Evolution of Biped Locomotion using Linear Genetic Programming

Krister Wolff

Department of Applied Mechanics, Chalmers University of Technology
41296 Göteborg, Sweden

Abstract

The development of anthropomorphic bipedal locomotion is addressed by means of artificial evolution using linear genetic programming. The proposed method is investigated through full rigid-body dynamics simulation of a bipedal robot with 26 degrees of freedom. Stable bipedal gait with a velocity of 0.054 m/s is realized. Locomotion controllers are evolved from first principles, i.e. the evolved controller does neither have any a priori knowledge on how to walk, nor does it have any information about the kinematics structure of the robot. Instead, locomotion control is achieved based on intensive use of sensor information. Also, the energy consumption of the robot is monitored during simulation, in order to yield a pressure on evolution to favor energy efficient gaits. In this linear genetic programming approach, randomly generated individuals undergo structural evolution. This is, to our knowledge, the first example of entirely model free evolution of bipedal gaits for a system with high number of degrees of freedom.

1 Introduction and motivation

There are numerous application areas for robots with anthropomorphic shape and motion capabilities. In a world where man is the standard for almost all interactions, such robots have a very large potential acting in environments created for people. They can function in certain areas which are not accessible for wheeled robots, such as stairways or natural terrain, for instance. Furthermore, robots capable of bipedal locomotion have the ability to interact with the environment using the whole body, and climb over large obstacles in its path.

From a control theory point of view however, bipedal walking robots are more difficult to deal with than wheeled robots. Wheeled robots are designed to maintain their wheels in contact with the ground at all times. Thus, stability is usually not an issue, except when the robot is on a steep slope. Legged robots, on the contrary, lift their feet off the ground to walk. Thus, the motion of walking dynamically changes the stability of the robot.

In bipedal walking a complete cycle is divided into two phases; the *single support phase* and the *double support phase*, taking place in sequence. During the single support phase one foot is in contact with the ground and the other foot is in swing motion, being transferred from back to front position. The so called *support polygon* is formed by the robot's stance foot in the single-support phase, and by the robot's feet in the double support phase. Further, bipedal locomotion is commonly divided into two main classes; *static gaits*, and *dynamic gaits*. The most fundamental issue of bipedal locomotion, apart from the realization of the relative motions of the mechanism's links, is how to preserve the walking balance in the system [41]. That question has long been the main topic of many scientific studies, and some of the more influential results will be briefly discussed in this section.

In the research literature on topics related to bipedal walking, the terms *stability*, *equilibrium*, and *balance* are often used interchangeably. Throughout this paper, we will use the following notions in order to avoid confusion; the term *stability* refers to a system which could be analytically

treated as stationary (i.e. the static case), whereas for a non-stationary system (the dynamic case), the terms *balance* and *equilibrium* are used.

By definition, in static walking the robot is restricted to move in such a way that, in order to avoid tipping over, the vertical projection of the centre-of-mass (PCOM) of the robot remains within the convex envelope of the support polygon at all times. Further, it is assumed that the robot's motions are so slow that dynamical effects, which could arise due to the amount of torque that is applied to the robot over time, could be neglected. Thus, the system's stability depends solely on the PCOM, as described above [24]. Such a gait is also referred to as a *statically stable gait* [16]. However, the resulting gait is usually too slow for practical use in real bipedal robots [8].

A dynamic gait is any gait that is not statically stable at all times [36]. In dynamic walking, in fact, the above described PCOM condition has very little to do with the equilibrium of the bipedal system; the PCOM is allowed outside the support polygon. Instead, dynamical effects which arise due to the link's velocities and accelerations are heavily involved for maintaining the walking balance [24]. Walking with dynamic balance provide higher locomotion speed and greater efficiency than walking with statically stable gaits, but the control problem of dynamic walking is harder [24].

Within the domain of dynamic walking, the idea of the centre-of-mass projection has got its generalized equivalents, such as the zero-moment point (ZMP) [43, 42, 39, 41, 40], and the foot rotation indicator (FRI) point [16], in order to deal with the walking balance problem.

The ZMP concept was introduced by Vukobratovic *et al.* around 1969 [43, 42], but the term ZMP was formally introduced a couple of years later [44, 39]. During the single-support phase only one foot is in contact with the ground, while the other is in swing motion. In order to then maintain dynamic equilibrium of the bipedal mechanism, the *ground reaction force* (GRF) should act at the appropriate point on the foot sole (of the stance foot) to balance all forces acting on the system during the motion. It should be clear that the boundary condition for dynamic equilibrium states that the torques acting around the horizontal axes (x and y), at the point where GRF is acting, will always be equal to zero. There may exist a torque around the vertical axis, but it is a realistic assumption that it is balanced by frictional forces (given that the friction coefficient is high enough), and it will not cause foot motion. Thus, the ZMP is defined as the contact point between the ground and the foot sole of the supporting leg where the torques around the horizontal axes, generated by all forces acting on the robot, are equal to zero. Any change in the walking dynamics will cause a simultaneous change of the GRF vector, altering its magnitude, direction, and acting point ZMP. During a dynamically balanced gait, the ZMP can move only within the supporting area. In the double-support phase, the positions of both feet are fixed relative to the ground, but also in this situation the ZMP should remain within the convex envelope of the support polygon. This is the fundamental meaning of the ZMP concept [41].

The ZMP concept has been involved in numerous practical applications of anthropomorphic locomotion mechanisms. The first one was the realization of a dynamically balanced bipedal gait in 1984, with the WL-10RD robot at the Waseda University in Tokyo [37].

An extension to the ZMP conception, named the FRI point, has been suggested lately [16]. In the case of a dynamically balanced gait, the FRI point may exit the physical boundary of the support area, which the ZMP never does. However, researchers in the field still disagree about the notions and interpretations of the ZMP and the FRI point. Generally accepted definitions of, and means of treating dynamic equilibrium in bipedal walking still remain elusive [16, 41, 24, 17, 40, 1]. Moreover, Goswami [17] have recently introduced an additional concept, the zero rate of change of angular momentum (ZRAM), in order to deal with balance maintenance in dynamic gaits. However, it is beyond the scope of this paper to further examine which one is the most adequate concept.

Conventionally, there are two main approaches to bipedal gait synthesis; *off-line trajectory generation* and *on-line motion planning* [45, 24]. The first approach, the off-line method, assume that there exists an adequate dynamic model of the robot and its environment, for the generation of periodic walking pattern [26, 21, 20, 18, 48, 37]. In order to keep the ZMP within the stable region, as described above, a desired ZMP reference trajectory is generated. The reference trajectory can for instance be generated

using *spline functions* [2]. Then, a body motion that adheres to the desired ZMP trajectory is derived, using knowledge about the inverse kinematics relationships and the system's dynamics. Only if the ZMP stays within the stable region, the generated gait can be implemented on the actual robot.

The second approach uses limited knowledge about the kinematics and dynamics of the robot and its environment. In on-line controller design, this is referred to as finding the *plant model* and *transfer function* of the system one wishes to control. A plant model describes the static relationship between input and output, and a transfer function includes dynamic effects as well. Given the transfer function, appropriate torques can then be generated by means of e.g. a PID regulator. This type of control scheme relies much on the feedback. Control methods for bipedal walking based on the dynamical equations of motion [15, 13] have been proposed. In order to decrease the demand for computational resources in real-time implementations, simplified models, such as the inverted pendulum model [23, 30], and a static motion strategy [49], have been proposed. With on-line methods higher walking robustness is achieved at the cost of an increase in demand for computational resources, compared with off-line methods.

Control policies based on classical control theory, like the ones outlined above, have been successfully utilized for bipedal locomotion in various implementations. However, the success of these methods relies on the calculation of reference trajectories for the robot to follow. That is, trajectories of joint angle, joint torque, or the centre-of-mass of the robot are calculated so as to satisfy constraints, such as the ZMP criterion [18, 2, 48, 13, 30, 12]. When the bipedal agent, either it is a real physical robot or a simulated equivalent, is acting in a well-known environment the abovementioned control methods should work well. However, the drawback of using such a method when acting in a realistic, dynamically changing environment is that reference trajectories can rarely be specified there. A bipedal robot will encounter unexpected situations in the real world, which cannot all be accounted for on beforehand. Therefore, alternative biologically inspired computational methods have been considered when generating gaits and other behaviors for bipedal robots. Such methods do not, in general, require any reference trajectory.

Some researchers have employed a connectionist approach, i.e. artificial neural network (ANN) based strategies, for control of bipedal walking. Katic and Vukobratovic have reviewed such methods in a recent paper [24]. Others, including the authors of this paper, advocate the use of evolutionary algorithms (EAs) for bipedal gait synthesis, e.g. [2, 8, 31, 7].

Although many works have been published on natural bipedal gait generation by means of using EAs, and in particular genetic algorithms (GAs), most of them concerned on a simplified simulation model of the biped, i.e. the 5-link model developed by Furusho and Masubuchi [14]. Such a simplified model might be useful for illustrating a gait generation method, but it has a severe limitation; only motion in the sagittal plane is considered. Of course, the impact of the proposed technique would have a greater impact if demonstrated in a 3-dimensional rigid-body simulation instead. There exist, however, some examples in the research literature of synthesizing locomotion for full rigid-body simulated bipeds, by incorporating EAs.

Pettersson [31] have reported on the development of a method for generating walking behaviors for bipedal robots. An adaptation of evolutionary programming (EP) to the case of finite state machines (FSMs) is used to operate both on the structure and on the parameters of the robotic brain. The method has been demonstrated on a simplified five link robot, constrained to move in the sagittal plane. Two test cases were used; energy optimization and robust balancing. In the case of energy optimization, a 134% improvement in walking length was obtained. In the case of robustness, FSMs evolved that could withstand some perturbations, which the initial FSMs could not. The authors have also recently reported ongoing research with evolution of RNNs, used as CPGs, for balancing behavior of a simulated bipedal robot [32].

