An Evolutionary Based Approach for Control Programming of Humanoids

Krister Wolff and Peter Nordin

Applied Quantum Physics and Complex Systems Laboratory,
Dept. of Microtechnology and Nanoscience, MC2,
Chalmers University of Technology,
S-412 96 Göteborg, Sweden
{wolff, nordin}@fy.chalmers.se
http://www.frt.fy.chalmers.se/cs/index.html

Abstract. We describe the first instance of a novel approach for control programming of humanoid robots, based on evolution. To overcome some of the difficulties with evolution on real hardware, we use a physically realistic simulation of the robot. The essential idea is to evolve control programs from first principles on a simulated robot, transfer the programs to the real robot, and continue to evolve on the robot. As the key motivation for using simulators, we describe an on-line learning experiment with a humanoid robot. The Genetic Programming system is implemented as a Virtual Register Machine, with a linear genome, and steady state tournament selection. Evolution created controller programs that made the simulated robot produce forward locomotion behavior. A further application of this system, with two phases of evolution, would be to have a flexible adaptation mechanism that can react to hardware failures in the robot.

1 Introduction

The applications of robots with human-like dimensions and motion capabilities are plentiful. In a world where man is the standard for almost all interactions, humanoid robots have a very large potential acting in environments created for humans. Both in industry and in academia walking humanoid robots attracts an accelerating interest [17]. In 1996, Honda Corporation presented the P2 humanoid robot, which is a biped robot that can walk like a human, even up and down stairs. A smaller and lighter robot, P3, was introduced in 1997 and in the year of 2000, they presented the humanoid robot ASIMO, which is conceived to function in an actual human living environment in the near future. The Sony Corporation has developed a small biped walking robot, SDR-4X, which is a platform for exploration of new possibilities for entertainment robots. Recently, in 2002, another Japanese company, Kawada Industries, Inc. introduced a humanoid robot named 'isamu' directed for use as a test bench for their product development.

Dealing with humanoid robots requires supply of expertise in many different areas, such as vision systems, sensor fusion, planning and navigation, mechanical and electrical hardware design, and software design only to mention a few. The objective of this paper, however, is focused on the synthesizing of biped robot gait.

Honda's ASIMO, as well as SONY Dream Robot, is walking under the Zero Moment Point walking algorithm, developed by Atsuo Takanishi at Waseda University [16]. The dynamic stability of a walking machine can be defined by the Zero Moment Point position [22] and [23]. This criterion states that the ZMP position of a biped robot is maintained within the footprint of the supporting leg. This approach, however, is based on derivation of an internal geometric model of the locomotion mechanism, and requires intensive calculations by the controlling computer, to be performed in real time.

Robots, designed in such a way that a model can be derived and used for controlling, shows large affinity with complex, highly specialized industrial robots, and thus they are as expensive as conventional industrial robots. Our belief is that for humanoids to become an everyday product in our homes and offices, affordable for everyone, there is need to develop low cost, relatively simple robots. Such robots can hardly be controlled the traditional way; hence this is not our primary design principle [5] and [13].

A basic condition for humanoids to successfully operate in human living environment is that they must be able to deal with unpredictable situations and gather knowledge and information, and adapt to their actual circumstances. For this reason, among others, we propose an alternative way for control programming of humanoid robots. Our approach is based on evolution as the main adaptation mechanism, utilizing computing techniques from the field of Evolutionary Algorithms.

The first attempt in using a real, physical robot to evolve gait patterns was made at the University of Southern California. Neural networks were evolved as controllers to produce a tripod gait for a hexapod robot with two degrees of freedom for each leg [Lewis et al, 1992]. Researchers at Sony Corporation have worked with evolving locomotion controllers for dynamic gait of their quadruped robot dog AIBO. These results show that evolutionary algorithms can be used on complex, physical robots to evolve non-trivial behaviors on these robots [9] and [10]. Gary Parker use Cyclic Genetic Algorithms to evolve gait actuation lists for a simulated, six legged StiquitoII robot [18].

However, dealing with biped locomotion leads us into a partly different problem domain. Evolution of static walking with a biped robot is a lot more difficult than it is with a robot that got a greater number of legs. A static gait requires that the projection of the center of mass of the robot on the ground lie within the support polygon formed by feet on the ground. This is obviously easier to fulfill with a robot that got four, six, eight or more legs. When a biped robot is walking (static), it is supported only by one foot at the ground during an appreciable period of time. Only this single foot then constitutes its support area. For a biped robot, the area of support is relatively small, compared to the altitude of its center of mass. The corresponding measure for a robot with four or more legs is more favorable. Therefore it is easier for a robot with many legs to maintain balance than it is for a biped robot, as the motion of walking dynamically changes the stability of the robot.

Evolving efficient gaits with real physical hardware is a challenge, and evolving biped gait from first principles is an even more challenging task. It is extremely stressing for the hardware and it is very time consuming. To overcome the difficulties with evolving on real hardware, we introduce a new method, based on simulation of the actual humanoid robot.

Nick Jakobi has developed a methodology for evolution of robot controllers in simulator, called 'minimal simulations', and shown it to be successful when transferred to a real, physical octopod robot [11]. This method, however, has not been validated on a biped robot. Recently, a research group in Germany reported an experiment relevant to our ideas, where they evolved robot controllers in a physics simulator, and successfully executed them onboard a real biped robot. They were not able to fully realize biped locomotion behavior, but their results were definitely promising [26].

