ZHAW School of Engineering
in collaboration with the
Artificial Intelligence Lab of the University of Zurich

Bachelor thesis
May 2009

A Robot Learning to Walk

Authors:
Urs Fässler, Nicolas Rüegg

Advisors:
Dr. R. Leuenberger (ZHAW),
M. Hoffmann (University of Zurich)

May 15, 2009

Abstract

The goal of this work was to develop a dynamic quadruped robot based on a minimalistic controller that exhibits a rich behavioral diversity including multiple gaits. The different patterns of locomotion are the basis for adaptation and research into embodied cognition.

A simulated model of the robot was used to facilitate optimization of the body and controller parameters. Using an evolutionary algorithm, suitable parameter values for the three gaits "walk", "trot" and "bound" were found.

Since a different body evolved for each gait, we tried to find a compromise morphology equally suitable for all gaits. In the process, we reached the limits of the existing framework for the optimization of robots. Thus, a new program called *Vidyaa* for the optimization of parameter values was developed. Due to its ability to nest optimization tasks, *Vidyaa* allows a new approach to these kinds of problems.

It is expected that the insights gained can be transferred to a real version of the robot.

Contents

List of Figures

List of Tables

Listings

1 Introduction

1.1 Overview

[1] In the project *"From Locomotion to Cognition"*, the University of Zurich's AI Lab engages several Ph.D. students. The main goal of this project is to find out what low-level sensory-motor tasks such as locomotion, navigation, and grasping, have to do with higher levels of cognition. Or to put it more provocatively, "What does walking have to do with thinking?".

A part of the project also includes the exploration of the design principles of biologically inspired legged quadrupeds. One of these robots is *Puppy*, an underactuated quadruped with four active hip joints and four passive knee joints. Touch sensors at the feet and potentiometers at the knee joints allow to measure feedback.

Figure 1.1 shows *Puppy* in the simulation.

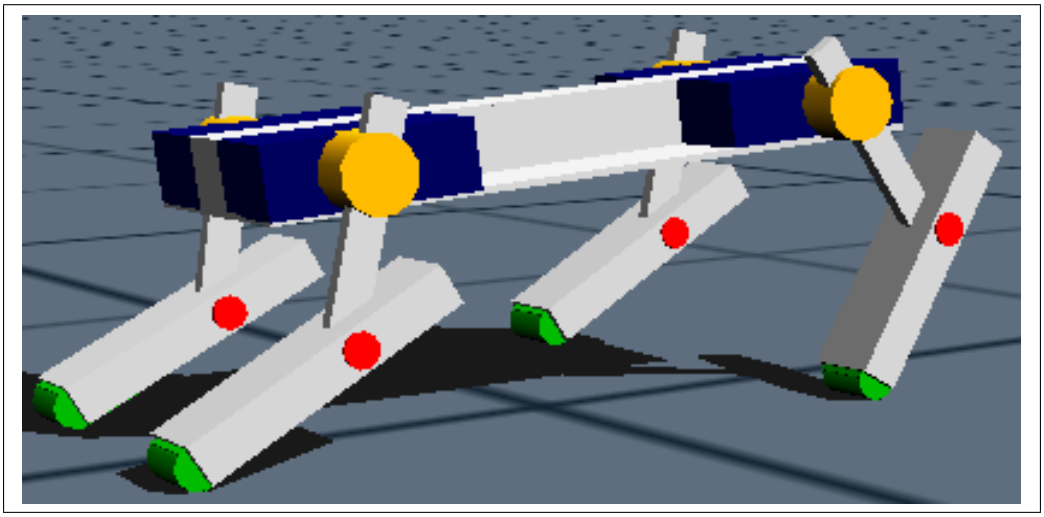

Figure 1.1: Default *Puppy* in *Webots*

In our first task, the aim was to find suitable body and controller parameters for the three gaits "walk", "trot" and "bound". By using an evolutionary algorithm, we tried to find as energy-efficient gaits as possible. The details of this process are described in chapter 2, *Development of Gaits*.

Since different body parameters turned out to be suitable for each gait, we tried to find a compromise body equally suitable for all gaits in our second task.

[1] Parts taken from [Hut08] and *http://ailab.ifi.uzh.ch/research/legged-locomotion*.

In the process, we reached the limits of the existing framework for the search of appropriate parameter values. Thus, we developed a completely new program called *Vidyaa* for the optimization of parameter values. This is described in chapter 3, *Vidyaa – Optimization Made Easy.*

In chapter 4 (*Conclusion*), we look back on our work and discuss possible future work.

Finally, the appendix contains a manual to the used framework for the improvement of robots, a manual for *Vidyaa* and the requirements which lead to its development.

Based on the insights and results of our work, future projects will try to use pattern generators with sensory feedback for the control of the locomotion (see also [RI08]). Also, it is hoped that the parameter values found and the corresponding gaits can be used as the base for more complex behavior such as avoiding obstacles.

1.2 Preliminary Work

1.2.1 Puppy – A Quadruped Robot

In 2004, Iida and Pfeifer [IP04] conducted experiments to explore rapid locomotion of a quadruped robot using *PuppyI*. *PuppyI* is a robot with four identical legs and a simple sine-wave generator as motor controller. This robot has subsequently been improved and extended with touch sensors at the feet and potentiometers at the knee joints allowing to measure feedback (often referred to as *PuppyII*).

To explore its capabilities and further optimize it, a *Webots*-model[2] was created. Based on this model, different body and controller parameters such as the leg length or the controller frequency are tested.

To conduct an automatic exploration and optimization of the parameters, M. Hoffmann wrote a framework including a simulated annealing algorithm. In 2008, it was extended by an evolutionary algorithm as part of a student project. [Hut08]

Let us have a look at this framework more closely, as it was intensely used during the first task.

1.2.2 Framework for the Improvement of Robots

Basically, the framework consists of two different parts. One part creates configuration files which define the body and controller parameters of the robot. The values which are written into these configuration files are determined by an optimization algorithm. The other part takes these configuration files as input and generates a so called *world*-file (*.wbt*) which determines the morphology of the robot and is finally read by *Webots*.

So far, three different optimization algorithms were implemented: Simulated annealing, an evolutionary algorithm and a systematic algorithm.[3]

[2] *Webots* is a simulation software for robots. It has been codeveloped by Cyberbotics and the Swiss Federal Institute of Technology in Lausanne (EPFL).

[3] In the strict sense, the systematic algorithm is not an optimization algorithm, because it does not optimize. It just tests all possible combinations of parameter values.

Simulated Annealing

Simulated annealing is an algorithm to locate a good approximation to the global minimum of a given function.

The following definition is taken from [wik09] and is a good description of the basic idea:

> Each step of the simulated annealing algorithm replaces the current solution by a random "nearby" solution, chosen with a probability that depends on the difference between the corresponding function values and on a global parameter T (called the temperature), that is gradually decreased during the process. The dependency is such that the current solution changes almost randomly when T is large, but increasingly "downhill" as T goes to zero. The allowance for "uphill" moves saves the method from becoming stuck at local minima – which are the bane of greedier methods.

The implemented version of the simulated annealing algorithm is described in detail in [Lun04].

Evolutionary Algorithm

The evolutionary algorithm is based on the principles of the theory of evolution. [4]

Every generation contains a certain number of individuals of which some individuals are selected as parents. Mutations (children) of the selected parents build the next generation.

The fitness of an individual is determined by simulating it with *Webots* and the higher its fitness is, the higher is its chance of becoming a parent.

In the implemented version, there are two different algorithms to select the parents. With the *leaders* algorithm, the parents are randomly chosen from the fittest N individuals. If the *roulette* algorithm is used, the parents are randomly chosen from the entire generation. The normalized fitness value of every individual is the probability with which an individual is selected as a parent. Individuals with a high fitness value are more likely to become parents (even more than once) than individuals with a low fitness value.

Additionally, the algorithm exhibits a so called *elitism* switch. If *elitism* is on, the very best individual of every generation is copied to the next generation without mutation.

Each parameter of a parent has a preset chance of being mutated. If a parameter is mutated, it is calculated as follows:

Be p_p the parent's parameter value and p_c the child's parameter value then:

$$\mu = (r_{\max} - r_{\min}) * \alpha \tag{1.1}$$

$$p_c = p_p + x_d() * \mu \tag{1.2}$$

[4] That is not entirely true: Unlike nature, this implementation just uses mutation and no cross-over.

Here $r_{\min}$ and $r_{\max}$ are the minimum and maximum values for the current parameter. The function $x_d()$ is a random number generator with the distribution d. Equation 1.2 is repeated until $p_c \in [r_{\min}, r_{\max}]$ is true.

To guarantee diversity, the parameter values of the first generation are uniformly distributed over the entire search space.

Systematic Algorithm

To allow for systematic testing of parameter values, we extended the framework with an algorithm which tests every possible combination of the free parameters. Unlike the previously described algorithms, the systematic algorithm is not a optimization algorithm.

Testing every possible combination of parameters is done by systematically increasing the value of the free parameters until every combination is tried. Of course, this is very time consuming with many free parameters and should be avoided for more than three free parameters.

The pseudo code of this algorithm can be found in the appendix.

