Albert-Ludwigs-Universität
Freiburg
Institut Für Informatik

Lehrstuhl für Grundlagen der Künstlichen Intelligenz
Prof. Dr. Bernhard Nebel

Gait Optimization for Humanoid Robots

Master Thesis

Julio César Pastrana Pérez

August 12th 2005

A los que me enseñaron a caminar:
A mis padres

To the ones who taught me how to walk:
My parents

Declaration

I hereby confirm that I have independently composed this Master Thesis and that no other than the indicated aid and sources have been used. This work has not been presented to any other examination board.

Freiburg, August 12th 2005

[illegible]

Contents

Chapter 1

Introduction

Humans learn to walk a few months after their birth. Before they take their first steps, they need to learn how to crawl, and then to stand up. Naturally, this learning process does not finish when executing the first step, it is a process that is constantly evolving until the individual is confident enough to walk over different types of surfaces without falling. After some years, the child has learned by trial and error how to balance, walk, run, etc. and to combine these movements into a flexible yet efficient form of locomotion.

Biped robots or human-like robots have recently become very popular in the scientist community. These types of robots are more likely to successfully function and interact in environments that are designed for humans. For example, human-like robots can mimic the actions of humans and can operate in the same environments without the need of special modifications. Furthermore, legged locomotion facilitates access to many different types of terrains that would be impossible to enter with a wheeled robot (i.e., buildings with stairs, steep inclines, or rough, uneven terrain). Nevertheless, making humanoid robots fully autonomous is far from trivial.

One fundamental problem when working with humanoid robots is to develop stable gaits that allow the robot to walk without falling. Moreover, if one wants the robot to walk at different speeds, the difficulty of controlling such a behavior increases substantially because each speed has its own characteristic set of joint movements. One alternative to achieve this is to perform a hand-tuning for each speed by using physical intuition. However, this process can consume a lot of time and requires a lot of human experience. An other possibility is to use an evolutionary learning technique to find the optimal joint motions for each speed and gait. Using such a technique, enables the robot to “learn” how to walk much in the same way as a child learns, constantly improving its walk as time goes on.

The main goal of this work is to use an evolutionary model, namely genetic algorithms, to optimize the speed coordination and control of a humanoid robot’s gait in order to make it walk in a stable manner as fast as it can. A genetic algorithm is a search technique based on the mechanisms of natural selection and natural genetics, used for the solution of search and optimization problems. At the end of the presented research, the robot should be able to accelerate through a finite number of discrete speeds before reaching a maximum stable speed.

Let us remark that the maximum walking speed is very important for the results of this work because the main contribution is an optimized walking gait that allows changes in speed and that enables a humanoid robot to walk at fast speed.

One immediate application is the RoboCup humanoid soccer league, where speed is a determinant factor in the game. The goal of RoboCup is to build a team of fully autonomous humanoid robots by the year 2050 that is able to defeat the human football soccer champion. This master thesis aims to contribute to the efforts of the scientific community to accomplish this goal.

This work focuses on the following objectives in order to make its contributions:

1. *Develop a gait engine (walking manner), whose parameters have to be optimized.*
2. *Develop the genetic algorithm that encodes the parameters of the gait engine as its genetic material.*
3. *Optimize the parameters of the gait engine using the simulator and afterwards a real humanoid robot.*

After the development of the robot’s gait engine, a hand-tuning is performed in order to generate some parameters that will allow the robot to move forward. This way, the learning process is not going to start from zero, therefore saving time.

To avoid damaging the humanoid robot and consuming human labor by performing several experiments with it, a simulator is used in order to test the learning technique. After the robot performs well in the simulator using the learned parameters, the optimization algorithm is used to optimize the parameters on the real robot.

The remainder of this thesis is structured as follows:

- Chapter 2 introduces Genetic Algorithms. It describes their history, explains how they work and gives a brief example.
- Chapter 3 describes the basics of human locomotion to clarify how humans perform this task.
- Chapter 4 gives an overview over related work, and describes the software and the hardware used during this work.
- Chapter 5 explains in detail the two proposed gait engines.
- Chapter 6 presents the optimization algorithms.
- Chapter 7 presents experimental results of the optimization process with the simulator and with the real robot.

Chapter 2

Introduction to Genetic Algorithms

This chapter gives an introduction to Genetic Algorithms, it clarifies the pertinent vocabulary and also explains the procedure of GAs with a brief example.

2.1 Historical Background

Genetic algorithms (GAs) are inspired by the theory of *"evolution of species by natural selection"* [1], a very well known theory published in 1859 by the naturalist Charles Darwin. In his work Darwin states that all living organisms have evolved through a process of natural selection. He explains how organisms that better adapt to the environment are more likely to survive and to produce descendants with similar or better characteristics that will ensure the endurance of the species.

Darwin suggested that any population has individuals that are slightly different from one another, and that these differences give some advantage to some individuals to survive long enough to be able to reproduce. As a consequence, the traits that help in the survival will pass on to the next generation.

Let us explain the idea of natural selection with a popular example: *"The English Peppered Moth"* [3].

In nature, the English peppered moths have varieties that differ in body and wing coloration. Such color variations go from white to black, although the most common variants are the speckled ones because most of the trees have speckled trunks. The darker moths are camouflaged less in such an environment and are easy prey for the predators. Thus, these moths will not live long enough to reproduce and pass on their traits.

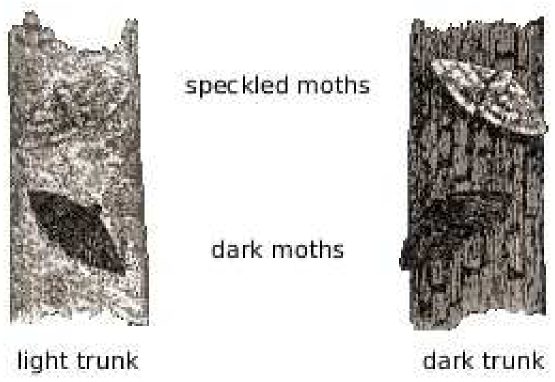


Figure 2.1: Example of environmental adaptation (image from [11]).

In the 1800s, the Industrial Revolution brought to England many industries that were burning coal to power the new inventions. Some cities started having serious problems with air pollution, which covered the trees in a fine black dust. Suddenly the moths that were camouflaged where easy prey and the black ones were able to pass on their traits. There was a moment when the majority of the moths were the black colored ones. After some years when most industries stopped burning coal, the population of speckled moths started growing again and the population of black moths decreased.

After all, it is possible to say that the natural selection is nothing else than the selection of the best adapted individuals to survive and to reproduce. If we want to see it from the genetic algorithms point of view, an adapted individual has a high *fitness value*.

The theory behind genetic algorithms is not limited to Charles Darwin's work. They also include concepts of genetics, such as the reasearch of Gregor Mendel, who is one of the most important pioneers of genetics. In his book "Experiments in Plant Hybridization" [2] he describes his experiments with peas and concluded that the characters or traits from parents pass unmodified to successive generations. He also suggested that the inheritance of each trait is determined by units or factors, now called *genes*.

The following example shows Mendel's hybridization of peas. Furthermore, one can say that this is the way in which genetic algorithms create new generations of individuals.

Let us suppose we have two pure varieties of peas, each one has two characteristics: color and shape. If we cross these varieties to create hybrid individuals, in future generations we expect to have peas with combined characteristics of both parents. Figure 2.2 shows the example.

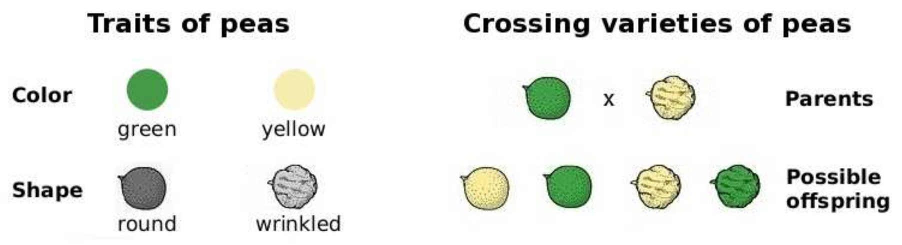


Figure 2.2: Exchanging traits of peas.

As we can see, after different combinations of the parents, the offspring inherit their traits. This process is called *recombination* process or *crossover* process.

On certain occasions, it can happen that new traits appear creating new varieties or completely new species. These changes are named *mutations*, which can lead to malfunction or death of the organism; but when these changes are favorable to the organism, they tend to accumulate over the generations. If one uses the example of Figure 2.2, one expects green and yellow descendants. However, if after the recombination process one sees peas of colors other than green or yellow, a mutation took place.

Charles Darwin's theory of the evolution of species by natural selection together with Gregor Mendel's theory of genetics form the *modern evolu-*

tionary synthesis or *Neo-Darwinism*, which states that a series of change mechanisms work inside the populations in order to optimize the organisms. These change mechanisms (e.g., recombination, selection, and mutation) are used by different search techniques in computer science trying to mimic what nature does but in this case to solve engineering problems.

2.2 Genetic Algorithms

A *Genetic Algorithm* is a heuristic[1] based on the mechanisms of natural selection and natural genetics, used for the solution of search and optimization problems.

Genetic Algorithms were first developed by John Holland at the beginning of 1960s, but it was not until 1975 that Holland made known the first achievements in his publication *"Adaptation in Natural and Artificial System"* [4]. He explains that developing the idea of GAs was not an easy task, because his research was facing two problems: To achieve a deep understanding of natural adaptation process and to find the way of designing artificial systems that mimic nature's evolution process.

Before further explanation of GAs, it is necessary to clarify some of the vocabulary borrowed from natural genetics. *Population*, is a group of organisms of the same species, in which an *individual* is a single organism. The genetic information or *genotype* of an individual is contained in *chromosomes*, which are composed of units called *genes*. These genes are entities encoding the information inherited from the parents, which can take different forms or values called *alleles*. The position of one specific gene in a string or chromosome, is called *locus*.

As an example, Figure 2.3 depicts an individual that has five chromosomes, each chromosome with two genes. These genes can take just binary values 0:1.

In nature an individual is composed by multiple chromosomes, for example, the human DNA has 23 chromosome pairs with an estimated of 3×10^9

[1]A heuristic is a technique of problem solving using exploration and trial and error methods.

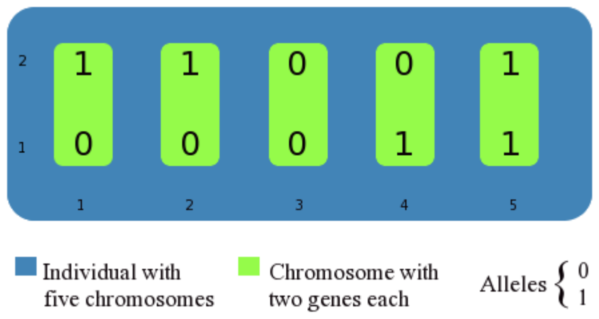

2
1
1 1 0 0 1
0 0 0 1 1
1 2 3 4 5
Individual with five chromosomes
Chromosome with two genes each
Alleles { 0 1

Figure 2.3: Example of an individual.

base pairs (A, C, T and G)[2]. Each chromosome contains many genes, humans are thought to have between 30,000 and 40,000 genes. All this genetic information combined is also called the *genome* of an individual.

The human genome is very complex, therefore full undertanding of it has not yet been achieved. But we should not be scared about the complexity of our DNA. The individuals used in this work are less complicated and the encoding representing them are not bases nor proteins, they are plain numbers.

Usually, genetic algorithms are used to solve problems with a large search space that needs to be explored to obtain an optimal solution. A search space is the set of all possible solutions, where one individual represents a potential solution.

Traditionally, the genome of an individual is coded as a binary string, that means that each gene has two posible alleles: 0 and 1, though this is not a restriction for not using other encodings such as numbers, characters, strings, or other data structures to store the genetic information.

The normal procedure of a GA is iterative, and starts with a population of individuals. This population can be created randomly or can be started with a set of known possible solutions. These individuals are called the *first generation*. The GA evaluates each individual of the population with a fitness function that assigns a score or a fitness value. This score will

[2]A: Adenine, C: Cytosine, T: Thymine and G: Guanine

tell us how well the current element of the population solves the problem. Then, a *new generation* is created using the individuals with the highest fitness values. The offspring are created through means of GA operators e.g., selection, crossover, mutation, which are explained in the following sections. The cycle is repeated until certain criteria are satisfied.

2.2.1 Fitness Function

The name fitness function is synonymous for objective function. Every objective function tries to optimize a function f, understanding optimization as the art of selecting the best alternative among a set of options in the search space $\mathcal{S}$.

$$f : \mathcal{S} \to \mathbb{R}^n, \qquad \mathcal{S} \in \mathbb{R}^n$$

The domain or search space $\mathcal{S}$ of f has all the candidate solutions, where these solutions are a subset of the real numbers $\mathbb{R}^n$.

Any optimization problem tries to find the maxima or minima of real valued functions. Let us note that it is posible to treat all minimization problems as a maximization ones if we follow the next statement:

$$min\ f(x) = max\ g(x) = max\ \{-f(x)\}$$

> "If the optimization problem is to minimize a function f, this is equivalent to maximizing a function g, where $g = -f$" [9].

The fitness function is in charge of giving every individual of a population a fitness value that tells how good it is in solving a given problem.

2.2.2 Selection

This operator is in charge of selecting the fitter elements of the population for subsequent recombination or crossover.

There are several ways of making the selection, just to mention some of them: Fitness proportionate selection [4], Boltzmann selection [7], tournament selection [6], rank selection [9], steady state selection [9] and others. Nevertheless, one of the most popular methods, and the one used in this work, is the *roulette wheel selection* also called fitness proportionate selection.

To implement this type of selection the following steps are needed:

1. Evaluate each individual k_i ($i = 1, \ldots, n$) of the population with the fitness function.
$$f_i = evaluate(k_i)$$

2. Calculate the probability of reproduction p_i for every individual.
$$p_i = \frac{f_i}{\sum_{i=1}^{n} f_i}$$

3. Calcualte a cumulative probability q_i for each individual k_i ($i = 1, \ldots, n$).
$$q_i = \sum_{j=1}^{i} p_j \qquad q_0 = 0$$
After creating the roulette wheel, one needs to spin it to select the parents that would exchange their genetic information. This spinning process is simulated as follows:

4. Generate a (float) random number r in the range $[0, 1]$.

5. If r is in the range $(q_{i-1}, q_i]$ select individual k_i.

To illustrate the above procedure, let us think about a population of four individuals. Table 2.1 shows their fitness value, their probability of reproduction and their cumulative probability.

	Individuals			
	k_1	k_2	k_3	k_4
f_i	5	20	10	25
p_i	0.08	0.33	0.17	0.42
q_i	0.08	0.41	0.58	1

Table 2.1: Example of roulette wheel selection method

Figure 2.4 shows the the roulette wheel created with the cumulative probabilities of Table 2.1. It is possible to see that the individuals with higher fitness values get bigger slices of the wheel, therefore they have better chances of being selected.

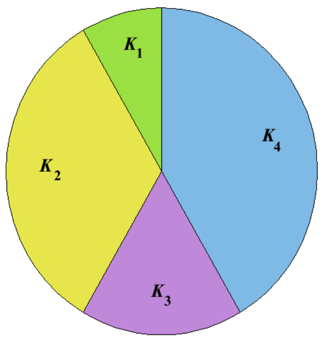


Figure 2.4: Roulette wheel

With this selection method, since the parents must be selected in pairs in order to exchange thir trairs, it is likely that some individuals are selected more than once. Of course, it is more probable that these individuals are the ones with high fitness values. But what happens if the same individual is selected twice before they mate in order to produce an offspring. Clearly, it is futile trying to produce a new individual when the two parents are equal becuse the resulting offspring will be equal when no mutation is present. In order to solve this problem, one can select a new couple of parents.

In the selection process there is an implicit elitism when selecting the best elements of a population and replacing the worst elements. This has the desired effect of allowing the generations to improve over time. However, a drawback is the that the individuals of a population tend to be very similar.

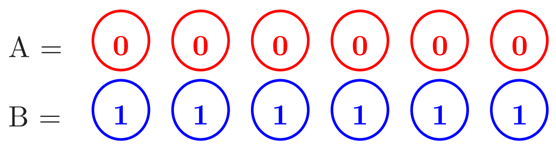
2.2.3 Crossover

This operator tries to mimic the biological recombination of two organisms. It is used to create new offspring by combining or varying the genomes of two parents. Section 2.1 mentioned a combination between two varieties of peas, see Figure 2.2. The idea behind this operator is very similar to the one of Mendel's experiments.