The rest of this paper is organized as follows; next to this section we describe an on-line learning experiment a humanoid robot, serving as a motivation for the work presented in the last section. That section is describing initial experiments in evolving biped walking from first principles on a simulated humanoid robot.

2 Background and Motivation

In this section we briefly describe an on-line learning experiment performed with our biped humanoid robot 'elvina'. However this experiment was fairly successful in evolving locomotion controller parameters that optimized the robot's gait, it pointed out some difficulties with on-line learning. We summarize the experiment here in order to exemplify the difficulties of evolving gaits on-line, and let it serve as an illustrative motivation for the work presented in the remainder of this paper.

2.1 Real Robot Experiment

Robot Platform The robot used in our experiments was 'elvina', which is a simplified, scaled model of a full-size humanoid with body dimensions that mirrors the dimensions of a human. It was originally developed as an alternative, low-cost humanoid robot platform, intended for research [27]. The 'elvina' humanoid is a fully autonomous robot with onboard power supply and computer, however many experiments are performed with external power supply. It is 28 cm tall and it weights about 1.49 kg including batteries. Each of the two legs has 5 degrees of freedom, of which 4 dof is active and 1 dof is passive. The head, the torso and the arms has 1 dof each, giving a total of 14 dof. The robot is equipped with a digital CMOS color camera, mounted in its head. The body also houses a near-infrared PSD (position sensitive detector) that is used to determine distances to nearby objects.

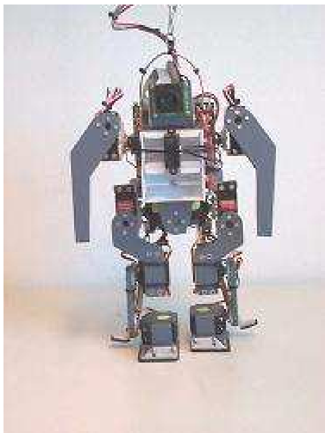

Fig. 1. Picture of the humanoid robot 'elvina'.

The robot has the 32-bit micro-controller EyeBot MK3 [3] onboard, carrying it as a backpack. The robot control programs are developed on a host computer, and after cross-compilation the binary file is downloaded to the EyeBot controller. All signal processing, including control, vision, and evolutionary algorithm, is carried out on the EyeBot controller itself. The body structure of the robot is constructed with the actuators as the main elements. The actuators that constitute the different sections of the body are connected to each other with parts made of PVC-plastic board so that they together form the robot body. This PVC material fulfills all necessary requirements since it is inexpensive and lightweight, yet strong and durable. The robot is assembled with standard off-the-shelf R/C servomotors as actuators. This kind of servo has an integrated closed loop position control circuit, which detects the pulse-code modulated signal that emanates from the controller board for commanding the servo to a given position. In this implementation, each servo is assigned a target position by the robot control program by addressing it an integer value within the interval $\{0, 255\}$. In its present status, the robot is capable of static walking.

Gait Generation Method The gait control method for this robot involves repetition of a sequence of integrated steps. Considering fully realistic bipedal walk, two different situations arise in sequence: the statically stable double-support phase in which the robot is supported on both feet simultaneously, and statically unstable single-support phase when only one foot of the robot is in contact with the ground, the other foot being transferred from the back to front position. When this sequence of transitions has been repeated twice, one can consider a single gait cycle to be completed. That is, the locomotion mechanism's posture and limb's positions are the same after the completion as it was before it started to move, and hence it's internal state is the same.

If we now study only static walk, i.e. the projection of the center of mass of the robot on the ground always lie within the support polygon formed by feet on the ground, there is obviously a number of statically stable postures in between the internal state of the robot and it's final state, during completion of a single gait cycle. Consider e.g. the situation when one is standing still on one leg, only supported by one foot on the ground. By interpolation between numbers of such, statically stable, consecutive states it is possible to make the robot to complete a single gait cycle. Then, by continually looping, biped gait is produced. In a more formalistic way:

Let j_i denote the state variable within the interval $\{0,255\}$ of the i:th degree of freedom of the robot, having k degrees of freedom in total. Now, for some internal state P of the robot, there is a unique vector $\mathbf{J}_P = [j_1, j_2, ..., j_k]$ describing that state. The robot's workspace $\mathbf{W}$ is defined by all possible linear combinations of the vectors $\mathbf{J}_1, ..., \mathbf{J}_j$, where $j = 256^k$. Impose the restriction that this state P should correspond to the robot being statically stable, and suppose that there exists n such statically stable states. Those states correspond to the vectors $\mathbf{J}_{P_1}, ..., \mathbf{J}_{P_n}$, and all possible linear combinations of these vectors then define the subspace $\mathbf{G}$ of $\mathbf{W}$. Here after, we will refer to $\mathbf{G}$ as the statically stable state space.

For the robot to complete a single gait cycle as described above, there must be at least m such statically stable internal states $P_1, ..., P_m$, of the robot to interpolate between. With the described methodology, we can find the statically stable internal states $P_1, ..., P_m$ of the robot by experimentally explore the workspace $\mathbf{W}$ of the robot until it is capable of performing a single gait cycle without falling. Thus, we have found one subspace $\mathbf{G}^*$ of $\mathbf{G}$. With this methodology however, one is restricted to produce static biped gait only.

Now, the vectors $\mathbf{J}_1, ..., \mathbf{J}_m$, defining $\mathbf{G}^*$, are stored as a list of integers, within the interval $\{0, 255\}$, in an array. By interpolating between consecutive vectors $\mathbf{J}_i$, and continually looping the array, the robot is able to produce stable, biped gait.

