thought of as a simple signal processor that can communicate with other neurons through their connections. An illustration of a neuron can be seen in Figure 3.2

Neural networks form the nervous system and appear with high density in brains and spinal cords of humans and animals. The human brain for example is made up of about 85 billion neurons, connected with $10^{14} - 10^{15}$ axons [13]. These control, amongst others, the muscles in the legs. The information for controlling the legs depends on the sensory information as well as a pattern generated independently of it.

Sensory information is collected by the sense organs which stimulate the so-called sensory neurons. They transmit information about positions of the joints, the tension in the muscles, as well as sense of movement and balance to the brain.

The legs are controlled by so-called motor neurons. These arise in the brain, most notably in the motor cortex and are called upper motor neurons. They pass

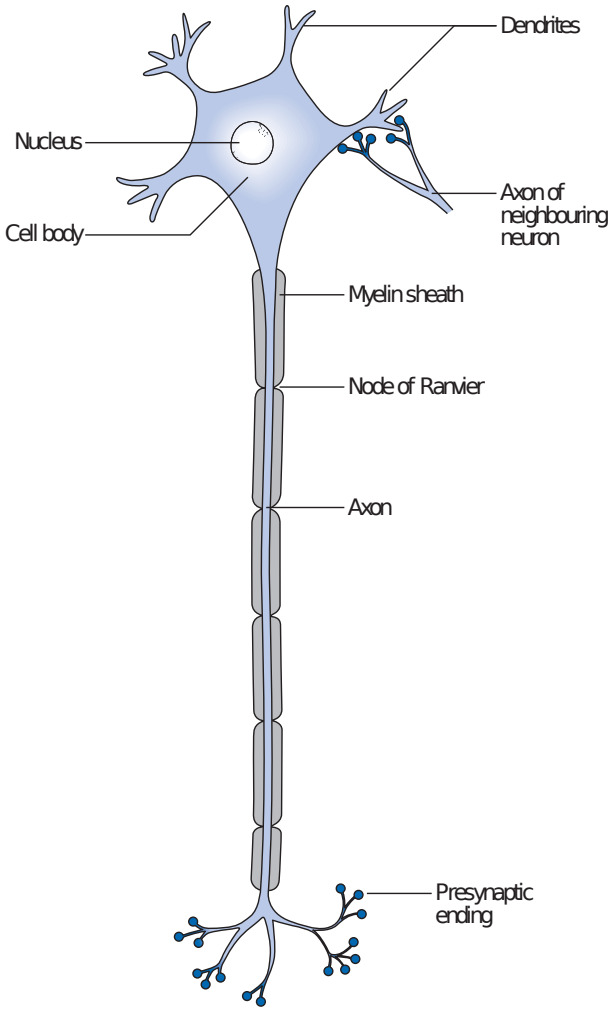

Figure 3.2: Structure of a neuron. Figure taken from [30].

the control signals down the spinal cord, to the peripheral areas where the lower motor neurons are located.

Pattern generators are located within the spinal cord itself and in the brain. The pattern and the signals from the upper motor neurons are passed to the lower motor neurons, which are connected to the muscle fibers that contract according to the transmitted control impulses [30].

The counterpart to the neural networks found in the nervous system are the bio-inspired artificial neural networks (ANNs). An ANN consists of artificial neurons, which are connected by weighted connections (as counterpart to the dendrites). It can be thought of as a directed graph with vertices (neurons) and edges (connections). A neuron is basically a transfer function σ that gets an input signal x and provides an output signal $o = \sigma(x)$. This output signal is multiplied with weight w of a connection (a floating point value) that can be positive or negative.

The neurons of an ANN are structured to form multiple layers, the first layer contains the input neurons (like the sensory neurons) and the last layer contains

Parameter	Min	Max	Mutable	Mutator
τ	1.0	100.0	true	*ConservativeGaussMutator*
bias	-1.0	1.0	true	*ConservativeGaussMutator*
weight	-5.0	5.0	true	*ConservativeGaussMutator*

Table 3.1: Details of the parameters for the CTRNN neurons.

the output neurons (counterpart to the motor neurons). In between can be a number of hidden neurons. The ANNs used in this work had zero to ten hidden neurons.

The input for this ANN was made up of servo angle, pattern generator, and sensor values. This information was scaled by the input neurons and distributed to all neurons connected to them. In the neurons the information was computed and then transmitted to other neurons until it reached the output neurons. The positions of the servos were then modified according to the activation of the neurons in the output layer.

Many different types of ANNs exist. For the experiments in this work a network with recurrent connections was used, called continuous-time recurrent neural network, or in short CTRNN.

3.3.1 CTRNN

Beer [2] describes the general form of a CTRNN neuron as follows:

$$\dot{y}_i = f_i(y_1, ..., y_N) \equiv \frac{1}{\tau_i}(-y_i + \sum_{j=1}^{N} w_{ji}\sigma(y_i + \theta_j)) \quad i = 1, 2, ..., N$$

where y is the state of each neuron, τ is its time constant ($\tau > 0$), w_{ij} is the weight of the connection from the j^{th} to the i^{th} neuron, θ is a bias term, and $\sigma(x)$ describes the activation function, see 3.3.2. In Beers original description, there is also a static input I_i for each neuron. But in the CTRNN used for this work, I is replaced by the input from the neurons in the input layer, which are only used to scale the sensor input and pass it to every non-input neuron in the network.

The aforementioned parameters are encoded in every individuals gene. They can be mutated upon reproduction, see Table 3.1 for details.

CTRNNs are the ANNs of choice here, because according to Beer

> "(1) they are arguably the simplest nonlinear, continuous dynamical neural network model; (2) despite their simplicity, they are universal dynamics approximators in the sense that, for any finite interval of time, CTRNNs can approximate the trajectories of any smooth dynamical system on a compact subset of $\mathbb{R}''$ arbitrarily well; (3) they have a plausible neurobiological interpretation, where the state y is

often associated with a nerve cells mean membrane potential and the output $\sigma(y)$ is associated with its short-term average firing frequency" [2].

This makes them very applicable for the problem stated in this thesis, because they are simple, offer a plausible neurological interpretation, and can approximate any dynamic system, which is needed to generate a walking gait.

The architecture of the networks used for the experiments is depicted in Figure 3.3. The input was coming from servo sensors, accelerometer, gyroscope, and sinus; the output was interpreted as position values for each servo. Not all input and output neurons were used in every experiment, they varied as described in Section 4.1.

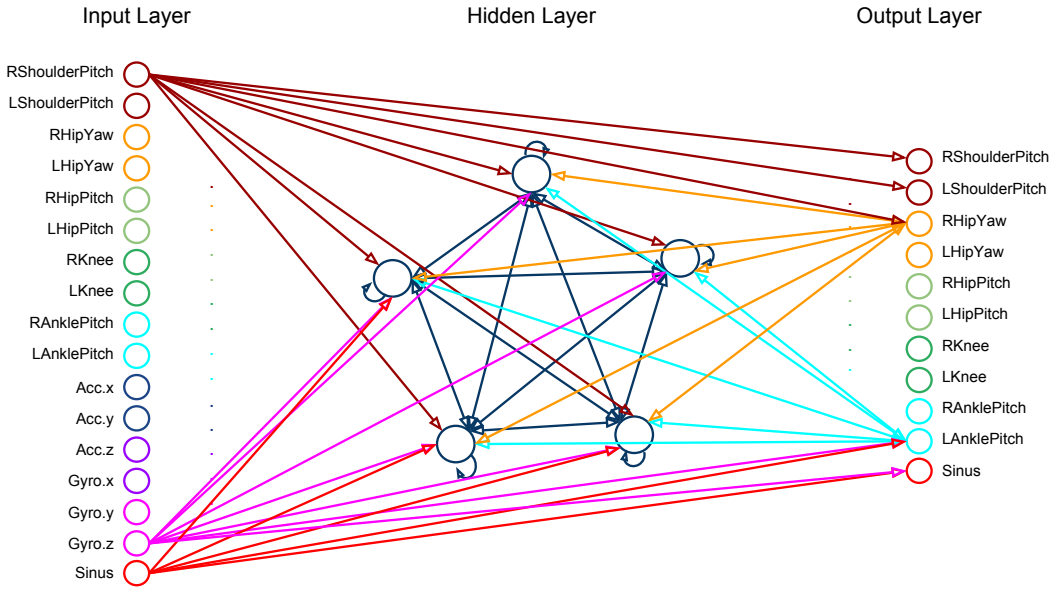

Figure 3.3: The connections are only exemplary; the network is fully connected (except that the input neurons have no incoming connections and the output neurons are not connected among themselves).

3.3.2 Activation Function

To calculate the output of a neuron, the incoming connections are summed up, where each weight is multiplied with the output value of the neuron at the origin of its connection. The heart of every neuron is the transfer function, which is fed with the sum of the incoming connections. The result (also called the activation of the neuron) contributes to the input of all neurons connected to this neurons outgoing connections.

In a closed loop setup the available input is from the sensors of the robot. In order to generate some type of oscillating activation, which is needed for a

continuous walk, the activation of some neurons has to be reciprocal to their input. For this work, as will be discussed in detail in 5.3.1, the hyperbolic tangent function, or short tanh function, was chosen to calculate the activation:

$$\tanh x = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

Bias

The bias is basically shifting the activation function to the left or to the right and makes it possible to get a certain activation for a certain input without having to modify the steepness of the activation functions curve. At the beginning the neurons in the output layer get an input of 0. This leads the neuron to output a certain value, according to its bias. If this value would set the servo to an extreme position, the robot falls over and the evaluation is aborted. Since the probability for outputting just the servos initial position, which makes the Darwin stand up, is small, most of the networks generated would fail. In order to avoid this the bias was used to ensure that the output at the beginning of execution matches the initial position. This approach was also discussed in the center-crossing paper [20] by Mathayomchan and Beer.

For a servo i with an initial position of $p_{init} \in [0, 1]$ (relative to the servos range), the bias θ_i for output neuron o_i with N incoming connections was calculated as follows:

$$\theta_i = -\frac{\sum_{j=0}^{N} w_{ij}}{2} - ln(\frac{1}{p_{init}}) - 1$$

So that the output generated by o_i was exactly the initial position of the servo for the zero-input:

$$f_i(0, ..., 0) = p_{init}$$

3.4 Experimental Setup

The different parts needed for an experiment consist of the tools produced for this thesis, as the Client-Server Architecture 3.4.4 and the *DarwinFramework* 3.4.3, as well as tools produced by third parties, such as the Webots simulator 3.4.1 and the Darwin model.