1.2.3 Additional Work Done

Besides the already mentioned implementation of the systematic algorithm, two parameters for the lateral (*C_Phase_lateral*) and anterior-posterior phase lag (*C_Phase_AP*) of the leg movements were introduced into the framework:

$$C\_Phase\_FR = C\_Phase\_FL + C\_Phase\_lateral \tag{1.3}$$

$$C\_Phase\_HL = C\_Phase\_FL + C\_Phase\_AP \tag{1.4}$$

$$C\_Phase\_HR = C\_Phase\_FL + C\_Phase\_lateral + C\_Phase\_AP \tag{1.5}$$

This allows a reduction from three to two dimensions for the exploration of the phase lags and is based on the assumption that it is sufficient to get all reasonable (symmetrical) gaits. In our case, the phase lag of the fore left leg (*C_Phase_FL*) is always zero which makes it even easier.

Additionally, we adapted the framework to Version 6.0.1 of *Webots* and corrected a few bugs.

2 Development of Gaits

2.1 Objective

By using *Webots* and an optimization algorithm, parameter values as good as possible should be found for different gaits of the simulated version of *Puppy*. The parameters available for optimization are listed in table 2.1.[1]

Parameter	*Description*	*Unit*
FrontFemurLength	Length of the front femur	m
FrontTibiaLength	Length of the front tibia	m
HindFemurLength	Length of the hind femur	m
HindTibiaLength	Length of the hind tibia	m
KNEEFSpringConstant	Front knee spring constant	
KNEEFDampingConstant	Front knee damping constant	
KNEEHSpringConstant	Hind knee spring constant	
KNEEHDampingConstant	Hind knee damping constant	
FrontKneeOffset	Front knee offset (position)	
HindKneeOffset	Hind knee offset (position)	
RobotCenterOfMass	Center of mass	relative to *Puppy*'s length
C_Frequency	Controller frequency	Hz
C_Phase_lateral	Lateral phase lag	relative to 2π (1 equals 2π)
C_Phase_AP	Anterior-posterior phase lag	relative to 2π (1 equals 2π)
C_Amp_F	Amplitude of the front leg (motor)	
C_Amp_H	Amplitude of the hind leg (motor)	
C_Offset_F	Reference position of the front motors	
C_Offset_H	Reference position of the hind motors	

Table 2.1: Parameters available for the optimization

Since different body parameter values for every gait were likely to result, the insights should be used to evolve a compromise morphology equally suitable for all different gaits.

It is expected that the parameter values found and the corresponding gaits can be used as the base for more complex behavior such as avoiding obstacles or catching a prey.

[1] Additional parameters are available, but they were not optimized.

2.2 A Side Note to Biology - Gaits in Quadrupeds

In nature, animals use different ways of locomotion to get from one location to another. While some animals walk, others swim, crawl or fly. Some animals are even able to move in two completely different ways. Many bugs can fly and walk. Salamanders can swim as well as walk and there are even flying fish.

As one can see, nature came up with many different patterns of locomotion. Even animals which usually just move on land can vary the way of locomotion. For instance, four-legged animals like cats and dogs have different gaits like walking, trotting and bounding. Although they are still using their legs to move forward, they change the way they are using them.

But why are different gaits needed and what is the reason to switch from one gait to another?

To answer this question, let us have a look at two different gaits of a house cat. When a cat roams unhurriedly, it usually just walks slowly (walking gait). However, when it is chased by a little child, it starts running as fast as possible to escape the expected threat (bounding gait). As one can easily see, different situations require different speeds. And since some gaits are more suitable for certain speeds, the animals switch between these gaits.

In general, most animals always try to use the most energy-efficient gait for the desired speed [HT81]. This means that if trotting needs less energy than walking at a certain speed, then trotting is chosen. Figure 2.1 shows the relation between energy consumption and speed in different gaits of a horse.

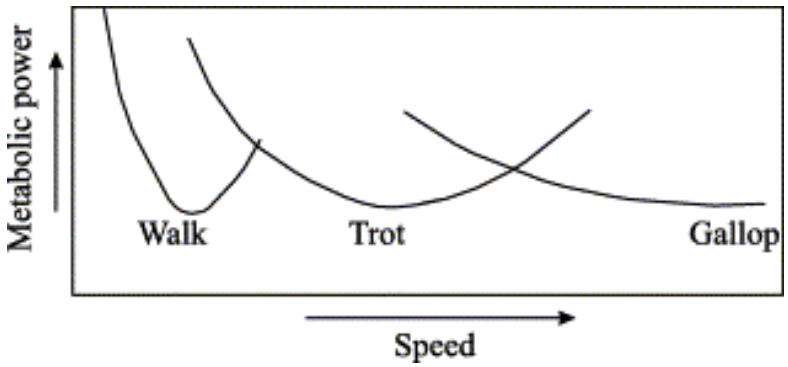

Figure 2.1: Energy consumption per traveled distance at different speeds of a horse [KIJ03]

To summarize this, animals use different gaits in different situations. Depending on the speed they try to achieve, they choose the most energy-efficient gait. Although other criteria like stability might have an influence, too, it is assumed that energy-efficiency is the main reason to switch between gaits.

More about gaits can be found in [JJRW07], [KIJ03] and [HT81].

2.3 Measurement of the Fitness

To get a measurement of the quality of a developed gait and robot, respectively, a so called fitness value is used. Depending on the desired features of a gait, a different approach to measure the fitness is chosen. For instance, there is *speed*, *stability*, the *energy consumption* or any combination of them, to name just a few candidates.

In the following section we will look at three different methods taken into consideration for our experiments.

2.3.1 Speed v

If we are interested in a fast gait, the most obvious measurement is the robot's average speed. The higher the speed v, the better the fitness of the gait.

In our experiments, we determined the speed by computing the traveled distance d from the robot's starting position to the robot's position when the simulation was finished and dividing it by the time t passed since the start of the simulation ($v = \frac{d}{t}$). The corresponding pseudo-code is listed in listing 2.1.

```
double calculateSpeed(void) {
        double xTransition = robotEndPositionX - robotStartPositionX;
        double yTransition = robotEndPositionY - robotStartPositionY;
        double timePassed = endTime - startTime;

        double xyTransition = sqrt(pow(xTransition,2) + pow(yTransition,2));
        return xyTransition/timePassed;
}
```

Listing 2.1: calculateSpeed()

The original code can be found in *puppySupervisorController.cpp* in the method *calculateCurrentSpeed()*.

2.3.2 Energy Consumption E

As in real animals, energy is one of the most important resources. Thus, it might be a good idea to include the energy consumption in the fitness function. Unfortunately, *Webots* does not provide a direct method to measure the energy consumption. Consequently, the user has no choice but to implement it by himself.

Measurement of the Energy Consumption

Basically, there are two straight-forward approaches to measure the energy consumption: where the energy flows in or out of the system. As shown in figure 2.2, there is just one source of energy in our system (servo motor). Measuring the power at this point is the easiest method. Another possibility, but more difficult to implement in *Webots*, would be to measure the loss of energy by damping.

Webots allows to measure the current position (angle x_t) as well as the current force (torque f_t) of a servo motor. Based on this data and the simulation interval Δt, the current power consumption P_t can be calculated. By adding up the power consumed in

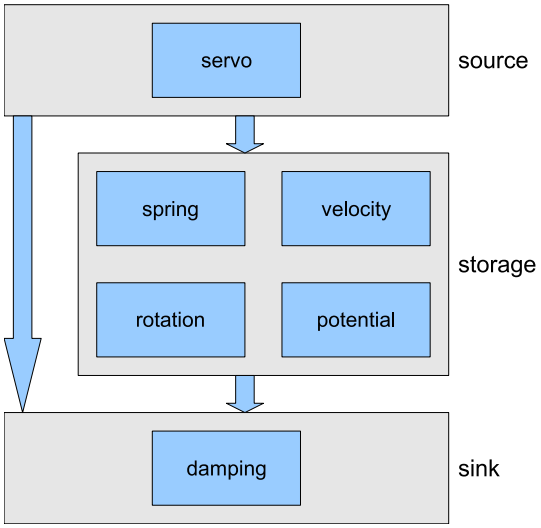

Figure 2.2: Energy flow in the robot

every time step, we can compute the energy consumption E of the servo motor as shown in equation 2.1.

$$E = \sum_t |\Delta t * P_t| \tag{2.1}$$

The unit for E is Joule ($1J = \frac{kg*m^2}{s^2} = 1Ws$).

Since *Puppy* uses a servo motor, there is no negative power[2]. Even when a force supports the movement of the motor, power is consumed to slow down the movement. For this reason the absolute value of the current power is used.

To compute the power consumption, two different methods were reviewed. The first one is based on the laws of physics, the second one uses a servo specific constant c_{wpt} which means watts per torque.

Physical Power Measurement

$$\Delta x = x_t - x_{t-1} \tag{2.2}$$
$$v = \frac{\Delta x}{\Delta t} \tag{2.3}$$
$$P_t = v * f_t \tag{2.4}$$

The unit for P is Watt ($1W = \frac{kg*m^2}{s^3} = 1\frac{J}{s}$).

Watts per Torque Power Measurement

$$P_t = c_{wpt} * f_t \tag{2.5}$$

[2]Power cannot flow back to the battery

Comparision of the Power Measurement Methods

To compare these two methods in a qualitative way, two experiments were conducted. In the first experiment a servo moved a mass by $\frac{\pi}{2}$ as shown in figure 2.3. The second experiment was conducted with a bounding *Puppy*.