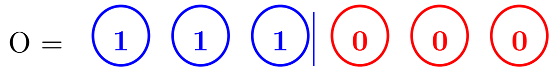
Several individuals with desired characteristics are selected and put into a so-called *mating pool*, where two individuals join to form parents in order to exchange their genetic material. The recombination of the genetic information between two indiviuals takes place after the selection process because it is desired that the best individuals of a population enter the mating pool.

There are several crossover techniques for organisms with different encodings of the genetic material. However, the idea of recombination can be applied independently of the encoding. To mention some: single point crossover, two point crossover, uniform crossover, etc.

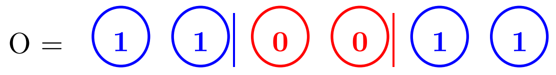
As an example, let us pick two parents A and B. The encoding of the genetic material is binary, and each parent has six genes.



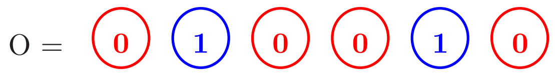
The *single point crossover* randomly chooses a locus and exchanges the genetic material before and after this point. Continuing with the example, let us suppose that the random point is 3. That means that the first three genes of parent B and the last three genes of parent A will be taken in order to form an offspring O.



The *two point crossover* does the same as the single point crossover, but in this case two points are randomly chosen. Now, if these two points are 2 and 4, the descendant of the example looks as follows:



The *uniform crossover* randomly selects from the genes of the parents. It is possible to say that each gene of the offspring is either from parent A or B, with 50% probability.



Usually, when implementing this operator, a crossover probability p_c is needed as a parameter. The crossover probability dictates what percentage of the population will be chosen to enter into the mating pool and exchange their genes to create a new generation. The part of the population that entered into the mating pool will subsequently be replaced by their offspring. The other part of the population that was not selected continues unchanged into the next generation. For example, when the crossover probability p_c is set to 0.80 then only 80% of the population is allowed to enter into the mating pool, where they will produce offspring. These offspring take the place of their parents in the new generation. The other 20% that was not allowed to produce offspring passes unchanged into the next generation, where they again have the possibility of being selected and sharing their genetic infomation.

2.2.4 Mutation

When using GAs, the individuals of a population tend to be very similar after some generations. This has the effect of stopping the evolution process or making it very slow. To prevent this, the mutation operator is used, ensuring genetic diversity.

The mutation operator has also the effect of exploring areas of the search space that have not been yet explored. This help us avoiding local minima.

One easy way of implementing a mutation is to generate a random (float) number r between [0,1] for each gene of every individual. If $r \leq p_m$, where p_m is the mutation probability, then the gene is altered by changing its value. For example, if the genetic coding is binary, 0 becomes 1 and 1 becomes 0.

2.2.5 Elitism

The basic genetic algorithm does not have the elitism operator. Nevertheless, the necessity of using it arises because the selection operator does not ensure selecting the best elements of each generation.

By introducing the elitist operator one treats the best element of the population better than the less fit elements, making sure that it will be present in future generations. Usually this operator is implemented by copying the best element into a separate structure. It will then replace the worst element of a generation whenever the current generation does not have a fitter element than the previews one.

Elitism increases the performance of a genetic algorithm because it prevents the loss of the best found solution.

2.2.6 GA Procedure

After describing the basic genetic operators, it is possible to formally define the procedure of a genetic algorithm:

Genetic Algorithm:

```
Input: A fitness function $f$
Output: An individual $x$ that maximizes the fitness function $f$
    $g \leftarrow 0$    $g$ = generation counter
    $g_{max} \leftarrow$ maximum number of generations
    initialize $P(g) \leftarrow \{x_1^g, \ldots, x_n^g\}$    $n$ = population size
                                                          $x_i^g$ = individual of the population $P(g)$
    evaluate $P(g)$
    while $g \leq g_{max}$ do
        $g \leftarrow g + 1$
        select new population $P(g)$ from $P(g-1)$
        crossover elements of $P(g)$
        mutate elements of $P(g)$
        evaluate $P(g)$
        elitism in the population $P(g)$
    end while
    determine the best $x_i$
    return $x_i$
```

The above algorithm starts by initializing the first population of individuals. This population $P(g)$ with n amount of individuals becomes the generation zero that has to be evaluated before entering the process that will generate a new generation. The evaluation process consists of asigning the fitness values to each member of the population (see Section 2.2.1). After assigning the fitness values, a new generation is created by means of the the five aforementioned genetic operators: select, crossover, mutate, evaluate and elitism (see Sections 2.2.2 - 2.2.5). This process is executed until the desired amount of generations is reached g_{max}.

There are some similar paradigms of evolutionary computation (e.g., evolution strategies, evolutionary programming, genetic programming), and it is not always easy to define the differences between GAs and other evolution approaches. Mitchell wrote: [8]

> "It turns out that there is no rigorous definition of genetic algorithm accepted by all in the evolutionary-computation community that differentiates GAs from other evolutionary computation methods. However, it can be said that most methods called GAs have at least the following elements in common: population of chromosomes, selection according to fitness, crossover to produce new offspring and random mutation of new offspring"

2.3 GA Example

This section describes the evolution process of genetic algorithms by means of an example. A very simple optimization problem is proposed, where the goal is to maximize a function $f(x)$ within the bounds [-3,3].

For the sake of explaining the procedure of the algorithm, a simple function is used $f(x) = \dfrac{\cos x^2}{1+x^2}$, where one can easily see that $arg\ \underset{x\in\mathbb{R}}{max}\ f(x) = 0$ and the global maximum is 1. This function is depicted in Figure 2.5.

First of all, it is necessary to define the encoding that will be used to represent the genetic information. A binary vector is used to represent the value of the variable x. Now, the question is, what should be the length of

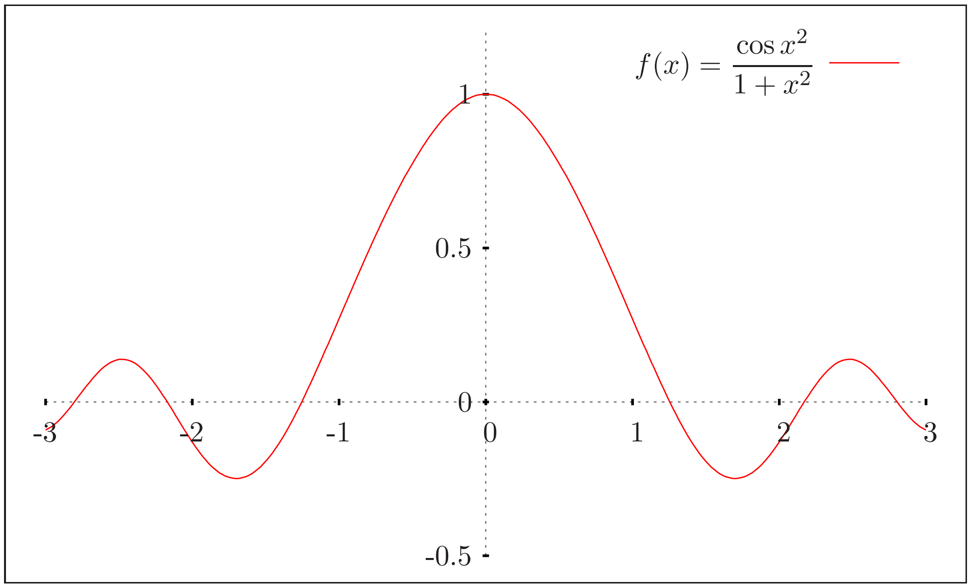


Figure 2.5: Example of optimization problem

the vector? This length depends on the desired precision, which is for this example 3 decimal places after the point. This means that we can represent 10^3 real numbers within an unitary interval. The interval [-3, 3] has a length of 6, therefore it has to be divided into at least $6 \cdot 10^3$ equal parts, in order to maintain the desired precision.

The length k of the binary vector is straightforward to calculate if we look at the next interval: $2^{k-1} < 6000 \leq 2^k$. This means that a 13 bit binary vector is sufficient.

$$\begin{array}{rccccccccccccc} & a_{12} & a_{11} & a_{10} & a_9 & a_8 & a_7 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 & a_0 \\ v_1 = & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ v_2 = & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array}$$

Vectors v_1 and v_2 represent the smallest and the largest values of the selected coding respectively. The mapping from a vector v to a real value x in the rage [-3,3] is done in two steps:

1. Convert binary vector from base$_2$ to base$_{10}$

$$(v)_{10} = \sum_{i=0}^{k-1} a_i \cdot 2^i = (\overbrace{a_{k-1}, \ldots, a_0}^{v})_2$$

2. Find the corresponding real value x

$$x = \mathit{lower\,bound} + (v)_{10} \cdot \frac{l}{2^k - 1}\ , \qquad (l = \text{interval length})$$

When applying this process to the vectors v_1 and v_2, we find that they represent the bounds [-3,3]:

$$x_1 = -3 + 0 \cdot \frac{6}{2^{13}-1} = -3\ , \qquad x_2 = -3 + 8191 \cdot \frac{6}{2^{13}-1} = 3$$

Evaluating a real value x_i is the same as evaluating its corresponding vector v_i. Naturally, the two steps described above are used to find the corresponding real value. For this reason, the function $eval(v_i)$ will be used as the equivalent of $f(x_i)$ shown in Figure 2.5.

To start the evolution process, the population size has to be defined, as well as the crossover probability p_c and the mutation probability p_m. Let us assume:

$$\mathit{population\ size} = 6, \quad p_c = 0.7, \quad p_m = 0.015$$

The initialization of the first generation consists of randomly creating six individuals, where the real values are: $x_1 = -2.5$, $x_2 = -1.8$, $x_3 = -1.0$, $x_4 = 1.0$, $x_5 = 0.5$ and $x_6 = 2.0$; and their corresponding binary vectors are the following ones:

$$\begin{array}{lccccccccccccc}
v_1^0 = & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
v_2^0 = & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
v_3^0 = & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
v_4^0 = & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
v_5^0 = & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\
v_6^0 = & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1
\end{array}$$

During the first evaluation phase, the fitness values of the vectors are:

$$\begin{array}{llcll}
eval(v_1^0) = & 0.0885284 & & eval(v_4^0) = & 0.1459633 \\
eval(v_2^0) = & 0.0121747 & & eval(v_5^0) = & 0.6161209 \\
eval(v_3^0) = & 0.1459633 & & eval(v_6^0) = & 0.0346356
\end{array}$$

The fitness of one individual can be negative. To avoid this problem, one can use offsets, thresholds or simply arbitrarily assigning all negative fitness values to 0. This process is called *fitness scaling*.

The next step is creating a new generation with the genetic operators *selection, crossover and mutation*. Clearly, the individual v_5 will have the best chances of being selected, since its fitness value is the best of the whole generation. But let us go through the process of building the roulette wheel. To do so, we need to calculate the fitness of the population, the probability of reproduction p_i and the cumulative probability q_i for each individual.

The fitness of the population is:

$$total\ fitness = \sum_{i=1}^{6} eval(v_i) = 1.0433862$$

The probability of reproduction for each individual is:

$$p_i = \frac{eval(v_i)}{total\ fitness}$$

The q_i values are the cumulative sum of the p_i values.

$$q_i = \sum_{j=1}^{6} p_j$$

Table 2.2 summarizes the computation of the p_i and q_i values.

	v_1^0	v_2^0	v_3^0	v_4^0	v_5^0	v_6^0
p_i	0.0848472	0.0116685	0.1398938	0.1398938	0.5905013	0.0331954
q_i	0.0848472	0.0965157	0.2364095	0.3763033	0.9668046	1

Table 2.2: Calculated p_i and q_i values.

The last step of the selection process is to spin the wheel. This is simulated by randomly generating float numbers within the range [0,1], and checking if they are in the interval $q_{i-1} < random\ number \leq q_i$. Below are listed the random numbers, the interval they belong to, and the selected individual.

random number	*interval*	*selected individual*
0.6653	(0.3763033, 0.9668046]	v_5^0
0.3493	(0.2364095, 0.3763033]	v_4^0
0.0683	[0, 0.0848472]	v_1^0
0.5308	(0.3763033, 0.9668046]	v_5^0
0.7263	(0.3763033, 0.9668046]	v_5^0
0.1985	(0.0965157, 0.2364095]	v_3^0

Table 2.3: Selection process.

The selected individuals of the population are the following ones:

$$\begin{array}{rccccccccccccc}
v_5^0 = & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\
v_4^0 = & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
v_1^0 = & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
v_5^0 = & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\
v_5^0 = & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\
v_3^0 = & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0
\end{array}$$

After the selection of the individuals, the following step is to select among these elements, the ones that will go into the mating pool. The probability of crossover $p_c = 0.7$, this means that 70% of the elements of the population will have the opportunity of sharing and propagating their genetic material. In order to select the parents, a random number r within the interval [0,1], is generated for each member of the population. If $r < p_c$ the current individual is selected for crossover. The results are shown in Table 2.4.

random number	$r<0.7$?	*selected parent*
0.4532	*Yes*	v_5^0
0.6971	*Yes*	v_4^0
0.7442	*No*	
0.9329	*No*	
0.2320	*Yes*	v_5^0
0.1280	*Yes*	v_3^0

Table 2.4: Parents selected to mate.

Two couples were randomly selected using the roulette wheel method previously described. The selected parents are (v_5^0,v_4^0) and (v_5^0,v_3^0).

Recombination is made using the single point crossover. In this example, two offspring are created from two selected parent couples, but the same process aplies when creating one offspring from one parent couple.

A random locus is generated for the first couple: locus = 3.

$$\begin{array}{rccc|cccccccccc}
v_5^0 = & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\
v_4^0 = & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1
\end{array}$$

The offspring are created:

$$\begin{array}{rccccccccccccc}
v_1^1 = & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
v_2^1 = & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1
\end{array}$$

A random locus is generated for the second couple: locus = 8.

$$\begin{array}{rcccccccc|ccccc}
v_5 = & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\
v_3 = & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0
\end{array}$$

The offspring are created:

$$\begin{array}{rccccccccccccc}
v_5^1 = & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
v_6^1 = & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1
\end{array}$$

After the crossover, the population is:

$$\begin{array}{rccccccccccccc}
v_1^1 = & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
v_2^1 = & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\
v_3^1 = & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
v_4^1 = & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\
v_5^1 = & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
v_6^1 = & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1
\end{array}$$

The two individuals printed in black (v_3^1, v_4^1) are the individuals from the previous generation that were not selected to mate. They are carried over into the next generation without any change.

The final step is mutation. This operator carries out a gene by gene operation; generating for each gene a random number r within the range [0,1] . If $r \leq p_m$ a mutation takes place. In the example, the population size is 6 and the length of each vector is 13, thus the total number of genes in the population is $6 \times 13 = 78$. The probability of mutation is $p_m = 0.03$, meaning that the number of genes to be mutated (on average) is $0.03 \times 78 = 2.34$. The mutation of a binary number is executed by flipping the value of the selected gene; if the value is 0 we change it to 1 and vice versa.

After running the mutation operator, the following genes were mutated:

random number	$r<0.03$?	*selected gene*
0.002211	*Yes*	35
0.011383	*Yes*	43

Table 2.5: Selected genes for mutation

The final population is shown below, with the mutated bits placed in a circle.

$$\begin{array}{lcccccccccccccc}
v_1^1 = & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
v_2^1 = & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\
v_3^1 = & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & \textcircled{1} & 1 & 0 & 1 & 0 \\
v_4^1 = & 1 & 0 & 0 & \textcircled{0} & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\
v_5^1 = & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
v_6^1 = & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1
\end{array}$$

At this point, we completed the first iteration of the loop shown in the genetic algorithm. To finish the example let us evaluate the elements of the above population:

evaluated value	*real value*
$eval(v_1^1)$=0.8835683	x_1=0.250
$eval(v_2^1)$=0.0388012	x_2=1.250
$eval(v_3^1)$=0.0876595	x_3=−2.488
$eval(v_4^1)$=0.9693107	x_4=0.125
$eval(v_5^1)$=0.6172870	x_5=0.499
$eval(v_6^1)$=0.1465646	x_6=−0.999

Table 2.6: Evaluation of the generation 1

It is possible to see that the genetic algorithm has found better individuals in this new generation because their fitness values are getting closer to the maximum. However, this behavior is not constant, it can be the case that a new generation is worse than the prior one.