3.4.1 Webots Simulator

For the executions of the experiments the Webots simulator was used. It is a professional development environment for mobile robots initially written by Dr. Olivier Michel at the Swiss Federal Institute of Technology in Lausanne, Switzerland in 1996. It is now maintained by Cyberbotics.

The interface of Webots and the environment used in this project can be seen in Figure 3.4. Webots allows the design of realistic environments and complex robotic

setups, because it offers the modification of graphical and physical properties of each object. It can therefore be used to simulate an experiment with a model of the DARwIn-OP in a realistic way. It offers a variety of simulated sensors and actuators to equip each robot[2], e.g. cameras, servo motors (rotational and linear), gyro, compass position, force, proximity, light, and touch sensors. The physics for the simulation is calculated using the Open Dynamics Engine.

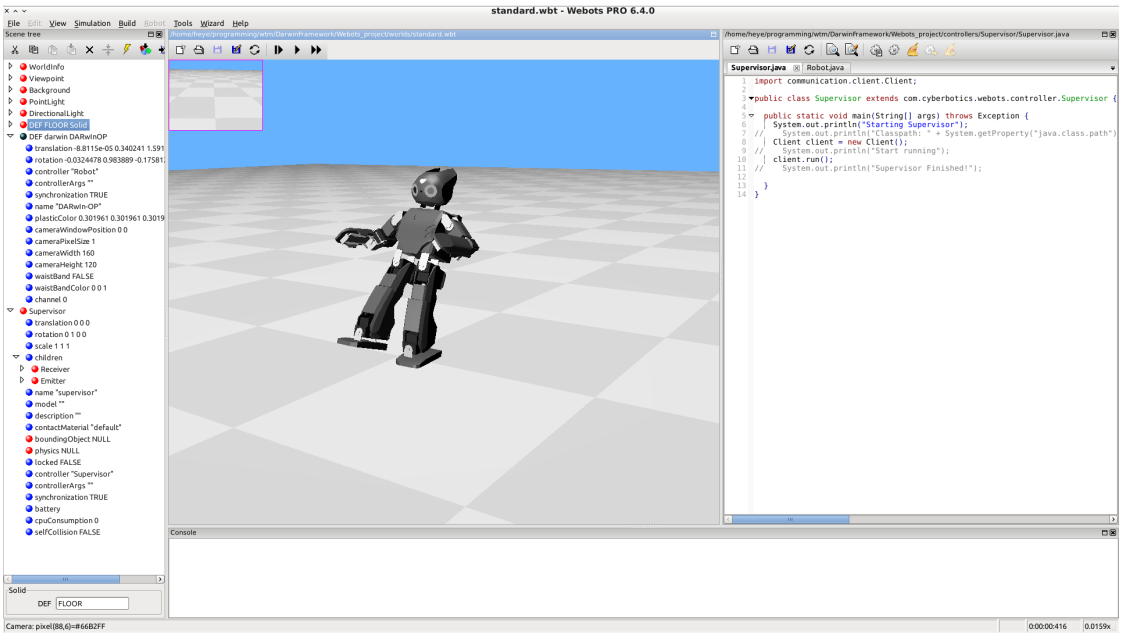

Figure 3.4: The GUI of the Webots simulator with a Darwin falling backwards.

In Webots the environment and robots are manipulated by controllers written in Java for this work, which is natively supported by Webots. For the experiments two controllers were used, one supervisor, which took care of tracking the Darwins position and would terminate the evaluation in cases discussed in 3.2.3, and the robot controller, which executed the neural network and connected the output neurons to the servos of the Darwin. The environment chosen was a simple plane with no obstacles perpendicular to the robots position. Its slope was 0°, except for two experiments that were performed on different sloped planes: $\{-0.5°, 0°, 0.5°\}$.

3.4.2 DARwIn-OP

The platform the algorithm was trained on is the DARwIn-OP. DARwIn-OP stands for "Dynamic Anthropomorphic Robot with Intelligence - Open Platform". It is a humanoid robot, designed by Virginia Tech and produced by the South Korean company Robotis. It is used, amongst other challenges, for robot soccer, e.g. in the RoboCup. In 2012, for example, it competed at the Robocup German Open in Magdeburg as well as at the Robocup World Championship in Mexico when it was

[2]Cyberbotics Manual 2013, 01 2013.

used by the "RoboCup AG" at the University of Hamburg. With its dimensions, 45cm tall, 2.8kg, and 20 degrees of freedom it satisfies the specifications for the *KidSize* league of the Robocup rulebook. Furthermore, it is equipped with a 3-axis gyroscope and accelerometer [8]. Figure 3.5 shows the Darwin with the servos used in this work.

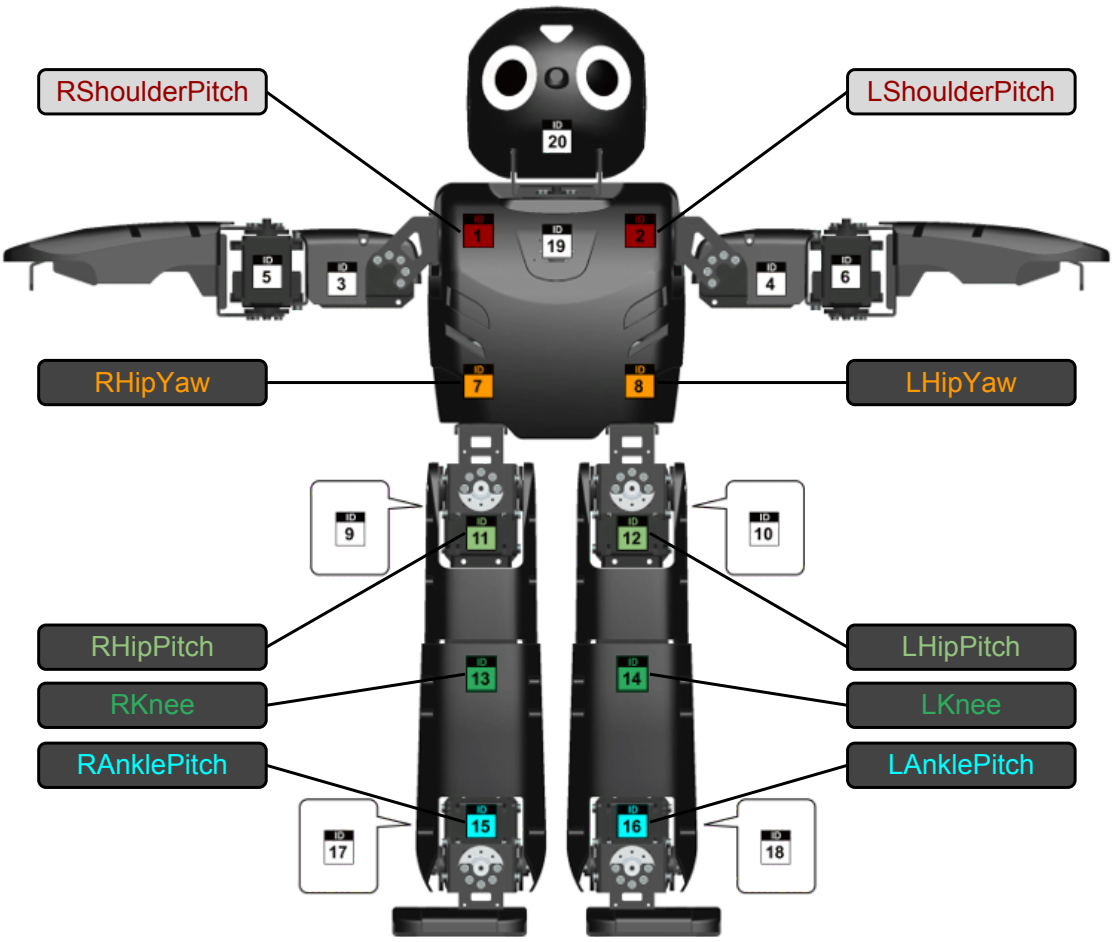

Figure 3.5: The Darwin with all the servos controlled by the network (base image taken from darwin-op.springnote.com).

The model used for the experiments was modeled by Cyberbotics and is included with the up to date version of the Webots simulator. It offers the complete physical model of the Darwin, equipped with all actuators and sensors found in the real Darwin, namely the 20 servos (therefore having 20 degrees of freedom), the accelerometer(x, y, z), the gyro (x, y, z) all simulated with physics close to reality, and for the sake of completeness, the LEDs, and the camera.

The robot has not been modified for the setup of the experiments, hence not equipped with foot contact sensors. This makes the task of learning to walk a lot harder, because many approaches by other research groups are based on the ZMP method, which requires contact sensors. Furthermore, the robots morphology and weight distribution is given, as explained before in Section 2 this brought in many

limitations, not only because the platform has to be realizable in the real world, but also because the model cannot be changed in the simulator in order to adapt it to the needs of the ANN for example.

But the great advantage of using a realistic model based on a real platform is: The gap between the simulation and the real world evaluation is much smaller and therefore more realistic to be overcome with a little handcrafting. Unfortunately a portation of the evolved walking gait to the real platform is not within the scope of this thesis.

In early experiments all servos were used as input sensors and could be controlled by the ANN. This led to a very high number of mutable parameters and an unnecessarily complex search space. To decrease the number of parameters, the use of servos that are not involved in the walking process were removed, namely the HeadPan and HeadTilt, the Elbows, and the ShoulderRoll servos. The ShoulderPitch was still kept for balance reasons.

3.4.3 The DarwinFramework

The whole implementation is based on the *DarwinFramework* written by the knowledge technology group (WTM) and extended for the needs of this work. It offers two different kinds of evolutions: standard evolution, using global selection operators and tournament selection, using local selection operators, as explained in Section 3.2.4. All individuals of this population can be arranged in a two dimensional grid to enable local tournament selection. These individuals and their ANNs encoded in the genes can be saved to disk, including all information about the experiment and other parameters. Supported are raw xml and zipped format, which compresses the files to a 25th of their original size. The genes themselves are versatile for any kind of data, it is only interpreted upon conversion to the CTRNN, which makes the *DarwinFramework* not only applicable for this work, but for many different purposes. The evolution process can be controlled through a graphical user interface (GUI). It provides tools for controlling the computers used for evaluation, the analysis of the experiments, and an overview of the population and the development of the evolution.