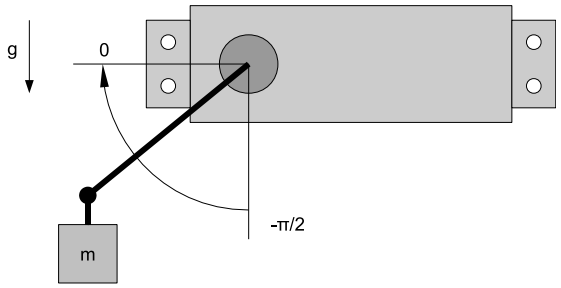

Figure 2.3: Design of experiment 1 (movement of a mass)

Experiment 1: Movement of a Mass Figure 2.4 shows the values measured for the angle and torque. The servo motor accelerated the mass until it reached the maximal velocity. In order to get realistic results, the following oscillation of the torque which is caused by the ongoing correction of the position was filtered out by a low pass filter.[3]

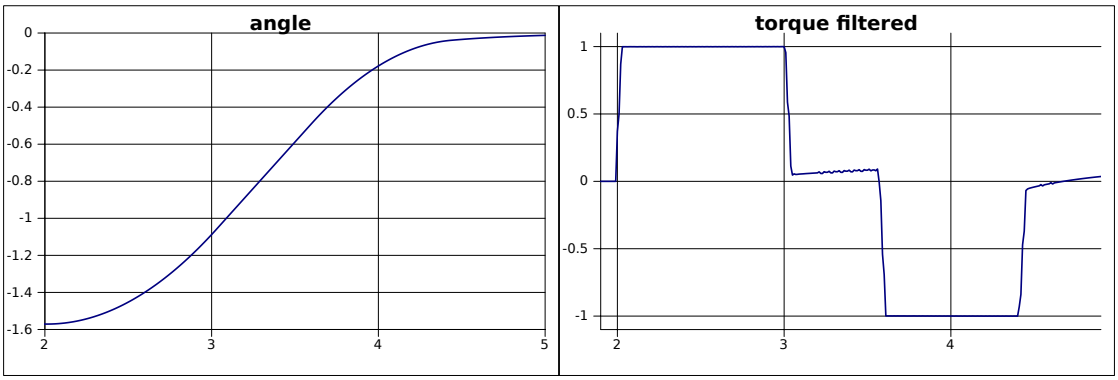

Figure 2.4: Measured data from experiment 1 (movement of a mass).

The calculation of the power and the energy consumption for both methods is shown in figure 2.5. The constant parameter c_{wpt} is hand tuned to a value of 0.53, such that the energy consumption is the same for both methods. $c_{wpt} = 0.53$ is a realistic value of a servo motor. As one can see in figure 2.5, at least in this experiment both methods are accurate.

[3]The low pass was realized by computing the average of four sequential values.

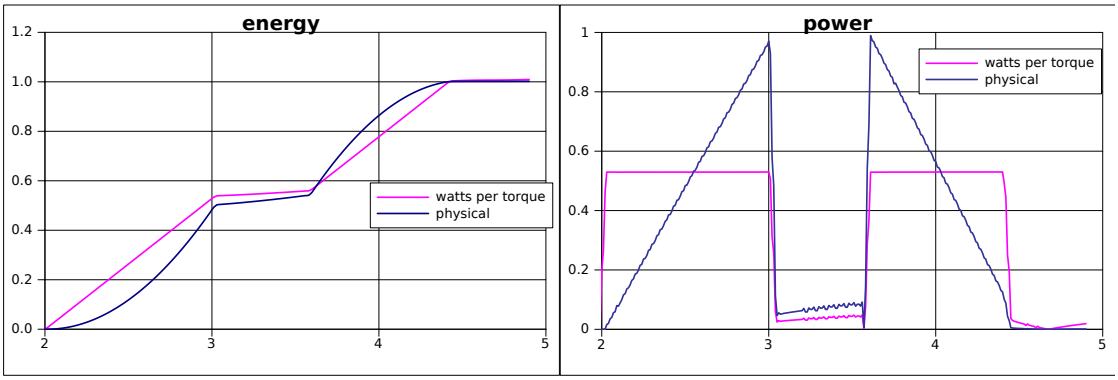

Figure 2.5: Comparision of the two methods for the power measurement in experiment 1. c_{wpt} set to 0.53.

Experiment 2: Puppy In this experiment, a servo motor of the hind hip of *Puppy* was used for the measurement. Figure 2.6 shows the values measured for the angle and torque. Although it is not filtered, there is no oscillation on the torque. Apparently, the servo does not oscillate if it is on-load operation. Thus, low pass filtering the torque is not necessary in our case.

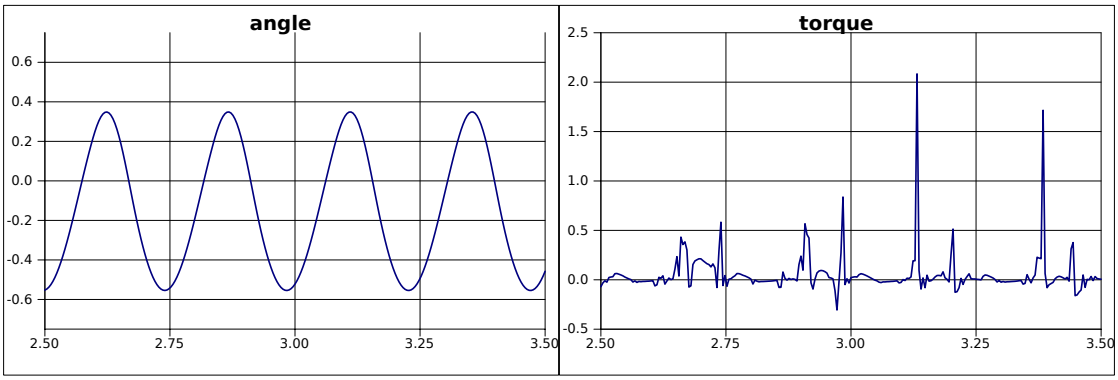

Figure 2.6: Measured data from experiment 2 (*Puppy*).

The calculation of the power and the energy consumption is shown in figure 2.7. Again, the constant parameter c_{wpt} is tuned by hand to a value of 7.7 . This is much more, but still a realistic value. It corresponds with the fact, that *Puppy* uses a more powerful servo.

As one can see in figure 2.7, after every period, the energy consumption results on the same level for both methods although the power consumption shows different peaks. This suggests both methods could be used.

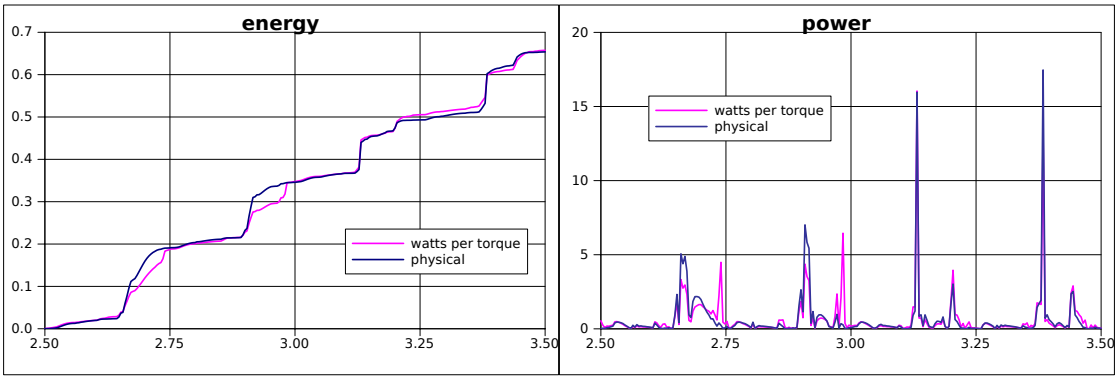

Figure 2.7: Comparision of the two methods for the power measurement in experiment 2. c_{wpt} set to 7.7.

Conclusion

Both measurement methods provide good results but since no hand tuning is required for the physical power measurement, it is easier to use. Nevertheless, the energy consumption by itself is not a good fitness value. Since not moving at all uses the least energy, a standing robot would be rated very good.

2.3.3 $\frac{d}{E}$ ratio

Although the energy consumption E by itself is not a good fitness value, it can definitely be combined with other measurements to build a useful fitness value f. Especially if we remember the curves of figure 2.1, the energy consumption seems to be an important criteria.

Combining the energy consumption E with the traveled distance d allows for the development of energy-efficient gaits without disregarding speed. Since this was one of our goals, we looked closer at the possibilities to combine these two values and came up with two different solutions.

The first idea was to divide the traveled distance by the consumed energy ($\frac{d}{E}$) which is the same as dividing speed by the power ($\frac{v}{P}$):

$$f(v, P) = \frac{v}{P} = \frac{\frac{d}{t}}{\frac{E}{t}} = \frac{d}{E} = f(d, E)$$

In this case, an increase of the traveled distance results in a higher fitness value. So does a decrease of the energy consumption. Additionally, an increase of the traveled distance with the factor two, also results in a fitness value twice as high if the energy consumption is kept on the same level. This makes sense and is exactly what we want. Nevertheless, it is not a perfect solution. If we use the evolutionary algorithm with the roulette wheel algorithm and we have one individual which is much better than the rest (what is not unlikely in the first generation), this is probably the only one which will be

chosen as a parent for the next generation which results in a very small diversity. This is not necessarily wrong and rather a problem of the optimization algorithm than of the fitness value. But one needs to keep this in mind when one decides to use this fitness value. Figure 2.8 shows the fitness function $f(d, E) = \frac{d}{E}$.