In order to control the movements of a limb, the partial movements of all involved joints must be coordinated and synchronized to get the desired motion. For this reason, a servo locomotion module has been implemented in software. Consider two consecutive vectors $\mathbf{J}_i$ and $\mathbf{J}_{i+1}$, which corresponds to two consecutive, statically stable states of the robot. The difference between the vectors corresponds to the translation of the robot's limbs, when interpolating between those vectors. Some of the limbs will have to make a large move, and some will only move very little, or not move at all. By subtracting the values of corresponding entries of the two vectors from each other, and divide the differences into small increments, sent to each servo to update its state, the robot's limbs is caused to simultaneously move from one position to another.

Evolutionary Gait Optimization Experiment In this sub-section we briefly describe an evolutionary experiment performed with our biped humanoid robot. The experiment is performed in order to optimize a by hand developed set of state vectors $\mathbf{J}_1, ..., \mathbf{J}_m$, defining a static robot gait (a subspace $\mathbf{G}^*$). The evolutionary algorithm used was a steady state evolutionary strategy [1] running on the robot's onboard computer. Individuals were evaluated and fitness scores automatically determined using the robots onboard digital camera and proximity sensor.

A population stemming from a manually developed individual was created with a uniform distribution over a given search range. Then four individuals were randomly selected from this initial population. These individuals were evaluated and their fitness was measured. The two individuals with the better fitness values were considered as parents and the two individuals with the lower fitness were replaced by the offspring's of the parent individuals. The selection, evaluation and reproduction phases of the evolutionary strategy was then repeated until the maximum number of trials was reached.

For this evolutionary strategies experiment we used an initial population

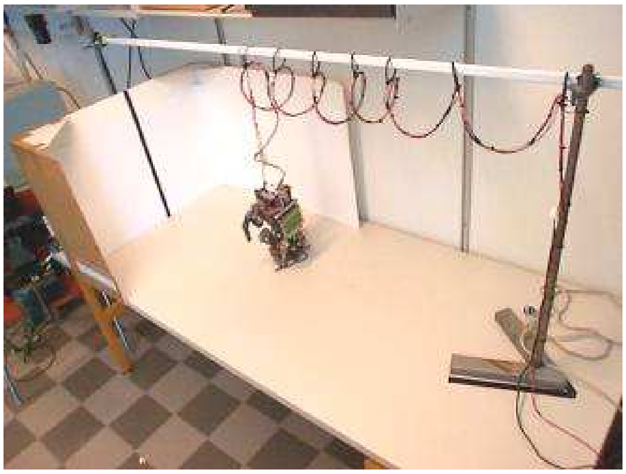

Fig. 2. The experimental environment.

of 30 individuals, and the method of tournament selection, with size four. The best-evolved individual and the manually developed individual were independently tested, and their performances were compared to each other's. The former one received a fitness score, averaged over three trials, of 0.1707, and the latter one, tested under equal conditions, got a fitness of 0.1051. Within this context, a higher fitness value means a better individual, and thus the best-evolved individual outperformed the manually developed individual both in its ability to maintain the robot in a straight course and in robustness, i.e. with a lesser tendency to fall over. By using this system, we evolved gait patterns that locomote the robot in a straighter path and in a more robust way, than the previously manually developed gait did.

For a more detailed description of the 'elvina' humanoid robot platform, and the evolutionary experiment performed with it, we refer to [24] and [25].

Observations To run such an evolutionary experiment as described above span over several days, and requires manual supervision all this time. Between each generation of four individuals evaluated, we paused the experiment for about 15 minutes in order to spare the hardware and especially the actuators. The main reason for this is that the actuators accumulate heat when they are running continuously under heavy stress. They then run the risk of getting overheated and gradually destroyed. We also observed that the position control circuit of the servomotors was sensitive to temperature. When commanding a servomotor to a given position by addressing it a fixed integer value within the interval {0, 255}, the physical angle of the servo's output shaft dislocates over time as the temperature of the servo increases. Since the robot's feet are coupled to each other via nine actuators their relative positions are then affected so much by this drift that it could cause the robot to fall. One way to handle this problem was, as mentioned above, to run the robot intermittent so that the servos maintain an approximately constant temperature.

Evolving efficient gaits with real physical hardware was a challenging task. In the six months of manually developing gaits and testing the evolutionary algorithm, frequent maintenance of the robot was indeed necessary. The torso and both the ankle actuators were exchanged once as well as the two hip servos. The most vulnerable parts of the robot were proved to be the knee servos. Both these servos were replaced tree times.

Obviously there are a number of difficulties related with evolving biped walking behavior on a real, physical robot. In an attempt to overcome some of the problems, we want to use a physically realistic simulation of the robot. The central idea in this concept is to evolve control programs from first principles on a simulated robot, transfer the resulting programs to the real robot and continue to evolve efficient gait on the real robot. Of course, there will arise other problems applying this method, as simulation systems always imply some simplifications of the real world.

3 Evolution Of Control Programs

3.1 Genetic Programming

In *The Origin of Species* [4], 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 and survival of the fittest. Theologians quickly labelled Charles Darwin "the most dangerous man in England", but his principle has become widely accepted among scientists and society in general today.

Over the past 35 years, the principle of natural selection and survival of the fittest has been successfully utilized on computers to optimize a solution towards

a pre-defined goal, and from this Evolutionary Algorithms have raised. Several research subfields, such as Evolutionary Programming [6], Genetic Algorithms [8] and Evolution Strategies [19] and [20] have emerged, and however they differ from each other in many aspects, they all mimic natural evolution.
Another flavor of Evolutionary Algorithms is Genetic Programming. While ES is associated with engineering optimization problems working on real-valued variables, and GA frequently operates on fixed length binary strings, GP deals with the evolution of computer programs. Genetic Programming, like other adaptive techniques as e.g. Evolutionary Algorithms, has applications in problem domains where theories are incomplete and insufficient for the human programmer, or when there isn't enough time or resources available to allow for human programming.