Summary

This chapter serves to introduce the basics of the genetic algorithms used during this work. First of all, a brief history of GAs was presented, followed by an explanation of the vocabulary used in the context of GAs.

A genetic algorithm was presented and all its functions were clarified, i.e. selection, crossover, fitness evaluation, mutation and elitism. Moreover, a detailed example was presented in order to clarify the GAs.

Chapter 3

Human Locomotion

In order to make a humanoid robot walk, it would be necessary to understand how humans perform this action. This chapter introduces the basic concepts of human locomotion, as well as some machine models that try to undstand the dynamics of human walking.

3.1 Bases of Human Locomotion

The study of human walking began when medicine practitioners started describing problems related to patients with various types of walking impediments. This research started analyzing normal individuals, with the aim of identifying certain parameters that would make it easier to diagnose the patients. After some time, a specific terminology started developing, enabling communication within the medical community.

First of all, it is essential to define the meaning of *walking*. "Walking is the main form of animal locomotion on land" [12], which humans carry out with the two lower extremities and the pelvis, also called the locomotor apparatus. When walking, "the primary task of the locomotor apparatus is to transport the head, arms and trunk" [13]. For this reason, we need to introduce the term *gait*, which is the "particular way or manner of moving on foot" [12]. For bipedal walking, the two basic gaits are walking and running. The main difference between the two gaits is that when running only one foot is in contact with ground at any time. When walking, a *double* *stance phase* takes place, which means that two feet are touching the ground during part of the gait cycle.

Executing a gait is to repeat a sequence of movements for the time that one is moving. Repeating these movements is called the *gait cycle*, which is defined between any two identical events in the walking cycle. Usually, the gait cycle is divided into two main periods, *stance* and *swing*. Stance is the period of time when the foot is in contact with the ground. Swing is the period time when the foot is in the air.

The stance phase is further divided into two types: double support and single support. The double support, properly called double-limb stance, is the time when both feet are touching the ground; this happens at the beginning and at the end of the stance phase. The single support, also called single-limb stance, is when one leg is supporting all the weight of the body, while the other leg is in its swing phase. The stance phase represents 60% of the time of the gait cycle and the other 40% belongs to the swing phase.

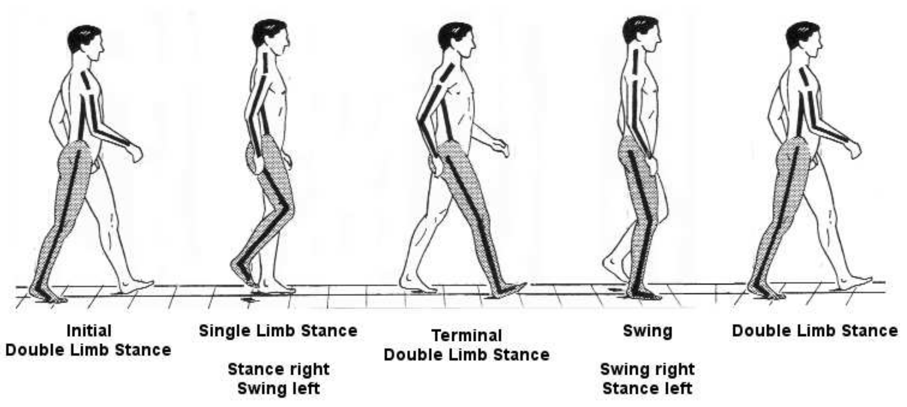


Figure 3.1: Walking gait cycle (image from [17])

Figure 3.1 shows a complete walking gait cycle. One can see that to produce the forward movement, the body weight has to be carried alternately on one leg, while the other performs the swing.

3.2 Gait Sub-phases

Even though a gait cycle can be described with two general phases, it can be further divided into eight sub-phases: initial contact, loading response, mid-stance, terminal stance, pre-swing, initial swing, mid-swing, and terminal swing (images from [13]).

1. Initial contact: It is the beginning of the cycle, when the foot of the limb that was in its swing phase touches the ground with the heel. It is also the start of the stance phase.

2. Loading response: After the heel struck the floor, the leg is absorbing the shock and getting ready to receive the body weight.

3. Mid-stance: Starts when the single limb phase starts, in other words, when the contralateral foot leaves the ground. It finishes when the frontal part of the swinging foot is aligned with the stance leg. It represents 50% of the single limb stance.

4. Terminal Stance: It completes the other 50% of the single limb stance. It finishes when the heel of the stance leg rises and the foot that was on the air touches the ground with the heel.

5. Pre-swing: This phase is starts the second double limb support of the gait. It begins with the heel strikes the floor and it ends when the toes leave the ground.

6. Initial swing: Starts right after the toes left the contact with the ground; it ends when the knee reaches the maximum flexion.

7. Mid-swing: Starts when the knee reaches its maximum flexion; it ends when the swinging leg is ahead of the body and its tibia is perpendicular to the ground.

8. Terminal swing: It is the last phase of the cycle. It starts after the leg was in vertical position, and ends when the heel touches the ground.

These eight phases describe how humans perform one cycle of the walking gait. However, there are other characteristics, such as the condition of the musculoskeletal structures, the nervous system, the ligaments, etc. that influence the performance and energy consumption.

3.3 Gait Classifications

During human walking, the center of mass (COM) is in constant movement. It translates in the direction of the body and also moves in lateral and vertical direction. As a consequence, the COM is most of the time outside the *base of support* [1]. One might think that this constant movement leads to an extreme waste of energy, nevertheless, human walking has evolved over millions of years reaching a high level of energy efficiency.

Bipedal locomotion requires a great amount of synchronization and coordination between the different muscles, ligaments, nerves, tendons, etc. to trigger the right movements at the right time. This is not an easy task, even though humans or other animals with this type of locomotion execute it effortlessly.

There are two main gait classifications, specifically static and dynamic gaits. The static gaits never let the center of mass outside the base of support, while dynamic gaits allow the center of mass to leave the base of support for a small period of time.

Static gaits are known to be the simplest and the most stable types of locomotion. Nevertheless, they tend to be slow. As afore said, the stability of the static gaits is due to the center of mass never leaving the support polygon. This means that if one stops the execution of the gait cycle at any time, there is no danger of falling over.

On the other hand, dynamic gaits are much more complex and harder to control. Humans seem to use a very efficient control model to balance when standing upright and when walking over any surface. One can note this when trying to stand straight and still, at all times we are correcting our balance.

[1]Also known as the polygon of support. The base of support is the polygon "formed by the limbs in order to maintain a balanced system and avoid tipping" [19].

Humans are said to walk dynamically stable, this stability is reached by constantly adjusting the position of the joints, muscles and other parts of the body.

When modeling human walking it is often compared to the motion of two coupled pendula. McGeer [20] concluded that the behavior of the leg performing the stance phase is analog to an inverted pendulum, and the leg in the swing phase is a normal pendulum attached at the hip, see Figure 3.2.

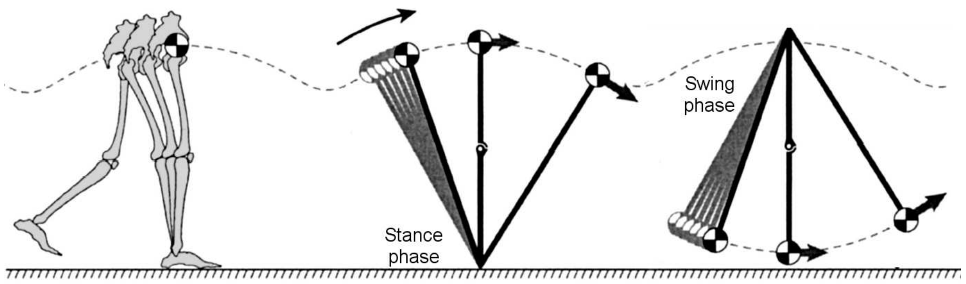


Figure 3.2: Pendulum analogy (image from [22]).

To prove the above statements McGeer designed a so-called passive walker, which walks with no energy input but the gravitational force created by a slope. The most simple passive walker is in fact a double pendulum with no extra joints. In further research McGeer [21] added knee joints to provide natural ground clearance, showing that the mechanism still works with the new feature (Figure 3.3). These bipedal systems have been studied with the intention of fully understanding the dynamics of human walking.

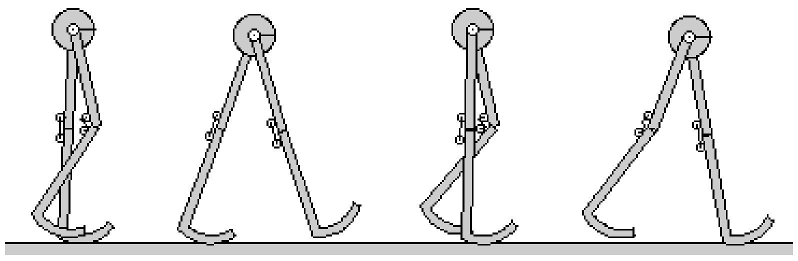


Figure 3.3: Passive walker with knees (image from[23])

Definitively designing and controlling a dynamic system that mimics the human gait is a difficult problem. The scientific community has not been able to define the specific criteria that would tell whether a system is dynamically stable or not. However, there is one approach besides the the inverted pendulum method - the so-called *Zero Moment Point* (ZMP) - originally proposed by Vukobratovic [24]. This approach is considered the state of the art in walking control, allowing the design of machines that can be called dynamically stable.

The following are definitions of the Zero Moment Point (ZMP) [26]:

- "The ZMP (Zero Moment Point) is defined to be a point on the ground at which the tangential component of the moment generated by the ground reaction force/moment becomes zero".

- "A humanoid robot can walk on a flat ground with keeping the dynamical balance if the ZMP is included in the convex hull of the foot supporting area."

In order to clarify the idea of the ZMP, we will use a classic problem in control theory *The cart-table problem* [25]. Suppose that we have a pedestal table whose mass is insignificant, and a cart of mass m that will be running on the table. See Figure 3.4.

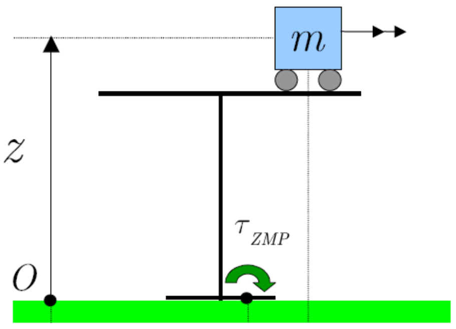


Figure 3.4: Cart-table model (image from [25])

The problem comes when the cart tries to go to the edges of the table. Let us think for a moment that the cart goes to one of the edges and stays there, obviously the table and the cart will fall because the cart is too heavy and the base of support of the table is too small. Now, let us think that the cart goes to the edge of the table with the proper speed and acceleration, stays there for some time and then it goes back to the center of the table. In this situation, it is possible that the table keeps the vertical position for some time. It is in this moment when we say that there is a ZMP at the base of the table.

The last example is an analogy for the foot of a bipedal walker during the stance phase, where the mass of the cart is replaced by the center of mass of the body. Figure 3.5.

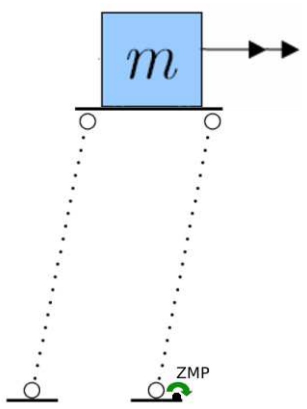


Figure 3.5: ZMP at the robot's foot

Summary

The aim of this chapter was to present some concepts of human locomotion. The phases of the human gait were introduced (i.e. stance phase, swing phase). The sub-phases of each phase were also described.

This chapter also clarified other important concepts such as the Static gait and the Dynamic gait. During this work, dynamic gaits are used in order to allow the robot to walk as fast as possible.

Chapter 4

General Setup

This chapter introduces the related work and also describes the hardware and software used during this work.

4.1 Related Work

Much research has been done on bipedal locomotion. However, most of this research focuses in trying to find better models to unerstand this type of locomotion [34, 35, 36, 37, 20]. The robots used in this research are basicly passive dynamic robots that move using the gravitational energy created by slopes. The simplicity of these type of robots support the idea that there is no need to use complicated control strategies in order to produce the bipedal walk.

> "The passive dynamic robots research aims to learn the roles of nerves and muscles in animals, and computer and motors in robots, by learning what can be done without them".

The use of physical simulations allow to test different types of learning techniques without using the real hardware. Sellers, et al. [38] use an evolutionary technique, namely a genetic algorithm, to generate walking and running gates with the help of a simulator. The evolution starts with chimpanzee-like gaits and finished with human-like gaits. It also compares the energetic costs of each gait.

Roberts, et al. [39] use also a physical simulator and a GA to tune the open-loop controlers of the gait and joints of a robot called ”GuRoo”. During this work most of the atention was focused on the smotheness and stability of the motions. Two genetic algorithms and a hand-tuned method were developed in order to optimized the motion of 15 joints. After the optimization process, the two GAs performed better than the hand-tuned method.

Wolff and Nordin [40] presented a very similar work, where the idea is to evolve a *gait control program* on a simulated robot for posterior evolution on a robot called ”Elvina”. The evolution process started with random individuals. Each individual is a list of instructions encoded as integers, with a maximum length of 256 instructions. After several generations of evolution with the simulator, the best individuals found in the process are taken for further evolution with the real robot. The final results indiacte that the robot was only able to walk staticly.

Röfer [41] describe a genetic algorithm approach to optimize the parameters of a gait. This is very similar to what this thesis presents. However, the gait engine used during their experiments is a quadrupedal gait (PWalk-style, see Figure 4.1) for the Sony Aibo robots. The final results tell that the maximum speed of the robot is 33.1cm/s, the maximum speed known for this type of robots.

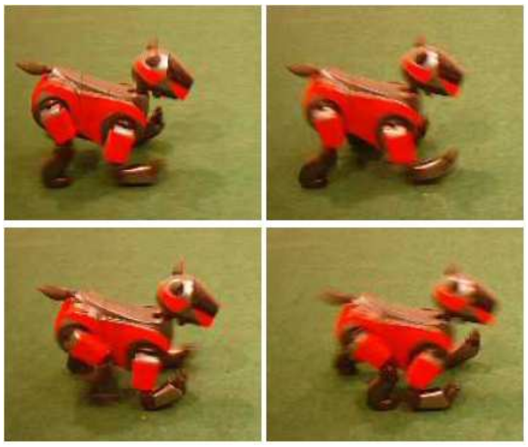


Figure 4.1: Aibo robot using the PWalk-style (Image from [41])

Kohl and Stone [42] propose a policy gradient reinforcement learning to optimize the parameters of a quadrupedal gait. The goal of this work is also to optimize parameters that will lead to the fastest possible walk. The robots learned during 3 hours with no human intervention, at the end of the optimization process the Aibo robot was able to walk at $291 cm/s$.

4.2 Hardware Description

This section describes the humanoid robot used to carry out the experiments in this thesis.

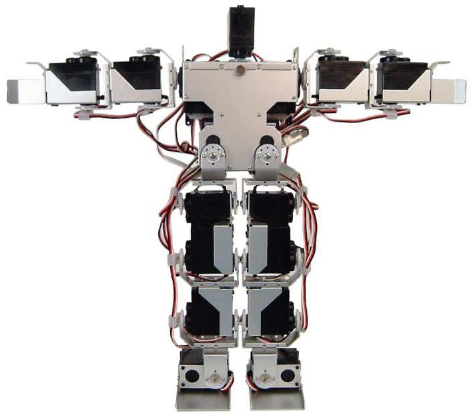


Figure 4.2: KHR-1 humanoid robot - original configuration.

The Humanoid Robot KHR-1 is an assembly kit manufactured by the Japanese company Kondo. It has 17 degrees of freedom (DOF), where each degree of freedom is represented by a servomotor.

The locomotor apparatus is composed of 10 DOF (each leg has 5 DOF), each arm has 3 DOF and 1 DOF for the head. The height of the robot is 34cm and its total weight is $1.2kg$.