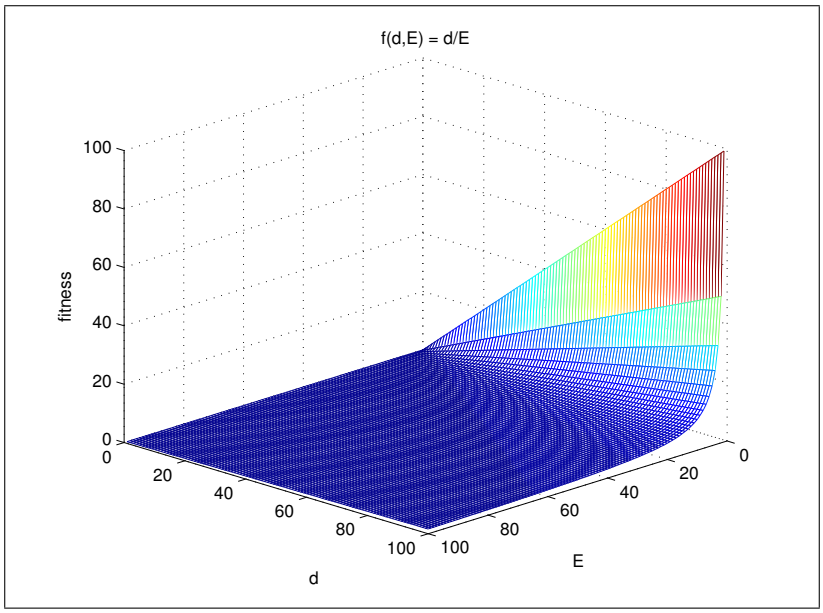

Figure 2.8: $f(d, E) = \frac{d}{E}$

The second idea was to subtract the consumed energy from the traveled distance ($f(d, E) = d - E$) as shown in figure 2.9.

As with the first idea, an increase of the traveled distance results in an increase of the fitness value. An increase of the energy consumption results in a lower fitness value. Intuitively, this is at least not wrong. Nevertheless, this fitness value turned out to be not very useful. The problem is, that we do not know in which range the values of d and E are going to be before we conduct the simulations. This leads to the problem, that an increase of the energy consumption by factor two makes almost no difference to the fitness value if the traveled distance has a much higher magnitude. This is demonstrated in figure 2.10 and practically means that although we are using half as much energy in one gait, it is just slightly more likely to survive in the evolutionary algorithm. Additionally, we could get fitness values below zero which is not very handy. Although this could be solved by adding the inverse energy consumption to the traveled distance ($d + \frac{1}{E}$), it would not solve the other problem.

As one can see, there is no silver bullet. But after balancing the advantages and disadvantages, we decided to go for the ratio $\frac{d}{E}$. It is more useful and easier to handle in combination with the optimization algorithms we used. Additionally, $\frac{d}{E}$ is independent of the simulation time t and thus allows the comparision between different simulations.

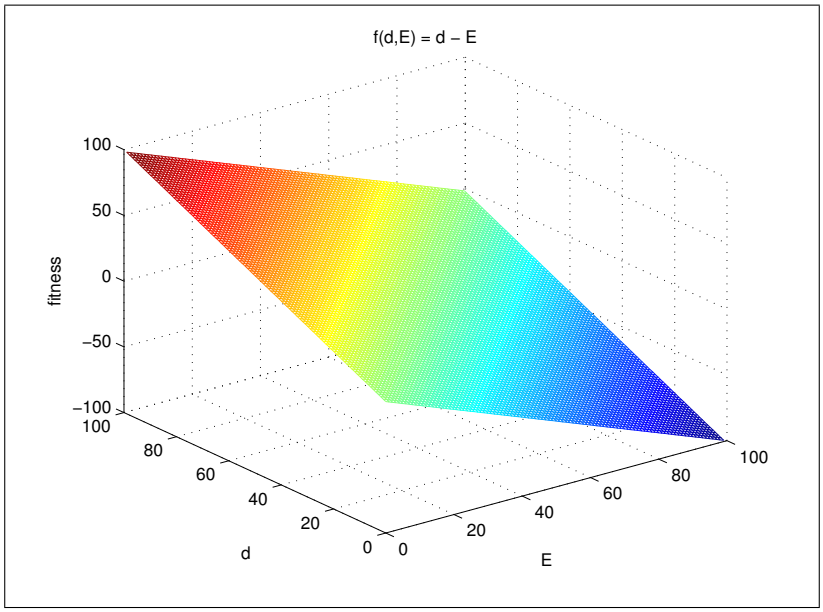

Figure 2.9: $f(d, E) = d - E$

2.4 Exploration of Parameter Values for Different Gaits

In the first series of experiments, the goal was to find good parameter values for defined gaits. The parameters were optimized using the existing framework for the improvement of robots. [Hut08]

The basic design of the experiments was as follows:

1. The desired gait was specified by constraining the lateral and anterior-posterior phase lag within a certain range.

2. By using an evolutionary algorithm with $\frac{d}{E}$ as the fitness value, energy-efficient body and controller parameters were found.

During these experiments, a simple feedforward sine-wave controller was used to control *Puppy's* leg movements. The motor position of every leg thus is described by equation 2.6.

$$A * \sin(2\pi f * t + \omega) + D \tag{2.6}$$

In the simulated version of *Puppy*, the front left leg is the reference ($\omega = 0$).

Details on this controller and also on a CPG[4] controller based on an adaptive oscillator can be found in [Hut08].

[4]CPG: Central Pattern Generator

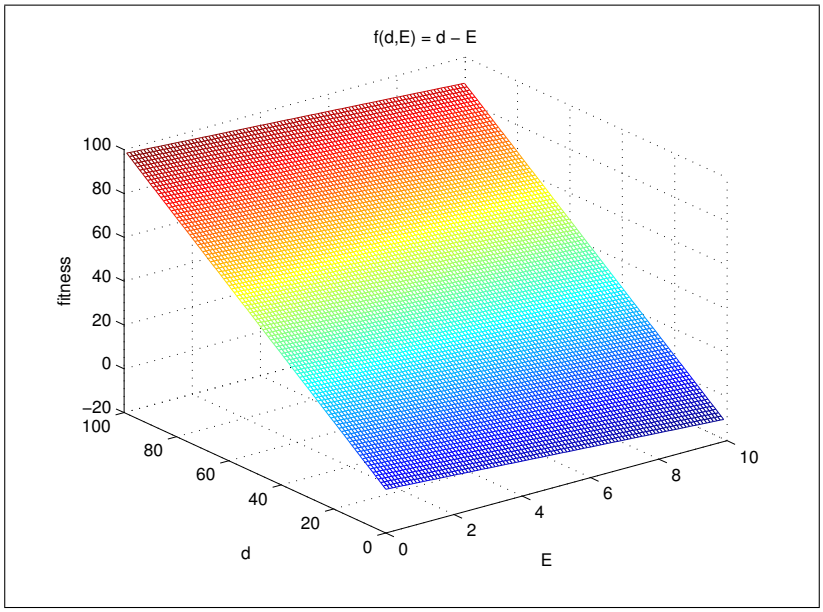

Figure 2.10: $f(d, E) = d - E$ with different scales. (50 - 2) and (50 - 1) almost result in the same fitness value, although we are using half as much energy in one gait.

2.4.1 Walking Gait

The goal of this first experiment, was to explore the parameter values for an energy-efficient walking gait. To make sure we would get a walking gait, we constrained the lateral phase lag to 0.4 – 0.6. and the anterior-posterior phase lag to 0.65 – 0.85.

Since we were particularly interested in an energy-efficient gait, we chose the ratio between the traveled distance and the consumed energy as the fitness value ($f = \frac{d}{E}$).

The other settings to this experiment were as follows:

Setting	*Value*
Generations	750
Population	80
Number of parents	8
Parent selection	roulette
Elitism	On
CHANGEPROBABILITY	5.56%

With a fitness value of 2.13513 ($d = 5.95674$, $E = 2.78987$), individual 62 of generation 561 turned out to be the fittest one:

Parameter	*Range*		*Fittest*
FrontFemurLength	0.05	0.12	0.108417
FrontTibiaLength	0.05	0.12	0.0598935
HindFemurLength	0.05	0.12	0.114643
HindTibiaLength	0.05	0.12	0.0760794
KNEEFSpringConstant	0.1	0.7	0.696502
KNEEFDampingConstant	0.001	0.01	0.00550196
KNEEHSpringConstant	0.1	0.7	0.593708
KNEEHDampingConstant	0.001	0.01	0.00101491
FrontKneeOffset	0.3	1.2	0.797224
HindKneeOffset	0.3	1.2	0.67988
RobotCenterOfMass	-1.0	1.0	0.525413
C_Frequency	0.5	4.0	1.69495
C_Phase_lateral	0.4	0.6	0.501812
C_Phase_AP	0.65	0.85	0.846008
C_Amp_F	0.5236	1.57	0.562042
C_Amp_H	0.5236	1.57	0.785213
C_Offset_F	-0.698	0.698	0.311935
C_Offset_H	-0.698	0.698	-0.0831453

Figure 2.11 shows this individual in *Webots*.