An example of using a GA to optimize the operational parameters of central pattern generators (CPGs), which were used to generate a walking pattern, was reported by Mojon [27]. Here, the model for the CPGs was based on the equations of the harmonic oscillator. A very realistic bipedal model was used. It was a model of a real physical humanoid robot, namely the QRIO[1] robot, developed by the

[1] http://www.sony.net/SonyInfo/QRIO/top_nf.html

Sony Corp. Sony's QRIO has 25 DOF, but the simulated model had "only" 21 DOF. The outcome of these simulations was that the simulated robot could walk in a straight line in a speed-equivalent of 1.5 km/h. This should be compared with the real QRIO robot that can walk in a speed of 2.5 km/h, on a flat surface. Also, the evolved gait did not look very realistic at all.

Recently, a research team from Australia used a standard GA with fitness proportionate selection scheme to evolve the parameters of a limited spline control system [4]. They managed to evolve a cyclic walking pattern, which could generate a slow forwards walk.

Reil and Massey [35] and Reil and Husbands [34] have used GAs to optimize fully recurrent neural networks (RNNs), used to control biped walking. They used a real-valued encoded GA to optimize weights, time constants, and biases in fixed architecture RNNs. Regarding the network's architecture; the fixed number of neurons was 10, and node 1 through 6 was considered as motor neurons. Connections between neurons could be removed, in the sense that its corresponding weight was set equal to zero. Their biped model has six DOF, and it consists of a pair of articulated legs connected with a link. Reil et al. reported results capable of biped walking in a straight line on a planer surface, without the use of proprioceptive input. Furthermore, simple sensory input to locate a sound source was integrated to achieve directional walking.

Ok et al. [29] have applied genetic programming (GP) to induce parts of a "nervous system" for a 3d simulated bipedal robot. The neural system represents a rhythm generation mechanism with CPG, consisting of neural oscillators, and a global feedback network. The authors have assumed that parameters and structures within single neural oscillators are known and fixed, and focused on creation of the feedback networks between the neural system and the body dynamics system. They managed to evolve a global feedback network which could generate bipedal gait, but only 4 steps of walking could be achieved with the simulated biped.

Using a genetic programming (GP) approach, Ziegler et al. [51] evolved gait controllers for a full rigid-body simulated bipedal robot. Their GP-system used linear genomes as representation. The best control program that emerged was capable of moving the biped forward by making fast, small steps.

The idea to utilize EAs for generating gait controllers for bipeds is not new, as shown by these examples. However, there exist only a few examples, to our knowledge, where one goes beyond parametric optimization, and optimize also the structure of the gait controller. Then, a more flexible scheme than the GA binary representation is required, e.g. GP [25].

While GAs usually operates on fixed-length binary strings, GP deals with the evolution of syntactically correct computer programs. The ordinary representation scheme in GP is called a *tree representation*. A GP tree consists of *functions* and *terminals* which, assembled into a structure, can be executed as a computer program. A terminal provides a value to the system, while functions process values contained in the system. The tree representation is a very flexible program structure. The set of functions can be very rich; any programming construct in any programming language may be used. In GP, there is much freedom to choose which functions to include. Further, the *size* of the GP-trees are allowed to vary during evolution.

Another sometimes used representation in GP is the *linear* representation, and consequently the variant of GP using that representation is referred to as linear genetic programming (LGP) [6]. LGP evolves sequences of *instructions* of an imperative programming language, e.g. C language or FORTRAN. LGP is, compared with tree-based GP, relatively easy to implement. Unlike tree-based GP, LGP also facilitates the use of *multiple program outputs*. This makes LGP ideally suited for the task of evolving controller programs for bipedal walking.

In this work, LGP is utilized to generate locomotion controllers from first principles for a simulated bipedal robot with 26 degrees of freedom (DOFs). It is noteworthy that in this robotic system there is no model of the bipedal system, neither any a priori knowledge on how to walk, available to the evolved controllers. Randomly generated individuals, or controllers, undergo evolutionary processes on the structural level. This is, to our knowledge, the first example of an entirely model free evolution of bipedal locomotion for a system with a high number of DOFs.

However, it should also be noted that learning such a complex task as bipedal locomotion from

scratch, starting with an empty brain and an already developed body, is far from trivial. For living animals in the nature, including humans, the body and the brain develop together. In this approach, the body of the bipedal does not undergo any evolutionary process. Still, this project is an attempt to generate a *robust* and *anthropomorphic* (i.e. human-like) bipedal gait by means of artificial evolution.

The rest of this paper is organized as follows: in Section 2 the physical simulation environment and the biped are described, together with the LGP system. Then, the simulation setup is described in Section 3. In Section 4 the results of the simulations are described, and in Section 5 the results are discussed. Finally, the paper ends with Section 6 Future work.

2 Method

In this section the methodology for the rigid-body simulations and the evolutionary method used here will be described. First, the physical simulation environment and the biped model are described. Second, the LGP system is explained.

2.1 Physical simulation

For simulating the articulated rigid-body dynamics the Open Dynamics Engine[2] (ODE) library is utilized. In ODE the equations of motion are derived from a Lagrange multiplier velocity-based model, and a first order integrator is used. The actual implementation of ODE puts emphasis on speed of execution, rather than on accuracy. It is therefore mainly used for qualitative engineering tasks involving rigid-body dynamics simulation.

2.1.1 Robot model

The biped model used here has no original equivalent design in the real world, but it could be seen as representing a generic bipedal robot model. It was created from the *body* and *joint* primitives available in the ODE simulation package. The robot model consists of 22 rigid-body parts, and 18 ODE joints. There are 8 *universal* joints, and 10 *hinge* joints used to connect the rigid-body parts into an articulated rigid-body structure (*universal* and *hinge* are the internal names of specific joint types in ODE). The rigid-body primitives used are 9 *capped cylinders* and 13 rectangular *boxes*. The robot's structure is defined using multiple chains, starting from its feet with each link described in terms of the previous links. This composition results in a 26 DOF bipedal model.

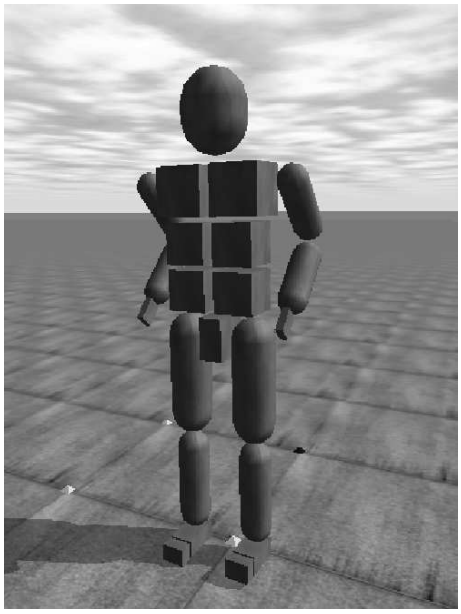

Figure 1: The biped model used.

2.1.2 Sensor feedback

Sensors monitoring the internal state of the robot, such as joint angles are referred to as *proprioceptive sensors*. In this setting, the *current joint angles* of the previous time step of the simulation are used by the evolved controller to compute the next set of motor signals for the robot.

Simulating a biped robot in a realistic environment most likely requires feedback loops between the robot's control system and the robot's body, as well as between the control system and the environment. The set of external sensors constitute the robot's "window" to the environment. Those sensors can measure quantities such as the robots acceleration or inclination relative a fixed coordi-

[2] http://ode.org/

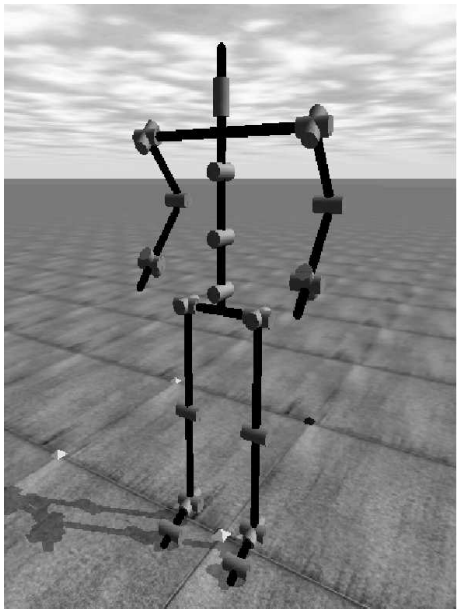

Figure 2: Schematic picture of the biped, showing its kinematics structure. The model consists of 19 links (black) interconnected by 18 joints (yellow), all in all resulting in 26 degrees of freedom.

nate frame, light intensity, external forces applied to the robots body, and sound waves, to mention a few examples.

A rigid body has six DOFs; three translational, and three rotational DOFs. In robotics, accelerometers are utilized to measure linear movement, and gyroscopes are used to measure rotations. To fully keep track of the movement of an object, tree accelerometers and three gyroscopes is sufficient. Mounted together, these form an inertial measurement unit (IMU). In this set of simulations, the robot was equipped with an *artificial balancing sense*, in the form of a virtual IMU in its head, and a three axis accelerometer in each foot. The linear accelerations and angular rates are obtained directly from the rigid body simulation in every time step.

2.1.3 Controller Model

Joints can be powered in ODE. i.e., for each joint in the bipedal model used here, there is a motor associated with it. The ODE package supplies a method, used in this investigation, to control the joints by simply setting a desired speed of the motor and a maximum force or torque that the motor will use to achieve that speed. Thus, in order to make the robot walk appropriate torque values need to be applied to the joints in each time step of the simulation.

In this implementation the speed and torque values have been pre-set, and the output of the evolved controller just sets the rotational direction (+) or (−) for each joint in the simulation. Appropriate values for *velocity* and *torque* were determined empirically. The generated controller takes as input the relative positions of the joints, i.e. the joint angles, additional sensor readings, and constants.