However GP is a relatively young research topic, methods for GP was suggested as far back as in the 1950s [7]. GP has been growing fast since the early 1990s, when John Koza published his book *Genetic Programming* [12], demonstrating the feasibility of GP with numerous applications. Koza has been instrumental in the development to establish Genetic Programming as a matured research field.

In Koza's formulation, program individuals are tree structures represented by LISP-S expressions, and by hierarchical subtree exchanging genetic crossover operator guarantee syntax preservation during evolution. Programs are stored as trees in a population and evaluated using a goodness criterion, or fitness function, to determine their quality. According to their individual quality, a selection of individuals is recombined using the genetic operators crossover and mutation. The resulting offspring are insert into the population, replacing some of the previous generation individuals. In our present work, however, we use a different variant of Genetic Programming, which is implemented as a Virtual Register Machine, and uses a linear representation of the programs.

Our primary goal is to utilize Genetic Programming for evolving locomotion control programs from first principles for our simulated biped robot, i.e. with no a priori knowledge for the robot on how to walk, information of morphology etc. The evolved programs take the robot's *current* internal state parameter values as input vector and return a vector predicting it's *next* internal state parameter values, in order to produce robust biped gait.

3.2 Physics Simulator Engine

The Open Dynamics Engine (ODE) is a free library for simulating articulated rigid body dynamics, developed by Russell Smith [21]. An articulated structure is created when rigid bodies of various shapes are connected together with joints of various kinds. ODE is designed for use in interactive or real-time simulation. It is particularly good for simulating moving objects in changeable virtual reality environments, because it is fast, robust and stable. Additionally, the user has complete freedom to change the structure of the system even while the simulation is running. ODE uses a highly stable, first order integrator, so that the simulation errors should not grow out of control. The physical meaning of this is that the simulated system should not "explode" for any reason. The simulation is based on a method where the equations of motion are derived from a Lagrange multiplier velocity based model. ODE has hard contacts, which means that a special non-penetration constraint is used whenever two bodies collide. Another, in simulators widely used method, is virtual springs to represent contacts. ODE has a built-in collision detection system with sphere, box, capped cylinder and plane as the collision primitives. Other features of ODE are arbitrary mass distribution of the rigid bodies, and a contact/friction model based on the Dantzig LCP solver described by [2]. The joint types implemented in ODE are ball-and-socket, hinge, hinge-2, fixed, prismatic slider and angular motor. The hinge-2 joint is the same as two hinges connected in series, with different hinge axes. The ODE library has a native C interface (even though ODE is mostly written in C++) and platform specific optimizations.

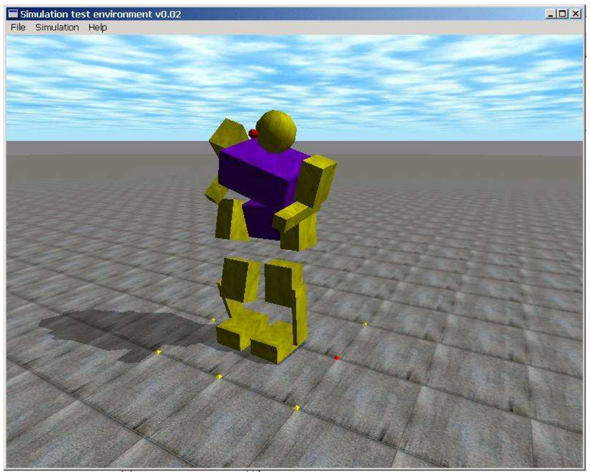

Fig. 3. Snapshot of the simulated humanoid robot. The body elements are directly connected to each other, although this is not visualized here.

Simulated Robot Model The robot model is qualitatively consistent with the real robot regarding the aspects of geometry, mass distribution, and morphology, see fig.3. It consists of 12 actuated joints and 13 body elements. It is constructed with its mass concentrated to the main body elements, which in the real robot correspond to the servo actuators, batteries and computer. The plastic body parts, interconnecting the servos to each other, are not rendered in the simulation, since their mass is very low compared to the total mass.

3.3 Symbolic Regression

The procedure of finding a symbolic equation, function, or program that fits given numerical data is called symbolic regression. A GP-system performing symbolic regression takes a number of numerical input/output relations, called fitness cases, and produces a function or program that is consistent with these fitness cases. Genetic Programming is ideal for symbolic regression, and most GP applications could be reformulated as an instance of symbolic regression [14].