3.4.4 Client-Server Architecture

There are a couple of reasons, why the development of a system for distributed evaluation seems a good investment:

Firstly the Webots simulator cannot reset the scene to the initial position without restarting the controllers. On the Cyberbotics website, it is stated[3], that the best way to do a reset of a scene is to save all data to disk, call the "revert()" function, and load the data again. This is necessary, because when the revert function is called, all unsaved data is lost due to reset of the Java controllers. Since the execution of one evaluation takes usually less than a second, this procedure sounds

[3]http://www.cyberbotics.com/dvd/common/doc/webots/guide/section6.3.html

inefficient. So the development of another solution, where the data is kept by another program was evaluated.

Another reason was that, as mentioned before, the usage of artificial evolution combined with tournament selection enables the parallel calculation of the fitness for the individuals in a population. This is very convenient since a large number of individuals had to be created and executed in order to get a decent result in the end.

Since the University of Hamburg owns a license server that can be used by ten different IP addresses in parallel, it was possible to distribute the tasks between up to ten different computers, which can run multiple instances of Webots. A task is simply an ANN to be evaluated. To realize the distribution, a server is started that sends out tasks to a list of machines, the clients, that then execute the tasks and return the fitness. In Figure 3.6 the architecture and the workflow is depicted.

The GUI is used to control the server and display the state of the evolution. The server is given a list of machines that are setup for evaluation (having environment variables set, the Webots simulator and the *DarwinFramework* installed). One Machine can run an arbitrary number of Webots instances, usually the number is chosen to be less or equal to the number of virtual cores. For each instance, a unique *MachineKey* exists. Before the machine can evaluate any task they have to be connected, meaning a secure shell protocol (ssh) tunnel back to the server has to be opened.

When the evolution (blue loop) adds a pending task, the server (green loop) starts the *PendingTasksWorker*, if it is not already running. The *PendingTasks-Worker* itself then starts an instance on a Machine and puts a *Handle* along with the *MachineKey* for it into the green loop. If all cores on all machines are occupied, the *PendingTasksWorker* simply waits. As soon as any instance notifies the server (registering at the *NetworkService*) that it can evaluate another task, a *WebotsHandle* is created and the ANN along with the experiment parameters is sent as a serialized Java class over the network to the Webots instance. Note that we have two different handles now: the *WebotsHandle*, handling an individual that is being evaluated and the *Handle* for the instance responsible for the running state of the Webots application. The *Handle* for the Instance is then put into the running state (starting the ping protocol to check if the belonging Webots instance is still running). Webots then executes the task, sends command-line output back and answers to the ping protocol, until the execution is finished and the fitness is calculated. A *Handler* takes care of the *WebotsHandle* and the *Handle* for the instance. Now the computation of the fitness takes place on the remote computer. The *Handler* waits for the result of the evaluation, which includes the fitness, the tracking of the walked path, and the records of input and output values of the network.

If the result is sent back, or the connection to Webots is lost, both handles are returned. The *Handler* reverts the Webots instance or restarts it, if the connection was lost. On return of the *Handle*, the *MachineKey* is available for another execution again. The Webots instance notifies the evolution about the result of the evaluation. If the evaluation has failed, the individual is added as a pending task

again. Otherwise the individual gets assigned the calculated fitness. As depicted in Figure 3.1 its square in the population is colored green (according to its fitness) from now on. As soon as a tournament is performed, the two winners are copied, mutated and added as a pending task on the server again.

If the Webots instance started by the *PendingTasksWorker* does not report back after 40 seconds it is killed and restarted. The threshold was set to 40 seconds, because that was twice the time the Webots simulator takes to start under normal conditions.

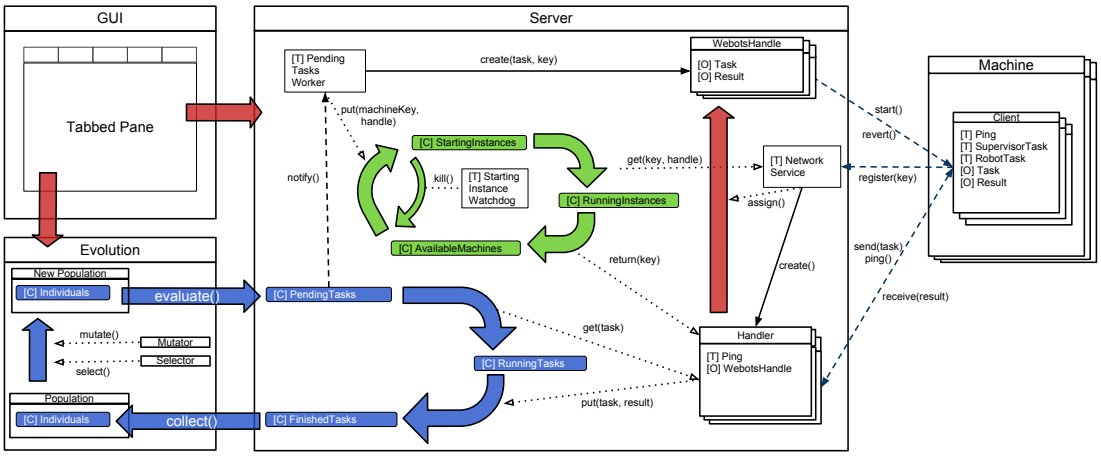

Figure 3.6: The architecture of the server and the flow of operations. The blue loop is the evolution, the green loop is the server.

3.5 Execution of an Experiment

Basically an experiment can be thought of as a random initialized population, consisting of a number of individuals, run with certain parameters as explained in 4.1. This population was the starting point for the evolution. The evaluation of the individuals was started and continuously executed until terminated, as explained in Section 3.2.6. As soon as a region with the dimensions of a tournament (starting at an arbitrary point) on the grid was completely evaluated, a tournament was performed.

3.6 Objective

The aim of this work was to evolve a locomotion for a humanoid robot, and therefore the expectations were to see the evolution converge towards a local minimum, which would be close to the way humans walk. Roughly speaking, the idea was to investigate a number of different configurations: Networks with a different number of neurons in the hidden layer or no hidden layer at all, as well as using a CPG as input besides the accelerometer and gyroscope sensors.

3.6.1 Evolution of Oscillation

As previously mentioned in Section 1, human walking is based on a pattern that is continuously repeated to move the legs in order to locomote. In order to imitate this pattern, the activation of the output neurons has to oscillate in a certain way. The shape of that pattern is unknown and it is the task of the evolution to generate it. As well as the tendency to generate a continuously oscillating behavior, CTRNNs also possess the possibility to generate a converging output. On convergence the servos stay at the same position, ergo no continuous locomotion is possible. So the first aim was to see a population evolve an internal pattern generator that could eventually lead to some kind of continuous movement.

3.6.2 CPG

To see if providing an oscillating pattern from an external input at the beginning could speed up the evolution, some experiments were to be performed with a central pattern generator (CPG) connected to one input neuron. The setup for this was as follows: The input from the CPG was fed into the sinus input neuron of the network. In order to provide a modifier for the wavelength, the output of the output neuron o_{sinus} was taken as input for the CPG. The pattern was generated by a sinus function receiving the current frame counter i as input and the activation of the output neuron o_{sinus}. So the input for the CPG-neuron was:

$$I_{sinus} = \sin(i * 0.1 + o_{sinus})$$

3.6.3 Hidden Layer

As described by Paul [25] it is possible in a certain setup to evolve locomotion without using a hidden layer. But this might not be true for all configurations, so for the setup used in this work, the question was: Is it possible to generate an oscillation pattern without a hidden layer and if so, is it possible to evolve a locomotion driven by this pattern? After failure to evolve any locomotion, the question changed to: How many neurons in the hidden layer are necessary for the development of locomotion?

Chapter 4

Results

96 runs within 17 different experiments have been performed, 143 runs including the preliminary and testing experiments, which are listed but not analyzed here.

4.1 Experiments

Table 4.1 shows a detailed overview of the parameters and results of all experiments. The abbreviations of the column titles are as follows:

E Number of the experiment.

Network Architecture of the ANN, specifically number of neurons in Input/Hidden/Output layer. See Table 4.2 for the sensors and servos attached to the different input and output neurons.

Arms (used) Describes the angle of the arms as well as whether they were used or not.

CPG States whether a central pattern generator was used.

$\boldsymbol{h_{min}}$ Is the minimum height of the robot. The measuring point lies in the center of the robots torso.

COD Is the expansion velocity of the circle-of-death, as explained in Section 3.2.3.

Slope Determines whether or not the experiments were performed on leveled or sloped ($-0.5°, 0°$, and $+0.5°$) ground.

Fit The version of the Fitness function. F* is not a single functions but stands for the developmental state of the fitness function, as explained in 3.2.3.

Succ The number of runs that successfully evolved a locomotion.

Ttl The total number of runs.

E	Network	Arms (used)	CPG	h_{min}	COD	Slope	Fit	Succ	Ttl
0	16/5/10	Bent (✓)		17cm	0m/s		F*	5	47
1	16/5/10	Bent (✓)		17cm	0m/s		F1	0	6
2	16/0/10	Bent (✓)		17cm	0m/s		F2	0	7
3	16/5/10	Bent (✓)		*[1]	0m/s	✓	F2	0	2
4	16/0/10	Bent (✓)		*	0m/s	✓	F2	0	4
5[2]	7/5/8	Bent	✓	17cm	0m/s		F2	0	9
6	14/5/8	Bent		17cm	0m/s		F2	0	1
7	16/5/10	Straight (✓)		17cm	$\frac{1}{20}$m/s		F3	1	5
8	14/5/8	Straight		17cm	$\frac{1}{20}$m/s		F3	0	9
9	14/9/8	Straight		17cm	$\frac{1}{20}$m/s		F3	1	1
10	7/5/8	Straight	✓	17cm	$\frac{1}{20}$m/s		F3	0	1
11	7/0/10	Straight (✓)	✓	27cm	$\frac{1}{20}$m/s		F4	0	2
12	16/0/10	Straight (✓)		27cm	$\frac{1}{20}$m/s		F4	0	11
13	16/2/10	Straight (✓)		27cm	$\frac{1}{20}$m/s		F4	0	7
14	16/3/10	Straight (✓)		27cm	$\frac{1}{20}$m/s		F4	0	10
15	16/4/10	Straight (✓)		27cm	$\frac{1}{20}$m/s		F4	1	4
16	16/5/10	Straight (✓)		27cm	$\frac{1}{20}$m/s		F4	3	12
17	16/10/10	Straight (✓)		27cm	$\frac{1}{20}$m/s		F4	1	4

Table 4.1: Overview of all experiments performed. E0 are the preliminary tests, E1-E11 explorative experiments, and E12-E17 the final experiments to analyze the impact of the number of neurons in the hidden layer.