Appropriate motor signals are generated from the raw output of the evolved individual by means of a modified signum function ξ. These signals are then sent to the robot for execution, see Fig. 3 for more details. The modified signum function is defined as follows:

$$\xi(x) = \begin{cases} -1 & \text{if } x < \kappa \\ +1 & \text{if } x > \kappa \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

The value of the parameter κ was set to 0.12 in the simulations. Furthermore, updated motor signals are sent to the robot at a constant time interval, covering a fixed number η of simulation time steps. The reason for not letting the motor signals be updated in each simulation time step was to avoid that joints could change their rotational direction in every time step, which would have resulted in rapid oscillations of the joints. Such movements are not desirable.

2.2 Linear genetic programming

The EA variant used here is referred to as linear genetic programming (LGP), and it follows the paradigm of genetic programming of automatic induction of syntactically correct computer programs. LGP evolves sequences of instructions of an imperative programming language, e.g. C language, FORTRAN, or machine code [28]. The instructions of LGP are restricted to operations (and conditional operations) that accept a minimum number of constants or memory variables, called *registers*, and assign the result to a register

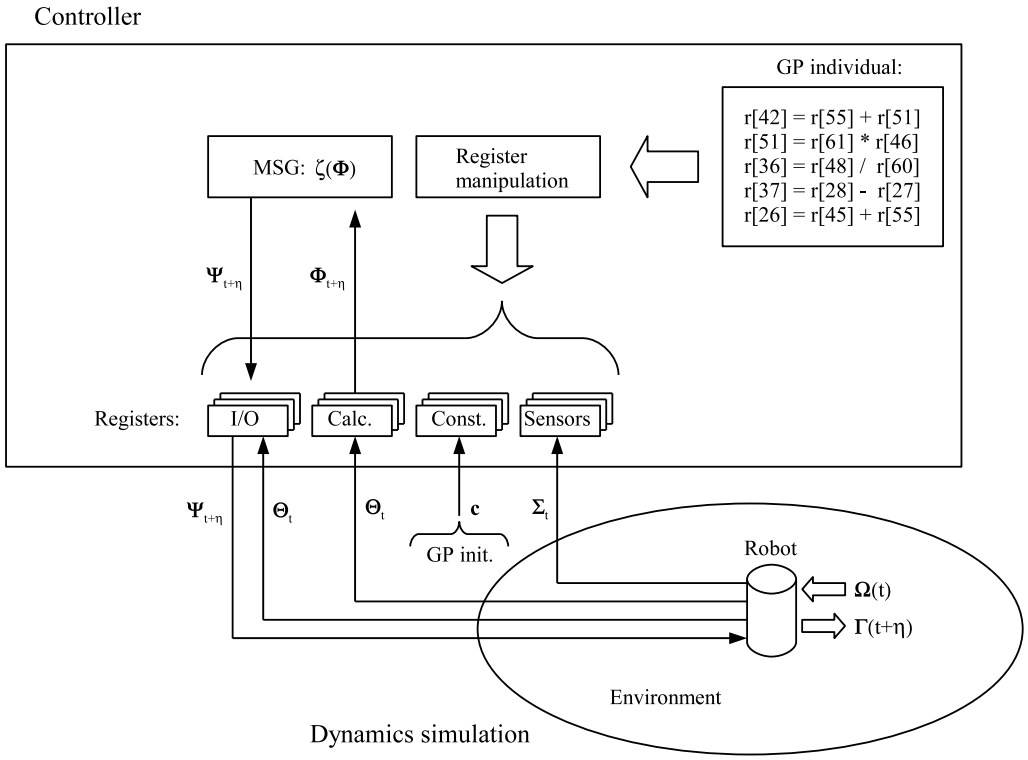

Figure 3: Schematic depiction of the information flow through the robot control system. At the discrete timestep t, the robot receives perceptual input $\mathbf{\Omega}(t)$ through its sensor channels. The sensor signals $\mathbf{\Sigma}_t$, are then fed into the *sensor registers* ($r_{55} - r_{66}$). Simultaneously, the robot's current joint angle positions $\mathbf{\Theta}_t$ are recorded in the *I/O-* and *calculation registers* ($r_0 - r_{25}$ and $r_{26} - r_{51}$ respectively). There is one register of each of these types associated with each degree of freedom. At the beginning of the GP run, the *constant registers* ($r_{52} - r_{54}$) were supplied with the values 0.001, 0.01, and 0.1 respectively. The VRM then executes the GP-individual (program), which manipulates the contents of the calculation registers. The I/O- sensor- and constant registers are *read only* during this stage. When the program execution has terminated (i. e. evaluation steps off the end of the program), motor signal generation (MSG) is initiated; a modified signum function ξ operates on the final contents $\mathbf{\Phi}_{t+\eta}$ of the calculation registers, and supplies the output $\mathbf{\Psi}_{t+\eta}$ to the I/O registers. These motor signals are then sent to the robot, which then in timestep $t + \eta$ execute these motor commands and accomplish some actions $\mathbf{\Gamma}_{t+\eta}$.

again, e.g. $r_0 := r_1 + 1$. In this section, the LGP concept will be explained in more detail.

2.2.1 Evolutionary algorithm basics

In *The Origin of Species* [10], it was argued that all existing organisms are the descendants of a few simple ancestors that arose on Earth in the distant past, and that the driving mechanism behind this evolutionary development was natural selection. Over the past 35 years or so, the principle of natural selection has been successfully utilized on computers to optimize a solution towards a pre-defined goal, and from this evolutionary algorithms (EA) have developed. Several research subfields, such as evolutionary programming (EP) [11], genetic algorithms (GA) [19], evolution strategies (ES) [33], and genetic programming (GP) [25] have emerged. Although they differ from each other in many aspects they all mimic natural evolution in

some ways. Based on very simple models of organic evolution, EAs are applied to various problems such as combinatorial optimization problems or learning tasks.

The main ingredients common for all types of EA are *population of solution candidates*, *variation operators*, *conservation operators*, *quality differentials*, and *selection methods* [3]. The first decision that has to be made, however, when designing an EA in order to solve a problem is how to define the *representation* of a solution.

In GAs solution candidates are represented as fixed-length strings of zeros and ones, and in ES the individuals are represented as vectors of continuous variables. In EP, the representation scheme consists of collections of finite state machines. In GP there are three principal representation structures used, which are called *tree*, *linear*, and *graph*. An assembly of solution candidates, or individuals, are simply called a *population*.

The *variation operator* ensure that new aspects of a problem are considered. In EAs this operator is called *mutation*. It comes with three EA parameters determining its strength, its spread, and its frequency of application. A very strong mutation operator would basically generate a random parameter at a given position within a solution. If applied to all positions within a solution, it would generate a solution completely uncorrelated with its origin, and if applied with maximum frequency within the entire population, it would erase all information generated in the population so far. Variation operators are means to increase the diversity in the population of solution candidates.

The *conservation operators* are used to consolidate what has already been learned by various individuals in the population, i.e. conservation operators are means to decrease the diversity. *Recombination*, or *crossover* of two or more solutions is the primary tool for achieving this goal. Provided the different parameters in a solution representation are sufficiently independent from each other, combinations of useful pieces of information from different individuals would result in better overall solutions. There are different ways to achieve a mixing of solutions. The most frequently used method involves two individuals recombining their information, although multi-recombinant methods are also used. Common recombination methods are one-point, two-point, or n-point crossover for discrete parameter values between two individuals, as well as intermediate recombination for continuous parameter values. A subset of the population can also be copied without change to the next generation, i.e. *reproduction*.

An important property of an EA is to have some *quality differentials*, i.e. a graded *fitness function* that distinguishes a better solution candidate from a good one, or a bad candidate from a really bad one. If there was only a binary fitness function, stating a solution by "1" and no solution by "0", then there would not be enough diversity among individuals to drive the evolutionary search process.

Since the population is finite, not all the individuals generated by the EA can be stored. Both the solution candidates to be included in further evolution and the ones to be replaced from the population are selected on the basis of the quality differentials. Following Darwin's tradition, this procedure is called *selection* [3]. The most commonly used selection method is *fitness-proportional selection*, which stems from the GA community. In GP *tournament selection* is also widely used. A generic evolutionary algorithm is summarized in Algorithm 2.1 below.

Algorithm 2.1 (Generic EA)

1. Randomly *initialize* a population of solution candidates.
2. By using a certain selection method, *select* individuals that are fitter than others. The fitness measure defines the problem that has to be solved by the algorithm.
3. *Generate new individuals* by applying one or more of the genetic operators reproduction, recombination, and mutation.
4. If the *termination criterion* is not met, go to 2.
5. *Stop*. The best solution found is represented by the fittest individual.

□

2.2.2 LGP fundamentals

When dealing with the actual implementation of LGP, the concept of the so called *virtual register*

machine (VRM) [22] is useful. The term is derived from a hypothetical model of computation, called a register machine [38].

A particular instance of a virtual register machine, VRM-$\mathcal{M}_{m,n}$, consists of a finite set of registers $R = \{r_1, ..., r_m\}$, each of which can hold a floating point value, a finite set of instructions $I = \{I_1, ..., I_n\}$. Such a register machine, implemented in a high level imperative language, is referred to as "virtual" because it has to be interpreted by software. The registers constitute the machine's mutable memory, and all program input and output is communicated through the registers. Additionally, registers may also be used to supply the program with constants, which can be synthesized in otherwise unused registers. Therefore, it seems natural to define a *register state* $\mathbf{S_r}$ as a vector of m floating point values

$$\mathbf{S}_r \equiv (r_1, ..., r_m)$$

Program inputs are supplied in the initial state $\mathbf{S}_{r,i}$ and outputs are taken from the final register state $\mathbf{S}_{r,f}$. Beside the required number of *input registers*, additional registers can be provided in order to facilitate calculations. These so called *calculation registers* (sometimes also referred to as *internal work registers*) are normally initialized with a constant value each time before a program is executed on the fitness cases. The VRM-$\mathcal{M}_{m,n}$ is allowed only to write to the calculation registers when executing the list of instructions (program). Thus, the input registers are write protected during this phase.