To enable the control of the servo motors, the robot has two RCB-1 electronic boards, each of which can control up to 12 servos. These electronic modules have also a serial interface (RS-232) that enables the communication with a PC. The power source of the robot is an Ni-Cd battery.

4.2.1 Hardware Additions and Modifications

In order to make the robot fully autonomous it needs to carry its own computer. Due to the size of the robot KHR-1, a lightweight device with reasonable computational power was selected. This device is a Fujitsu Siemens Pocket Loox 720, whose total weight is $170gr$.

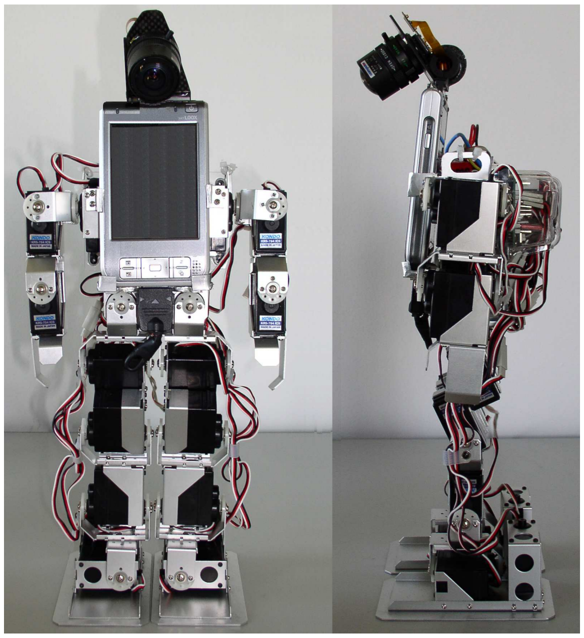


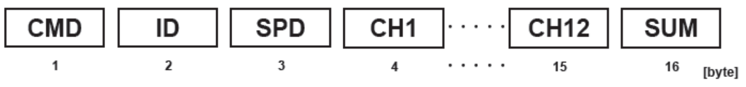
Figure 4.3: KHR-1 After some hardware modifications
The total weight of the robot is $1.55kg$

To keep track of the environment a FlyCam-CF camera was added, which has a special lens that allows a wider field of view. Additionally, other two servo motors were attached to the feet, with the purpose of giving the ability of omnidirectional walking.

The head DOF was not needed and so we removed it. Figure 4.2 shows the robot as it looks after the modifications.

4.2.2 Communicating with the Robot

As aforementioned, the communication with the robot is established by means of a RS-232 serial interface. The RCB-1 control modules have predefined commands that enables the manipulation of each servo motor. Each of the two control boards is in charge of one part of the body, one for the upper body and one for the lower body. The structure of the commands to manipulate the servos is as follows:



- CMD indicates the command to use. In the case of manipulating the positions of the servos it is FD_{hex}.
- ID is associated with the identifier of the RCB-1 board. The module that controls the upper body has an ID=0_{hex} and the one that controls the lower body has an ID=1_{hex}.
- SPD is the desired speed at which the servos will move. The range of this variable is $[0_{hex}\ ,\ 7_{hex}]$. We set the speed to 0_{hex}, which is the fastest one.
- CH1 to CH12 indicate the desired positions of the servo motors. The range is 0 - 180 degrees. See Figure 4.4.
- SUM is the checksum of the packet.

The frequency at which the packets are sent to the robot is $50Hz$. At higher frequency rates, it was noticed that the control boards get confused, causing the servo motors to execute unexpected movements. The $20ms$ between each data packet, are further divided into $10ms$ periods to separate the packets addressed to the upper and lower body boards.

It is essential to mention that the only feedback from the robot is an acknowledgment packet communicating the reception of a packet. Therefore, after sending a request, we expected the robot to do as indicated without any guaranties. The servo motors that the robot had at the time of the experiments were of the type that are not able to give feedback of their current position.

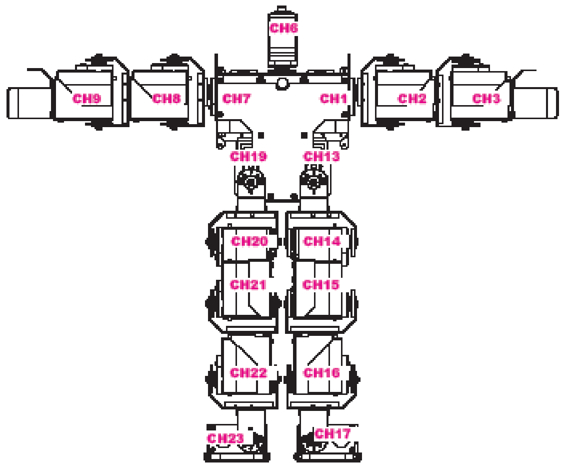


Figure 4.4: KHR-1 Servo-Channel association.

4.2.3 Motion Convention for the Joints

When the robot is turned on, it starts at a so-called *home position or zero position*, which is the starting position of each servo motor when beginning an operation. The home position of KHR-1 is an upright position, as shown in Figure 4.3. To see a full table with the zero position of each servo, see the appendix.

The estimation of a new target position is calculated relative to the zero position of each servo. Additionally, a motion convention for the servo joints was defined to be consistent with the mirrored movements of the body, and also to be able to create behaviors that both legs can use.

This convention has positive and and negative motions for each joint. Figure 4.5 shows the positive trajectories of each servo of the locomotor apparatus. The negative motions are in counter directions to the ones showed.

For example, if one wants the robot to extend the legs at right angles to the trunk, the hip joints (CH13 and CH19) have to move 90 degrees in positive the direction.

The movements of the upper body joints were defined in a similar way, however, they are not used during the experiments.

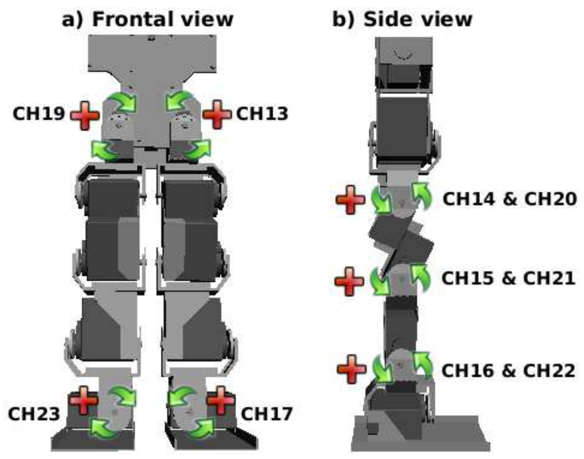


Figure 4.5: Motion Convention for Servo Joints

4.3 Software Description

4.3.1 The Simulator

One of the main disadvantages of the evolutionary algorithms is the number of iterations they need to do in order to find good solutions. They are also intense with the computational resources when operating.

It was clear that it is very difficult or impossible to run enough experiments in the real robot so that a genetic algorithm can sufficiently optimize the parameters of a gait engine. Therefore, a simulator was used. This simulator is a virtual reality environment created with the Open Dynamics Engine (ODE) [27] and the OpenGL API [28].

Every single part of the robot is simulated by a rigid body with the shape and weight of the real part. These parts are interconnected by joints that simulate the behavior of the servo motors (see Figure 4.6).

The simulator gives the opportunity of working with a fully functional virtual KHR-1 robot, without having to be concerned about damaging the hardware. It also allows a greater number of experiments, without human labor.

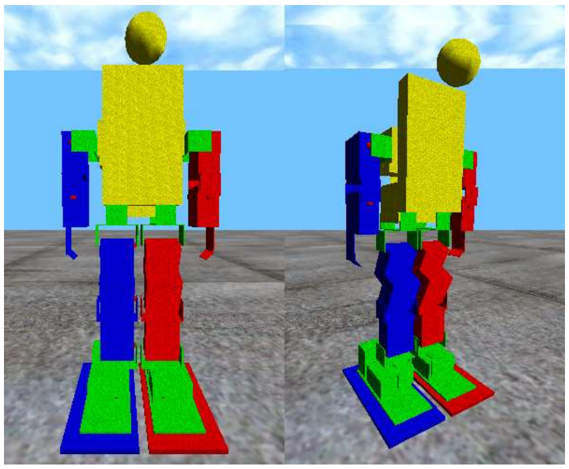


Figure 4.6: Simulation of KHR-1.

4.3.2 Behavior Control

A hierarchical architecture [29] was used to control the robot. This architecture consists of various levels with different behaviors. The complexity of these behaviors increases when going up in this hierarchy, however, their frequency of execution decreases. This means that the fastest and least complex behaviors stay at the lowest part of the hierarchy.

Having a layered structure permits the upper layers to use the lower ones to increase their level of complexity. For instance, one can think of a behavior at a lower layer that interacts with each joint of a body part. Then an upper layer can use this behavior to interact with the body part instead of just interacting with individual joints. If one goes one level up, one can create behaviors that communicate with body parts, thus, allowing the creation of behaviors that include different parts of the body.

The communication between layers is accomplished by means of sensors and actuators, each of which is connected just to one layer. These sensors and actuators are updated every time their corresponding layer performs its duties.

For example, Layer 0 used during this work has sensors for each joint of the leg. Layer 1 has sensors such as leg angle and leg extension. Figure 4.7 depicts the sensors of each layer.

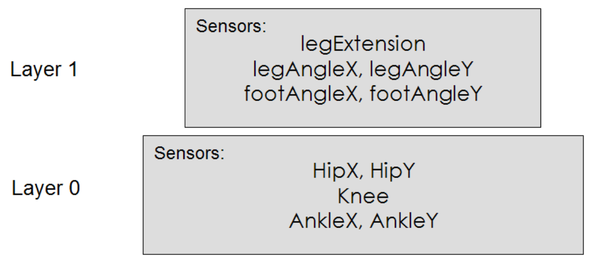


Figure 4.7: Sensors in each layer for each leg

The sensors that have the X refer to the ones that take care of the motion in the frontal or lateral plane. The sensors with a Y refer to the motions in the sagittal plane. See Figure 8.1 in appendix (page 113) to identify the motion planes.

The HipX sensors are directly related to the position of the servos CH13 and CH19. The HipY sensors are connected to the servos CH14 and CH20. The Knee sensors are associated to the servos CH15 and CH21. The AnkleY sensors are related to the servos CH16 and CH22. The AnkleX sensors are associated to the servos CH17 and CH23. See Figure 4.5.

The legExtension actuators control the extension of the legs when walking. The legAngleY sensors control the pendular motions of the legs during the gait cycle, these motions are performed in the sagittal plane. The legAngleX sensors control the motion of the legs in the frontal plane. The actuators footAngleX and footAngleY help keep the feet plates parallel to the hip.

Section 4.2.2 mentions that new positions are sent to the robot every $50Hz$, meaning that layer 0, placed at the bottom of the control architecture, runs at the same speed to maintain the generation of the target positions. The layer on top of layer 0 runs at $25Hz$. Layer 0 and layer 1, were sufficient to create a behavior that was able to handle the complexity of the gait cycle.

4.3.3 GA Application

Even though the implementation of the genetic algorithms that was used to optimize the gait parameters has not been mentioned, it is essential to mention that the GA was implemented as an independent application due to some problems with the determinism of the simulator.

The problem with the simulator arose when training the learning methods and the applications were still implemented as one program. It was realized that evaluating the same set of parameters with the simulator would give very different results. One could not know if a given set of parameters were the reason for the robot to fall immediately or to walk long distances.

The real cause of the problem was never found. Although, it is probable that the problem arose from not properly resetting all the variables and buffers to their starting values in the ODE library.

Under these circumstances, when there is not consistent feedback from the simulator, it is very difficult to train a learning machine. Nevertheless, it was possible to get consistent results from the simulator by restarting the whole application (behavior control and simulator). Therefore it was decided that the implementation of the genetic algorithm had to be done as an independent application.

4.3.4 Communication between the Programs

The GA application, the simulator and the behavior control needed a way to communicate with each other, in order to perform the experiments either with the simulator or with the real robot.

Two applications coexisted in one program (the simulator and the behavior control). The behavior control has a communication buffer that captures the generated target positions for the servo motors. After capturing the new positions, it either sends them to the simulator or to the real robot, depending on the type of experiment.

In Section 4.3.3, it was mentioned that the GA application had to be implemented as a separated program. To achieve the communication with the behavior control program, the GA program writes a text file with all the data that needs to be tested. After that, it executes the behavior control program and waits for feedback. When the behavior control program starts, it expects the information file to be present. Then, when the simulator and behavior control are finished testing the data, a new file is written to give the fitness evaluation to the GA application.

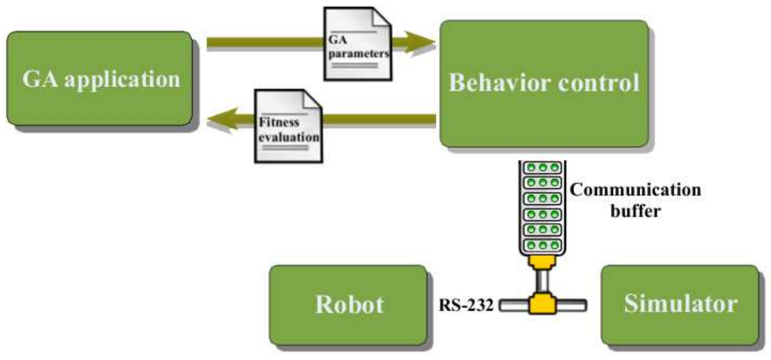


Figure 4.8: Communication diagram

After writing the file for the GA, the behavior control and simulator program terminates its execution. Then, the GA program reads the evaluation file. This is done systematically until the genetic algorithm has reached the stopping criterion, which the evolution during certain number of generations. Figure 4.8 depicts the communication diagram when carrying out experiments.

Summary

Chapter 4 presented the robot KHR-1 used during this work, as well as the hardware modifications made to it. The software was also presented (Simulator, Behavior control).

Chapter 5

Gait Engines

This chapter discusses the two optimized gait engines: the gait G4P and the gait G19P. The names of the gaits are just an acronym of the words *Gait*, *number of parameters selected to be optimized* and *Parameters*. For the G4P gait, 4 meaningful parameters were selected, whereas for the gait G19P, 19 parameters where selected.

5.1 Starting to Walk

Before introducing the proposed gaits, it will be useful to describe how the robot rocks from side to side, leaving the body weight on one leg. First imagine that the robot forms a parallelogram with its feet, hip and legs (Figure 5.1). In order to start rocking, one needs to control four joints of the lower body: HipX and AnkleX joints (Figure 4.4). The ankle servos rotate in the same direction at the same time, as well as the servos at the hip but in counter direction to the ankle servos.

This periodic motion produces a something similar to that of the inverted pendulum mentioned in Section 3.3, although in the robot's frontal plane. Moreover, it is possible to see that one of the legs supports all the body weight when rocking towards its direction. During this support phase it is possible to command the robot to lift the contralateral foot for a small period of time by shortening the leg, and then straightening it back out

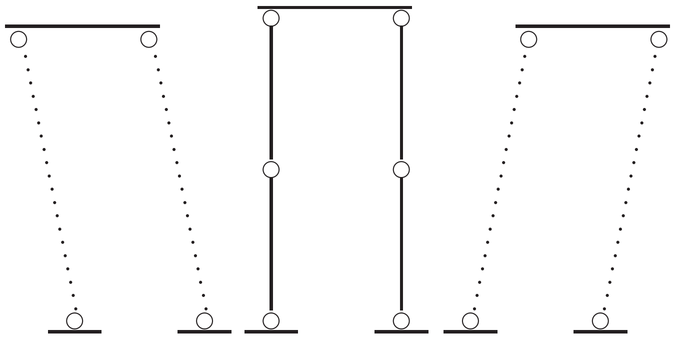


Figure 5.1: Parallelogram formed by the robot
One can see that the two foot plates are parallel to the hip plate

again to continue the swinging motion. If one let the robot perform this periodic action, one will realize that the robot is marching in place.

To shorten the leg one needs to put into action more servo joints: HipY, Knee and AnkleY (Figure 4.4).

A desired characteristic, when making the leg shorter, is that the foot plate stays parallel to the hip, as shown in Figure 5.1. Now, let us assume that the legs of the robot are symmetrical and that the distance between the hip and the knee is equal to the distance between the knee and the ankle. This assumption makes easier the estimation of the new positions of the servo motors.