When speaking about hidden neurons an abbreviation will be used: no hidden neurons (H0), two hidden neurons (H2), three hidden neurons (H3) and so on. An illustration of the network architecture of type H5 can be seen in Figure 3.3, H2 - H10 are set up the same way, only with a different number of hidden neurons, H0 has only connections from the input layer directly to the output layer.

4.1.1 Oscillation

The first 19 runs (E1 to E4) were performed on H0 and H5 on even ground and sloped ground. None of these runs led to any kind of locomotion. But in two of the ten H5 runs a vibrating pattern was evolved, see Figure 4.1 for an example.

In most runs the evolved solutions tended to be very careful with movements

[1] No early termination.

[2] Due to bug this experiment was performed with an invalid number of output neurons, therefore it cannot be executed with the software anymore since the bug has now been fixed and an invalid output vector size is rejected.

Servo/Sensor	Input 7	Input 14	Input 16	Output 8	Output 9	Output 10
ShoulderPitch			✓			✓
HipYaw		✓	✓	✓	✓	✓
HipPitch		✓	✓	✓	✓	✓
Knee		✓	✓	✓	✓	✓
AnklePitch		✓	✓	✓	✓	✓
Accelerometer	✓	✓	✓			
Gyroscope	✓	✓	✓			
CPG (Sinus)	✓				✓	

Table 4.2: The different input sources and controllable servos used in different setups, as well as the CPG. Since each configuration was unique regarding the use of neurons in the input or output layer, the sensors and servos attached can be identified simply by the number of neurons.

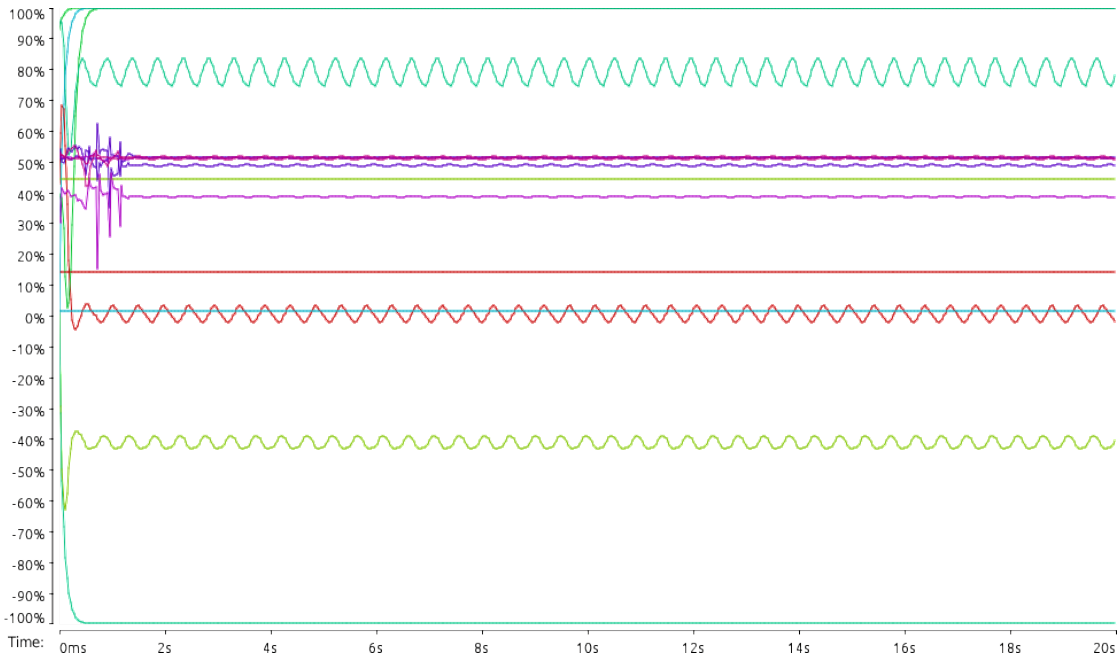

Figure 4.1: Vibrating pattern of run E1_R4.

and usually kept standing close to the initial position, instead of moving a servo and therefore risking instability. In many of the runs, where an oscillating pattern was evolved, the first servos that started to move continuously were the left or right ShoulderPitch servos; hence it was suspected that these are used to evolve a pattern generator without having to use servos that directly destabilize the stand.

This led to the assumption that in general the input of the servo angles was used to generate a pattern. So a series of runs (E8 and E9) were performed where input and output of the ShoulderPitch was removed, ergo the arms could

not be modified by the ANN at all. In both experiments, a locomotion evolved, however this locomotion had a very slow velocity and an even less anthropomorphic appearance compared to the techniques evolved in the experiments where the arms were used. Furthermore, disabling the arms seemed to decrease the probability of evolving a locomotion, so the arms were enabled again in later experiments. So, apparently the arms were not required, but were helpful in generating a pattern.

4.1.2 CPG

In an attempt to reduce the search space (in experiment E5[3], E10, and E11) all input of the servo sensors was removed and replaced by a central pattern generator (CPG). But once again, in order to keep stability the networks tended to suppress the input from the CPG so that the robot would not move away from its starting point. This can be seen in Figure 4.2. Nothing even close to a recurring pattern not to mention locomotion was evolved in any of the 12 runs, therefore no further experiments with a CPG were performed from this point on.

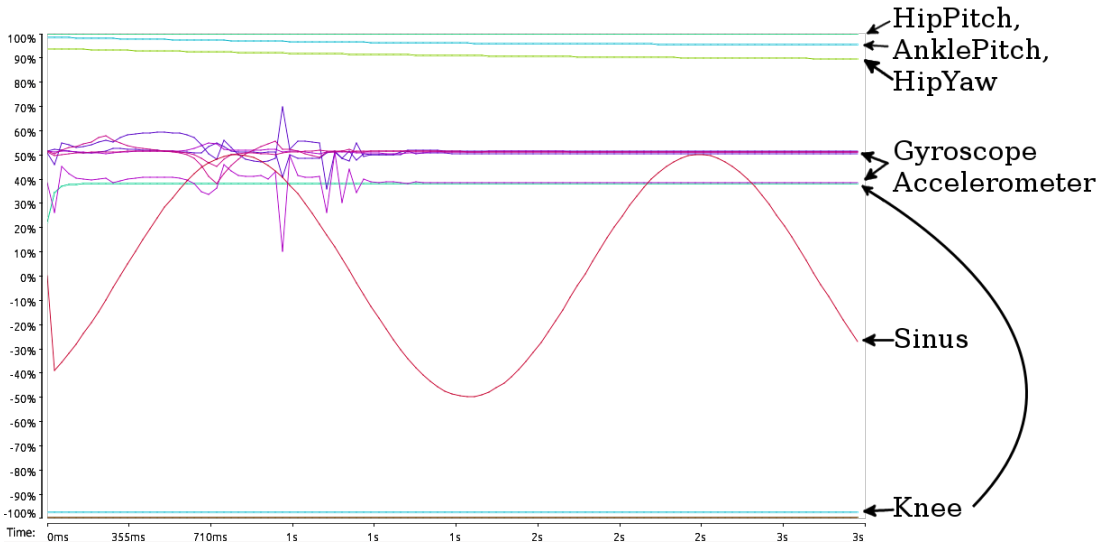

Figure 4.2: Input of the CPG, accelerometer, gyroscope, and output to the servos displayed in one graph (experiment E11_R1). The CPG input was apparently suppressed and the robot kept standing still.

4.1.3 Hidden Layer

Based on the preliminary runs (E0), a locomotion only evolved with neurons in the hidden layer. The interest grew into determining how important the neurons in the hidden layer are in order to produce any kind of locomotion. A standardized

[3]Due to a bug in the software, that has been fixed, the E5 experiment does not provide an output neuron for the o_{ij} value and cannot be executed because the number of output neurons does not match the required size.

experiment configuration was defined, using the ShoulderPitch servos, but no CPG, termination on position below 27 cm and a circle-of-death, with a propagation velocity of $\frac{1}{20}$m/s (as explained in Section 3.2.3). This configuration was used to perform runs with six different architectures, namely H0, H2, H3, H4, H5, H10.

Within experiments E12 to E14 (configurations H0, H2, and H3), 28 runs were performed. None of these resulted in some kind continuous locomotion, but rather in a step forward or backward and in certain cases a slow leaning over in order to optimize the covered distance. In contrast to that, in the experiments E15 to E17, five of the 20 performed runs evolved a locomotion.

4.2 Locomotion

In those cases, when a locomotion evolved, the steps were rather small and one leg would not pass the other. Specifically we want to examine run 12 of experiment 16 (with 5 hidden neurons), because the solution found was highly optimized and a good example of a successful locomotion. The optimization can be observed in visualization of the tracked robot positions in Figure 4.4. The corresponding graph with servo, accelerometer, and gyroscope input is shown in Figure 4.3.