Finally, one or more registers may be defined as *output register(s)*, either among the calculation registers or among the input registers. The LGP structure facilitates the use of multiple program outputs. By contrast, functional expressions like trees calculate one output only [6].

In addition to the external register state, VRM-$\mathcal{M}_{m,n}$ maintains an internal state: the *program counter*, PC. The program counter is an integer that selects which instruction to fetch and execute. Branch instructions modify the PC to point to the branch's target; all other instructions always increment the PC to point to the next instruction. By definition, in LGP a *program* $\mathbf{P}$ is a vector of n instructions

$$\mathbf{P} \equiv (I_1, ..., I_n)$$

Table 1: Instruction Set

Instruction type	*General notation*	*Input range*
Arithmetic operations	$r_i := r_j + r_k$ $r_i := r_j - r_k$ $r_i := r_j \times r_k$ $r_i := r_j / r_k$	$r_i, r_j, r_k \in \mathbf{R}$
Trigonometric functions	$r_i := \sin r_j$	$r_i, r_j \in \mathbf{R}$
Conditional branches	$if(r_j > r_k)$ $if(r_j \leq r_k)$	$r_j, r_k \in \mathbf{R}$

The program counter PC naturally corresponds to an index i, of $\mathbf{P}$. A program terminates when $PC = n$, i.e. when evaluation steps off the end of the program.

The choice of *instruction set* should be based on the same principles as in tree-based GP. The ability of GP to find a solution strongly depends on the expressiveness of the instruction set. On the other hand, the dimension of the search space, i.e. all possible programs that can be built from these instructions, increases exponentially with the number of instructions and registers. Therefore, the set of possible instructions is limited to those of Table 1. The presence of the periodic `sine` operator in the operator set is strongly motivated by the fact that locomotion in biological creatures is cyclic in nature. To assure semantic correctness, a slightly different division operator (`div`) than the standard division operator is defined. It works exactly as the standard division operator, except for zero denominator input. In that case, the *protected division operator* returns a large constant value, here set to 10^8.

In LGP conditional branching is usually interpreted in the following way: if the condition in the `if` statement evaluates to `true`, the subsequent instruction is executed. If, on the other hand, the condition in the `if` statement evaluates to `false` that instruction is skipped, and program execution jumps to the next instruction instead, see Example 2.1. When using conditional branching, the control flow of the program may be different for different input situations, which is the motivation for including branching in this investigation.

Example 2.1 (Linear Program Execution)

This example shows a simple *register manipulating program* in C notation. The code from line 2 to line 10 constitutes the linear program, or instruction body, and the other code is just a wrapping for managing the GP program. The execution of the linear genome starts at the *topmost* instruction, i.e. `r[42]=r[55] * r[51]`, and proceeds *down* the list, one instruction at a time. Suppose that the first `if` statement (line 3) evaluates to `true`, and that the second `if` statement (line 7) evaluates to `false`. Then, the instructions on lines 2, 3, 4, 5, 6, 7, 9, 10 will be executed, but *not* the instruction on line 8.

```
1:  void GP(double* r){
2:     r[42] = r[55] * r[51];
3:         if(r[18] > r[32]);    \\true
4:     r[51] = r[61] * r[46];
5:     r[36] = r[48] - r[60];
6:     r[37] = r[28] * r[27];
7          if(r[15] > r[27]);    \\false
8:     r[42] = r[18] + r[45];
9:     r[37] = r[53] / r[36];
10:    r[26] = r[45] + r[55];
11:     return;
12: }
13: int main(void){
14:     double reg[80];
15:     ...
16:     GP(reg);
17:     ...
18:     return 0;
19: }
```

□

The actual register machine uses instructions for two or three registers. Three-register instructions operate on two arbitrary registers and assign the result in a third register (e.g. `r[i]=r[j]+r[k]`), while two-register instructions operate on only one operand (e.g. `r[i]=sin(r[j])`).

The term 'register' here can be misleading. The GP system *does not* operate on the actual CPU registers of the computer. The registers in this context are simply allocated as arrays of integer values in the RAM. Each instruction consists of four elements, encoded as integers, and the whole individual is a linear list of such instructions, see example 2.2 below.

Before the *initialization* process can begin, the maximum allowed genome length has to be defined. In LGP, the length of an individual is simply the number of basic instructions it contains. A linear genome individual is initialized in the following way:

Algorithm 2.2 (Initialization)

1. Randomly choose a *length* of the genome within the permitted interval.
2. Add a randomly chosen *instruction* to the genome.
3. Fill out the instruction with *register references* to randomly chosen registers from the register set and/or randomly chosen constants from the constant range.
4. Repeat step 2 and 3 until the number of instructions added equals the length chosen in step 1.
5. Repeat step 1 to 4 for every individual in the population.

□

So far, only steady-state *tournament selection* has been used. The following tournament selection scheme was used: two individuals are randomly picked from the population to compete against each other, and the best individual is selected with probability one. This procedure is carried out twice, and the two selected individuals mate and produce two new individuals. The newly created individuals are then inserted into the population, on the two worst individuals' place, which are then erased from the population. The quality of the randomly initialized individuals is usually very low. The initial population is transformed in an evolutionary search process by means of the *genetic operators*, crossover and mutation.

Crossover works by swapping parent individuals linear genome segments. Two crossover points in the first parent's genome are randomly chosen, and then two other crossover points in the other parent is chosen. The instructions in-between the crossover points are swapped, and the resulting individuals are the new offspring. This crossover operator allows the genome lengths of the individuals to vary over time, see figure 4.

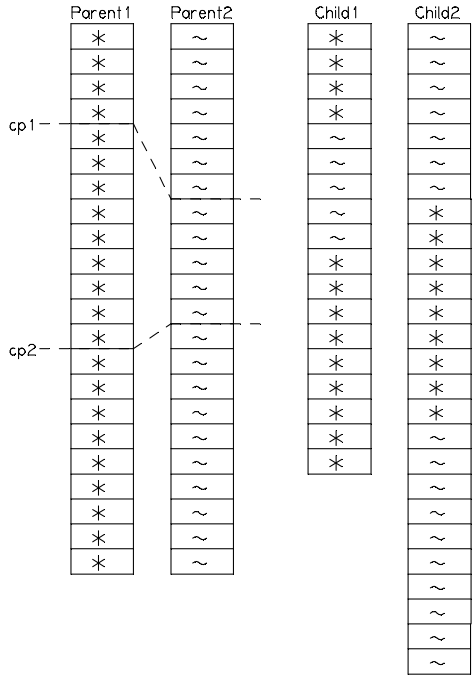

Figure 4: Length-varying two-point linear GP Crossover. Selecting different crossover points in both parents and swapping the intermediate genome segments varies the length of the genomes. The tokens ($*$, $\sim$) represent arbitrary instructions from the set.

In LGP, the mutation operator works on two different levels. The mutation operator first selects *which* instruction to be mutated, and then it makes one or more changes to that instruction. There are normally three types of change that can occur:

- Change any of the register destinations to another randomly chosen register.
- Change the operator in the instruction to another randomly chosen operator.
- Change a constant to another randomly chosen constant.

The atomic units of the mutated instruction (operator, registers and/or constant) must be members of the original sets.

In all GP-applications, finding a proper *fitness function* that guides the artificial evolution in the desired direction is of great importance. The primary goal of this project includes producing a sustainable, robust, anthropomorphic (i.e. human-like) bipedal gait. Defining the characteristics of a sustainable, robust gait is almost self evident; the biped should never fall, no matter how long time a good individual is being executed, and it should handle perturbations from the environment in a robust way. One can easily find proper measures for this, like 'time-of-walking', 'walking-speed', etc. Defining a measure for 'anthropomorphic' in this context is harder. Up to now, we simply let this be judged by the humans watching it.

2.2.3 EA for bipedal walking

Since the simulation is entirely deterministic are all individuals evaluated under identical conditions. They all start from the same upright pose, with the same orientation. The execution time for an individual is maximum N simulation time steps (evaluation time = $N/0.009$ s, where $N = 4000$), and if an individual causes the robot to fall before this time is reached, the evaluation is terminated. In the beginning of a simulation, the great majority of individuals are terminated before the intended evaluation time. The settings of the EA are summarized in table 2. The evolutionary algorithm used in these simulations was a steady-state tournament selection algorithm, utilizing two-point crossover and mutation, and with the following execution cycle:

Algorithm 2.3 (Evolutionary Algorithm)

1. Randomly *initialize* a population of individual solutions.
2. *Evaluate* all individuals according to the *fitness* measure, Eq. 2. The simulation comprises the following steps:
   a. Create an instance of the simulated robot.
   b. Record the initial position in 3d-space of all the robot's limbs.
   c. Execute the individual for N simulation time steps.
   d. Record the final position of all the robot's limbs.
   e. Compute the fitness value.
   f. Destroy the simulated robot.

3. Perform tournament *selection*.

4. Apply genetic operators *crossover* and *mutation* on the winners to produce two children.

5. *Replace* the two losers in the population with the offspring.

6. *Evaluate* the children according to the fitness function (as described above in step 2).

7. If the *termination criterion* is not met, go to step 3.

8. *Stop*. The best individual represents the best solution found.

□

2.2.4 Encoding scheme for walking

In the current implementation a chromosome takes form of an array of integers, as shown in example 2.2, where each row in the chromosome is interpreted as an instruction. An instruction is encoded in the following way: the first and second elements of an instruction refer to the registers to be used as arguments, the third element corresponds to the operator, and the last element is a register reference for where to put the result of the operation. The arithmetic operators are encoded as `add`=1, `sub`=2, `mul`=3, `div`=4, `sine`=5 in the genome. Conditional branching operators are encoded in the third element as 6=`if(r[j] > r[k])`, and 7=`if(r[j[ <= r[k])`. When decoded, the first line (instruction) in example 2.2 is interpreted as `r[42]=r[55] * r[51]`.