Let us imagine that a fully extended leg has 100% of its *leg extension* (LE). Then, the angle α that has to be added or subtracted to the hip, knee and ankle joints in order to create a new LE is calculated as follows:

$$\alpha = \arccos(\textit{new LE}) \tag{5.1}$$

new LE is the desired percentage of the leg extension.

Figure 5.2(a) shows a fully extended leg, which is formed by two parts of equal length. When a leg is shortened, as shown in Figure 5.2(b), it forms a triangle whose angles can be calculated.

Making one of the legs shorter when rocking is dangerous, in the sense that the robot can fall if the leg is not lifted at the right moment and for the

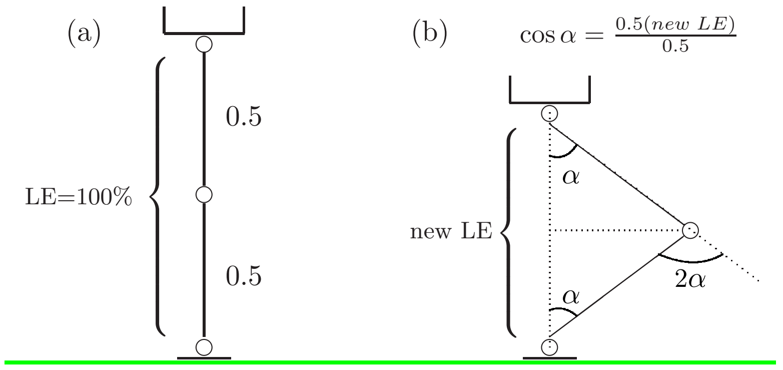


Figure 5.2: Calculating a new leg extension
(a) Fully extended leg. (b) Shortened leg and location of the α angles.

right amount of time, while the opposite leg supports the body weight. The rocking motion itself can also make the robot to fall over if the side to side motion, also called the lateral displacement motion, is considerable.

Figure 5.3 illustrates how the robot looks when marching in place. It also shows the trajectory of the servo motors, according to the motion convention described in Section 4.2.3.

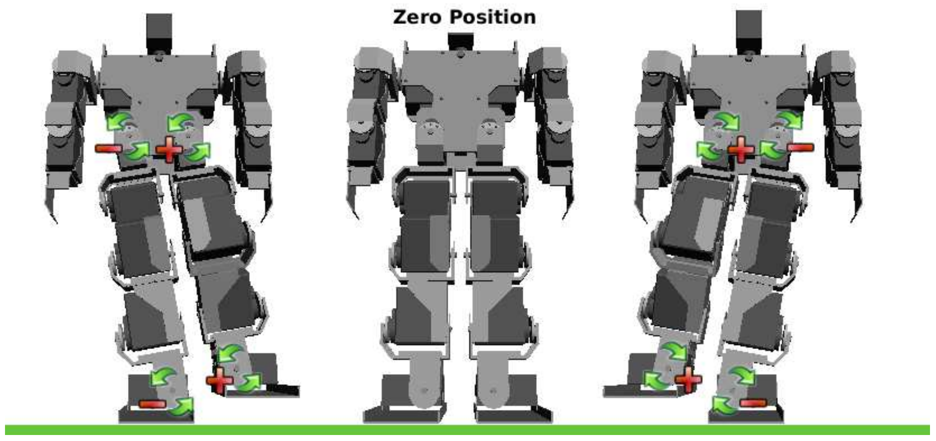


Figure 5.3: KHR-1 marching in place

Marching in place can be considered a prerequisite for walking, although the robot is not advancing. One can also say that it is walking with the length of the steps equal to zero.

The robot has to perform the lateral motion because it the only way to shift the weight of the body to one leg, leaving other leg free to swing forward.

5.2 Timer structure

The implementation of a timer is required to coordinate all the movements of the robot. The timer used during this work runs within the interval $[-\pi, \pi]$, where each lapse of time $t_{new} = t_{old} + \Delta t$. Δt is a positive number that determines the speed at which the timer runs. An interval of 2π was selected because the most common periodic functions, such as sine or cosine, have this period. Having a timer running within the interval $[-\pi, \pi]$ and using periodic functions, makes the implementation of repetitive events relatively straightforward.

Furthermore, each leg uses this general timer but with an offset of $\pm\pi/2$. To distinguish between each leg, a *leg sign* was assigned: one leg has $+1$ and the other leg has -1. This allows the creation of an antiphase effect when calculating the time offset for each leg.

$$\textit{timer of leg} \ = \ \textit{general timer} \ + \ (\textit{leg sign} \cdot \frac{\pi}{2}) \tag{5.2}$$

Calculating the timer for each leg.

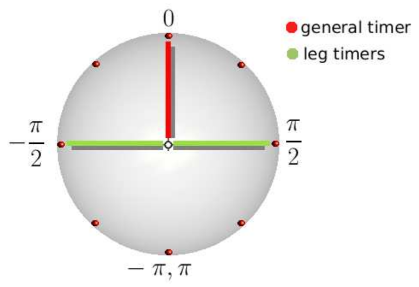


Figure 5.4: General timer and leg timers

5.3 Gait G4P

The first gait engine presented is the gait G4P, where four parameters are selected to be optimized. The purpose of selecting just four parameters is to investigate their importance during the optimization process while the other parameters are fixed to a constant value.

The gait engine G4P uses the basic rocking from side to side, described in Section 5.1, as the principle to start walking. It is essential to discuss the conjunction of this behavior with the timer described in section 5.2, in order to achieve the correct coordination of the movements.

First, let us start with one servo motor placed at its zero position with a stick attached to its rotor (Figure 5.5). In order to produce the desired rocking motion, one can compute the servo positions using a sinusoidal function, where the amplitude indicates the rotational displacement.

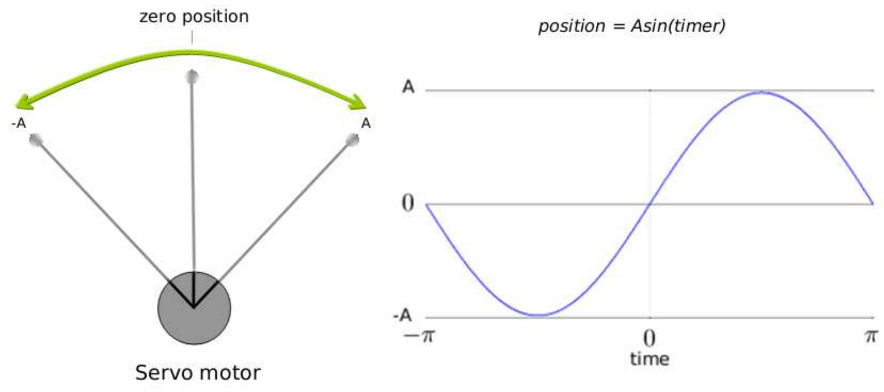


Figure 5.5: Making a servo to rock from zero to -A, to zero, to A, to zero.

The same principle is applied to generate the positions of the servos that make the robot swing from side to side. The functions used by the gait G4P are sinusoidal, moreover, the generated positions are stored in the actuators $LegAngleX$ and $FootAngleX$ of each leg, see Section 4.3.2.

A parameter called *lateral amplitude* controls the rocking motion of the robot in the frontal plane, indicating how far the robot should move from side to side.

The following procedure generates the rocking motion:

$timer$ = The timer of the corresponding leg. (5.3)

LegAngleX = lateralAmplitude$\cdot$ sin($timer$)

FootAngleX = -0.5$\cdot$lateralAmplitude$\cdot$ sin($timer$)

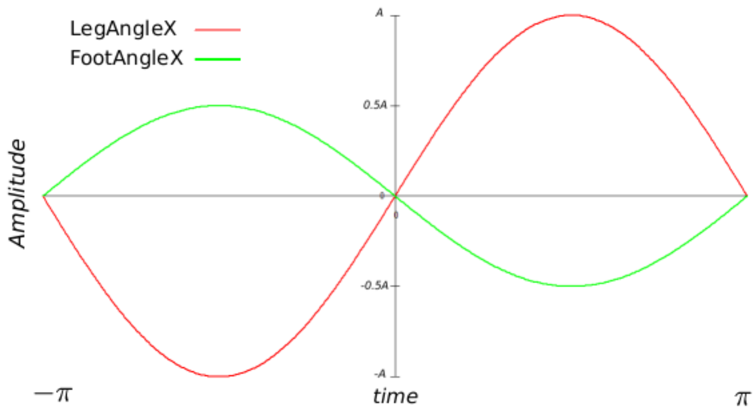


Figure 5.6: G4P - Positions generated for LegAngleX and FootAngleX
This motion can be visualized in Figures 5.1 and 5.3

One can see in Figure 5.6 that the displacement of the joints at the hip is twice the displacement of the foot joints. The factor 0.5, obtained by manual calibration, increases the lateral inclination of the robot when performing the rocking motion. The motion at the hips tries to decrease this lateral inclination so the robot does not fall.

The next step, is to shorten the leg that is not carrying the robot's weight in order to lift the foot plate off the ground. This leg shortening is carried out as mentioned in Section 5.1, though one needs to fix a small error with the calculations, because the distance between the joints is not equal, as required by this method.

The *LegExtension* actuator is the one in charge of keeping the current leg extension. The range of this parameter goes from 0 to 1, meaning 0% to 100% of the leg extension respectively. However, the robot is not able to shorten the leg less than 60% due to the hardware design. At first the leg extention during the support phase was set to 100%, completly straightening the leg. It was then observed that the robot could acheive higher speeds when the leg was left slightly bent. This is further explained in Section 7.1.

The leg shortening in the gait G4P has a procedure to compute the servo positions. This procedure reduces the extension of the leg in the second half period of a sinusoidal function, se Figure 5.7. Before the leg shortening begins, the leg is perfoming the support phase and has to have a longer extention than in the swing phase in order to properly carry the weight of the robot. It can be seen that the green line representing the support phase in Figure 5.9 corresponds to the time when the leg has a longer extention. When the blue sinusoidal swing phase begins, then the leg shortening also takes place roughly at the same time; that is to say that the two motions swing and shortening are related in time. Moreover, the shortening of the leg is increased when the steps are big and the shortening is reduced when the steps are small by relating the sagittal amplitude and the amplitude of the sinusoidal function that shortens the leg.

The amplitude of the leg shortening function is called *ShorteningAmplitude*. The frequency of the same function, which plays a role in the speed of the shortening of the leg is called *ShorteningSpeed*.

Again, there are some constant values that were manually set or used values obtained from experiments in the simulator. Such is the case for the time when the robot starts bending the leg ($ShiftedTimer$), the speed at which the robot shortens the leg *ShorteningSpeed*, etc. These values are shown in the procedure 5.4.

```
                                                                        (5.4)
$timer$ = The timer of the corresponding leg.
ShiftedTimer = $timer - 1.15$
ShorteningAmplitude = 0.35
ShorteningSpeed = 2.9
NewAmplitude = ShorteningAmplitude + 0.45 · (SagittalAmplitude)
if (ShiftedTimer $\geq \frac{-\pi}{\text{ShorteningSpeed}}$) and (ShiftedTimer $\leq \frac{\pi}{\text{ShorteningSpeed}}$)
    ShorteningFactor = NewAmplitude·$(-1 - \cos(\text{ShorteningSpeed}\cdot\text{ShiftedTimer}))$
end if
```

5.4: G4P - Procedure to shorten the leg

As aforementioned, the shortening factor has to take place during the swing phase. To assure that these two events coincide, a shifted timer was set to the value $timer - 1.15$. The shortening speed was set to 2.9, meaning that the swing motion will be executed in the same amount of time no matter how large the size of the step. The $NewAmplitude$ is the amplitude of the cosine function used to shorten the leg. Its minimum value is the shortening amplitude, which was set to 0.35. This means that when the robot is marching in place, the amplitude of the shortening function is 0.35. As the sagittal amplitude (step size) is increased, the $NewAmplitude$ is also increased, which in turn increases the amplitude of the cosine funtion. The end result is that the robot lifts its foot higher off the ground as the step size increases.

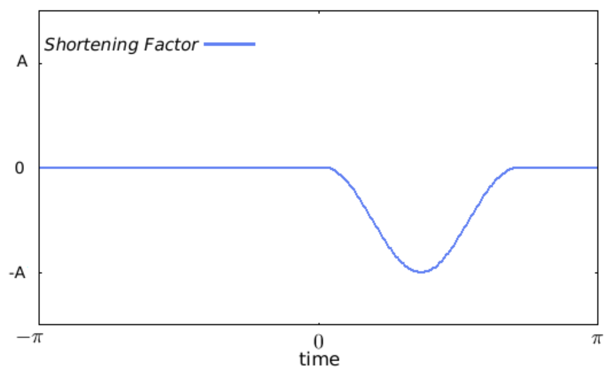


Figure 5.7: G4P - Function to shorten the leg. The frequency of the function is fixed, but the amplitude depends on the size of the step and a constant value ($ShorteningAmplitude$) that was fixed at 0.35.

Once the robot is able to rock from side to side and it is also able to shorten the legs, the next step is to produce the forward motion. This gait creates the forward motion by producing target positions for the actuator $LegAngleY$, which controls the pendular motions of the legs in the sagittal plane.

A leg performing its support phase behaves like an inverted pendulum that carries the body weight in the direction that the robot advances. Figure 5.8(a) illustrates this situation. On the other hand, when the leg performs its swing phase, it behaves as a normal pendulum attached at the hip. Figure 5.8(b) depicts the leg of the robot when swinging. Notice that the leg is not yet shortened.

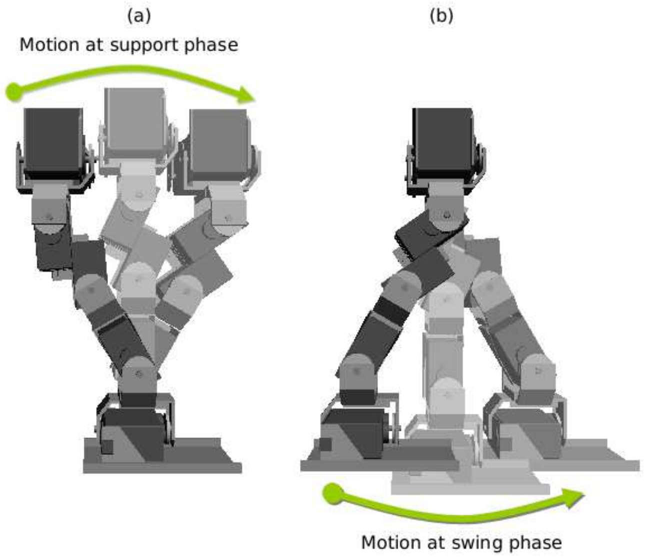


Figure 5.8: Pendular motions of the support leg and swing leg

From the above image, one can learn that the support motion and swing motion are performed by the servos HipY and AnkleY of each leg, see Section 4.3.2. To perform the support phase, the servo positions are generated by a straight line with a gentle slope. The servo positions for the swing phase are generated by a sinusoidal function. See Figure 5.9

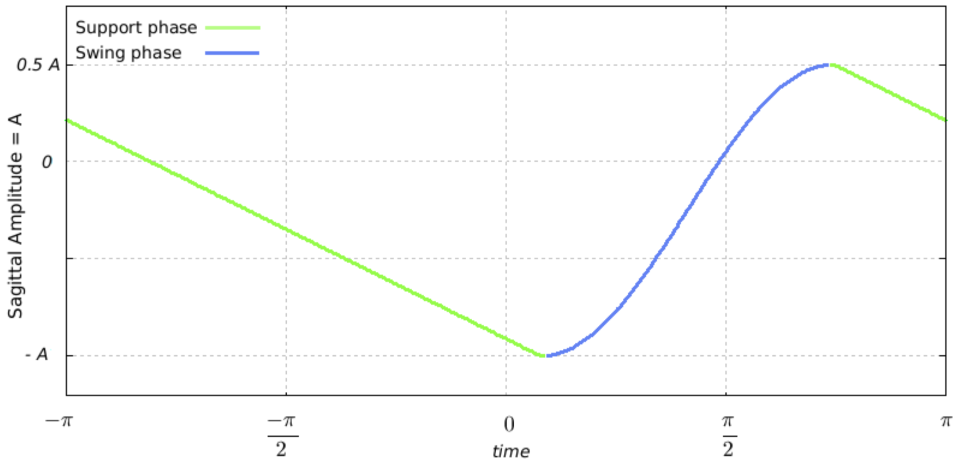


Figure 5.9: G4P - Servo postions for the support and swing phases

The straight line generates positions for the $LegAngleY$ that take the weight of the body gently to the front, whereas the sinusoidal function generates positions that produce a fast swing motion. These two motions are related to the size of the step, which is controled by a the parameter *Sagittal Amplitude*. The sagittal amplitude controls the amplitude of the pendular motions of the support and swing phases. For example, when the value of this parameter is large, then the step size is also large and vice versa.