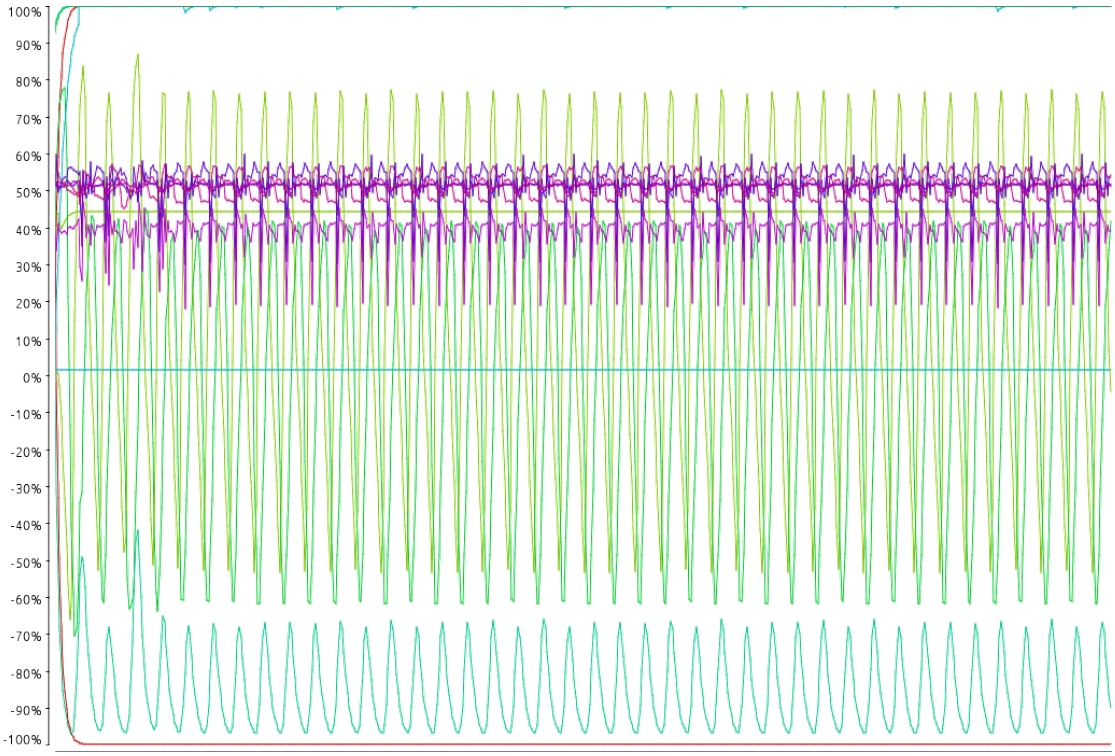

Figure 4.3: A pattern recurring over 20 seconds, causing the robot to perform about 42 steps.

Now we want to do a detailed analysis of the sequence of one step. It was performed within the first seconds of the evaluation of the individual with the highest

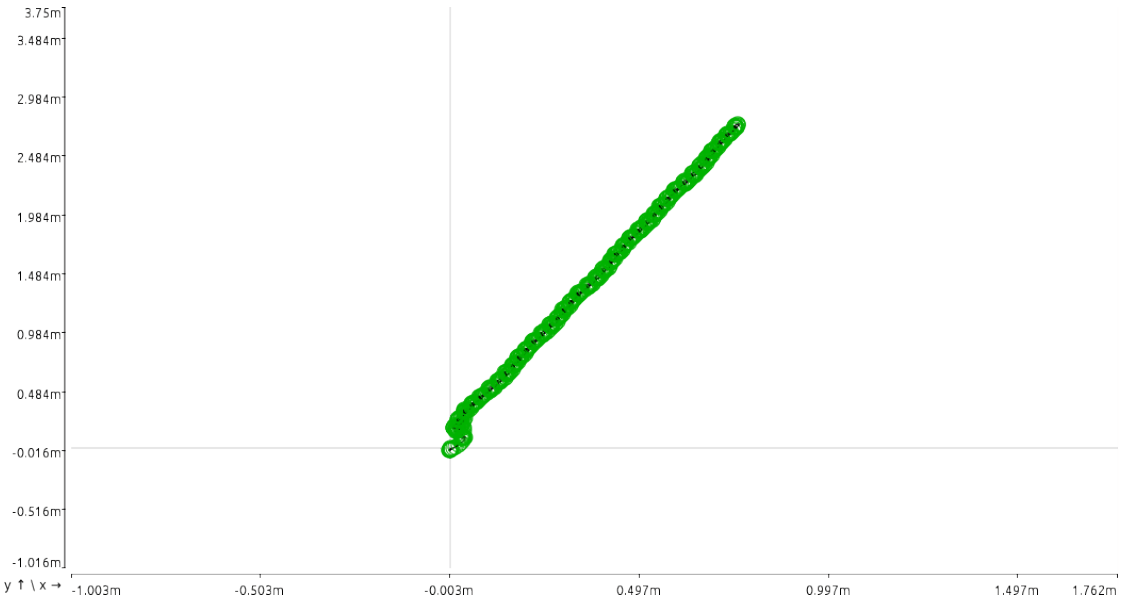

Figure 4.4: The path walked, this is highly optimized for distance traveled over time. The initial point is at (0m, 0m).

fitness in E16_R12. The view in the simulator (Figure 4.6) and the corresponding part in the graph (Figure 4.5), is a recurring pattern, where the left HipPitch plays the main roll.

1. Initial position.
2. Lifting the left foot off the ground.
3. Pushing it forward.
4. Both feet on the ground.
5. Lifting right foot a little.
6. Pulling right foot forward.
7. Going into initial position again.

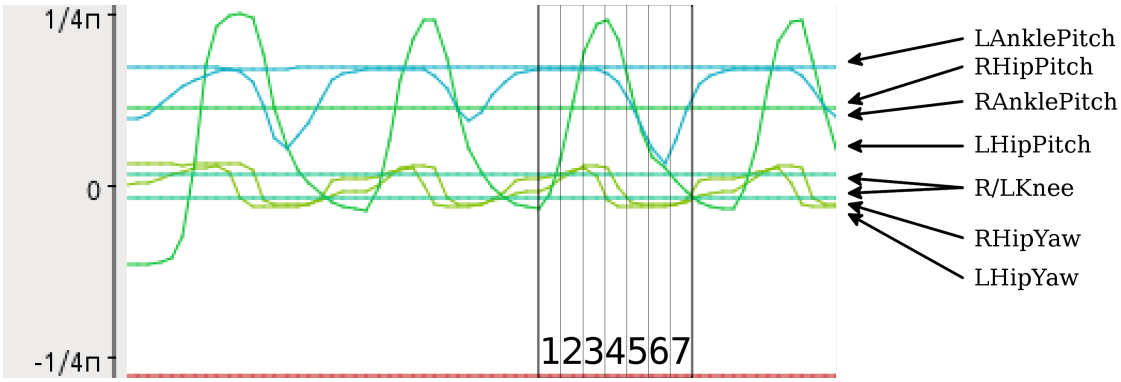

Figure 4.5: The sequence in the graph, excerpt of 2.25 seconds, from $-\frac{\pi}{4}$ to $\frac{\pi}{4}$.

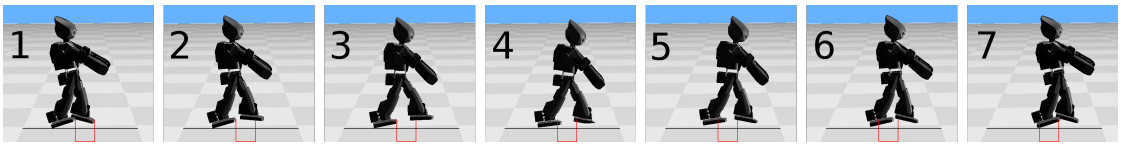

Figure 4.6: The sequence of a step, from 1.47 seconds to 1.85 seconds. Initial tips of feet depicted with red lines.

Chapter 5

Discussion

5.1 Problems Encountered in the Tools Used

Next to the problems encountered during the configuration of the parameters for the experiments, there were also quite a number of technical issues. Prior to being able to perform the experiments needed, the tools for execution have to be developed. This was not always straightforward because a number of problems have emerged out of the incorporation of many different tools, namely:

Network Communication Connecting to many computers in parallel, tunneling through the firewall of the university, the hassle with ssh, network security mechanisms, and the general problems that come with programming.

Webots and the Darwin Model Incorporation of the controllers and their messaging system and the inability to run in headless mode and accessing the servos and sensors of the Darwin model.

All these took a long time to be solved. This section is supposed to provide possible solutions for future usage of these tools.

5.1.1 Network Communication

Java already offers a Transmission Control Protocol (TCP) socket implementation that can be used fairly easily to send data. This all sounds very doable, but there were some difficulties to overcome.

Webots itself has to be started by an external process, which can be done using Secure Shell (ssh). At first ssh was used through Java's runtime utilities, which turned out to be a quite messy solution, since there were many instances of ssh running in parallel, on the server as well as on the client machine that sometimes caused hangups. Replacing external ssh with the Jsch library gave everything a cleaner touch, but caused issues with *strictHostChecking*, "known_hosts" file, and environment variables that were not loaded at the beginning. But these issues could be overcome by disabling *strictHostChecking* and adding every machine manually to the "known_hosts" file. Exporting the variables manually from the ".bashrc" file

using `$ eval $(grep VAR_NAME .bashrc)`, and logging in on the graphical login screen.

Jsch requires a graphical login. Which is very inconvenient for remote access, which will be explained later on. A graphical login was also required for Webots, since it needs a display to run on and cannot be forwarded due to a bug. This is a big drawback, because a couple of reasons can lead to a state, where the machine is not able to perform any further experiments. Furthermore, after a varying amount of time (from one minute to two weeks) Webots does not start anymore and gives one of the following error messages on startup: (1) No GTK display could be found, (2) The MIT_MAGIC_COOKIE is invalid, or (3) The OpenGL driver reports a failure. The only solution found for these problems was a reboot of the machine. However, not only Webots caused problems, unrelated issues like expiring *Kerberos* tickets (that should have renewed themselves, but didn't) appeared when least expected. These were solvable by logging out of the session or restarting, but afterwards the required graphical login to the machine can only be done manually by someone present on the machine.

Currently the best solution for the communication via the network works in the following way (where S is the computer on which the server is running): A reverse ssh tunnel is established by S to each client machine. Every application on the client machine is now able to use this tunnel to communicate with the server running on S, as if the server would be running locally on the machine itself. The only requirements for S is, that it has to be connected to the Internet and to the "Fachbereich Informatik - Virtual Private Network" (short FBI-VPN), while the physical location of S can reside anywhere in world. Through this tunnel, a TCP/IP-connection is established over which messages can be sent in packets. These packets carry for example the serialized ANNs or the calculated fitness for an individual.

In order to be able to control the execution of the experiments a GUI for the Client-Server Architecture was developed. It offers an overview of the whole execution process, including the machines, every instance of Webots, every experiment performed, the whole population, and the progress of evolution. Furthermore, it is possible to change the distribution of tasks to different machines in case some machines should fail during a run.