Example 2.2 (Linear Program Encoding)

This individual comprises nine instructions. The genome shown in this example is also shown in decoded form in Example 2.1.

```
55, 51, 3, 42,
18, 32, 6, 41,
61, 46, 3, 51,
48, 60, 2, 36,
28, 27, 3, 37,
15, 27, 6, 33,
18, 45, 1, 42,
53, 36, 4, 37,
45, 55, 1, 26,
```

□

The information flow between the control system and the robot is communicated through the registers. Currently, there are 67 registers used. At timestep t sensor signals are fed into the *sensor registers* ($r_{55} - r_{66}$). Simultaneously, the robot's current joint angle positions are recorded in the *I/O-* and *calculation registers* ($r_0 - r_{25}$ and $r_{26} - r_{51}$ respectively). Registers for ephemeral constants ($r_{52} - r_{54}$) are initiated at the beginning of the GP run. An ephemeral constant is initialized with a random value when created, and then it retains that value throughout its lifetime during a run. When the VRM executes the GP-individual (program) the contents of the calculation registers are manipulated. When the program execution has terminated, motor signal generation (MSG) is initiated; a modified signum function (Eq. 1) operates on the final contents of the calculation registers, and supplies the output to the I/O registers. These motor signals are then sent to the robot, which then in timestep $t + \eta$ executes the motor commands. A more detailed description of information flow in the controller is given in figure 3.

3 Simulations

A similar simulation using a simplified version of the LGP-system described in this paper was conducted by Wolff and Nordin [47] (Paper III in [46]). The bipedal model used was also simpler, i.e. it had a smaller number of DOF. The only feedback to the control system was the current joint angles of the biped.

The biped model used in the simulations described in this paper has a more human-like appearance. Further, the control system is enhanced with readings from accelerometers, and from a virtual inertial measurement unit (IMU). The LGP system is augmented with conditional branching as well.

In this section, the organization of three different sets of simulations is described. In the first set of simulations approximately 80 test runs were conducted in order to develop, test and verify the functionality of the evolutionary system with physics simulation. Preliminary results were obtained, which motivated some modifications and improvements of the evolutionary simulation system. These simulations are referred to as the *groundwork simu-*

Table 2: Koza style tableau, showing parameter settings for the evolution of locomotion control programs for the simulated humanoid robot.

Parameter	Value
Objective	Approximate a function that produce robust biped gait
Terminal Set	66 integer registers
Function Set	`add, sub, mul, div, sine, branching`
Raw Fitness	According to Eq. 1, scalar value
Standardized Fitness	Same as Raw Fitness
Population Size	between 8 and 512 individuals
Initialization Method	Random Gaussian distribution
Simulation Time	corresponding to 36 s. of real time
Crossover Probability	Between 0.0 and 1.0%
Mutation Probability	1.0
Initial Program Length	Gaussian distribution, between $\langle 32 \rangle$ and $\langle 512 \rangle$ instructions.
Maximum Program Length	1024 instructions
Maximum tournament number	25000
Selection Scheme	Tournament, size 4
Termination Criteria	Max tournament number

lations (Section 3.1). These simulations constitute the foundation for the setup of the simulations coming next.

As described in Section 3.2, *main simulations setup*, 60 independent simulation runs were performed mainly to investigate how the outcome is affected by specific parameter choices for the LGP system and the robot. Finally, a number of *additional simulations* are appended (Section 3.3). These were carried out in order to investigate some possible future enhancements of the evolutionary method.

3.1 Groundwork simulation setup

Here, the organization of the initial set of simulations is outlined. The fitness measure is described in detail, and its configuration is motivated. An implicit fitness measure, i.e. *energy discharging*, is introduced as well. The four different stopping criterions used are described as well.

3.1.1 Fitness Measure

Several different approaches to fitness measure was investigated, the first one investigated by Wolff and Nordin [47]. The initial assumption mentioned there, that it should be important to include the "height above ground of the robot" to the fitness measure, was dropped. The reason for this was that here, when this measure was included, it was found that the robot was prevented from moving freely enough to really do any advancement in efficient walking. To simply drop that term seemed to be the best choice here. Another problem arose when the gait controllers had reached the level of performance to maintain the robot in an upright pose and balance it. In order to reach higher levels of fitness they just let the robot stand idle until there was a very short moment of evaluation time left, and then take a large leap forward. Then, they get rewarded for good distance coverage over the trial, and the fact that the robot would have fallen

to the ground if evaluation was not terminated at that moment did not affect the evolutionary process. This became visible only when individuals were examined after the evolutionary run. Individuals can be recapitulated for infinitely long time afterwards. However, a proper fitness measure was found to be:

$$F = \sum_{t=0}^{N} (\mathbf{r}_R + \mathbf{r}_L)\hat{\mathbf{y}} \tag{2}$$

where N is the number of time steps in the simulation, $\mathbf{r}_R$ and $\mathbf{r}_L$ are the position vectors to the robot's right and left foot respectively, and $\hat{\mathbf{y}}$ is the unit vector along the y-axis, which is the forwards direction of the robot. One way to cope with the programs keeping the robot idle most of the time and activating it at the end of the evaluation, is simply to give it a (small) reward in every time step. The fitness function above is a sum of the contributions given for the robots actions in each time step. Thus the individual will be given higher fitness if the robot *gradually* approaches end point of its locomotion trajectory, rather than if it stand still until evaluation has almost finished, and then takes a large leap. That is, if the end point is the same in both cases.

3.1.2 Energy Discharging

In the early runs, most individuals hardly walked at all, and those who did, they walked in a very unnatural way. For those individuals, some of the joints showed "oscillating" behavior, i.e. they rapidly switch direction of revolution between each controller time step η. The oscillations were hardly visible for an observer, but nevertheless the robot slowly drifted across the floor. Another peculiar means of locomotion that emerged was robots walking sideways, like a crab fish do. That kind of gait is faster than the oscillating gait described above, but neither it is anthropomorphic. In order to prevent such misbehaviors to emerge, an energy discharging function was added. Human bipedal locomotion is very energy efficient, compared to the walks of humanoid robots. As an example, the state-of-the-art Honda humanoid Asimo uses at least 10 times the energy (scaled) of a typical human [9].

In each evaluation round the energy consumption of the biped was monitored as it moved its joints, and it was only allowed to use a specific amount of energy. The assumption was that this should yield a pressure on evolution to favor specific gaits that are energy efficient, and thus more anthropomorphic gaits ought to emerge.

Energy consumption, E_{jt}, of the j:th joint during timestep t equals the work performed in that joint during the timestep t. The total energy consumption of the biped, E_{tot} then, is the sum of E_{jt} over all joints $j_i \in \{j_0, j_1, ..., j_n\}$ and all timesteps $t_i \in \{t_0, t_1, ..., t_n\}$. Further, the work performed by a (generalized) force in circular motion, moving from an angle φ_a to φ_b, is defined by the following relationship:

$$W_{ba} = \int_a^b |\mathbf{M}| d\varphi \tag{3}$$

where $\mathbf{M}$ is the applied torque. In the simulation, time is discretized and the applied torque is constant during each timestep. Thus, the total energy consumption is given by:

$$E_{tot} = \sum_{j,t} (\varphi_{b,jt} - \varphi_{a,jt}) \, |\mathbf{M}_{ba,jt}| \tag{4}$$

When the total energy consumption E_{tot} equals some predefined value, evaluation of that individual is terminated what so ever. This can be thought of as a motor battery that discharges as the robot moves, and when the battery eventually gets empty, the robot stops.

3.1.3 Termination of Individuals

There are several ways in which the evaluation process of an individual could be terminated. First of all, there is a maximum allowed evaluation time of 4000 timesteps. When this time has passed, termination is definite. Second, if the current individual causes the biped to fall, evaluation will also be interrupted. Third, too high energy consumption (as described in section 3.1.2) could impose the termination of an individual as well. Last, in order to speed up the evolutionary process, another continual monitoring of individuals has also been introduced. A conditional termination criterion is specified according to the following expression:

$$\frac{F(i) + i_c}{t_i} < \frac{F_c + i_c}{i_c} \tag{5}$$

where $F(i)$ equals the fitness contribution at timestep t_i. F_c and i_c are constants, set to 20.0 and 1000 respectively. The interpretation of the above inequality is that the fitness contribution in each timestep should grow at least linear with time. The right hand side of Eq. 5 is a constant, specifying the minimal growth rate accepted. If the expression evaluates to *true* at some point, evaluation of that individual is terminated immediately. This constraint has the effect on evolution that individuals which spend most of their evaluation time standing idle are terminated earlier. Thus, a large portion of the expensive simulation time is freed up. Further, a positive side effect is that such individuals receive lower fitness scores.

3.1.4 The role of body proportions

In human development, basic skills like walking are acquired in an early age, usually before the age of 2.5 years. At that age, however, the body development is far from accomplished. That is, a child's head is extremely large in proportion to the rest of the body, compared to an adult's body. The head of a very young child make up about a quarter of their total height [5].

For that reason, simulation runs with different body proportions of the bipedal has been conducted. That is, evolution with the body proportions of an adult, and the body proportions of a child was examined.

3.2 Main simulation setup

The aim of these simulations was to investigate how the outcome was affected by specific parameter choices of the system. Parameters of the LGP system were investigated, as well as a parameter specific to the actual control problem. LGP specific parameters was *population size* $P(t)$, expectation value of *initial genome length* of the population $\langle L_i \rangle$, *crossover probability* p_c, and *mutation rate* r_{mut}. The problem specific parameter examined was the *initial energy level* E_i of the motor battery. Initially, these parameter values were set to their *default values*, which were determined empirically. The following default parameter values were used; 128, 128, 0.8, 0.2 and 128, respectively.

Table 3: Parameter settings examined, the default values are typeset in *italic*.

Parameter	Values
$P(t)$	8, 32, *128*, 512
$\langle L_i \rangle$	32, 64, *128*, 256
p_c	0.0, 0.2, *0.8*, 1.0
r_{mut}	0.1, *0.2*, 0.4, 0.8
E_i	32, *128*, 256, 512

3.3 Additional simulation setup

In this section, a number of supplementary runs will be described. Those runs were conducted because the results from the main simulation was not entirely convincing, so possible ways of improving the setup need to be investigated.