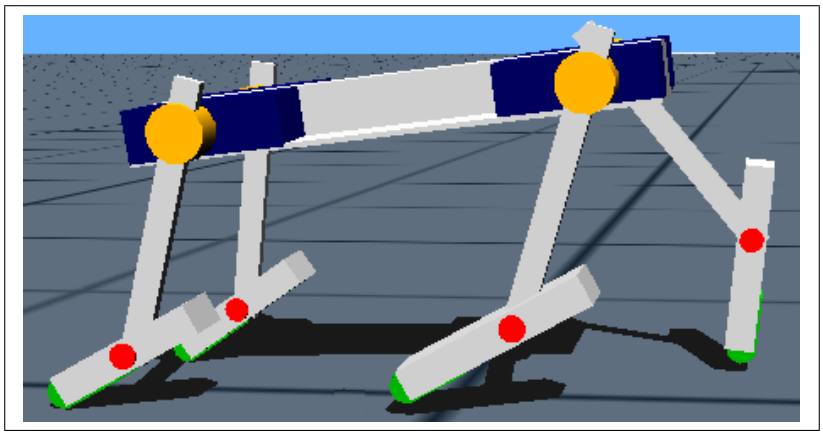

Figure 2.11: Fittest walking individual in *Webots*

Although several individuals with a similar or even the same fitness value occured up to the 750th generation, none of them reached a better fitness value.

Apparently, long femurs are an advantage and the knee spring constants were almost maximized. The center of mass is set quite far to the front and the body itself is tilted. With an anterior-posterior phase lag of 0.846008, the gait almost looks like a pace gait and while walking, its front feet often touch the ground.

Systematic Exploration of the Controller Frequency

Since we were interested whether we can vary the speed of the gait simply by modifying the frequency, we also conducted a systematic experiment. By systematically varying the frequency, we wanted to find out, how frequency, speed and energy consumption are related.

Apart from the frequency, we fixed all parameters to the above values.

Figure 2.12 shows the behavior of the energy consumption and speed when the frequency is increased. As one can see, the fitness reaches its maximum around 1.7Hz. Between 1 and 2Hz the robot is stable[5] but then suddenly becomes unstable.

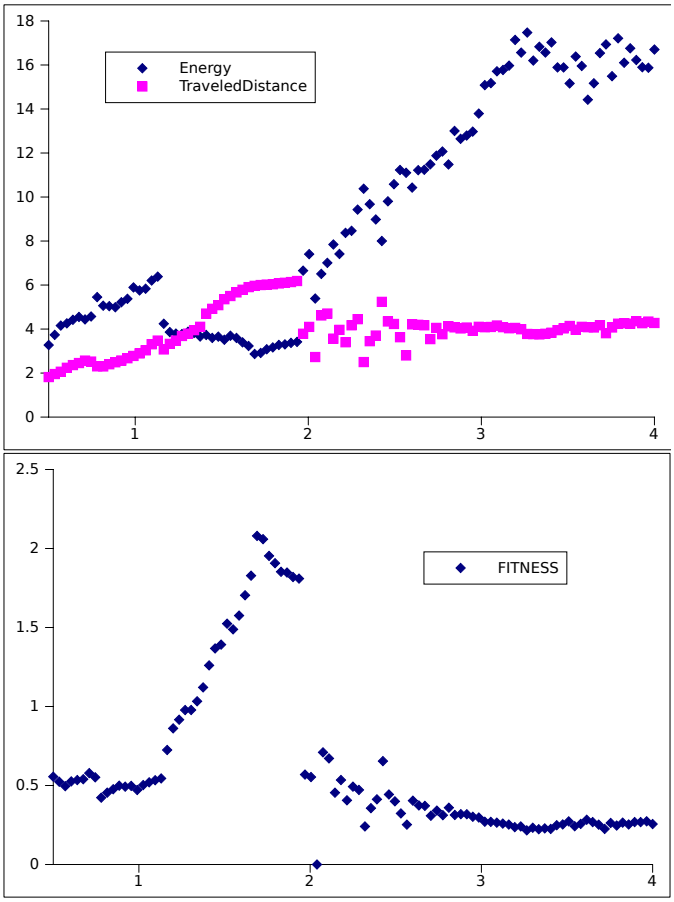

Figure 2.12: Systematic variation of the frequency in the walking gait. Tumbling robots' fitness value is set to 0.

[5]With "stable", we mean a robot which can vary its frequency and amplitude without tumbling.

2.4.2 Trotting Gait

The experiment for the exploration of the parameter values for an energy-efficient trotting gait was done in a similar way as the experiment for the walking gait. To make sure we would get a trotting gait, we constrained the lateral and anterior-posterior phase lag to 0.45 – 0.55.

Again, we chose the ratio between the traveled distance and the consumed energy as the fitness value ($f = \frac{d}{E}$).

The other settings to this experiment were as follows:

Setting	*Value*
Generations	750
Population	80
Number of parents	8
Parent selection	roulette
Elitism	On
CHANGEPROBABILITY	5.56%

With a fitness value of 1.43427 ($d = 5.80094$, $E = 4.04452$), individual 53 of generation 127 turned out to be the fittest one:

Parameter	*Range*		*Fittest*
FrontFemurLength	0.05	0.12	0.0858701
FrontTibiaLength	0.05	0.12	0.0674084
HindFemurLength	0.05	0.12	0.115045
HindTibiaLength	0.05	0.12	0.0667877
KNEEFSpringConstant	0.1	0.7	0.576192
KNEEFDampingConstant	0.001	0.01	0.00563574
KNEEHSpringConstant	0.1	0.7	0.634036
KNEEHDampingConstant	0.001	0.01	0.00405694
FrontKneeOffset	0.3	1.2	0.800005
HindKneeOffset	0.3	1.2	1.00625
RobotCenterOfMass	-1.0	1.0	0.599965
C_Frequency	0.5	4.0	2.37565
C_Phase_lateral	0.45	0.55	0.502542
C_Phase_AP	0.45	0.55	0.540277
C_Amp_F	0.5236	1.57	0.534679
C_Amp_H	0.5236	1.57	1.3026
C_Offset_F	-0.698	0.698	0.0595042
C_Offset_H	-0.698	0.698	0.0435751

Figure 2.13 shows this individual in *Webots*.

The fittest trotting individual has a very similar body to the one evolved for the walking gait. Compared to the previously found walking gait, the trotting gait has a

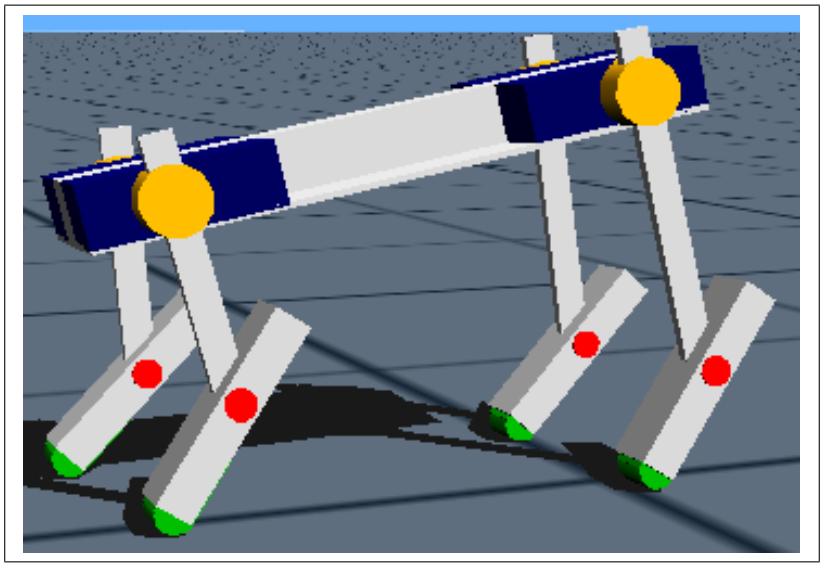

Figure 2.13: Fittest trotting individual in *Webots*

very low fitness value. It consumes about 40% more energy for the same distance. Again, its front feet often touch the ground while trotting.

Systematic Exploration of the Controller Frequency

As in the experiment for the walking gait, we conducted a systematic experiment to find out how frequency, speed and energy consumption are related.

Apart from the frequency, we fixed all parameters to the above values.

As one can see in figure 2.14, the evolved trotting *Puppy* is much more stable in terms of frequency than the walking version.

2.4.3 Bounding Gait

As in the other experiments, we made sure we would get a bounding gait by constraining the lateral phase lag to 0.0 and the anterior-posterior phase lag to 0.4 – 0.6.

Again, we chose the ratio between the traveled distance and the consumed energy as the fitness value ($f = \frac{d}{E}$).