5.1.2 Webots and the Darwin Model

In order to communicate with the simulation environment of Webots so-called controllers are used. These are written in Java for this work and have to be compiled by the Webots Java compiler. But in order to keep the most pats of compilation within the *DarwinFramework* the controller classes in Webots solely initiate a new class residing in the *DarwinFramework* that handles the real work. In order to access the sensors and servos, a *RobotController* is needed. This *RobotController* can only control the robot, while another controller (the *SupervisorController*) has to be used to modify the scene or get position information about the robot. These two can communicate via a messaging system that was setup with the same interface as

the inter computer communication. This messaging system is used to transmit for example the whole ANN serialized as a java object from the *SupervisorController* to the *RobotController* for evaluation.

During and after execution Webots tries to write to four different files: ".webotsrc", ".webotsrc.swp", "webots.log", and "webotsOgre.log", which are all located in the home directory. This leads to conflicts when multiple instances try to write to the same file at the same time. To get around this the environment variable `HOME` was modified to point to a unique directory in "/tmp" for each instance. These directories were deleted after the Webots instance was killed.

Webots would sometimes get stuck or simply crash. In order to avoid a deadlock of some *Handlers* in the server when waiting for the result of an evaluation that, due to a crash in Webots, would never be sent back, a ping protocol was introduced. It sends a ping and waits for a pong response until it sends a ping again. If a timeout was reached (no response for more than 20 seconds), the individual was set into the pending state again, the Webots instance was killed and a new Webots instance was launched.

But if the instance got stuck on shutdown it might try to write to one of the files that resided in a deleted directory. In those cases Webots displayed an error message on the graphical screen. While this message is displayed, the process is in "D" state, which means, that it cannot be killed by any command, not even "kill -9". So the computer has to be accessed manually again, to close this error dialog.

As previously mentioned, the University of Hamburg owns a license server. Webots requires a connection during the whole execution time to this server. If the server is not responding or not accessible Webots would not start or continue the execution and would display an error message that has to be closed manually.

5.2 Fitness Function

The fitness function is a very important part of any evolutionary algorithm and requires a lot of careful tuning. In this section the successive improvement of the fitness function will be explained by combining the observed behavior and the actions taken to fight certain problems.

5.2.1 Development of F*

F* denotes the fitness function in the developmental stage during the preliminary experiments. Thus all experiments performed with this ever-changing fitness function are not comparable to each other. This is the reason why it does not have a number assigned to it.

The fitness of an individual is calculated, after the evaluation is over, based on the actions taken during evaluation. Therefore, a termination criterion for an evaluation must be defined at first. Since the space the robot could travel was not limited, the first termination criterion of the evaluation was time. After 20 seconds the evaluation was terminated.

So the initial fitness function (first implementation of F*) simply rated the overall distance (measured in x and z direction) traveled away from the starting point within 20 seconds. The robot's position is defined by the center of its torso. As depicted in Figure 5.1, this fitness function did not describe the robots action well, because a small step did not give a good fitness compared to falling down.

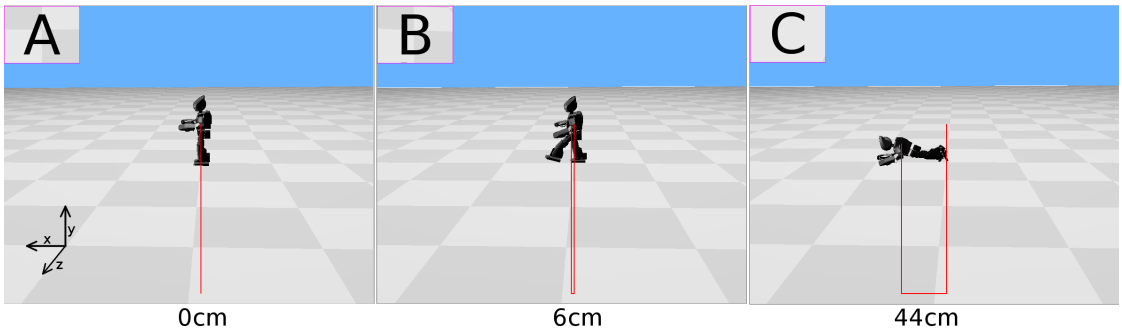

Figure 5.1: A: Initial position (before execution). B: Walking one step, then standing in a stable position (after execution). C: Falling over (after execution).

A fallen down robot was not able to get up, so when falling down, the robot could only crawl to improve its fitness. This was not desirable, because crawling is not allowed in humanoid soccer. In order to penalize this action, the experiment was terminated as soon as the robots height was below a certain level h_{min} and two penalties were applied: The 0.74cm (twice the robots initial height) was subtracted from the overall distance traveled and the evaluation was terminated. In order to reward robots for not causing an early termination, the runtime τ of the experiment was also taken into account when rating the fitness. h_{min} was 17cm (half of the robots height) for the fitness functions F1-F3 and for the last experiments increased to 27cm because sometimes the robot would fall but keep its chest above 17cm by pushing itself up with the arms.

Now the evolution was more likely to evolve solutions that avoided falling down during execution time. But this led often to a solution, where the robot stood, and waited until the experiment was over in order to get the time reward τ. In other words, the activation of the neurons converged and the CTRNN got into an equilibrium state after the first seconds of execution.

Important for locomotion is a pattern that causes the servos to move continuously. To make an oscillating pattern more attractive, a further reward was introduced: β, continuous movement of the servos. Therefore, the movement of the servos was recorded and integrated at the end. Now another very popular minimum was found, standing at the initial position and waiting for the end of execution to move the servos as much as possible in the last second so that the robot would fall or jump but not get below 17cm before the evaluation had ended.

In order to decide whether the movement was indeed over time, the analysis was divided into six equally long time pieces, β_0 to β_5. If the integrated value of any timepiece was below 0.25°, the reward of 1.5 was not given. But at first the robot would simply stay and move its arms to get movement over time. So the

analysis was based solely on the knees because the robot was not able to stay at the same spot when moving the knee servos.

After the introduction of this rule, the robots started to move their knee servos as much as possible, which is not very efficient. Overall it would be preferable, if all servos were moved as efficiently as possible. This led to the introduction of α which gave a reward reciprocal to the integrated value scaled to the interval $[0, 1.5]$. But now the solutions standing at one spot were rewarded for their "very efficient" use of the servos. So the reward for efficient movement was only given, when the integrated servo values of the knee where continuous.

5.2.2 F1-F4

This all helped to reduce the amount of runs with no developed locomotion. But further tuning was performed after experiment E1 did not produce any solution with locomotion. In order to balance the reward a little more towards walking, the individual weights for some parts were changed in F2.

Continuous movement (β) raised from 2.0 to 4.0.

Absolute walking distance (δ) raised from 3.0 to 3.5.

Integrated distance (Δ) raised from 15.0 to 19.25.

Not knowing that the number of hidden neurons was an important factor in evolving locomotion, the results of the next couple of experiments were disappointing, because none of them did develop any kind of locomotion behavior. Almost all of the experiments resulted in standing solutions, some of them jumping away at the end. Indeed only three of those experiments had the correct preconditions, more than three neurons in the hidden layer. This led to further tuning of the fitness function, thus F3 was introduced. It featured a new instrument to force individuals to walk: an expanding circle-of-death. It killed the individual, as soon as it came within this circle. The expansion speed was set to $\frac{1}{20}$m/s, so that the radius of this circle was one meter at the end of the experiment. Furthermore, the arms were set to be straight instead of bent during execution.

This produced two runs resulting in locomotion. But still some runs ended with standing individuals or individuals that managed to push themselves up with their arms, staying above the 17cm mark. So again the fitness function was changed and F4 was created. The only difference being that the minimum height was increased from 17cm to 27cm.

When looking at the results produced so far, no runs with no hidden neurons resulted in locomotion. A suspicion arose, that the number of neurons in the hidden layer played an important role. So finally experiments with different numbers of hidden neurons were performed with no other parameters changed. These were the last experiments with F4 being the final fitness function. And indeed, four of the 20 experiments resulted in locomotion. This indicates that the probability to

generate a solution with locomotion is somewhere around 25% using this fitness function and more than three hidden neurons in the neural network.

After all experiments had been performed, a bug in the fitness function was found. This bug does not affect the results, but it makes the calculation of the integrated distance (Δ) implausible. The distance is taken in intervals of half a second, but measured only to the last frame. Therefore, the reward erroneously is given, when the robot is only moving from one point to another, but not actually moving away from the point where the last measurement was taken. The intention of measuring only two times a second was to avoid giving a reward in this kind of situation. It might be possible that even better results are achieved, with experiments performed using F5, where the bug will be fixed.

5.3 Comparing Objective and Outcome

The objectives were to evolve oscillation, test if the integration of a CPG would introduce further improvement, compare the impact of neurons in the hidden layer, and finally develop a locomotion. First of all, it was discovered, that the choice of the activation function has a great impact on the development of oscillating patterns.

5.3.1 Activation Function

A popular transfer function for ANNs is the sigmoid function:

$$f(x) = \frac{1}{1 + e^{-x}}$$

But during early test phases it was discovered that the activation of the CTRNN always converged to an equilibrium state after a few seconds. As said before, an oscillating pattern is needed, because a CTRNN with converging output causes the servo position to stay at a certain angle, so the robot cannot move forward. This is the reason for choosing the tanh function over the sigmoid function.

As Kamps et al. [5] pointed out, the standard sigmoid neuron is the more biologically plausible approach, but with an input of 0 it has an activation around 0.5, which tends to increase general network activation and can therefore lead to convergence of high activations. They found out that an anti-symmetric smashing function

$$f(x) = \frac{2}{1 + e^{-x}} - 1$$

works better because the activation is 0 for $f(0)$. The hyperbolic tangent is a more natural function because it does not have to be stretched and shifted and is very close to the anti-symmetric smashing function. It can be deduced in the following way:

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1} = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

The experiments performed after the transition to the tanh function have shown that an oscillating pattern is more easy generated by an ANN using neurons with the tanh function than the sigmoid function.

5.3.2 CPG

As shown in the results, the input from the external pattern generator is ignored after a small number of generations. It is possible that at early stages of evolution the fitness function forces the individuals to keep stability, so it becomes impossible to incorporate the pattern while no stable step is developed. Therefore, the individuals that have a stable standing pose get better rewards and the evolution tends to ignore the pattern input in favor of stability.