3.3.1 Robot controlled by dual controllers

When starting to walk the robot's configuration is transformed from the initial standing posture, into a cyclic motion pattern. Once the cyclic phase is entered, a configuration similar to the initial static configuration is not likely to occur again. At least not until the walking stops. It is reasonable to assume that the *transformation phase*, from static posture to dynamic walking, should best be handled by one controller, or brain, and the cyclic walking phase by another brain. Therefore, an approach with dual controllers, executed in series, was investigated as well. Control of the robot is then switched from the first brain (B1) to the second brain (B2) after a fixed amount of execution time. Here, the switching occurs after 3 s.

In this approach the genomes is divided in two parts, B1 and B2. Apart from that, everything else works in the same way as before, regarding the LGP system. However, the crossover operator is restricted to only exchange genetic material between the B1 part of the first individual, with genetic material of the B1 part of the second individual. Correspondingly, only B2 material can be exchanged with B2 material, between individuals.

3.3.2 Counting the number of foot steps

In order to prevent the robots from taking many short, small steps during walking, individuals were punished for the number of steps they used when

walking. A function was introduced that counted the number of walking steps, and then individuals were punished accordingly by reduced fitness rewards. This was done by simply multiplying the fitness score according to Eqation 2, with the following factor:

$$F_s = \frac{1}{\sqrt{s_{l,r}}} \tag{6}$$

where $s_{l,r}$ is the number of foot steps during walking, with the left and right foot respectively. Thus, the total fitness contribution is reduced according to a factor inversely proportional to the square root of the number of footsteps.

3.3.3 Obstacle in front of the robot

In an attempt to force individuals to make the robot to take larger and higher foot steps, an obstacle was placed directly in front of the robot at the evaluation. The height of the obstacle was varied, in different runs, between one fifth, and one half of the height of the robot's feet.

The simulation was based on the following assumption: if the robot's first step is high enough to climb the obstacle in front of it, it is more likely that the robot will continue to take that high foot steps. Further, that one individual will also get higher fitness than other individuals, which cannot cope with the obstacle, and thus give rise to more offspring. Hence, individuals taking high foot steps will be promoted by evolution.

3.3.4 A lifting force applied to the robot

It is very a difficult task to learn to walk, just starting from a standing pose. In an attempt to make it easier for individuals to start walking, an approach was tried were an *upwards* directed force was attached to the robots body during evaluation. This setup resemble to some extent the situation were an infant is learning to walk, supported by his parent or another adult person: when the child starts to lose balance, the supervisor (adult person) can quickly give him support and lift him up again, so that he can continue with the training again. In this simulation, the *spring force* was applied to the robot's body, which prevented the robot from completely falling to the ground. The spring force obeyed Hooke's law, $F = C \cdot \Delta l$, where C is the so called spring constant, and Δl is the vertical displacement from the starting position. Note that there are *no* horizontal components of the spring force, just the vertical one.

4 Results

4.1 Groundwork simulation results

Apart from the testing, verification, and fine-tuning of the LGP system with physics simulation, the major contribution from these simulations is that adding the *energy discharging function* improved the results by a great portion. More anthropomorphic gaits evolved, and both the oscillating behaviors and the crab-like walking behavior ceased to emerge from the evolution.

The case when comparing evolution with different robot morphologies (Section 3.1.4) did not yield any specific result. Similar gaits emerged with both morphologies, approximately equally often.

4.2 Main simulation results

This section presents the outcome of 60 independent simulation runs. Due to limitations of time and computational power, not all of the 1024 possible combinations of alternative parameter settings were examined. Instead the parameters were varied, one at a time, from their default value settings. Thus, we obtain 20 unique parameter combinations. The actual settings for each run are clear from table 4, as well as the resulting fitness values. For statistical significance all runs were repeated three times with identical parameter settings.

The best over-all fitness values were obtained for parameter settings equal to 128, 128, 0.8, 0.2, and 256. As clear from table 4 default parameter values of $P(t)$ and r_{mut} produced the best fitness values, of respective partition. A (default) value of 0.8 for the parameter p_c also gave the best result of its partition, at least when looking at the $\bar{f}_{\max}$ and $\bar{f}_{avg}$ fitness values. On the contrary, when considering only the best fitness value of a single individual of that partition, a p_c value of 0.0 (i.e. only mutation was used) outperformed the others, however with the worst value of $s(\bar{f}_{\max})$. The best fitness values for $\langle L_i \rangle$ and E_i were obtained for values of 32 and 256, respectively.

Table 4: Fitness values of the best individuals, $\max(\bar{f}_{\max})$; averages of the best individual fitness values, $\bar{f}_{\max}$, with standard deviations $s(\bar{f}_{\max})$; averages of the population fitness values, $\bar{f}_{avg}$. All values were found after 25000 tournaments. Each one of the five partitions of the table shows the variations of a single parameter value. Numbers in bold represent the best fitness values of each partition.

$\max(\bar{f}_{\max})$	$\bar{f}_{\max}$	$s(\bar{f}_{\max})$	$\bar{f}_{avg}$	$P(t)$	$\langle L_i \rangle$	p_c	r_{mut}	E_i
5728	4635	1777	3475	8	128	0.8	0.2	128
5839	5015	1144	4701	32	128	0.8	0.2	128
7303	**6760**	480	**6535**	128	128	0.8	0.2	128
6954	6604	486	5032	512	128	0.8	0.2	128
7449	**6900**	586	**6546**	128	32	0.8	0.2	128
5817	5065	848	4864	128	64	0.8	0.2	128
7303	6760	480	6535	128	128	0.8	0.2	128
6176	5232	1194	4007	128	256	0.8	0.2	128
7943	6054	1662	4621	128	128	0.0	0.2	128
6580	5468	973	5303	128	128	0.2	0.2	128
7303	**6760**	480	**6535**	128	128	0.8	0.2	128
7197	6687	578	6419	128	128	1.0	0.2	128
6417	5170	1109	4679	128	128	0.8	0.1	128
7303	**6760**	480	**6535**	128	128	0.8	0.2	128
6797	6292	580	5933	128	128	0.8	0.4	128
6308	6194	148	4517	128	128	0.8	0.8	128
2049	1270	581	1101	128	128	0.8	0.2	32
7303	6760	480	6535	128	128	0.8	0.2	128
8958	**7598**	1375	**7368**	128	128	0.8	0.2	256
7076	6352	660	6020	128	128	0.8	0.2	512

Summarizing the results of table 4; best fitness values of each partition of the table, according to $\bar{f}_{avg}$, were obtained for the following five parameter values: 128, 32, 0.8, 0.2, and 256. The best individual found had a fitness value of 8958. During its evaluation time, which corresponds to 36 s of real time simulation, it covered a distance of 1.93 m.

A second series of simulation runs were performed with the parameter value settings equal to 128, 128, 0.8, 0.2, and 256. Then, the best individual obtained had a fitness value $\max(\bar{f}_{\max})$ of 9044, and the averages of the best individual fitness values $\bar{f}_{\max}$ were equal to 7483. The deviation of fitness values compared with the best runs of the first series was within the margin for error.

The graphs of figure 6 show covered distances, $d(t)$, for the best individual found and two other individuals, for comparison. As clear from the figure, the best individual (i.e. the one labeled "8958") traveled the longest distance during the evaluation time, and thus received highest fitness score. At the end of the evaluation period, however, it fell backwards. That occurence is indicated by the graph $d(t)$ dropping off at the end. The other two individuals both remained on their feet for the whole evaluation period of 4000 timesteps, but they received lower fitness values. Successful individuals in these simulations all exhibit cyclic locomotion behavior, exemplified by the graphs in figure 7.

4.3 Additional simulation results

Only a few preliminary test runs have been performed with these simulation setups. The setup with a lifting force applied to the robot (Section 3.3.4) did not yield any improvement. No walking behavior at all emerged.

The case with an object placed in front of the robot (Section 3.3.3), no individual so far have

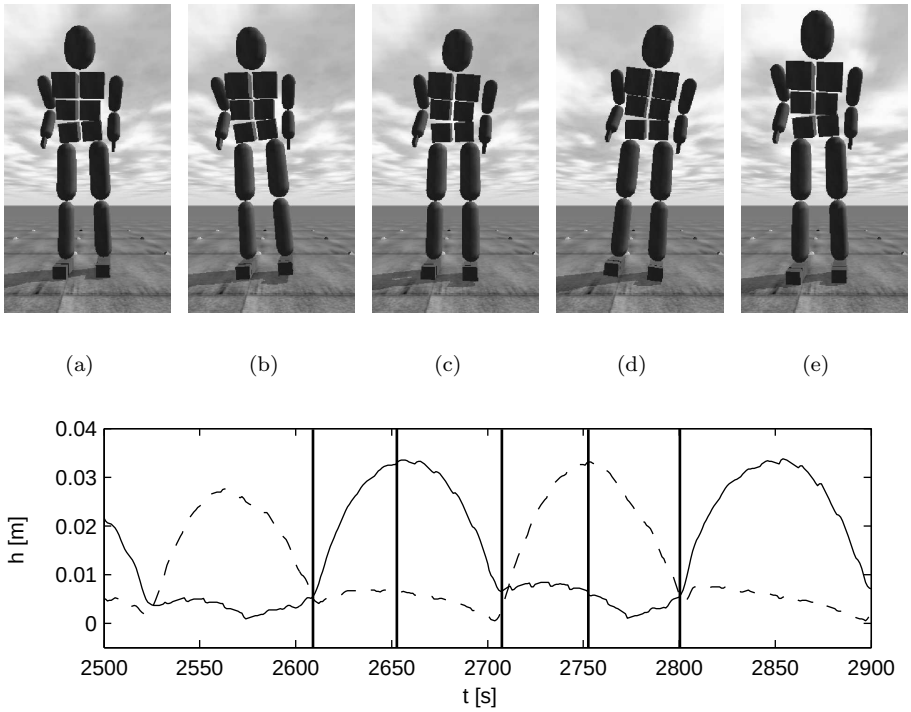

Figure 5: *Top panel*: Walking scene of the bipedal robot. Starting from the left, double support phase is depicted (a), followed by the single support phase with left foot in swing motion (b). Then again a double support phase (c), followed by single support phase with the right foot in swing motion this time (d). Finally, the gait cycle is completed with a double support phase (e). *Bottom panel*: Plot of the depicted gait cycle. The graphs show the height above ground, $h(t)$, of the CoM point of the left foot (solid line) and the right foot (dashed line), respectively. Horizontal positions of the vertical lines indicate corresponding double- and single support phases (a) throgh (e).

managed to climb the obstacle. Thus, no walking behavior emerged.