The other settings to this experiment were as follows:

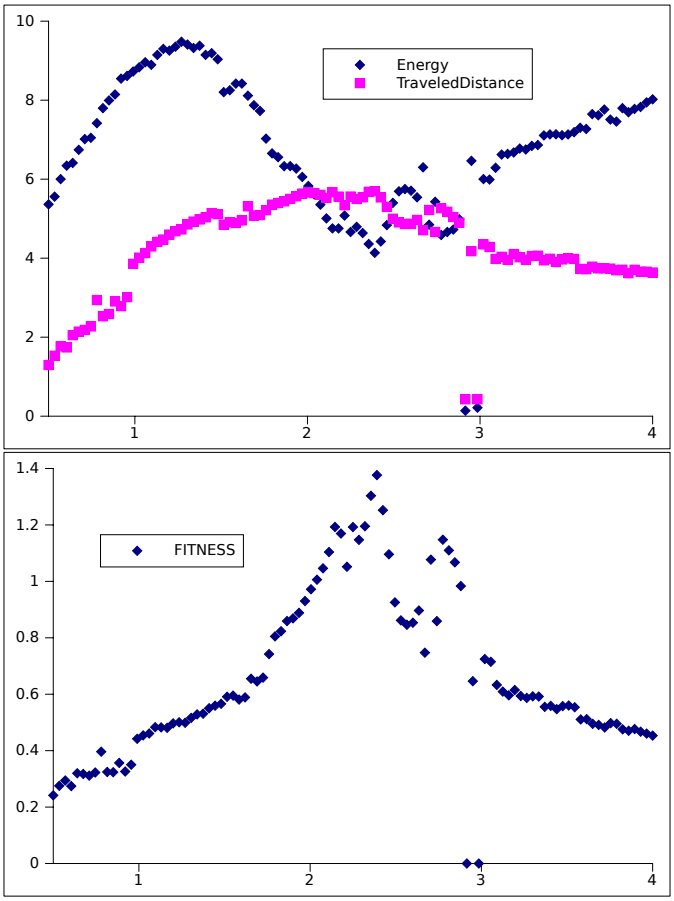

Figure 2.14: Systematic variation of the frequency in the trotting gait. Tumbling robots' fitness value is set to 0.

Setting	*Value*
Generations	750
Population	80
Number of parents	8
Parent selection	roulette
Elitism	On
CHANGEPROBABILITY	5.56%

With a fitness value of 2.58945 ($d = 6.07075$, $E = 2.34442$), individual 15 of generation 392 turned out to be the fittest one:

Parameter	*Range*		*Fittest*
FrontFemurLength	0.05	0.12	0.107495
FrontTibiaLength	0.05	0.12	0.0593474
HindFemurLength	0.05	0.12	0.0785541
HindTibiaLength	0.05	0.12	0.0816153
KNEEFSpringConstant	0.1	0.7	0.695015
KNEEFDampingConstant	0.001	0.01	0.0089879
KNEEHSpringConstant	0.1	0.7	0.688352
KNEEHDampingConstant	0.001	0.01	0.00114522
FrontKneeOffset	0.3	1.2	0.478145
HindKneeOffset	0.3	1.2	0.951577
RobotCenterOfMass	-1.0	1.0	0.318758
C_Frequency	0.5	4.0	3.39166
C_Phase_lateral	0.0	0.0	0
C_Phase_AP	0.4	0.6	0.400149
C_Amp_F	0.5236	1.57	0.525028
C_Amp_H	0.5236	1.57	0.857557
C_Offset_F	-0.698	0.698	-0.261618
C_Offset_H	-0.698	0.698	-0.290135

Figure 2.15 shows this individual in *Webots*.

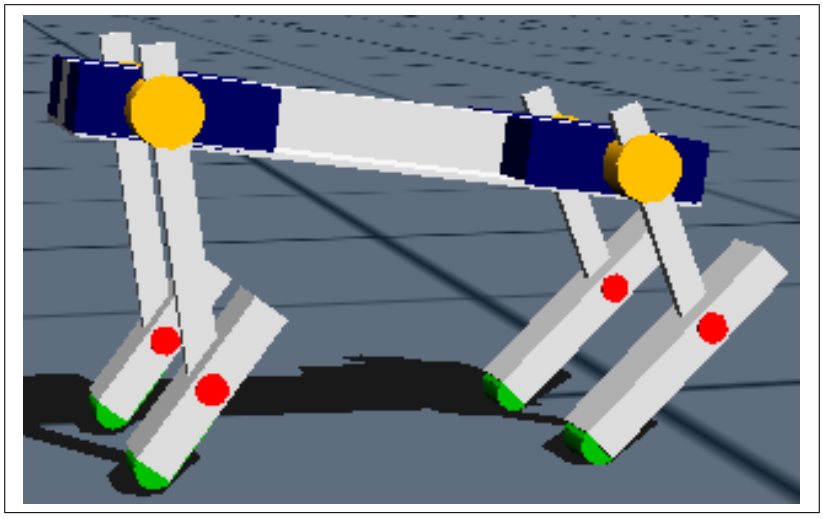

Figure 2.15: Fittest bounding individual in *Webots*

The bounding individual reaches the highest fitness value of all individuals analyzed so far. Compared to the best trotting and walking individuals, it has a more upright posture and shorter hind femurs.

Since the anterior-posterior phase lag was minimized, we assume it would make sense to allow for a smaller anterior-posterior phase lag in future experiments.

Systematic Exploration of the Controller Frequency

To find out how frequency, speed and energy consumption are related, we systematically varied the frequency while the other parameters were fixed to the above values.

As figure 2.16 demonstrates, the bounding *Puppy* behaves very instable around the highest fitness value but is stable in the range from 1 to 2Hz.

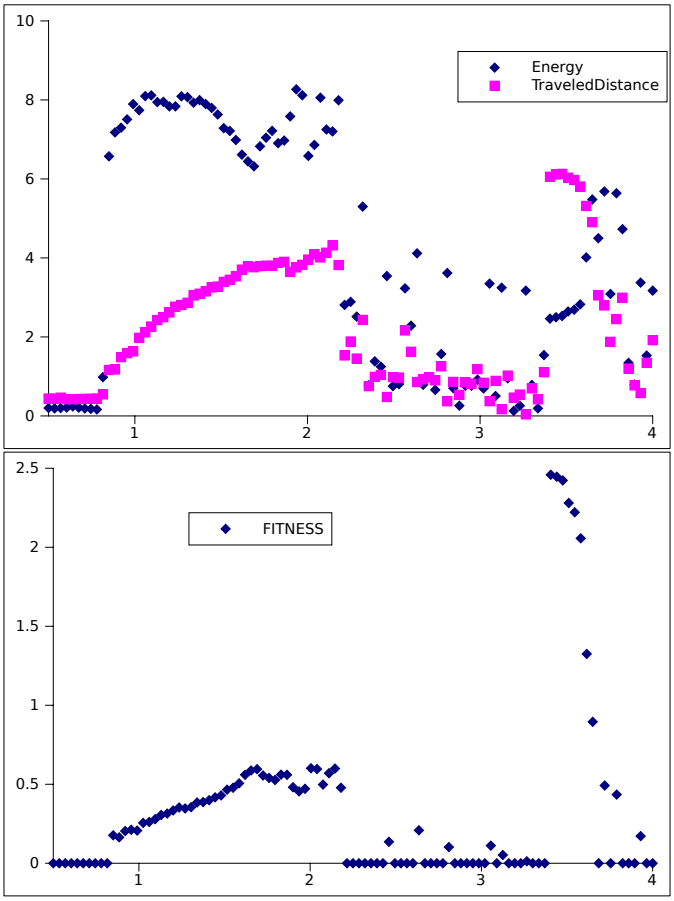

Figure 2.16: Systematic variation of the frequency in the bounding gait. Tumbling robots' fitness value is set to 0.

2.5 Compromise Morphology

So far, several experiments were conducted to find good body and controller parameters. As a reminder, the basic design of these experiments was as follows:

1. The desired gait was specified by constraining the phase lags within a certain range.
2. By using an evolutionary algorithm, suitable body and controller parameters were found.

This design of the experiments resulted in robots which are very good at a specific gait but are not necessarily good at other gaits or not even stable within the same gait (robot tumbles as soon as the frequency is varied).

While this type of experiment allows us to draw conclusions for the relationship of parameters under certain conditions, it does not allow us to generate robots which perform well in all different gaits and are guaranteed to be stable within a certain gait.

Since the aim of the second task was to develop a robot which performs well in all gaits, we designed a new type of experiment to find a compromise morphology for the robot.

2.5.1 Design of the New Experiments

The new experiment should provide the following results:

- A robot which performs well and is stable in all three gaits.
- The robot should be able to vary the speed within a gait by adjusting the frequency.

To summarize it, we need a new, holistic approach. The same robot needs to be able to perform well in all different gaits and must be able to vary the speed within a gait. One possible design of an experiment which could achieve the desired results is described in the box on page 22.

A Specific Experiment for Puppy

1. Divide the parameters into body and controller parameters. Controller parameters can be changed during the lifetime of a robot, body parameters can not.
2. Birth of a robot with a given body.
   a) This robot is now in a phase we call "childhood". During its childhood, it should learn to walk, trot and bound. To learn these different gaits, the robot has a certain amount of "time". Within a gait, the robot must be able to vary the speed by systematically varying the frequency.
   b) Just the robots which perform well in all gaits are likely to survive.

This idea results in a three-level experiment. The highest level might be an evolutionary algorithm exploring the body parameters. For every given body, a simulated annealing algorithm could try to find acceptable gaits by exploring the controller parameters and within each gait, the controller frequency could be systematically changed within a certain range to figure out which gaits are stable.
The results of every level are then returned to the experiment on the higher level in a way that the higher level can include it in its decisions (e.g. by a fitness value).
This design of the experiment is also illustrated in figure 2.17.