Our problem can thus be formulated: approximate a function that takes the robot's current internal state parameter values as input, and returns a vector predicting it's next internal state parameter values in order to produce robust biped gait:

$$f(j_1, j_2, ..., j_k) = [j'_1, j'_2, ..., j'_k] \tag{1}$$

3.4 Virtual Register Machine

The Genetic Programming representation used for this problem of robot control program induction is an instance of a Virtual Register Machine, VRM(k, l). It has k I/O registers and l internal work registers. In the current implementation of our system, l equals k. The function set consists in the present of arithmetic functions ADD, SUB, MUL, DIV, where DIV is protected division, and SINE. We now define a register state vector $\mathbf{Reg} \equiv [Reg_1, ..., Reg_k]$ of k integers, each of the elements corresponding to one of the actuated joints of the simulated robot. All program input/output is communicated through the states of the I/O registers. That is, program inputs are supplied in the initial state **Reg**, and output is taken from the final register state Reg'. Further, the I/O register state vector is initially copied into the internal work registers. We can do this in a straight forward manner, since we have imposed that the number of I/O registers, k, equals the number of work registers, l. The Virtual Register Machine is allowed writing only to the internal work registers when looping the program instructions. The I/O registers are write-protected in this phase, and their final state is updated after the end of the program execution cycle, before they are passed to the robot and then updating it's internal state. With this set-up, registers may be used to initially supply the program with constants, or alternatively, the program may synthesize necessary constants in registers. Therefore, this GP-system doesn't require any random initialized constants to be used.

3.5 Linear Genome Representation

Each individual is composed of simple instructions between input and output parameters. Each instruction consists of four elements, encoded as integers, and the whole individual is a linear list of such instructions:

```
 8, 22, 3, 12,
19, 11, 2, 16,
15, 12, 3, 12,
 8,  3, 4, 19,
12, 12, 4, 21,
 1,  6, 5, 12,
20,  3, 1, 19,
 9, 12, 2, 21,
23,  5, 3, 19,
16,  9, 3, 14,
13, 21, 5, 19,
 6, 13, 5, 14,
16, 22, 3, 16,
16,  3, 4, 18,
 8, 19, 2, 13,
20,  5, 3, 20,
13,  6, 1, 14,
```

Parsing the individual above, and print out the instructions in 'C-style' looks like this:

```
Reg12 = Reg8 * Reg22;
Reg16 = Reg19 - Reg11;
Reg12 = Reg15 / Reg12;
Reg19 = Reg8 / Reg3;
Reg21 = Reg12 / Reg12;
Reg12 = sin(Reg1);
Reg19 = Reg20 + Reg3;
Reg21 = Reg9 - Reg12;
Reg19 = Reg23 * Reg5;
Reg14 = Reg16 * Reg9;
Reg19 = sin(Reg13);
Reg14 = sin(Reg6);
Reg16 = Reg16 * Reg22;
Reg18 = Reg16 / Reg3;
Reg13 = Reg8 - Reg19;
Reg20 = Reg20 * Reg5;
Reg14 = Reg13 + Reg6;
```

This example individual consists of 17 instructions, and thus have $4 \times 17 = 68$ genes. The encoding scheme is as follows; the first and second elements of an instruction refers to the registers to be used as arguments, the third element corresponds to the operator, i.e. ADD, MUL, and so on, and the last element is a register reference for where to put the result of the operation. The meaning of the first line (instruction) here is: multiply register 8 with register 22 and put the result in register 12. The operators take two arguments, except when the operator is SINE, which of course only take one argument. In this case, the SINE operator is applied to the first element in the instruction, and the second element is simply discarded. A mutation on that element will thus have no effect on that individual's genotype. The register references 1-11 are assigned to I/O-registers, and register references 12-23 are assigned for the internal work registers.

Table 1. Encoding scheme for the operator gene.

Operator	Encoding
ADD	1
SUB	2
MUL	3
DIV	4
SINE	5

3.6 Evolutionary Algorithm

At the beginning of the evolutionary process, the population is filled with randomly created individuals. The length, or number of instructions, of an individual is chosen randomly with Gaussian distribution, with expectation value 20. The maximum length is restricted to 256 instructions. The genes are created with a uniform distribution over their respective search range; 1-23 for the two first genes of an instruction, 12-23 for the last gene, and 1-5 for the third gene, which corresponds to the function set.

Our GP-system is a steady state tournament selection algorithm, with the following execution cycle:

```
1. Select four members of the population for tournament.
2. For all members in tournament do:
   a. Create an instance of the simulated robot.
   b. Record the position in 3d-space of all the robot's limbs.
   c. Execute the individual for 2500 simulation time steps.
   d. Record the final position of all the robot's limbs.
   e. Compute the fitness value (see below).
   f. Destroy the simulated robot.
3. Perform tournament selection.
4. Apply genetic operators on the winners to produce two children.
5. Replace the two losers in the population with the offspring.
6. Go to step 1.
```

The individuals are evaluated (evaluation cycle starting with point 2a. above) under identical conditions, since the simulation is entirely deterministic. They all start from the same standing upright pose, with the same orientation. The execution time for individuals are 2500 simulation time steps (corresponding to approx. 20 seconds of real time simulation), and if an individual cause the robot to fall before this time is completed, the evaluation is terminated. In the beginning of an experiment, a great majority of individuals are terminated before the intended time.

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	24 integer registers,
Function Set	ADD, SUB, MUL, DIV, SINE
Raw Fitness	According to eq. (2), scalar value
Standardized Fitness	Same as Raw Fitness
Population Size	800
Initialization Method	Random
Simulation Time	2500 simulation time steps
Crossover Probability	100%
Mutation Probability	80%
Initial Program Length	Gaussian distribution, expectation value 20.
Maximum Program Length	256 instructions
Maximum Tournament Number	None
Selection Scheme	Tournament, size 4
Termination Criteria	None (determined by the experimenter)