5.3.3 Hidden Layer

The CTRNNs used in this work have shown that they can evolve rhythmic behavior on their own under certain circumstances. The number of neurons in the hidden layer is apparently a crucial point for an oscillating pattern which is required for continuous locomotion. This leads to the conclusion that the internal recurrent connections are crucial for the development of a recurring pattern. So far no upper limit of hidden neurons has been found and it is unclear if it exists. Unfortunately the time for this work is limited and no further experiments on this subject could be performed, so that no explanation or proof for this indication can be delivered here.

5.3.4 Locomotion

The overall objective of this work was to evolve some kind of walking behavior similar to human walking since this seems to be a very effective method of movement (a local optimum). So a good result was expected to show similar behavior in the simulator. One has to take into account that human walking is learned by falling down and getting back up again. The learning is not done by random mutation but some kind of feedback, collected from falling down. Furthermore, the human walking apparatus offers much more detailed "sensor information" such as pressure on the feet and joints. Another impact was the limitation of sensors and available servos as well as the absence of passive controllers such as spring connections between joints.

When seeing robots walk in humanoid soccer leagues the walking motion is not very anthropomorphic, but rather jerky and apathetic. Therefore, the expectation that the results may differ was formed.

In cases, where some kind of locomotion evolved the behavior was in an upright position and rather stable, but unfortunately not humanlike. But the gaits that evolved reminds one of the techniques used in some martial arts (e.g. Wing Chun which is a gung fu style), which are developed with maximum stability in mind. So it is not surprising that a good local minimum would look like this kind of

walk. For a detailed comparison of human walking and the evolved locomotion see Figure 5.2.

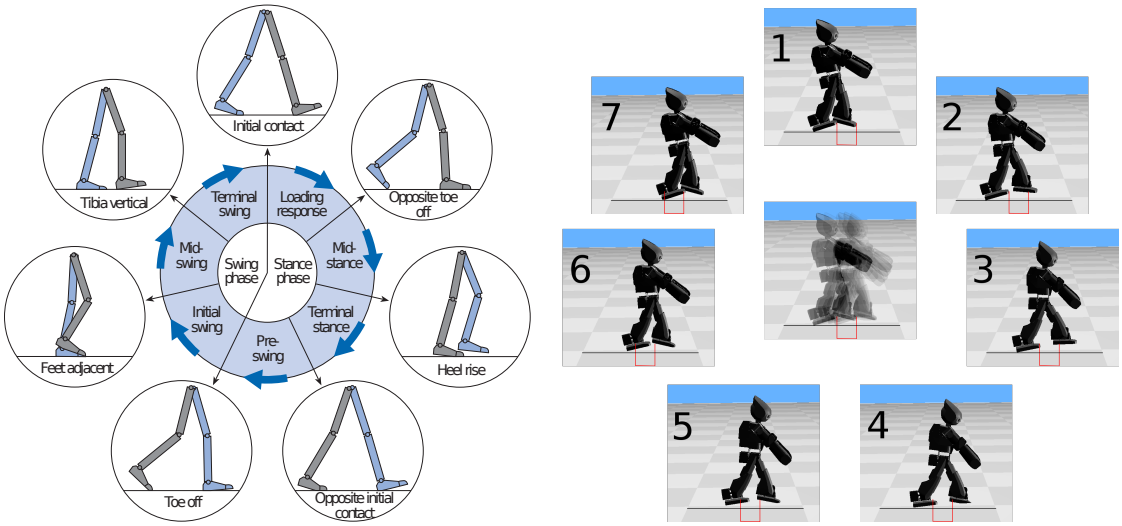

Figure 5.2: Comparison of the human walking sequence with the evolved sequence. (Left figure taken from [30].) In the middle of the right figure resides an alpha blending of all images merged together.

The scope of this thesis is unfortunately not large enough to include all different variations of parameters or also to add other techniques such as different servo connections using springs and such. So a lot of possible branches in the search space have been cut off and therefore it might be possible that no humanlike locomotion lies within the given search space.

Another problem could be that a creature with the Darwins morphology and degrees of freedom is simply not able to walk like a human.

Velocity

The velocity of the walks was quite reasonable. No assumption about the expected speed was made prior to the results, but the gaits of humanoid robots in the *KidSize* league are usually not very fast. In this work the robots walked as much as 4.62m of absolute distance (E17_R5) and an integrated distance of 5.65m (E15_R4) within 20 seconds. It is very likely that these speeds cannot be reached in the real world, because they are optimized to the simulated physics. Furthermore, the servos would probably overheat and wear down fast. But it is still a remarkable speed given the size of the robot and the limitations of the platform due to the realistic model of the Darwin.

5.4 Analysis of the Evolutionary Progress

When looking at the fitness graphs recorded over time, one can see the expected appearance of a normal evolution: A big step upward followed by optimization

over many generations, resulting in a logarithmic looking graph. But at some point even an extensive amount of evaluations did not lead to a better solution. This happened much quicker in cases when the robot did not evolve a walk, but rather only one step, or jumped from the starting point and therefore fell down. When a gait was evolved that enabled the robot to move away from the starting point at a constant speed, the evolution improved over a longer period of time and usually with an increasing amount of larger steps. In some cases the early individuals showed a rather staggering walk, which would improve over many generations to a straight directed walk with much higher fitness levels. A good example of this is E16_R12 where oscillation after 560 tournament and locomotion after about 1200 tournaments evolved and where improvement was still made after 8560 tournaments. A graph of the max fitness and average fitness can be seen in Figure 5.3.

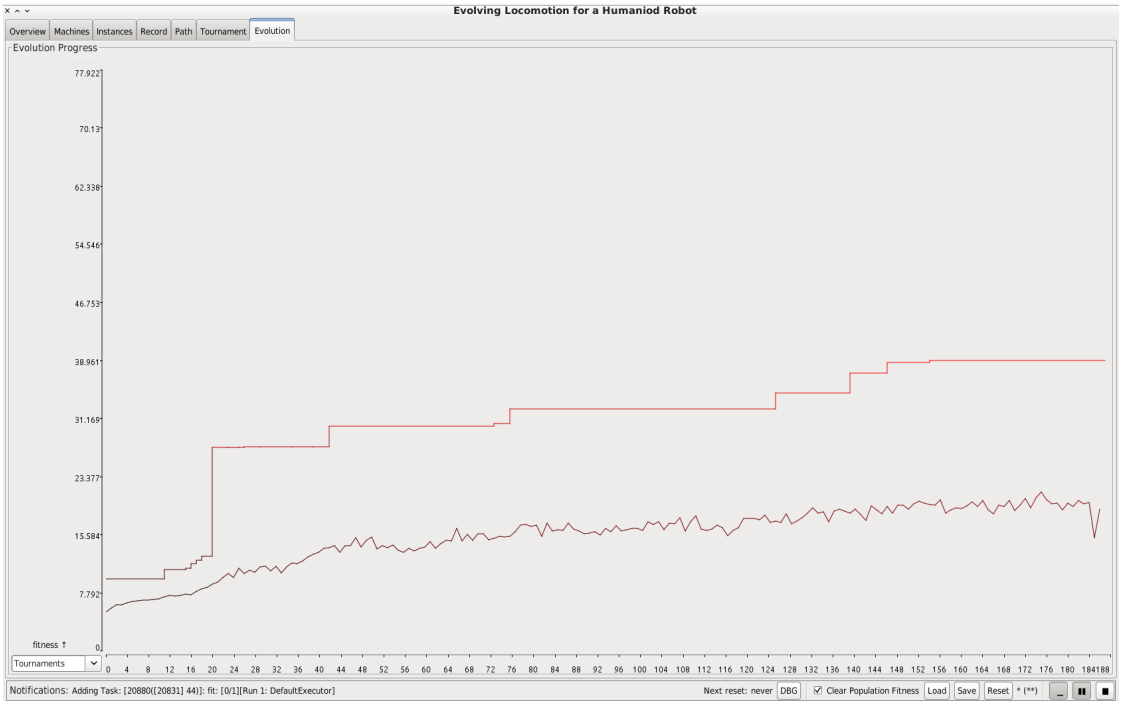

Figure 5.3: Max fitness and average fitness of run E16_R12, every datum is equal to 56 performed tournaments.

Chapter 6

Summary and Conclusion

In summary: 17 different experiments have been performed, where seven different walks have been evolved. When including the preliminary experiments, more than one million individuals have been evaluated for this work, with most of these dying in the process of evolution.

1. Within this work, a Client-Server Architecture offering a GUI was developed in order to control the DarwinFramework. This provides genetic algorithms of genes to evolve by mutating them within a population of individuals. These genes can be converted to ANNs that can be evaluated remotely and in parallel with Webots on the Darwin model and saved to be loaded and evaluated further at a later point. The framework is versatile; and it is possible to evolve a kick or a stand up animation with small effort.

2. This framework has the capability to develop locomotion for a complex humanoid robot, with a success rate of 25% in experiments E15 to E17. The parameters needed to acquire this rate have been defined in this thesis as:

   (a) The final fitness function (F4).

   (b) Input from accelerometer, gyroscope, and ten servo sensors as described in Section 4.

   (c) Four or more neurons in the hidden layer.

   (d) Control of ten servos also discussed in Section 4.

3. The results indicate that the number of neurons in the hidden layer is crucial because experiments with three or less neurons did not develop a single locomotion within 41 runs.

4. Furthermore, the tanh function was found to deliver much better results when focusing on oscillation compared to the sigmoid function.

5. An artificial CPG is not helpful for the pattern generation process. In successful cases a pattern is generated within the network itself.

6. Furthermore: Webots is not suited for remote execution of experiments. However, within this work solutions for the problems caused by Webots have been discussed so that it is possible to control it by a remote server.

6.1 Outlook

In future experiments, the optimal number of neurons in the hidden layer can be evaluated, either by setting the number of neurons by hand, or by allowing the addition of neurons during mutation. Another interesting question is: is it really impossible to generate a walk with three or less neurons in the hidden layer or is it simply just that the probability is therefore much decreased? An answer to this question would be useful and furthermore, an evaluation of what changes in the setup are needed to enable ANNs with less hidden neurons to develop a walk could lead to interesting findings.