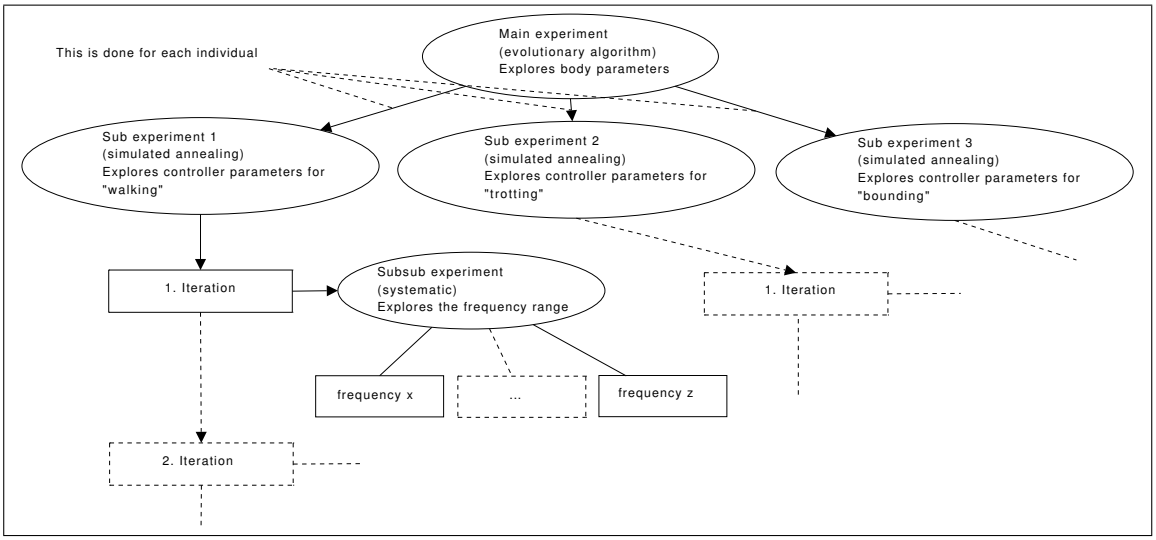

Figure 2.17: Design of a complex experiment

Although this biologically inspired experiment seems reasonable, it has a huge drawback: Due to its massive demand of computing power, it can not be realized. A small calculation shows this in detail:

ipg: individuals per generation (evolutionary algorithm)
spg: steps per gait (simulated annealing algorithm)
ss: systematic steps (systematic algorithm)
sps: seconds per simulation (approximately 3 seconds)

$$\begin{aligned} 500generations * 50ipg * 3gaits * 100spg * 10ss * 3sps &= 225'000'000s \\ &= 3'750'000m \\ &= 62'500h \end{aligned}$$

Obviously, evolution in nature is a parallel process for a reason.

By exchanging the evolutionary algorithm with a simulated annealing algorithm, the number of robots to simulate would decrease dramatically:

$$\begin{aligned} 100steps * 3gaits * 100spg * 10ss * 3sps &= 900'000s \\ &= 15'000m \\ &= 250h \end{aligned}$$

With an execution time of less than two weeks, this experiment can definitely be conducted and with some fine-tuning, it would probably be possible to reduce the execution time to less than five days.

Unfortunately, the design of the framework did not allow to conduct experiments as a part of another experiment and thus it was impossible to conduct the suggested experiments.

Consequently, we had the choice between extending the old framework or writing a new program from scratch. After weighing the pros and cons, we came to the conclusion to write an entirely new program for the optimization of the robot's parameters.

2.6 Summary

For all gaits, energy-efficient parameter values were found. While the trotting gait is stable within the entire frequency range, especially the bounding gait is very unstable for frequencies higher than 2Hz.

In general, smoothly moving individuals achieved better fitness values than abruptly moving ones.

The walking gait found is very similar to the pace gait. A more characteristic walking gait could be achieved by constraining the anterior-posterior phase lag to a smaller range around 0.75.

The evolved walking and trotting individual both lack ground-clearance. Since most surfaces are uneven, this could lead to very instable behavior if transferred to a real *Puppy*.

Conducting some tests, we accidentally discovered the following walking version of *Puppy*:

Parameter	*Value*
FrontFemurLength	0.077682
FrontTibiaLength	0.088447
HindFemurLength	0.117036
HindTibiaLength	0.061377
KNEEFSpringConstant	0.399338
KNEEFDampingConstant	0.002245
KNEEHSpringConstant	0.229442
KNEEHDampingConstant	0.004256
FrontKneeOffset	0.857459
HindKneeOffset	0.695156
RobotCenterOfMass	0.633376
C_Phase_lateral	0.485147
C_Phase_AP	0.810948
C_Amp_F	0.558569
C_Amp_H	0.828399
C_Offset_F	-0.014544
C_Offset_H	0.038788

As figure 2.18 shows, it features a very stable walking gait. In future experiments, sta-

ble gaits could be promoted by varying the frequency within a simulation and including it in the fitness value. Additionally, varying the amplitude with the frequency possibly brings improvement.

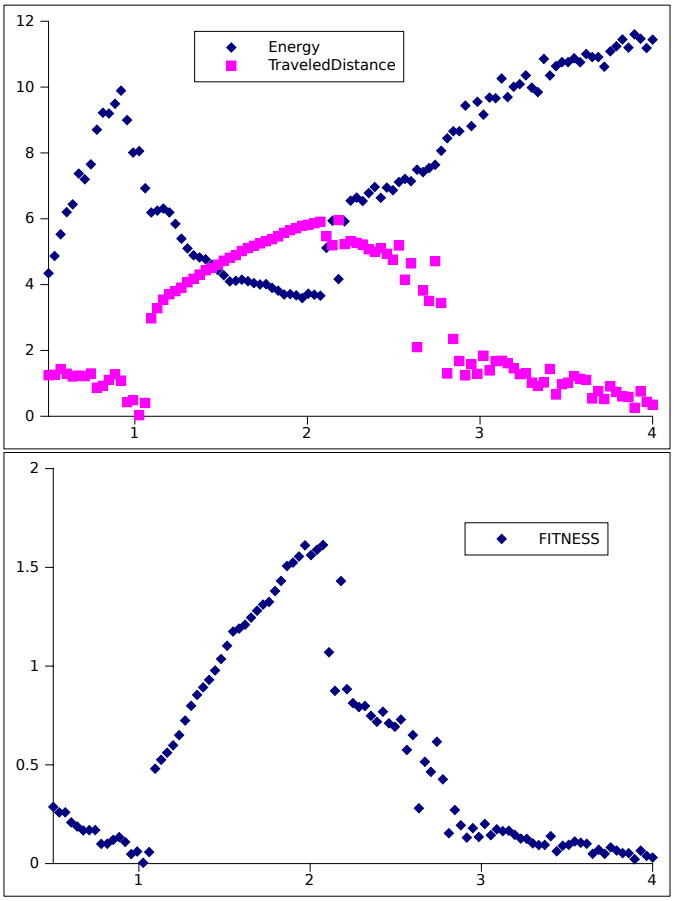

Figure 2.18: Stable walking version of *Puppy*. Tumbling robots' fitness value is set to 0.

For further analysis, a video and the experiment results of each gait can be found in the appendix.

3 Vidyaa – Optimization Made Easy

3.1 Overview

Based on the insights from the development of gaits for the *Puppy* robot, we developed a new program called *Vidyaa* for the optimization of parameter values. What the individual parameters represent is of no interest to the program itself, which allows to optimize a neural network as well as the hardware parameters of a certain machine or any other set of parameters.

To distinguish how good a set of parameter values is, the parameters are tested using an external, user-defined program. This simulation software then returns a so called *fitness value* which is used by *Vidyaa* to decide on its further steps.

The optimization algorithm used can be freely selected by the experimenter, and unlike most other optimization tools, *Vidyaa* allows the combination of several algorithms for the optimization. For instance, it is possible to optimize some parameters with an evolutionary algorithm and some other parameters with a simulated annealing algorithm.

The user defines the optimization task by configuring so called experiments. These experiments can be combined in a hierarchical structure which allows the optimization of parameters on different levels with different algorithms.

In reference to its ability of using optimization algorithms to optimize parameters, we called it *Vidyaa* – the Sanskrit expression for *learning*, *knowledge science*.

3.2 Requirements

Before we started with the actual implementation of *Vidyaa*, we made a list of requirements in which the following use case was identified as the primary use case:

> A user would like to optimize different parameter values of any system. Thus, he designs an experiment by specifying which optimization algorithms to use on which parameters. By defining a hierarchical experiment, the user is also able to conduct an experiment within another experiment. Or to put it differently, he is able to nest optimization tasks.
>
> After he has done the configuration, he starts the experiment and waits until it is finished. During the experiment, all intermediate results and parameter values of every step are saved into a logfile in order to be able to reproduce the results.
>
> After the experiment finishes, he analyzes the results by evaluating the collected data with adequate tools.

The conducted *FURPS* analysis[1] also identified the need for a centralized configuration, easy supportability and extensibility as the major requirements. In addition, a user and programmer guide were expected.

Additional requirements and the details of the *FURPS* analysis can be found in the appendix.

3.3 Implementation

The following sections are just a summary. A detailed documentation on *Vidyaa* is provided in the *Programmer-* and *User Manual* in *Vidyaa – Optimization Made Easy* (see appendix).

3.3.1 Basic Idea

In *Vidyaa*, the user defines the optimization task by configuring so called experiments. An experiment contains the parameters to optimize and the algorithm to use for the optimization. Experiments can be combined in a hierarchical structure. From this follows that for every step of an experiment, the entire sub-experiment is conducted, too. This is illustrated in figure 3.1.