Fitness Calculation As 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 for the experiment was to produce a "human-like", bipedal gait without the robot falling. To accomplish this task, the individual controlling the robot should; (i) locomote the robot as straight forward as possible, and (ii) keep the robot in an upright pose during the movement. Hence, the proper measurements to feed the fitness function with are related to the height maintained by the robot, and the covered distance during simulation. Taking this as starting point, we initially experimented a lot with different variants of the fitness function, before we could find a function, consistent with our intentions. Another commonly used feature in robotics-GP is that the fitness function should have a rewarding part and a punishing part. Explicitly formulated in mathematical terms, the proper fitness function was found to be:

$$f = W\left[1.0 - \frac{h_{start}}{h_{stop}}\right] - (d_{left} + d_{right}) \tag{2}$$

where h_{start} is the height of the robot at the starting position, h_{stop} is the height when evaluation terminates (either the simulation is fully completed, or it is terminated before the intended time, caused by the robot falling). The height measure is applied to the position of the robot's head, however one could take the height of any body part. The second term is a measure of the distance covered by the robot during evaluation, applied to its feet. The robot is always starting with its feet in origo (in xy-plane), forward direction being along the negative y-axis. Forward movement is thus resulting in a value < 0 for the second term, but since the second term is preceded by a minus sign, the contribution to fitness will then be a value > 0. The first term will give a positive contribution to fitness if $h_{stop} > h_{start}$, negative contribution in the case when $h_{stop} < h_{start}$, and zero contribution if $h_{stop} = h_{start}$. Thus we have a fitness function rewarding forward locomotion and keeping the upright pose, and punishing backward movements and falling. The W in the first term is a weight, scaling the mutual relation of rewarding and punishing. After some tweaking, it was found to work best when set to a value in the order of 10.

Genetic Operators By crossover and mutation, variations are introduced to the genetic material. The crossover operator exchanges blocks of code that are cut out at instruction boundaries, and mutation operator changes the inside of an instruction by mutating register references and operators.

We use only two-point string crossover, with 100% probability for crossover, divided mutually on the rate 4:1 on homologous and non-homologous crossover. The homologous crossover acts by randomly picking equivalent two points in both parent's genome, and simply swap the instructions between the two points. This operation is genome-length preserving. Non-homologous crossover differs in that the selected points must not be corresponding. This allows for the crossover operator to vary the length of the genomes.

Mutation operates only on one individual. When an individual is chosen for mutation, the mutation operator works by randomly selecting an instruction from that individual, and makes a change in that instruction. It makes that change either by changing any of the register references to another randomly chosen register reference from the register set, or the operator in the instruction may be changed to another operator that is in the function set. The probability for an individual to undergo mutation is 80%.

4 Results

When observing the experiments in run-time, it is compelling how quickly the simulated robot learns. In the first couple of hundred tournaments, a great majority of the individuals cause the robot to fall almost immediately in the beginning of the evaluation cycle, and the greater part of them tip over backwards. Maybe one out of ten individuals fall to the fore, which is a good starting point of taking a step ahead. Rather soon, however, one can observe the opposite situation, one out of ten individuals' overturn backwards and the rest fall ahead. This was not the desired goal for the evolution, but we regard this as being the first refined behavior that emerged.

The next observable stage of development in the evolution is when a large fraction of individuals is keeping the robot at a standstill, almost motionless, on its feet. In the beginning of our experiments, we faced some problems with evolution converged to this state. By increasing the population size and making some adjustments to the fitness function (mainly by decreasing the weight w, giving lesser punishment for tipping over), we could guide the evolution towards the desired goal. The mix of individuals showing this behavior, and individuals with a more 'energetic' behavior guarantee sufficient diversity of the population for evolution to proceed.

The final results of these experiment was indeed consistent with our initial objectives. That is, evolution created controller programs that made the simulated robot produce forward locomotion behavior. Some of the resulting programs made the robot walking forward in a spiral manner, with small movements, and others produced gait patterns with more lively movements. When tested, some of the individuals managed to keep the robot on its feet for the whole evaluation time (2500 simulation time steps), but when executed for a longer time, the robot usually ended up overturned. Nevertheless, a division of evolved programs could accomplish the task during the test run, without ever tipping over the robot. Fig. 4 and fig. 5 displays some statistics from a representative run. In

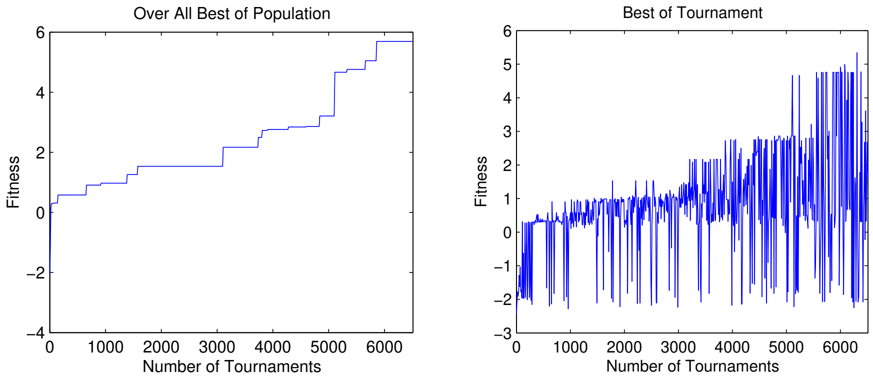

Fig. 4. Fitness value of over all best individual in the population (**a**) and fitness value of the best individual in every tournament (**b**).

these experiments we did more than thirty independent runs, ranging from a few thousand tournaments, up to more than 80000 tournaments. The way fitness was defined (eq.2), a fitness value < 0 correspond to the robot falling backward, and a small positive value (typically ranging from ~ 0.3 to ~ 0.6) correspond to the robot immediately falling ahead, while a value around 1.5 indicate a standstill. In figure 4a, one can observe how the best individual performed those behaviors;

falling backward in the first few hundred tournaments, falling ahead in the first thousand tournaments, and standing still up to the 3000 tournaments. Fitness values in the range of ~ 1.5 to ~ 2.5 indicate some good locomotion, but usually ended up with the robot overturned, and fitness > 2.5 was successful locomotion behavior.