The procedure 5.5 shows how to produce the positions of the support and swing motions used by the gait G4P. It depicts several constant values, which are other parameters, whose values were manually tuned or experimentally found with the help of the simulator. For example, the parameter $SwingStart = 0.25$ tells time when the swing phase stars, the parameter $SwingSpeed = 1.5$ tells speed of the swing, etc.

```
(5.5)
$timer$ = The timer of the corresponding leg.
$\text{SwingStart} = 0.25$
$\text{SwingSpeed} = 1.5$
$\text{tmpAngle} = -(\frac{1.5}{2\pi-(\pi/\text{SwingSpeed})} \cdot (timer - \text{SwingStart}) + 1)$

**if** ($timer \geq \text{SwingStart}$)
    **if** ($timer < \text{SwingStart} + \frac{\pi}{\text{SwingSpeed}}$)
        $\text{tmpAngle} = -0.75(\cos(\text{SwingSpeed}(timer - \text{SwingStart})) + 0.33)$
    **else**
        $\text{tmpAngle} = -(\frac{1.5}{2\pi-(\pi/\text{SwingSpeed})} \cdot (timer - \text{SwingStart} + \frac{\pi}{\text{SwingStart}}) - 0.5)$
    **end if**
**end if**
$\text{LegAngleY} = \text{SagittalAmplitude} \cdot \text{tmpAngle}$
```

5.5: G4P - Procedure to generate the positions of support and swing motions.

The above procedure is divided into two parts. The first part covers the time interval of $[-\pi, SwingStart]$. During this period of time, the *tmpAngle* receives the positions generated by the equation of a straight line. The second part corresponds to the time interval $[SwingStart, \pi]$. During this period of time, a cosine function generates the positions that produce the swing motion. After the cosine function reaches its apex, the positions are then generated using the same equation as the first straight line but shifted in time. It is possible to see from this procedure that the support phase lasts longer than the swing phase. This actually mimics the gait of a human, which reduces the time when the body is most unstable - i.e. when only one foot is touching the ground.

With the purpose of keeping the foot plate parallel to the ground, the calculated $LegAngleY$ from the hip joints must be added to the position of the ankle joints.

After computing the $ShorteningFactor$, it is necessary to subtract it from the $LegExtension$ in order to calculate the new leg extension during the swing motion.

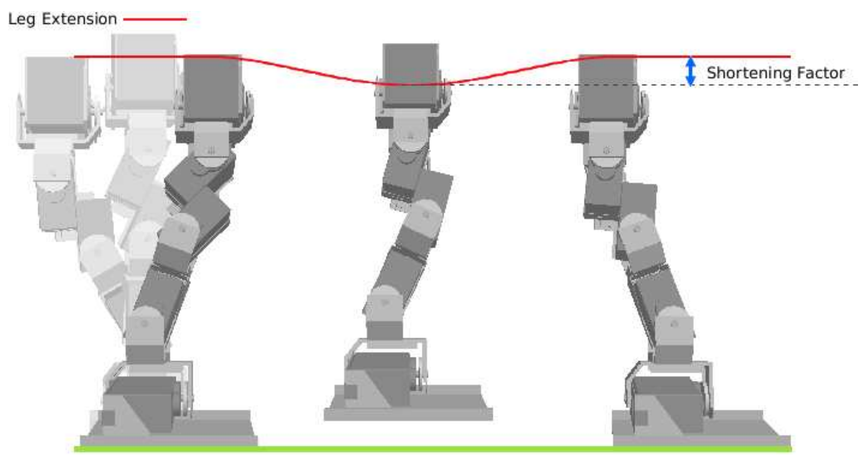


Figure 5.10: G4P - Shortening the leg during the swing phase

Figure 5.10 depicts the leg extension before, during, and after the swing phase. The red curve depicts the percentage of the leg extension during the gait cycle. One can also realize that the leg extension is constant until it is time to swing the leg. Furthermore, one should remember that the other leg is performing the support phase, transporting the rest of the body forward.

Finally the values are added together in order to compute the new servo positions.

$$\begin{aligned}
&\text{Knee} = -2\alpha \\
&\text{AnkleY} = -\text{LegAngleY} + 0.8\alpha - \text{Inclination} \\
&\text{AnkleX} = \text{FootAngleX} \\
&\text{HipY} = \text{LegAngleY} + 1.2\alpha \\
&\text{HipX} = \text{LegAngleX}
\end{aligned} \tag{5.6}$$

5.6: G4P - Procedure compute the new servo positions

The α angles calculated using the equation 5.1 in order to produce a new leg extension have a small error. This error is introduced by the fact that the distance between the hip-knee and knee-ankle joints is not equal in the KHR-1 robot. In order to correct the α angles, they are multiplied by a factor of 0.8 and 1.2 as can be seen in the procedure 5.6.

When calculating the servo positions for the AnkleY sensor, a new parameter was introduced to compensate for the weight of the Pocket PC and the camera. The so-called *Inclination* parameter adds some degrees to the general inclination of the robot to prevent the robot from falling forwards.

All these actions complete one step in half the time of the total timer, and two steps or one cycle during the total time. Naturally, the speed at which this timer runs affects the performance of the gait.

The G4P gait is a very simple one, where most of the values were determined by human experience and continues manual tuning. Nevertheless, some of these parameters, four to be precise, were optimized as explained in the next chapter.

5.4 Gait G19P

The second proposed gait engine is the G19P, which in total has 19 parameters. This gait engine is different from the G4P because it does not bind the movements of the robot and treats the motion of every joint independently. Of course, this leads to a more complicated gait that has more parameters to take into consideration.

The *timer* of the gait is identical to the one used in the G4P. However, the total time is divided into two equal parts of π length, one dedicated to the support phase and one to the swing phase. The time from $[-\pi, 0]$ is the time of the support phase and the time from $[0, \pi]$ belongs to the swing phase. This division was done to separate the two main gait phases, meaning that the leg performing either support or swing phase has to perform its duties between these time boundaries.

First of all, let us start with the implementation of the rocking motion of the robot. If one remembers, the gait G4P performs this using the procedure 5.3, binding the motion of the servos at the feet, as well as the motion of the servos at the hip. The difference with the G19P gait is that each servo has its own function to generate the positions. These positions are stored in the actuators $FootAngleX$ and $HipAngleX$.

Before clarifying the procedures that generate the positions, the desired motion of the joints is explained. Initially the joints are at the zero position, as the time advances, the angle of the joints increase until they reach the apex of a sinusoidal function, and then begin to decrease until the joints again reach the initial position. Figure 5.11 gives and example of this motion at the foot joints.

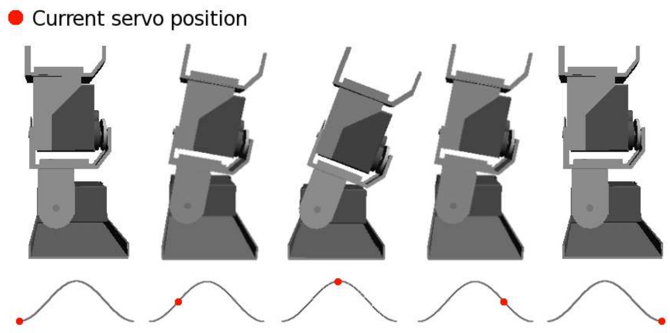


Figure 5.11: G19P-Flexion of the ankle joints based on a sinusoidal function

The gait G19P assumees that when one of the gait phases starts (swing or support), the four servos that generate the rocking motion must be at their zero position. This means that if these servos move during the swing or support phase, they have to return to the zero position before they switch from the current gait phase.

Using periodic functions, such as cosine or sine, to generate the servo positions simplifies the control of the joints because one can make sure that the motion of the servos is going to finish at the zero position within the π time interval.

A simple cosine functions has three basic parameters: Amplitude, Frequency, Phase shift. These parameters control the rotation of the servo, the speed of rotation and the time when the motion has to start and finish respectively.

$$A \cos(B(time + C))$$

A = amplitude, B = Frequency and C = Phase shift

In order to generate the positions for the servos that have something to do with the rocking motion of the robot, the following procedures are going to use periodic functions to produce servo motions with an amplitude of motion A, with speed of motion B and with a phase shift C.

The *FootAngleX* positions are generated by the procedure 5.7, which is subdivided into two parts. One is in charge of the motion during the support phase $[-\pi, 0]$ and the other is in charge of the motion during the swing phase $[0, \pi]$. After checking if the *timer* corresponds either to the support or swing phase, the procedure waits to move the servo joint from its zero position until the assigned phase shift $\frac{C_n}{B_n}$ indicates it. When one period of the cosine function is over, the procedure ensures that the joint moves stays at its zero position.

(5.7)

```
if $timer$ between $[-\pi, 0]$      Support phase
    if $(timer \geq -\pi + \frac{C_1}{B_1})$ and $(timer \leq \frac{-2\pi}{B_1} + \frac{C_1}{B_1})$
        FootAngleX $= \frac{A_1}{2}(1 - \cos(B_1(timer - \frac{C_1}{B_1} + \pi)))$
    else
        FootAngleX $= 0$ (zero position)
    end if
end if

if $timer$ between $[0, \pi]$      Swing phase
    if $(timer \geq \frac{C_2}{B_2})$ and $(timer \leq \frac{2\pi}{B_2} + \frac{C_2}{B_2})$
        FootAngleX $= -\frac{A_2}{2}(1 - \cos(B_2(timer - \frac{C_2}{B_2})))$
    else
        FootAngleX $= 0$ (zero position)
    end if
end if
```

5.7: G19P - Procedure to generate the FootAngleX positions

In total, six parameters are controlling the motion of the AnkleX joints in the procedure 5.7: A_1, B_1, C_1, A_2, B_2, C_2. The A parameters are the amplitude of their corresponding cosine curve, which means how much the joint will twist. The B parameters are the frequency and represent the speed of the motion. Finally, the C parameters represent the time translation or phase shift of each function.

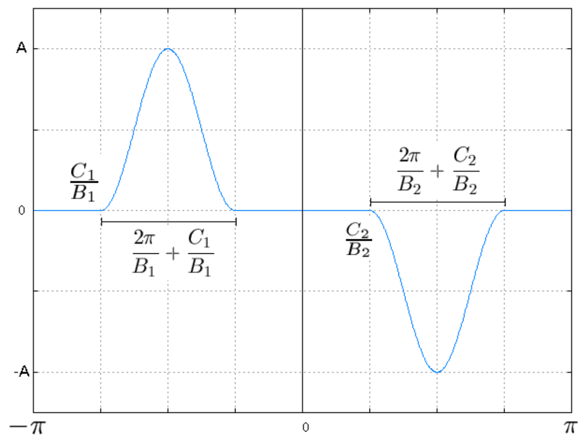


Figure 5.12: G19P - Example of parameter curves

Figure 5.12 depicts an example of the servo positions generated by the last procedure, where the parameters A, B and C were arbitrarily chosen. It is possible to see that each of the two curves has three parameters.

The G19P uses the same method as with the AnkleX joints, to generate the motion at the HipX joints. Again, the parameters A, B and C are going to be used to create the positions of the servos. The procedure 5.8 shows this.

(5.8)

```
if $timer$ between $[-\pi, 0]$    support phase
    if $(timer \geq -\pi + \frac{C_3}{B_3})$ and $(timer \leq \frac{2\pi}{B_3} + \frac{C_3}{B_3} - \pi)$
        HipAngleX $= -\frac{A_3}{2}(1 - \cos(B_3(timer - \frac{C_3}{B_3} + \pi)))$
    else
        HipAngleX $= 0$ (zero position)
    end if
end if

if $timer$ between $[0, \pi]$    swing phase
    if $(timer \geq \frac{C_4}{B_4})$ and $(timer \leq \frac{2\pi}{B_4} + \frac{C_4}{B_4})$
        HipAngleX $= \frac{A_4}{2}(1 - \cos(B_4(timer - \frac{C_4}{B_4})))$
    else
        HipAngleX $= 0$ (zero position)
    end if
end if
```

5.8: G19P - Procedure to generate the HipAngleX positions

Six new parameters are introduced: A_3, B_3, C_3, A_4, B_4, C_4. Moreover, these parameters have the same meaning as afore explained.

As carried out in procedure 5.7, procedure 5.8 also checks if the timer of the leg is at swing or support phase. Then, it keeps the HipX servos at its zero position until the time shift $\frac{C_n}{B_n}$ indicates the the servo should start its motion with the positions generated by the cosine functions. After one period of these cosine functions, the servo joints should be again at its zero position.

Figure 5.13 depicts an example of the positions calculated by procedures 5.7 and 5.8, where each curve is in charge of controlling one servo. The servos that are involved in the rocking motion, namely the servos at the hip and at the feet (actuator HipX and AnkleX).

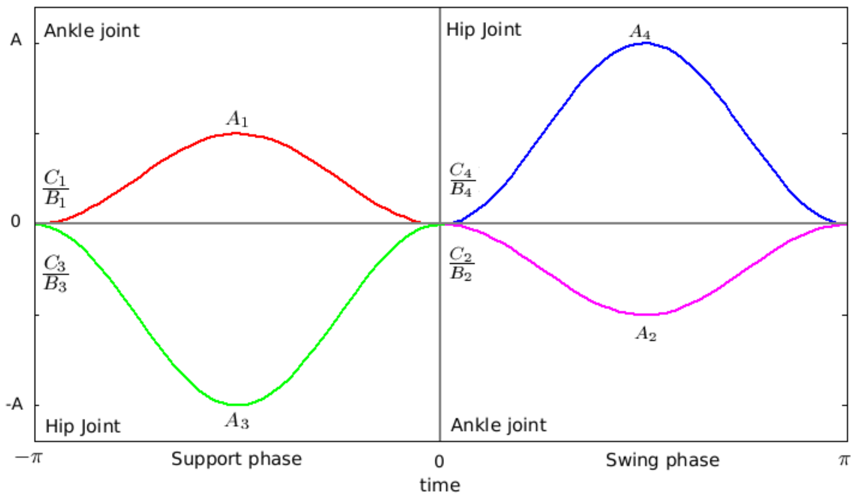


Figure 5.13: G19P - Servo positions at the hip and ankles that make the robot swing from side to side.

Before further explanation, it essential to clarify the relation between the parameters A, B and C because they have strong dependencies and also form something similar to a hierarchy.

If the parameter A (amplitude) is different from zero, the function exists and the servo joint will move from the zero position. The parameter B (frequency or speed of motion) has a minimum value of 2, this is because the original assumption tells us that if the servo moves from the zero position, it has to come back to it, in order to be ready for the next movement. Since the timer was divided into two halves of π length, the slowest functions that can fulfill this requirement are the ones with a frequency of 2 and zero time translation. Now, one can think about the strong relation between the parameter B and the parameter C (time translation). Where each value of B has its own range of time translation.

In order to calculate the maximum phase shift, it is necessary to calculate the period of the function. Since the normal sine and cosine functions have a period of 2π, the function $f(t) = \sin(Bt)$, $B > 0$ completes one cycle as Bt varies from 0 to 2π . This means that $0 \leq Bt \leq 2\pi$. Solving this inequality for t, one obtains $0 \leq t \leq \frac{2\pi}{B}$. The last means that the functions completes one cycle as t varies from 0 to $\frac{2\pi}{B}$ and has a period of $\frac{2\pi}{B}$. To calculate the maximum phase shift one must subtract the calculated period from the maximum possible period, which is in our case π.

$$\begin{gathered}\text{maximum shift} = \pi - \tfrac{2\pi}{B}, \quad 2 \leq \mathrm{B} \leq 3.5 \\ \text{maximum C} = \text{maximum shift} \cdot \mathrm{B} \\ \text{A complete cycle is delimit in the range:} \\ \left[\mathrm{C}, \mathrm{C} + \tfrac{2\pi}{B}\right]\end{gathered} \tag{5.9}$$

The generation of the forward motion is similar to that in the gait G4P. The only difference is that the G19P uses half cycles of sinusoidal functions to generate both support and swing phases. Moreover, these functions are also delimited by the support and swing boundaries $[-\pi, 0]$ and $[0, \pi]$ respectively.