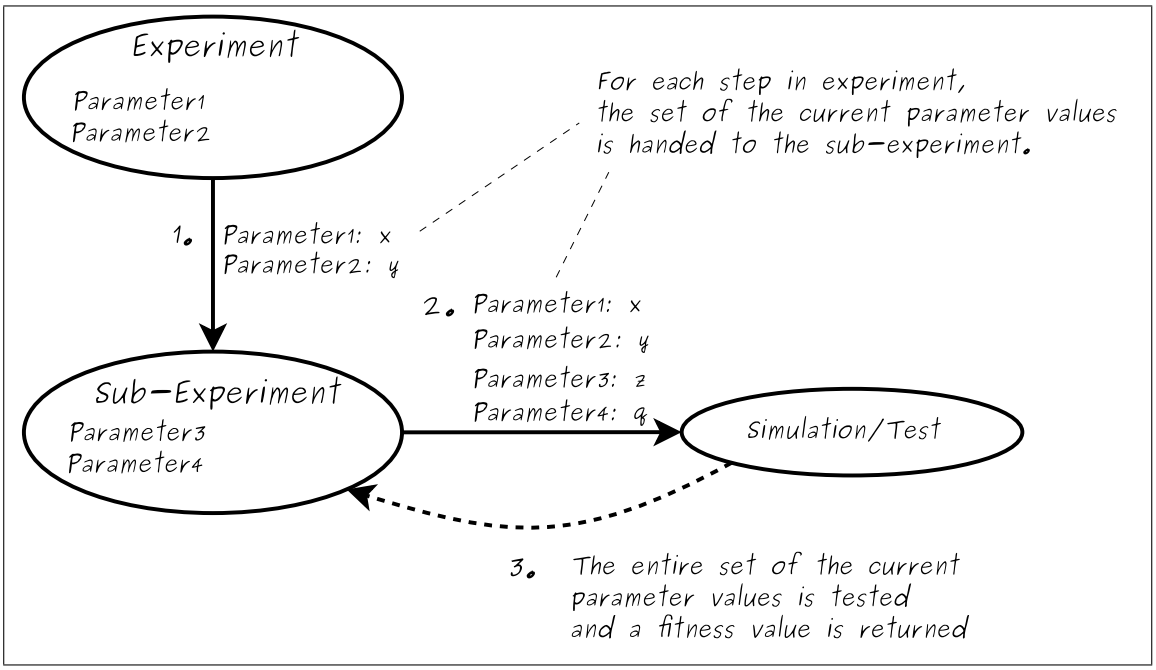

Figure 3.1: The optimization process

The experiment on the first level creates a set of parameter values and passes them to its sub-experiment. The sub-experiment then extends this set of parameter values and

[1] *FURPS*: Functional Requirements, Usability, Reliability, Performance and Supportability

finally passes them to a simulator (or test program), which tests the current values and returns a fitness value. Based on the fitness value, the sub-experiment decides on its next step. When the sub-experiment is finished, it returns a fitness value to the experiment on the next higher level, which then also decides on its next step.

To put it differently, experiments pass their current parameter values to their sub-experiments. That is, a set of parameter values is passed down the hierarchy and ends with a test (simulation) of the parameter values. Every level in between can add or modify something.

3.3.2 Architecture

When *Vidyaa* was designed, corresponding to the requirements, the main objectives were extensibility, flexibility and ease of use. The implemented architecture directly originates from these objectives and many characteristics are quite easy to understand when this is kept in mind.

One of the most basic principles is the separation of the actual optimization from the testing of the parameters. This increases the flexibility as well as the extensibility of the software.

Furthermore, setting up optimization tasks was made easier for the user by choosing XML as the format for a central configuration file. At the same time, existing XML parsers made it easy for a programmer to interprete the configuration file.

Concerning *Vidyaa*'s internals, it has been tried to keep it as simple as possible, without cutting back on flexibility or functionality.

Basically, in *Vidyaa*, an optimization task is regarded as being an *Experiment*. Each experiment can contain other experiments, which allows the nesting of optimization tasks.

3.4 Features

With *Vidyaa* we developed an easily extensible tool for the optimization of parameter values. Its most important features are:

- Stable platform for optimization tasks.
- Centralized configuration in an XML file.
- Human- and machine readable logfile.
- Different log levels configurable.
- Nesting of optimization tasks.
- Separation of the test/simulation program and the actual optimization.
- Availability of two optimization algorithms (simulated annealing, evolutionary algorithm).

- Availability of one algorithm for the systematic testing of parameter values.
- Easily extensible with own optimization algorithms or experiment types.
- Detailed documentation (*User Manual* as well as a *Programmer Manual*) provided.

To allow contributions from others, *Vidyaa* has been published under `http://vidyaa.origo.ethz.ch` as open source software (GPL).

3.5 Roadmap

Although *Vidyaa* is already of great value, there are still many ways to extend and improve it in future versions. Table 3.1 shows a few features considered useful or important by the authors.

Item	Complexity
Ports for other platforms	low
Validation of the configuration file.	low
Enhancement of the data logger	low
New experiment type "Serial"*	medium
Tool for the analysis of the logfile	medium
Automated unit tests for *Vidyaa*	medium
Parallelization of optimization tasks	very high
Network-compatible version	very high
New algorithms	depending on the algorithm

Table 3.1: *Vidyaa* Roadmap

*The idea of the experiment type "Serial" is as follows: A *Serial Experiment* contains a list of experiments, of which the best parameters of one experiment are used as the base for the next experiment.

4 Conclusion

In this work, a simulated model of the *robot* was used to facilitate optimization of the body and controller parameters.

Using evolutionary algorithms, we found body and controller parameters for *Puppy* which feature a high energy efficiency for each of the three gaits "walking", "trotting" and "bounding". The evolution of a given gait was achieved by constraining the range of the possible phase lag between the leg movements. By defining the fitness value as the ratio between traveled distance d and the energy consumption E ($f(d, E) = \frac{d}{E}$), we optimized for the maximal energy efficiency. The development of defined gaits by constraining the phase lag is based on the assumption that our quadruped robot would use the same patterns of locomotion as four-legged animals. However, this needed experimental verification. By freeing the phase lag, one could find out which gaits the robot would evolve by itself. It might also be possible, to govern the evolution of specific gaits by differently weighting parts of the fitness function. For instance, we assume faster gaits would evolve when the fitness function $f(d, E) = \frac{d^2}{E}$ would be used since the traveled distance would be of higher magnitude than the energy consumption.

Since for each gait, different body parameters turned out to be suitable, one should try to find a compromise morphology. Using *Vidyaa* and based on the suggested design of the experiments in section 2.5, this should be possible.

One unique feature of the evolved gaits is that they have a very low ground-clearance. Since most surfaces are uneven, this could make the gaits unstable if transferred to a real robot. Conducting the same experiments with a higher surface friction or adding small pebbles in the simulation could help to increase the ground-clearance in the evolved gaits.

In our experiments, a simple sine-wave generator was used as the controller for the leg movements. It might be possible to however achieve much better gaits by using a CPG controller and including sensory feedback. This could also help with the ground-clearance problem since irregular movements of the legs would be possible.

Finally, the robot should know when to switch between the gaits. Based on the measurement of the energy, this would result in an energy efficient behavior of the robot. Possibly, this could be implemented using a winner-take-all neural network.

Concerning *Vidyaa*, maintenance and further enhancement has beend planned. Parallelization of the optimization algorithms would help a lot to decrease the execution time of optimization processes. Together with a network-compatible version, this could make *Vidyaa* to a powerful tool to solve optimization problems. To allow contributions of others, *Vidyaa* has been published under *http://vidyaa.origo.ethz.ch* including its source code. Currently, it has been tested under Linux but porting it to other platforms can be easily done.

Bibliography

[HT81] D. F. Hoyt and C. R. Taylor. Gait and the energetics of locomotion in horses. *Nature*, 292:239–240, July 1981.

[Hut08] Stefan Hutter, 2008. Co-evolution of Morphology and Controller of a Simulated Underactuated Quadruped Robot using Evolutionary Algorithms.

[IP04] Fumiya Iida and Rolf Pfeifer. "Cheap" Rapid Locomotion of a Quadruped Robot: Self-Stabilization of Bounding Gait. In *Proceedings of the Eigth Conference on Intelligent Autonomous Systems, IOS Press: Amsterdam*, pages 642–649, 2004.

[JJRW07] Thilo Pfau Justine J. Robilliard and Alan M. Wilson. Gait characterisation and classification in horses. *The Journal of Experimental Biology*, 210:187–197, 2007.

[KIJ03] D. C. Kar, K. Kurien Issac, and K. Jayarajan. Gaits and energetics in terrestrial legged locomotion. *Mechanism and Machine Theory*, 38(4):355 – 366, 2003.

[Lun04] Max Lungarella. *Exploring Principles Towards a Developmental Theory of Embodied Artificial Intelligence.* PhD thesis, University of Zurich, Switzerland, 2004.

[RI08] Ludovic Righetti and Auke Jan Ijspeert. Pattern generators with sensory feedback for the control of quadruped locomotion. In *2008 IEEE International Conference on Robotics and Automation*, 2008.

[wik09] Simulated annealing. `http://en.wikipedia.org/wiki/Simulated_annealing`, 05 2009.