As depicted in fig. 4a, the currently best individuals in the population showed progress from the beginning of the evolution and continued to develop over time. Examining fig. 4b show that there is a fraction, relatively constant over time, of very poor individuals in the population. This pool of not-so-good individuals is probably required to maintain the diversity of the population, and thus avoiding that evolution get stuck in the state of inactive individuals.

Typically, the program length decrease below the initialization length in the beginning of a run, but after a short while it starts to increase above that threshold, and finally it stabilize around some value. See figure 5. In all experiments we used the same initialization program length, with gaussian distribution and expectation value 20. It was observed that the program length, averaged over the whole population, did never go below the value 13, and never above 50, and it usually stabilized somewhere around 30.

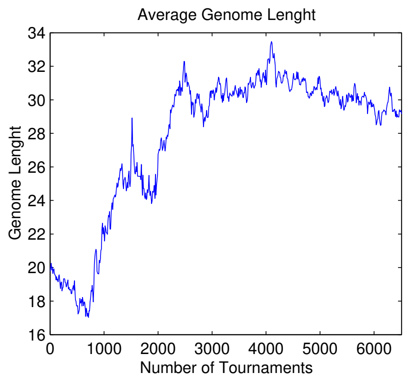

Fig. 5. Average genome length of all individuals in the population, length being defined as the number of instructions in an individual.

5 Summary and Conclusions

We describe the first instance of a novel approach for control programming of humanoid robots. It is based on evolution as the main adaptation mechanism, utilizing computing techniques from the field of Evolutionary Algorithms. However, evolving on real hardware is a challenging task, and in an attempt to overcome some of the difficulties, we use a physically realistic simulation of the robot. The central idea in this concept is to evolve control programs from first principles on a simulated robot, transfer the resulting programs to the real robot and continue to evolve efficient gait on the real robot. As the key motivation for using simulators, we briefly describe an on-line learning experiment performed with our biped humanoid robot 'elvina'.

The 'elvina' robot is a simplified, scaled model of a full-size humanoid with body dimensions that mirrors the dimensions of a human. It is a 28 cm tall, fully autonomous robot with onboard power supply and computer and it has 14 degrees of freedom. The robot is equipped with a digital color camera, and a near-infrared PSD that is used to determine distances to nearby objects. The experiment was performed in order to optimize a by hand developed set of state vectors, defining a static robot gait. We used a steady state evolutionary strategy, running on the robot's onboard computer. Individuals were evaluated and fitness scores automatically determined using the robots onboard digital camera and proximity sensor. By using this system, we evolved gait patterns that locomote the robot in a straighter path and in a more robust way, than the previously manually developed gait did.

However, we want to use physically realistic simulation of the robot, to evolve control programs from first principle. For this we use the Open Dynamics Engine, which is a free library for simulating articulated rigid body dynamics.

The Evolutionary Algorithm is an instance of Genetic Programming, implemented as a Virtual Register Machine with 12 internal work registers and 12 external registers for I/O operations. The individual representation scheme is a linear genome, encoded as an array of integers. The selection method is a steady state tournament algorithm, with size four. By crossover and mutation, variations are introduced to the genetic material. The crossover operator exchanges blocks of code that are cut out at instruction boundaries, and mutation operator changes the inside of an instruction by mutating register references and operators.

The final results of these experiment was consistent with our initial objectives. That is, evolution created controller programs that made the simulated robot produce forward locomotion behavior. Some of the resulting programs made the robot walking forward in a spiral manner, with small movements, and others produced gait patterns with more lively movements.

Current versions of the simulation system and the robot, however, do not allow the evolved programs to be directly downloaded to the robot. Further investigations and improvements are needed. To begin with, we must implement a subsystem of the simulated robot's control system and program interpreter on the real robots micro controller. Further, the real robot has an active feedback system, consisting of a color camera and a distance sensor, which will be implemented on the simulated robot as well. Since the development of the robot platform is an ongoing process, various other sensors, will shortly be implemented on the robot. Then, the simulated robot should of course reflect all aspects, morphological and perceptual, of the real robot.

With this system of two phases of evolution, it will be possible to have a flexible adaptation mechanism that can react to hardware failures in the robot, e.g. if an actuator or sensor break down. By extracting information about malfunctioning parts and do off-line evolution with a modified model of the robot, it will become possible to react to the changes in the robot morphology. For robots working in hazardous environments, or in applications with remote presence robots, this feature would be very useful.