When counting the number of foot steps of the robot (Section 3.3.2), the best individual found so far managed to walk a few steps, and then it stood idle until it got terminated, accumulating some fitness meanwhile. Most runs in the main simulation did better off.

However, in the simulation with two brains serially controlling the robot, individuals emerged that has almost as high fitness values as the best individuals from the main simulation runs. But, considering the way they walk, it looks even better than the best individuals from previous runs. That is, the gaits look more anthropomorphic.

5 Discussion and Conclusions

The primary goal of the simulations reported in this paper was to evolve robust, anthropomorphic bipedal locomotion. Evolution generated controller programs that made the simulated robot stride across its environment by means of bipedal locomotion. When executed afterwards, a couple of the best individuals were actually capable of producing locomotion behavior several times longer than the evaluation time of 36 s. For instance, the individual labeled '5592' in this paper was executed for a time period of more than 20 minutes, and the robot was still walking forward. However, no one individual exceeded a walking speed of 5.37 cm/s. Clearly, this is a very moderate walking speed, considering the

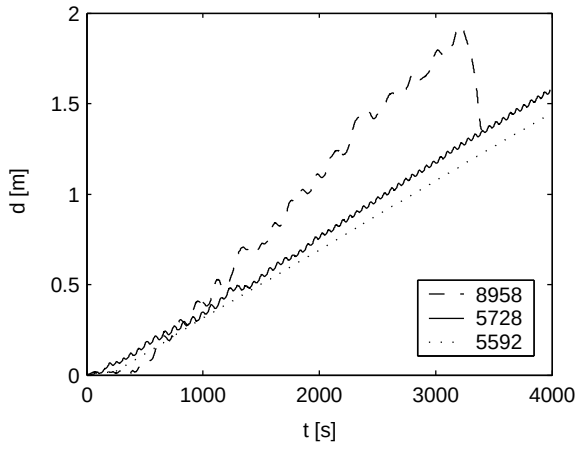

Figure 6: Graphs of covered distances, $d(t)$, during 4000 evaluation timesteps, for three different individuals. These individuals received fitness scores of 8958, 5728 and 5591, respectively. The individual with fitness value 8958 was actually the best individual found in all the runs.

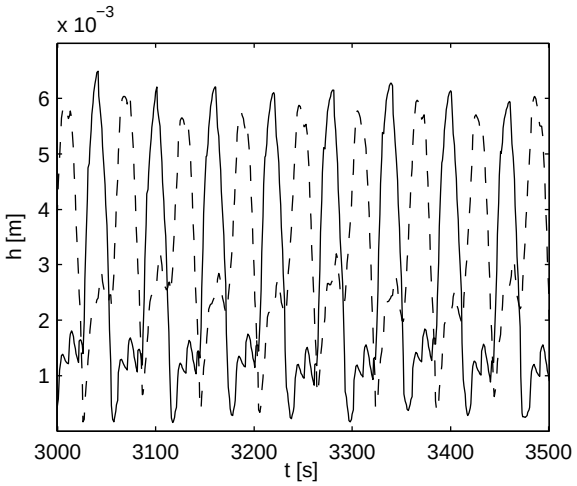

Figure 7: Time series of resulting body dynamics, of the individual of fitness 5728. The graphs show the height above ground, $h(t)$, of each foot, during 500 timesteps (equal to 4.5 s) of simulation. The fact that both curves never indicate an altitude of zero (which is supposed to happen when a foot touches the ground) is a consequence of the fact that both feet were constantly tilted some unspecified angle away from the horizontal plane, which the graphs were not compensated for. However, since this individual performs a cyclic walking pattern each local maximum indicates a single footstep. The figure contains a sequence of 9 steps of the right foot (dashed line), and 8 steps of the left foot (solid line).

fact that the dimensions of the bipedal robot corresponds to the size of a (small) human being. Even the best individuals lifted their feet only a small distance above the ground when walking, and also used a very short step length. Given these characteristics, low walking speed will of course be the result. These individuals simply did not activate the knee joints of the biped very much. When walking, these individuals activated the hip joints primarily. This kind of locomotion behavior is somewhat dissimilar to human gait. However, apart from these defects, good individuals generally made the robot walk in an upright, forward directed manner. Further, some of the successful individuals clearly utilized the inherent dynamics of the physical system when walking.

Evolution is a theory of gradual, hereditary change. When observing the evolutionary process of any one of the successful simulation runs described here, that statements is clearly supported. For example, the individual previously referred to as '5728' in this paper shows an atypical way of walking with its left foot slanted "toe up", i.e. only the heel is in contact with the ground. Individual '5728' emerged as the best one of the run in tournament number 18839, but the above mentioned peculiarity was observed in evolution as early as in tournament 180, for an individual with fitness equal to 444. It could very well be possible that different "tracks" of anomalies, e.g. like the one described above, could co-exist during evolution, but that was not investigated here.

Figure 8 show fitness progress during a representative evolutionary run. Typically, there are several distinct increments in the fitness curve of the best individual. Some of these fitness leaps correspond to what could be thought of as transitions in evolution. In the initial randomly generated population, almost all individuals fall over after a few seconds of evaluation, thus they receive poor fitness scores. Rather soon however, individuals learn to stand up without falling over. At this stage, they

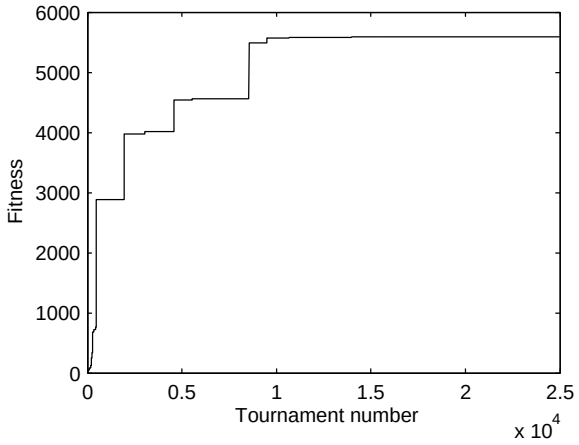

Figure 8: Fitness progress during an evolutionary run, exemplified by the run which generated individual of fitness '5728'.

receive slightly higher fitness, but they get terminated according to the fourth termination criterion (Eq. 5). The next transition occurs when the best individuals get more movable, mainly by activating the robot's hip joints. Gradually, the magnitude of these motions increase, which results in a few small forward steps. Hence, fitness scores reach the next level, despite most of the individuals fall over after a while. The evolutionary stage following from that include individuals that can continue to walk forward without falling at all, but they all get terminated, again because of the fourth criterion. The last great transition arises when a majority of individuals begin to walk just slightly faster, but receive much better fitness scores. They manage to adapt their walking pace to just above the threshold defined by the fourth criterion (Eq. 5), and as a consequence they walk almost the full evaluation time, terminated only because of the energy discharging criterion. After that, evolution favors those individuals that can cut down the energy cost. During that phase only small fitness increments were observed.

6 Future work

In order to fully accomplish the goals of this project, there are several alternative extensions or modifications left to investigate.

In this study a population of individuals, or programs, is evaluated for the task of bipedal locomotion. When evaluating individuals in the simulation, all individuals have to start the robot from the same upright standing position (similar to the 'attention' stand). The activity of bipedal walking is mainly cyclic to its nature. It might therefore seem unnatural to commit charge of control to one single closed-loop program in such a position, when that posture is not present as any of the postures in the walking cycle. Accordingly, we have carried out preliminary simulations using two individuals controlling the robot. One program is controlling the robot for the first 3 s of evaluation time, and then control is taken over by a second program. Thus, we have one *start-up program*, and one *cyclic program* in the same individual. The results so far, which are based on a few simulation runs only, look promising. However, the evaluation time greatly increases since there are two genomes, instead of one, in each individual which should evolve into successful individuals. Currently, the point at which control is taken over by the second individual is specified at before hand. A natural extension would thus be to let this moment be decided upon during execution of the individual instead. A suitable *behavior selection* method to do this would be the utility functions (UF) method for instance [32].

Even the best individuals generated in the simulations described in this paper are impaired by serious shortcomings, as mentioned in the previous sections. First of all, the foot steps are too short, and second, the knee joints are used very little. Before any really anthropomorphic gaits can be expected to emerge, such weaknesses must be eliminated.

Evolving robot controllers for such a complex task as bipedal locomotion requires realistic simulation of the bipedal robot model. Such simulations are computationally very costly. During an evolutionary run, most of the computation time (maybe as much as 99%) is spent on evaluating individuals by means of full rigid-body simulations. Only a small fraction of the time is allotted to EA related tasks (mutations, crossover etc.). An approach to speed up the evolutionary search process has been proposed by Ziegler et al. [50]. In their method a second GP system is involved, which evolves meta-models of the first GP system's fitness function. Hence, time consuming evaluations can

be replaced by faster classifications of individuals, or fitness value estimations. In their simulations (evolving gaits for a 12 DOF, four-legged robot) they could save as much as 50% of the evaluations with this method, compared to "standard" fitness evaluations.