The fact that the CPG was ignored raised some important questions: could the reason for this be the fitness function? It is possible that it denies paths through the search space leading to a stable locomotion when using a CPG. It may be required to restructure the concepts and aims of the developed fitness function, which requires some effort but it could lead to even better and faster developing results because as said before: a recurring pattern is crucial for locomotion.

Fixing the bug in the fitness function mentioned in Section 5.2.2 and performing further experiments with the fixed version. Furthermore, since the fitness function is rather complex, it might be possible to achieve equally good results when leaving out some terms, e.g. the reward for continuous movement, which is now indirectly forced by the expanding circle-of-death.

Also a different kind of mutation operator could be introduced, for example crossover recombination.

It may also be possible to perform incremental evolution, where the fitness function changes over time, so at first a stepper is evolved which is then further evolved with a fitness function rewarding distance walked. And after a stable walk has been evolved the focus could be shifted to walk along certain waypoints for example.

Furthermore, an evolved walk can be ported to the real platform in order to see how well it performs outside the simulator. In relations to simulators, another interesting question would be: how does the algorithm run in a different simulator with a different physics engine?

Appendix A

Nomenclature

Used terms in alphabetical order:

ANN Artificial Neural Network

COD Circle-of-death (used in F3 and F4), see also 3.2.3 and 5.2.2.

CPG Central Pattern Generator

CTRNN Continuous-Time Recurrent Neural Network

Darwin DARwIn-OP

DARwIn-OP Dynamic Anthropomorphic Robot with Intelligence - Open Platform

DNA Deoxyribonucleic Acid

E*x* Where $x \in \{0, ..., 17\}$, notation for experiment x.

F*x* Where $x \in \{*, 1, 2, 3, 4\}$, notation for Fitness Function of version x. F* denotes work-in-progress version, see also 5.2.1 and 5.2.2.

GUI Graphical User Interface

H*x* Where $x \in \{0, 2, 3, 4, 5, 9, 10\}$, notation for network with x neurons in the hidden layer.

R*y* Where $y \in \{0, ..., 47\}$, notation for run y, therefore Ex_Ry denotes run y of experiment x.

IP Internet Protocol

TCP Transmission Control Protocol

ODE Open Dynamics Engine

SSH Secure Shell

FBI-VPN "Fachbereich Informatik - Virtual Private Network"

WTM Knowledge Technology Group

ZMP Zero-Moment Point

Appendix B

Additional Proof

B.1 Fitness Function Formula in Detail

Detailed formula for the treatment of the servo movement in the fitness function.

Where N_f is the number of frames, N_s is the number of servos and k is the index of the knee servo. α describes the efficiency of servo movements, β describes the continuity of servo movements.

$$\beta_i = \sum_{j=i}^{N_f} |servo_{(i*6+j-1),k} - servo_{(i*6+j),k}| * 2, for\ i \in \{1, ..., 6\}$$

if β_0 to β_5 are all above 0.25° then:

$$\alpha_{tmp} = \sum_{i=1}^{N_f} \sum_{j=1}^{N_s} |servo_{(i-1),j} - servo_{i,j}|$$

$$\alpha = \max(0, N_f * \frac{300 - (\alpha_{tmp} - 1.5)}{300 * 640})$$

and

$$\beta = 1$$

else:

$$\alpha = 0$$

$$\beta = 0$$

Appendix C

Fun Facts

- 14 locomotions evolved in six different experiments.
- 17 different experiments were performed, plus preliminaries.
- 142 runs performed.
- About 16k phrases in this thesis.
- More than 25k lines of code in Java all in all.
- More than 80k lines of code were written, more than 65k removed specifically for this thesis, including configuration and GUI files.
- More than one million individuals were created.
- About five gigabyte consumed by saved populations, compressed to about 210 megabyte.

Bibliography

[1] U. Asif and J. Iqbal. A comparative study of biologically inspired walking gaits through waypoint navigation. *Advances in Mechanical Engineering*, 2011, 2011.

[2] R.D. Beer. On the dynamics of small continuous-time recurrent neural networks. *Adaptive Behavior*, 3(4):469–509, 1995.

[3] James J Collins and SA Richmond. Hard-wired central pattern generators for quadrupedal locomotion. *Biological Cybernetics*, 71(5):375–385, 1994.

[4] Charles Darwin. *On the Origin of Species*. John Murray, 1859.

[5] M. de Kamps and F. van der Velde. Neural blackboard architectures: the realization of compositionality and systematicity in neural networks. *Journal of neural engineering*, 3(1):R1, 2006.

[6] Jacques Duysens and Henry WAA Van de Crommert. Neural control of locomotion; part 1: The central pattern generator from cats to humans. *Gait & posture*, 7(2):131–141, 1998.

[7] A.E. Eiben and J.E. Smith. *Introduction to evolutionary computing*. springer, 2007.

[8] Daniel D. Lee Seung-Joon Yi Stephen McGill Yida Zhang et al. Robocup 2011 humanoid league winners. Technical report, Autonomous Intelligent Systems, Computer Science, Univ. of Bonn, Germany GRASP Lab, Engineering and Applied Science, Univ. of Pennsylvania, USA RoMeLa, Mechanical Engineering, Virginia Tech, USA, 2011.

[9] Mariano Garcia, Anindya Chatterjee, and Andy Ruina. Speed, efficiency, and stability of small-slope 2d passive dynamic bipedal walking. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 3, pages 2351–2356. IEEE, 1998.

[10] Kyrre Glette and Mats Hovin. Evolution of artificial muscle-based robotic locomotion in physx. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1114–1119. IEEE, 2010.

[11] Ambarish Goswami, Bernard Espiau, and Ahmed Keramane. Limit cycles and their stability in a passive bipedal gait. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 1, pages 246–251. IEEE, 1996.

[12] Milton Roberto Heinen and Fernando Santos Osorio. Applying genetic algorithms to control gait of simulated robots. In *Electronics, Robotics and Automotive Mechanics Conference, 2007. CERMA 2007*, pages 500–505. IEEE, 2007.

[13] Suzana Herculano-Houzel. The human brain in numbers: a linearly scaled-up primate brain. *Frontiers in Human Neuroscience*, 3(31), 2009.

[14] R. Hirose and T. Takenaka. Development of the humanoid robot asimo. *Honda R&D Technical Review*, 13(1):1–6, 2001.

[15] Auke Jan Ijspeert. A connectionist central pattern generator for the aquatic and terrestrial gaits of a simulated salamander. *Biological cybernetics*, 84(5):331–348, 2001.

[16] Kenji KANEKO, Fumio KANEHIRO, Shuuji KAJITA, Hirohisa HIRUKAWA, Toshikazu KAWASAKI, Masaru HIRATA, Kazuhiko AKACHI, and Takakatsu ISOZUMI. Humanoid robot hrp-2. *IEEE 2004: International Journal of Humanoid Robotics*, 2:1083–1090, 2004.

[17] Hiroshi Kimura, Seiichi Akiyama, and Kazuaki Sakurama. Realization of dynamic walking and running of the quadruped using neural oscillator. *Autonomous Robots*, 7(3):247–258, 1999.

[18] Hiroshi Kimura, Yasuhiro Fukuoka, and Ken Konaga. Adaptive dynamic walking of a quadruped robot using a neural system model. *Advanced Robotics*, 15(8):859–878, 2001.

[19] Y. Kuroki, M. Fujita, T. Ishida, K. Nagasaka, and J. Yamaguchi. A small biped entertainment robot exploring attractive applications. *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, 1:471–476, 2003.

[20] B. Mathayomchan and R.D. Beer. Center-crossing recurrent neural networks for the evolution of rhythmic behavior. *Neural Computation*, 14(9):2043–2051, 2002.

[21] Tad McGeer. Passive dynamic walking. *The International Journal of Robotics Research*, 9(2):62–82, 1990.

[22] G. McHale and P. Husbands. Gasnets and other evolvable neural networks applied to bipedal locomotion. *From Animals to Animats*, 8:163–172, 2004.

[23] G. Mendel. Versuche über pflanzenhybriden. *Verhandlungen des naturforschenden Vereines in Brunn 4: 3*, 44, 1866.

[24] N. Ouannes, N.E. Djedi, Y. Duthen, and H. Luga. Gait evolution for humanoid robot in a physically simulated environment. *Intelligent Computer Graphics 2011*, 374:157–173, 2012.

[25] C. Paul. Sensorimotor control of biped locomotion. *Adaptive Behavior*, 13(1):67–80, 2005.

[26] T. Reil and P. Husbands. Evolution of central pattern generators for bipedal walking in a real-time physics environment. *Evolutionary Computation, IEEE Transactions on*, 6(2):159–168, 2002.

[27] J.H. Solomon, M. Wisse, and M.J.Z. Hartmann. Fully interconnected, linear control for limit cycle walking. *Adaptive Behavior*, 18(6):492–506, 2010.

[28] R. Téllez, C. Angulo, and D. Pardo. Evolving the walking behaviour of a 12 dof quadruped using a distributed neural architecture. *Biologically Inspired Approaches to Advanced Information Technology*, 3853:5–19, 2006.

[29] Miomir Vukobratović and Branislav Borovac. Zero-moment point – thirty five years of its life. *International Journal of Humanoid Robotics*, 1(01):157–173, 2004.

[30] Michael W. Whittle. *Gait Analysis: An Introduction (Fourth Edition)*. Heidi Harrison, 2007.

[31] Inoue Yamaguchi, Soga and Takanishi. Development of a bipedal hmanoid robot: Control method of whole bondy cooperative dynamic biped walking. *IEEE 1999: International Conference on Robotics & Automation*, 1:368–374, 1999.

Erkärung der Urheberschaft

Ich versichere an Eides statt, dass ich die vorliegende Bachelor Thesis selbstständig und ohne unerlaubte Hilfe Dritter angefertigt habe. Alle Stellen, die inhaltlich oder wörtlich aus anderen Veröffentlichungen stammen, sind kenntlich gemacht. Diese Arbeit lag in gleicher oder ähnlicher Weise noch keiner Prüfungsbehörde vor und wurde bisher noch nicht veröffentlicht.

Ort, Datum Unterschrift

Enverständniserklärung

Hiermit erkläre ich mich mit der Veröffentlichung der vorliegenden Bachelor Thesis durch die Bibliothek des Fachbereichs Informatik an der Universität Hamburg einverstanden.

Ort, Datum Unterschrift