References

1. Banzhaf, W., Nordin, P., Keller, R.E., and Francone F. D. 1998: Genetic Programming An Introduction: On the Automatic Evolution of Computer Programs and Its Applications. San Francisco: Morgan Kaufmann Publishers, Inc. Heidelberg: dpunkt verlag.
2. Baraff, B., and Witkin, A. 1997: "Physically Based Modeling: Principles and Practice" Online Siggraph '97 Course Notes. Carnegie Mellon University, Pittsburgh, PA, USA. `http://www-2.cs.cmu.edu/~baraff/sigcourse/index.html` Last Visited: 01/21/2003.
3. Bräunl, T. 2002: EyeBot Online Documentation. `http://www.ee.uwa.edu.au/~braunl/eyebot/` Last visited:01/21/2003.
4. Darwin, C. 1859: "On the Origin of Species by means of Natural Selection or the Preservation of Favored Races in the Struggle for Life" Murray, London.
5. Dittrich, P., Burgel, A., and Banzhaf, W. 1998: "Learning to move a robot with random morphology" In: Phil Husbands and Jean Arcady Meyer, editors, First European Workshop on Evolutionary Robotics. Berlin: Springer-Verlag, pages 165-178.
6. Fogel, L. J., Owens, A. J., and Walsh, M. J. 1966: Artificial Intelligence through Simulated Evolution. New York, USA: Wiley.
7. Friedberg, R. M. 1958: "A Learning Machine - Part I", IBM Journal of Research and Development, USA: IBM, 2(1), pages 2-11.
8. Holland, J. 1975: Adaptation in Natural and Artificial Systems. Ann Arbor, MI, USA: The University of Michigan Press.
9. Hornby, G.S., Fujita, M. Takamura, S., Yamamoto, T., and Hanagata, O. 1999: "Autonomous evolution of gaits with the Sony quadruped robot" In Proceedings of the Genetic and Evolutionary Computation Conference. San Francisco: Morgan Kaufmann Publishers, Inc.
10. Hornby, G.S., Takamura, S., Yokono, J., Hanagata, O., Yamamoto, T., and Fujita, M. 2000: "Evolving robust gaits with AIBO". IEEE International Conference on Robotics and Automation, New York: IEEE Press, pages 3040-3045.
11. Jakobi, N. 1998: "Running across the reality gap: octopod locomotion evolved in a minimal simulation" In: Phil Husbands and Jean Arcady Meyer, editors, First European Workshop on Evolutionary Robotics. Berlin: Springer-Verlag, pages 39-58.
12. Koza, J. 1992: Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, MA, USA: MIT Press.
13. Langdon, W. B., and Nordin, P. 2001: "Evolving Hand-Eye Coordination for a Humanoid Robot with Machine Code Genetic Programming" In Proceeding of EuroGP 2001. Lake Como, Milan, Italy: Springer Verlag, pages 313-324.
14. Langdon, W. B., and Poli, R. 2002: Foundations of Genetic Programming Springer-Verlag, ISBN 3-540-42451-2, 274 pages.

15. Lewis, M. A., Fagg, A. H., and Solidum, A. 1992: "Genetic programming approach to the construction of a neural network for control of a walking robot" Proceedings of the IEEE International Conference on Robotics and Automation. New York: IEEE Press.
16. Li, Q., Takanishi, A., and Kato, I. 1992: "Learning Control of Compensative Trunk Motion for Biped Walking Robot based on ZMP Stability Criterion" IEEE Int. Conf. on Intelligent Robots and Systems, pages 597-603.
17. Nordin, P., and Nordahl, M. G., 1999: "An evolutionary architecture for a humanoid robot" In Proceedings of the Fourth International Symposium on Artificial Life and Robotics. Oita, Japan.
18. Parker, G. B. 1998: "Generating Arachnid Robot Gaits with Cyclic Genetic Algorithms" In Proceedings of the Third Annual Genetic Programming Conference. Morgan Kaufmann, pages 576-583.
19. Rechenberg, I., 1965: "Cybernetic Solution Path of an Experimental Problem". Royal Aircraft Establishment Translation No. 1122, Farnborough Hants: Ministry of Aviation, Royal Aircraft Establishment.
20. Schwefel, H. P. 1995: Evolution and Optimum Seeking. New York, USA: Wiley.
21. Smith, R. 2002: Open Dynamics Engine v0.030 User Guide. `http://opende.sourceforge.net/ode-0.03-userguide.html` Last Visited: 01/21/2003.
22. Vukobratovic, M. Frank, A. A., and Juricic, D. 1970: "On the stability of biped locomotion" IEEE Transactions on Biomedical Engineering, BME-17 (1), pages 25-36.
23. Vukobratovic, M., Borovac, B., and Surdilovic, D. 2001: "Zero-Moment Point - Proper Interpretation and New Applications" In Proceedings of the 2nd IEEE-RAS International Conference on Humanoid Robots, Humanoids 2001. Waseda University, Tokyo, Japan: Institute of Electrical and Electronics Engineers, Inc., pages 237-244.
24. Wolff, K., and Nordin, P. 2001: "Evolution of Efficient Gait with an Autonomous Biped Robot using Visual Feedback" In Late-Breaking Papers of the 2001 Genetic and Evolutionary Computation Conference, GECCO 2001. San Francisco: Morgan Kaufmann Publishers, Inc., pages 482-489.
25. Wolff, K., and Nordin, P. 2002: "Evolution of Efficient Gait with an Autonomous Biped Robot using Visual Feedback" In Proceedings of the Mechatronics 2002 Conference. University of Twente, Enschede, the Netherlands.
26. Ziegler, J., Barnholt, J., Busch, J., and Banzhaf W. 2002: "Automatic Evolution of Control Programs for a Small Humanoid Walking Robot" 5th International Conference on Climbing and Walking Robots (CLAWAR).
27. Ziegler, J., Wolff, K., Nordin, P., and Banzhaf W. (2001). "Constructing a small humanoid walking robot as a platform for the genetic evolution of walking" In U. Rückert, J. Sitte and U. Witkowski, editors, Proceedings of the 5th International Heinz Nixdorf Symposium: Autonomous Minirobots for Research and Edutainment, AMiRE 2001 (pp. 51-59). Paderborn, Germany: Heinz Nixdorf Institute, University of Paderborn.