The *LegAngleY* actuator is used in each leg to store the position of the leg when transporting the body weight or when swinging. If one remembers the parameter sagittal amplitude of the gait G4P, it is the one that controls the size of the step. The gait G19P also uses a parameter that is going to control the amplitude of the sinusoidal functions generating the pendular motions. The name of this parameter is α, Figure 5.14 depicts the robot at the double stance phase, where one leg is preparing to swing and the other one is preparing to support. It is at this moment when it is possible to identify the α angles.

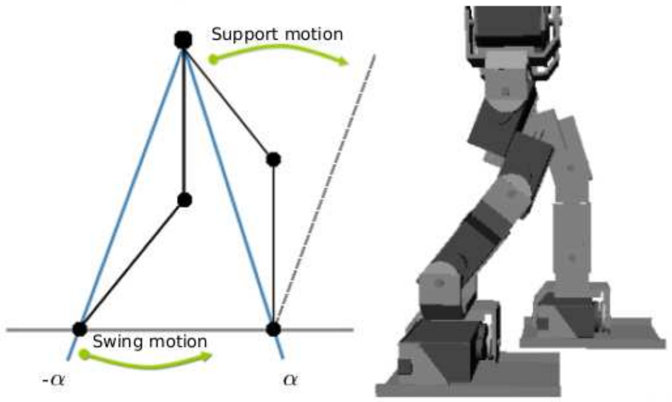


Figure 5.14: G19P - Robot at double stance phase having the legAngleY sensors with α values

One can see that both legs will exchange their roles. The support leg that originally has a leg angle equal to α, will perform the inverted pendulum motion until it reaches a leg angle equal to $-\alpha$. On the other hand, the swinging leg starts with a leg angle equal to $-\alpha$ and it will perform the normal pendulum motion until it reaches a leg angle equal to α.

The following procedure shows how to compute these positions.

(5.10)

```
Motion of the support leg
**if** $timer$ **between** $[-\pi, 0]$
    **if** $(timer \geq -\pi + \frac{C_5}{B_5})$ **and** $(timer \leq \frac{\pi}{B_5} + \frac{C_5}{B_5} - \pi)$
        **LegAngleY** $= \alpha \cos(B_5(timer - \frac{C_5}{B_5} + \pi))$
    **else if** $(timer < -\pi + \frac{C_5}{B_5})$
        **LegAngleY** $= \alpha$ (starting angle)
    **else if** $(timer > \frac{\pi}{B_5} + \frac{C_5}{B_5} - \pi)$
        **LegAngleY** $= -\alpha$ (final angle)
    **end if**
**end if**

Motion of the swing leg
**if** $timer$ **between** $[0, \pi]$
    **if** $(timer \geq \frac{C_6}{B_6})$ **and** $(timer \leq \frac{\pi}{B_6} + \frac{C_6}{B_6})$
        **LegAngleY** $= -\alpha \cos(B_6(timer - \frac{C_6}{B_6}))$
    **else if** $(timer < \frac{C_6}{B_6})$
        **LegAngleY** $= -\alpha$ (starting angle)
    **else if** $(timer > \frac{\pi}{B_6} + \frac{C_6}{B_6})$
        **LegAngleY** $= \alpha$ (final angle)
    **end if**
**end if**
```

5.10: G19P - Procedure to generate the forward motion

Procedure 5.10 introduces four new parameters B_5, C_5, B_6, C_6, which have the same meaning as in the other functions, speed of motion and phase shift respectively. The procedure first verifies if the leg is either in its swing or support phase.

If the leg performs its support phase, the procedure keeps the starting α value until the time shift $\frac{C_5}{B_5}$ indicates that the cosine function should generate the new positions that are going to take the leg angle to a $-\alpha$. When the cosine function has reached its minimum value, the procedure keeps the leg angle in $-\alpha$ until the leg has to perform its swing motion.

If the leg is performing its swing phase, the procedure keeps the leg angle with the starting $-\alpha$ value until the time shift $\frac{C_6}{B_6}$ agrees that the cosine function should start producing new positions. After the cosine function has reached the apex algorithm keeps the leg angle with the α value until it switches to the support phase.

Note that both cosine functions have the same amplitude α, this is done because it is desired that the pendular motions have the same amplitude.

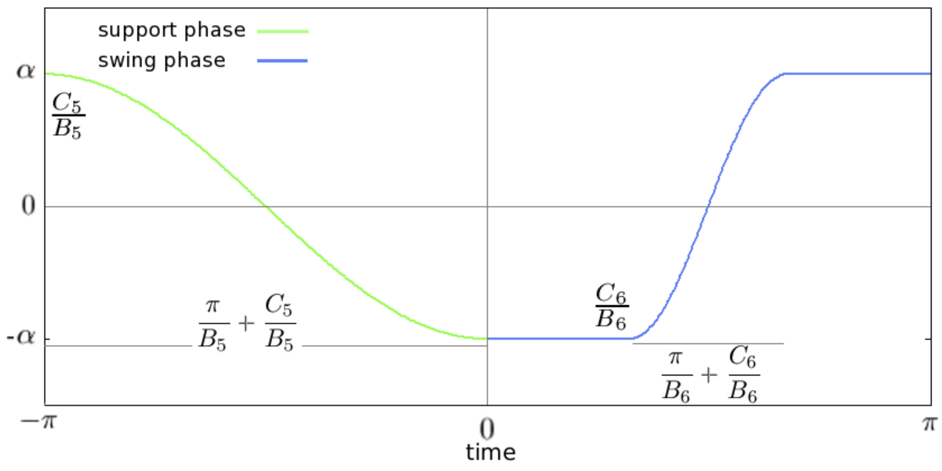


Figure 5.15: G19P - Functions of support and swing phases

Figure 5.15 shows an example of the functions that produce the transition of the *LegAngleY* from α to $-\alpha$ during the support phase, and from $-\alpha$ to α during the swing phase. One can also see that there are some constant parts at α and $-\alpha$ in the graphic; this is due to the decision of leaving the leg angle at its position until it is time to start the transition. This figure also shows the slowest possible transition during the support phase produced by a B (frequency) to be equal to 1. Naturally, if the speed of the transition increases, one will observe some constant lapses of $\pm\alpha$, similar to the ones depicted in the swing phase.

Changing the α angle allows us to change the size of the step. If its value is zero the size of the step is zero, letting the robot to march in place. Further explanation of how to calculate the step size is given in the next chapter.

The parameters B and C have the same relation as afored explained. In order too calculate the maximum phase shift of these functions, a similar approach to the one described in procedure 5.9 is used.

$$\begin{gathered}\text{maximum shift} = \pi - \tfrac{\pi}{B}, \quad 1 \leq \mathrm{B} < 3 \\ \text{maximum C} = \text{maximum shift} \cdot \mathrm{B} \\ \text{Half of the cycle is delimit in the range:} \\ \left[\mathrm{C}, \mathrm{C} + \tfrac{\pi}{B}\right]\end{gathered} \tag{5.11}$$

In this case, the lowest frequency is 1. This means that the requirement of reaching the angle $-\alpha$ starting at α, or vice-versa, cannot be achieved by a functions that cannot reach half of their period within a π period of time.

As well as the gait G4P, the gait G19P has an actuator in each leg to store the current leg extension when executing the gait. The *LegExtension* actuator has a range of [0,1], which means 0% to 100% of the leg extension. The idea of the leg extension is the same as explained in Section 5.1.

When presenting the gait G4P, it was mentioned that the legs of the robot KHR-1 do not fulfill the requirements of the method to calculate a new leg extension because the distance between the joint (hip-knee-foot) are not equal. In order to solve this situation, a new approach is introduced.

In the gait G19P, the leg extension is calculated with the law of cosines; this is accomplished by treating the legs as a triangle of which the length of its sides are known. The length of two of the sides of the triangle is fixed by the hardware design of the KHR-1 robot. Specifically, the distance from the servo motors at the hip (CH14 and CH20) to the servos at the knee (CH15 and CH21) is 5.2cm, and the distance from the servos at the knee to the servos at the feet (CH16 and CH22) is 5.6cm. The last side of the triangle is the desired leg extension. When the legs are fully extended they have a length of 10.8cm, Figure 5.16 (a), this means that the computation of the desired leg extension is carried out by multiplying $LegExtension$ times 10.8cm.

To fully understand the method, let us name each of the aforementioned sides: side a = 5.2, side b=5.8 and side c = $10.8 \cdot LegExtension$. The computation of the inner angles α, β and θ formed by the triangle depicted in Figure 5.16 (b) is straightforward using the law of cosines.

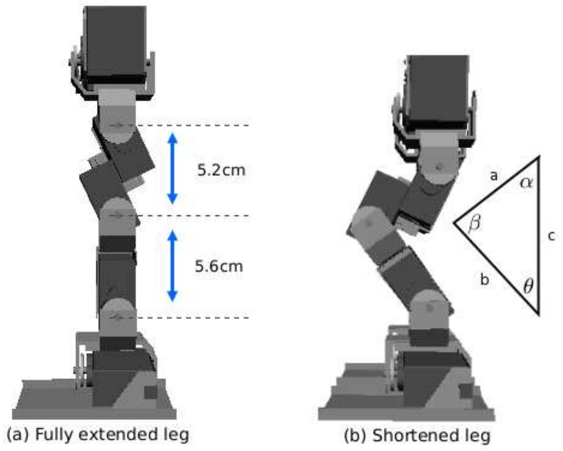


Figure 5.16: G19P - Calculating the leg extension

$$\theta = \arccos\left(\frac{b^2+c^2-a^2}{2bc}\right) \quad \alpha = \arccos\left(\frac{a^2+c^2-b^2}{2ac}\right) \quad \beta = \arccos\left(\frac{a^2+b^2-c^2}{2ab}\right)$$

The next step is to clarify the leg shortening during the swinging motion of the legs. When one of the legs starts to swing, it is assumed that the leg shortening must start as well. This is done by binding the shortening motion and the swinging motion. The parameters frequency B_6 and time shift C_6 of the procedure 5.10 are the ones that create this binding.

Figure 5.10 depicts the desired shortening of the leg when it is performing the swing motion. As well as explained for the gait G4P, the $LegExtension$ is constant during the whole gait cycle until the swinging leg moves forward.

The following procedure produces the desired shortening motion:

(5.12)

```
if $timer$ between $[-\pi, \pi]$
    if $(timer \geq \frac{C_6}{B_6})$ and $(timer \leq \frac{\pi}{B_6} + \frac{C_6}{B_6})$
        ShorteningFactor $= A_7 \sin(B_6(timer - \frac{C_6}{B_6}))$
    else if $(timer < \frac{C_6}{B_6})$
        ShorteningFactor $= 0$
    else if $(timer > \frac{\pi}{B_6} + \frac{C_6}{B_6})$
        ShorteningFactor $= 0$
    end if
end if
```

5.12: G19P - Procedure to shorten the leg when swinging

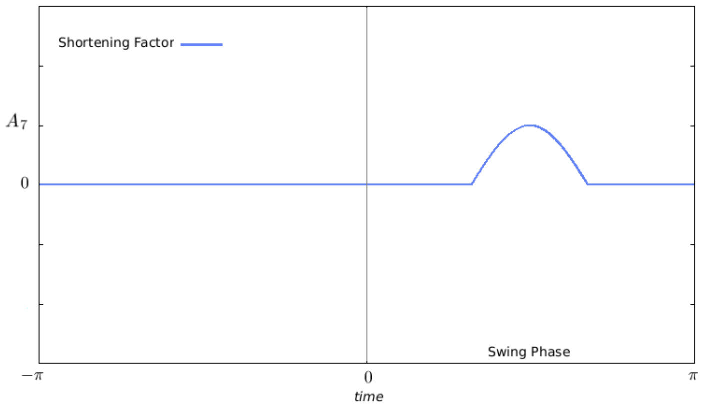


Figure 5.17: G19P - Shortening Factor function

It is possible to see in the procedure 5.12 that the parameter A_7 is the only new parameter. The frequency and the time shift belong to the swing motion. The amplitude A_7 indicates how much the leg is going to bend during the swing motion. To compute the new leg extension one must subtract the $ShorteningFactor$ from the $LegExtension$.

The procedure makes sure that the $ShorteningFactor$ is equal to zero during the entire gait cycle, and just when it is time to swing the leg, this factor increases its value following the positions computed with the sine function. Figure 5.17 depicts an example of the shortening factor.

Finally, one must calculate the servo positions for each leg, which is the same as the last step of the gait G4P:

(5.13)

$$\text{Knee} = -\pi + \beta$$

$$\text{AnkleY} = -\text{LegAngleY} + \alpha$$

$$\text{AnkleX} = \text{FootAngleX}$$

$$\text{HipY} = \text{LegAngleY} - \theta - \text{Inclination}$$

$$\text{HipX} = \text{LegAngleX}$$

5.13: G19P - Procedure compute the new servo positions

The inclination parameter has the same function as in the gait G4P, except that in the gait G19P it is placed at the hip, not at the ankle. The value of this parameter is 4°.

Summary

This chapter introduced the two gaits used during this work G4P and G19P. For both gaits, it was explained how to produce the servo positions for each of parameter.

Chapter 6

Gait optimization using Genetic Algorithms

During this chapter, the genetic algorithms that optimized the parameters of the gait engines G4P and G19P will be introduced. A discussion of the optimization problem is also put forth.

6.1 Problem formulation

After the introduction of the gait engines, it is time to mention the optimization problem that was solved during this work. Both gaits, G4P and G19P, were manually tuned in order to make the robot walk. Nevertheless, one can never be sure if the parameters that one thinks are the best for the gait engine are correct.

First of all, it is important to mention that the problem consisted of making the robot walk as fast as possible with the same set of parameters. This means that the robot should be able to accelerate from a zero speed to a maximum speed, after passing through a certain number of intermediate discrete speeds. Clearly, this is a very strong assumption because every walking speed might have its optimal set of parameters.

Note that optimizing the parameters of a gait engine is different for every surface. Although the principle of walking is the same, the value of the parameters change from one surface to an other. The main goal of this research was to make the robot walk as fast as possible over a carpet, similar to those used in the robocup leagues [30]. Making a bipedal robot walk over this type of surface is more difficult than just walking on smooth hard surfaces. This is because this surface is somewhat unpredictable, with bulges that affect the performance of the gait.

When the robot is at zero speed, it is marching in place. Then gentle changes in the size of the step, namely in the sagittal amplitude parameters, will create the forward motion. These changes in the step size are done in accordance with the step frequency and a linear increase in the speed. At the beginning of the investigation, the changes in the step length were carried out by linearly increasing the size of the step, not taking into account the step frequency. This led to an uneven treatment of the step frequencies, where the lowest step frequencies were the ones taking advantage of this situation, leaving out possible optimum step sizes for the medium and higher step frequencies.

In order to clarify how to increase the size of the step, the *step frequency* has to be introduced as one of the parameters that will be optimized. As its name indicates, the step frequency is the number of steps that the robot has to perform within one second. This parameter is also considered to be one of the most important ones when developing a gait engine.

As mentioned in section 5.2, the general timer runs within the range $[-\pi, \pi]$. It was also mentioned that in this time interval a cycle of the gait has to be completed. This gives an interval of size π to perform one step. Since the rate at which the position packets are sent to the robot is $50Hz$, it is possible to calculate Δt of the general clock that will satisfy the desired step frequency.

$$\Delta t = \frac{\pi \cdot step\ frequency}{50} \tag{6.1}$$

The time needed to complete one step is given by:

$$t = \frac{\pi}{50 \cdot \Delta t} \tag{6.2}$$

After calculating the time required to execute one step, one can calculate the size of the step needed to keep up a certain speed using the linear equation of motion $d = v \cdot t$. Here, d is the length of the step, v the desired speed and t is the time required to complete one step.

Figure 6.1 shows the location of the α angle that needs to be calculated in order to produce the step of size d.