References

[1] M. Abdallah and A. Goswami. A biomechanically motivated two–phase strategy for biped upright balance control. In *Proceedings of the International Conference on Robotics and Automation, ICRA'05*. IEEE, 2004.

[2] T. Arakawa and T. Fukuda. Natural motion generation of biped locomotion robot using hierarchical trajectory generation method consisting of GA, EP layers. In *Proceedings of the International Conference on Robotics and Automation, ICRA'97*, pages 211–216. IEEE, 1997.

[3] W. Banzhaf, P. Nordin, R. Keller, , and F. Francone. *Genetic Programming – An Introduction: On the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1998.

[4] A. Boeing, S. Hanham, and T. Bräunl. Evolving autonomous biped control from simulation to reality. In *Proceedings of the Second International Conference on Autonomous Robots and Agents, ICARA'04*, Palmerston North, New Zealand, 2004. IEEE, Massey University.

[5] B. A. Bogin. *Patterns of Human Growth*. Cambridge University Press, 2 edition, 1999.

[6] M. Brameier. *On Linear Genetic Programming*. PhD thesis, Dortmund University, 2003.

[7] G. Capi, Y. Nasu, L. Barolli, K. Mitobe, and K. Takeda. Minimum energy gait synthesis for walking biped robots based on genetic algorithms. In *Proceedings of the International Conference on Production Engineering, Design and Control PEDAC'01*, volume 2, pages 903–912, 2001.

[8] M. Y. Cheng and C. S. Lin. Genetic algorithm for control design of biped locomotion. *Robotic Systems*, 14(5):365–373, 1997.

[9] S. H. Collins, A. L. Ruina, R. Tedrake, and M. Wisse. Efficient bipedal robots based on passive-dynamic walkers. *Science*, 307:1082–1085, 2005.

[10] C. Darwin. *On the Origin of Species by means of Natural Selection or the Preservation of Favored Races in the Struggle for Life*. Murray, London, 1859.

[11] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, New York, NJ, 1966.

[12] Y. Fujimoto and A. Kawamura. Simulation of an autonomous biped walking robot including environmental force interaction. *IEEE Robotics & Automation Magazine*, pages 33–42, jun 1998.

[13] Y. Fujimoto, S. Obata, and A. Kawamura. Robust biped walking with active interaction control between foot and ground. In *Proceedings of the International Conference on Robotics and Automation, ICRA'98*, volume 3, pages 2030–2035. IEEE, 1998.

[14] J. Furusho and M. Masubuchi. A theoretically motivated reduced order model for the control of dynamic biped locomotion. *Journal of Dynamic Systems, Measurement and Control*, 109:155–163, 1987.

[15] J. Furusho and A. Sano. Sensor-based control of a nine-link biped. *International Journal of Robotics Research*, 9(2), 1990.

[16] A. Goswami. Postural stability of biped robots and the foot-rotation indicator (FRI) point. *The International Journal of Robotics Research*, 18:523–533, 1999.

[17] A. Goswami and V. Kallem. Rate of change of angular momentum and balance maintenance of biped robots. In *Proceedings of the International Conference on Robotics and Automation, ICRA'04*, volume 4, pages 3785–3790. IEEE, 2004.

[18] K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka. The development of Honda humanoid robot. In *Proceedings of the International Conference on Robotics and Automation, ICRA'98*, volume 2, pages 1321–1326. IEEE, 1998.

[19] J. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, MI, USA, 1975.

[20] Q. Huang and Y. Nakamura. Sensory reflex control for humanoid walking. *IEEE Transactions on Robotics and Automation*, 21(5):977–984, 2005.

[21] Q. Huang, Y. Nakamura, and T. Inamura. Humanoids walk with feedforward dynamic pattern and feedback sensory reflection. In *Proceedings of the International Conference on Robotics and Automation, ICRA'01*, pages 4220–4225. IEEE, 2001.

[22] L. Huelsbergen. Toward simulated evolution of machine–language iteration. In *Proceedings of the '96 Conference on Genetic Programming*, pages 315–320, Stanford University, CA, USA, 1996.

[23] S. Kajita and K. Tani. Adaptive gait control of a biped robot based on realtime sensing of the ground profile. In *Proceedings of the International Conference on Robotics and Automation. ICRA'96*, pages 570–577. IEEE, 1996.

[24] D. Katic and M. Vukobratovic. Survey of intelligent control techniques for humanoid robots. *Intelligent and Robotic Systems*, 37(2):117–141, 2003.

[25] J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* The MIT Press, Cambridge, MA, USA, 1992.

[26] Q. Li, A. Takanishi, and I. Kato. Learning control of compensative trunk motion for biped walking robot based on zmp stability criterion. In *Proceedings of the '92 International Conference on Intelligent Systems*, pages 597–603. IEEE, 1992.

[27] S. Mojon. Using nonlinear oscillators to control the locomotion of a simulated biped robot. Master's thesis, Computer and Communication Sciences, BIRG, EPFL, Lausanne, Switzerland, 2004.

[28] P. Nordin. *Evolutionary Program Induction of Binary Machine Code and its Applications.* PhD thesis, Dortmund University, Dortmund, Germany, 1997.

[29] S. Ok, K. Miyashita, and K. Hase. Evolving bipedal locomotion with genetic programming —a preliminary report—. In *Proceedings of the Congress on Evolutionary Computation, CEC'01*, pages 1025–1032, Seoul, South Korea, 2001. IEEE.

[30] J.H. Park and H.C Cho. An on-line trajectory modifier for the base link of biped robots to enhance locomotion stability. In *Proceedings of the International Conference on Robotics and Automation, ICRA'00*, pages 3353–3358. IEEE, 2000.

[31] J. Pettersson, H. Sandholt, and M. Wahde. A flexible evolutionary method for the generation and implementation of behaviors for humanoid robots. In *Proceedings of the 2nd International Conference on Humanoid Robots, Humanoids'01*, pages 279–286, Waseda University, Tokyo, Japan, 22-24 November 2001. IEEE-RAS, Humanoid Robotics Institute.

[32] J. Pettersson and M. Wahde. Application of the utility function method for behavioral organization in a locomotion task. *IEEE Transactions on Evolutionary Computation*, 9(5):506–521, oct 2005.

[33] I. Rechenberg. Cybernetic solution path of an experimental problem. *Royal Aircraft Establishment Translation*, (1122), August 1965.

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

[35] T. Reil and C. Massey. Biologically inspired control of physically simulated bipeds. *Theory in Biosciences*, 120(3–4):327–339, 2001.

[36] C. Ridderström. *Legged locomotion: Balance, control and tools - from equation to action.* PhD thesis, The Royal Institute of Technology, Stockholm, Sweden, 2003.

[37] A. Takanishi, M. Ishida, Y. Yamazaki, and I. Kato. The realization of dynamic walking by the biped walking robot WL-10RD. In *Proceedings of the International Conference on Advanced Robotics, ICAR'85*, pages 459–466, 1985.

[38] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proc London Math Soc*, 42:230–265, 1936.

[39] M. Vukobratovic. How to control artificial anthropomorphic systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 3(5):497–507, 1973.

[40] M. Vukobratovic and B. Borovac. Zero-moment point – thirty five years of its life. *International Journal of Humanoid Robots*, 1(1):157–173, 2004.

[41] M. Vukobratovic, B. Borovac, and D. Surdilovic. Zero-moment point - proper interpretation and new applications. In *Proceedings of the 2nd International Conference on Humanoid Robots, Humanoids'01*, pages 279–286, Waseda University, Tokyo, Japan, 22-24 November 2001. IEEE-RAS, Humanoid Robotics Institute.

[42] M. Vukobratovic, A. A. Frank, and D. Juricic. On the stability of biped locomotion. *IEEE Transactions on Bio-Medical Engineering*, 17(1):25–36, 1970.

[43] M. Vukobratovic and D. Juricic. Contribution to the synthesis of biped gait. *IEEE Transactions on Bio-Medical Engineering*, 16(1), 1969.

[44] M. Vukobratovic and J. Stepanenko. On the stability of anthropomorphic systems. *Mathematical Biosciences*, 15:1–37, 1972.

[45] M. Wahde and J. Pettersson. A brief review of bipedal robotics research. In J. van Amerongen, J. Jonker, P. Regtien, and S. Stramigioli, editors, *Proceedings of the 8th Mechatronics Forum International Conference, Mechatronics'02*, pages 480–488, Enschede, the Netherlands, 24-26 June 2002. IEE, Drebbel Institute for Mechatronics, Twente University.

[46] K. Wolff. *Evolutionary Humanoids for Embodied Artificial Intelligence.* Licentiate thesis, Chalmers University of Technology, Göteborg, Sweden, December 2003.

[47] K. Wolff and P. Nordin. Learning biped locomotion from first principles on a simulated humanoid robot using linear genetic programming. In E. Cantú-Paz, editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'03)*, volume 2723 of *LNCS*, pages 495–506, Chicago, 12-16 July 2003. AAAI, Springer Verlag.

[48] J. Yamaguchi, E. Soga, S. Inoue, and A. Takanishi. Development of a bipedal humanoid robot-control method of whole body cooperative dynamic biped walking. In *Proceedings of the International Conference on Robotics and Automation, ICRA'99*, pages 368–374. IEEE, 1999.

[49] Y.F. Zheng and J. Shen. Gait synthesis for the SD–2 biped robot to climb sloping surface. *IEEE Transactions on Robotics and Automation*, 6(1):86–96, 1990.

[50] J. Ziegler and W. Banzhaf. You can judge a book by its cover – evolution of gait controllers based on program code analysis. In *Proceedings of the 6th International Conference on Climbing and Walking Robots, CLAWAR'03*, pages 205–212. Professional Engineering Publishing, 2003.

[51] J. Ziegler, J. Barnholt, J. Busch, and W. Banzhaf. Automatic evolution of control programs for a small humanoid walking robot. In *Proceedings of the 5th International Conference on Climbing and Walking Robots, CLAWAR'02*. Professional Engineering Publishing, 2002.