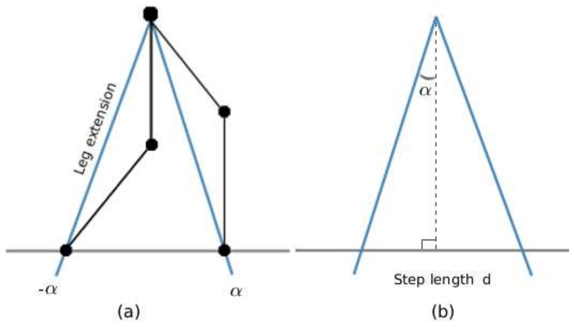


Figure 6.1: Step size according to speed

After observing the above image, the computation of the α angle for the $legAngleY$ is straightforward:

$$\alpha = \arcsin\left(\frac{d}{2 \cdot LE}\right) \tag{6.3}$$

where: d = step length in cm.
LE = current leg extension in cm.

Calculating the size of the step and sending the robot the corresponding servo positions does not give us any guaranties that the robot will move the joints exactly to those positions when walking. Due to the robot's lack of sensors, it very difficult to implement a fast and reliable gait engine, not to mention that some servo joints lose track of their position from time to time.

The acceleration in both gaits (G4P and G19P) is constant. For the gait G4P the increments in speed were carried out during the whole time of the gait cycle. Therefore, the speed changes were smooth and gentle. On the other hand, the gait G19P waited until one cycle of the gait was finished to increase the speed. The changes in this gait are a little more drastic, $0.25 cm/s$ increment every new cycle.

6.2 Proposed algorithms

This section presents the algorithms and the other parameters selected to be optimized. Two algorithms are introduced, one for each of the gaits described in chapter 4.

6.2.1 Algorithm for the gait G4P

When optimizing the gait G4P, just four meaningful parameters were selected. The following list shows these parameters:

1. Step frequency	2. Lateral amplitude
3. Leg extension	4. Inclination

Table 6.1: Four selected parameters to be optimized, see Section 5.3

The above parameters were the ones that after the optmization process allowed the robot to walk faster than with the selection of other parameters. This is further explained in chapter 7.

The learning or optimization process did not start from the very beginning. It started with a set of parameters that already made the robot walk. Furthermore, the selection of the four parameters was made by experience and by using the results from the simulator.

Before describing the algorithm, it is essential to introduce the encoding used to represent the individuals. Since there were no dependencies between the parameters, a simple array of four chromosomes, each containing one gene, was used. Moreover, the values of these parameters were of type float.

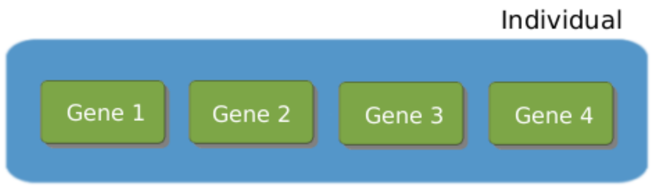


Figure 6.2: G4P Encoding of an individual

The following algorithm is the same as the one shown in Section 2.2.6. However, it is again here depicted because it is desired to illustrate the correct data input and the names used during the implementation.

GENETIC ALGORITHM FOR THE GAIT G4P:

Input: Initial individual x_0 with four parameters

Output: Individual x with four parameters that maximize the speed when walking

```
$g_{max} \leftarrow$ maximum number of generations
$n \leftarrow$ population size
$Ub \leftarrow \{ub_1, \ldots, ub_4\}$ upper bounds of parameters
$Lb \leftarrow \{lb_1, \ldots, lb_4\}$ lower bounds of parameters
$x_0 \leftarrow$ parameters of the initial individual
$g \leftarrow 0$    $g =$ generation counter
initialize $P(g) \leftarrow \{x_1^g, \ldots, x_n^g\}$    $x_i^g =$ individual of the population $P(g)$
evaluate $P(g)$
while $g \leq g_{max}$ do
    $g \leftarrow g + 1$
    create new generation $P(g)$ from $P(g-1)$
    evaluate $P(g)$
    elitism $P(g)$
end while
determine the best $x_i$
return $x_i$
```

The function called *initialize* takes the four parameters of the initial individual x_0 and generates n individuals for the initial population. This is done by adding Gaussian noise [33] to each parameter of the initial individual. The Gaussian noise is generated by a function that produces random numbers with a probability density of the Normal distribution.

The median of the Gaussian function is zero ($\mu = 0$) and the standard deviation is proportional to the allowed range of each parameter ($\sigma = 0.1 \cdot |ub - lb|$). If the value of the generated parameter is not within the lower and upper bounds, a new parameter is generated until this requirement is fulfilled.

(6.4)

```
Input: parameter k from $x_0$
Output: New parameter value
 $p \leftarrow$ parameter k
 $\mu \leftarrow 0$
 $\sigma \leftarrow 0.1 \cdot |ub_k - lb_k|$      limits of parameter k
 $new\ parameter \leftarrow p + GaussianNoise(\mu, \sigma)$
 while ( $new\ parameter < lb_k$ ) or ( $new\ parameter > ub_k$ )
         $new\ parameter \leftarrow p + GaussianNoise(\mu, \sigma)$
 end while
 return $new\ parameter$
```

6.4: G4P - Procedure initialize.

The next function is *evaluation*. The task of the evaluation is to assign the corresponding fitness value to the individual after the parameters were tested either with the simulator or with the real robot.

This evaluation process is very simple. First the parameters are sent to the testing process. Meanwhile, the robot (the simulated one or the real one) is waiting at the starting coordinate (0,0) on the walking surface. Then it starts to try to walk forward using the received parameters. It was already mentioned that the walking speed is going to increase, because we are looking for a set of parameters that can make the robot achieve maximum speed. Obviously, the robot is going to fall when it can not reach higher speeds or when it can not further spread its legs. Therefore, the fitness of the individual will depend on the distance that the robot was able to walk before falling.

When walking, the robot does not always go in a straight line. Therefore, walking in a curve or not over the x axis was penalized. The fitness value for each individual is the x component of the displacement, as shown in figure 6.3. One may think that penalizing an individual because it did not produce a straight motion is trying to maximize a function different from the one of maximizing the speed. In fact, the intention of penalizing is to produce some individuals that reach high speeds and that also walk in a straight line.

$$fitness_i = displacementX \tag{6.5}$$

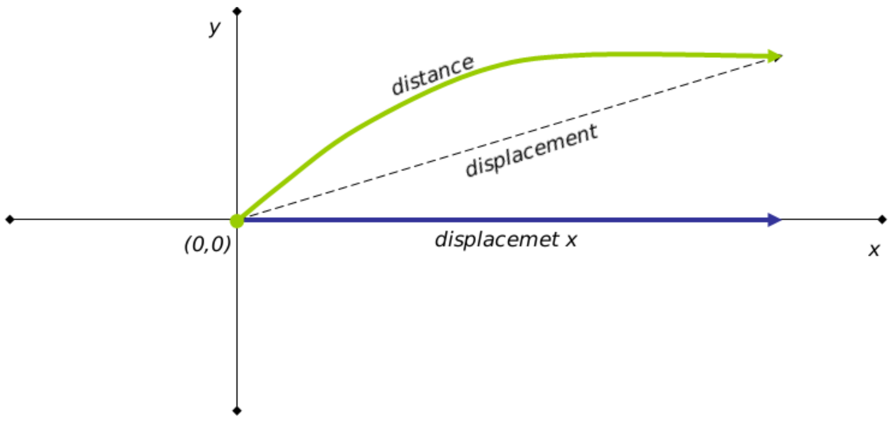


Figure 6.3: G4P Fitness of an individual

The ideal fitness would have been the distance walked by the robot, but with the KHR-1 robot it is impossible to calculate the odometry. With the simulator, it was possible to compute the odometry, although it was decided to use the displacement as the fitness value to be consistent with the experiments with the real robot. When performing the experiments with the real robot, the fitness of each individual was measured manually.

The *create new generation* function is in charge of selecting the parents and exchanging their genes to create a new individuals that will form the new generation.

At this point all the individuals of a generation have been evaluated. The next step is to create the roulette wheel to select the best individuals that are going to enter the mating pool for recombination.

The purpose when using the fitness proportionate selection is to give the best chances of selection to the fittest individuals, and less chances to the individuals with low fitness, thus, accelerating the convergence to an optimum. This is desired because the number of possible iterations is limited to a small number when experimenting with the real robot. Nevertheless, the drawback when using the roulette wheel selection is that the population tend to become very similar in a short amount of time. In order to solve this inconvenience, the Gaussian noise was introduced.

Let us assume that the parents A and B have already been selected. It is also known that for gait G4P there are just four genes to be exchanged. The next step is to apply the *uniform crossover* to select a gene from either from parent A or parent B with 50% of probability. After selecting the gene, some Gaussian noise is added, in order to introduce more possible values for each parameter and to solve the problem of the selection of the same parents.

Uniform crossover was selected because it has two important characteristics: "recombination potential" and "exploratortion power" [10]. The recombination potential is the ability to combine all the genetic material of the parents. The exploration power is the ability to search into the set of all possible solutions to a problem (search space).

The exchange of genetic information by the selected parents is carried out as shown in procedure 6.6.

The standard deviation σ, used to calculate the Gaussian noise, is influenced by the distance between the parameter values of the parents. When these values are too close together or equal, the standard deviation is calculated using the upper and lower bounds of the parameter.

The values of $\alpha = 0.2$ and $\beta = 0.02$ were determined by performing several experiments with different values in the simulator. The α factor uses the difference between the gene values of the parents to generate the standard deviation, but if these two values are too close to each other (almost equal), the algorithm uses the minimum standard deviation. The α_{min} is calculated using the difference between the upper and lower bounds of parameter times the β factor.

After calculating the correct standard deviation σ, the next step is calculating the Gaussian noise as a the disturbance or mutation number that will be added to the value of the selected gene. If the value of the new gene is outside of the boundaries, a new value will be generated until it is within the limits of the corresponding parameter. These offspring can be seen as mutated individuals, although these mutations are controlled by the generation of values close to the original gene value of the parent.

```
Input: $A \leftarrow$ first selected parent                                   (6.6)
       $B \leftarrow$ second selected parent
Output: $O$ offspring with 4 genes
 $\alpha \leftarrow 0.2$
 $\beta \leftarrow 0.02$
 $k \leftarrow 1$
 while ( $k \le nv$ )    $nv = 4$ Number of genes
    $gene_k \leftarrow$ (select either gene $A_k$ or $B_k$, each with 50% of probability)
    $\mu \leftarrow 0$
    $\sigma \leftarrow \alpha \cdot |A_k - B_k|$         using the gene value of the parents
    $\sigma_{min} \leftarrow \beta \cdot |ub_k - lb_k|$     using the limits of the parameter
    if ( $\sigma < \sigma_{min}$ )
       $\sigma \leftarrow \sigma_{min}$
    end if
    $O_k \leftarrow gene_k + GaussianNoise(\mu, \sigma)$
      while ( $O_k < lb_k$ ) or ( $O_k > ub_k$ )
           $O_k \leftarrow gene_k + GaussianNoise(\mu, \sigma)$
      end while
    $k \leftarrow k + 1$
 end while
 return $O$
```

6.6: G4P - Procedure to create a new individual.

In order to create a new population of n individuals, procedure 6.6 has to be executed n times. For each execution, new parents A and B are selected by spinning the roulette wheel.

The last function is the one that performs the *elitism*. After evaluating each generation, the best and the worst evaluated individuals is determined. Moreover, the best evaluated individual over the whole learning process is saved in an independent structure. If the best evaluated individual of a generation is not better than the best evaluated individual of the whole process, the worst evaluated individual of the generation is replaced by the best evaluated individual of the whole process. The elitism function helps to improve the performance of the optimization process because it does not lose the best individual found in the process.

The results obtained with this algorithm are shown in Section 7.1.

6.2.2 Algorithm for the gait G19P

This algorithm was in charge of training all the 19 parameters of the gait G19P at the same time. The following list shows these parameters:

1. Step frequency
2. Leg Extension

Support footAngleX

3. amplitude (A)
4. frequency (B)
5. time shift (C)

Support hipAngleX

6. amplitude (A)
7. frequency (B)
8. time shift (C)

Support motion

9. frequency (B)
10. time shift (C)

11. Shortening Factor

Swing footAngleX

12. amplitude (A)
13. frequency (B)
14. time shift (C)

Swing hipAngleX

15. amplitude (A)
16. frequency (B)
17. time shift (C)

Swing motion

18. frequency (B)
19. time shift (C)

Table 6.2: List of all 19 parameters for the gait G19P
For more details, see Section 5.4.

Due to the dependencies between the parameters the encoding of these parameters was not as direct as in the last algorithm (G4P). In order to implement the encoding, one needs to remember that the parameters that generate the trajectories of the joints form something similar to a hierarchy (A, B and C). If amplitude A is different from zero, the joint is going to move from its zero position. For every different value of the speed of motion or frequency B, there are different ranges for the time shifting C. see Section 5.4.

The idea of this hierarchy allowed us to think about creating chromosomes that contain all the information about the motion of a servo joint during the gait cycle, specifically the parameters A, B and C. These chromosomes with one or more genes (parameters) are going to be exchange in the mating pool when performing the crossover process. Four chromosomes have 3 genes, each of these four chromosomes corresponds to the motion of each servo joint that has something to do with the rocking motion of the robot. Three other chromosomes have 1 gene, namely: Step frequency, Shortening factor and Leg extension. Finally the last two chromosomes are the ones that control the pendular motion of the legs. They have 2 genes, frequency and time shift, because the amplitude of these pendular motions is controlled by the constant augmentation in speed. Figure 6.4 depicts the encoding used to optimize the gait G19P.

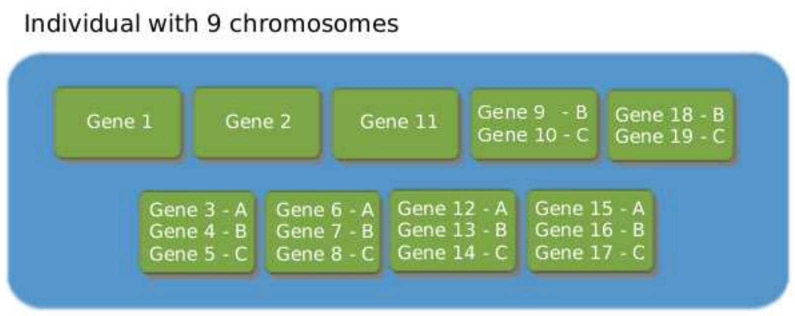


Figure 6.4: G19P Encoding of an individual

The algorithm that optimizes gait G19P is the same one used to optimize the gait G4P. Again, the learning process does not start from zero. An individual x_0 containing 19 parameters that makes the robot walk is inputed into the algorithm as the starting point of the optimization process.

The *fitness function* has a minor change. The fitness of an individual will be the displacement plus the x-component of the displacement, see Figure 6.3. This is done because it was considered that just taking the x-component of the displacement into account was too severe with the individuals that walked long distances but their walking direction was curved. When optimizing just 4 parameters one can afford this type strictness because the search space is small. However, when optimizing 19 parameters the search space is considerably large to afford penalization. Instead of penalizing, a reward *displacementX* for walking straight is used.

$$fitness_i \; = \; displacement + displacementX \tag{6.7}$$

The main differences between the algorithms lie in the function *create new generation*. The functions *elitism* and *evaluation* are exactly the same as the ones used by the previous algorithm.

The algorithm just receives 13 upper and lower bounds as input and not 19, which is the number of parameters to optimize. The other six bounds correspond to the time shift variables (C values). Due to the difficulty of giving the right ranges for the time shift, it was decided that the algorithm had to deal with this with some freedom when setting the limits. The difficulty lies in the fact that for each frequency B, there is a new maximum time shift $maxC$, see Section 5.4. Therefore, every time the frequency was changed, the range for the time shift variable changed to [0,$maxC$].

$$Ub \leftarrow \{ub_1, \ldots, ub_{13}\} \text{ the upper bounds of parameters.}$$
$$Lb \leftarrow \{lb_1, \ldots, lb_{13}\} \text{ the lower bounds of parameters.}$$

The *initialization* function evaluates the firts individual x_0. Then it creates n copies of the starting individual to complete the size of the population with the same individual. Afterwards, the function *create new generation* generates the following generation.

The function *create new generation* is very similar to the one presented for the algorithm that optimizes the gait G4P. The selection of the individuals that are going to exchange the genetic material is carried out in the same way. This algorithm also uses the uniform crossover to select the chromosomes that are going to be part of the new offspring. However, due to the number of parameters, it was considered that adding Gaussian noise every time a crossover is performed is not a desired feature because the good parameters may die out before recombining with other good parameters, eliminating the possibly producing better individuals. Therefore, it was decided that the algorithm has to perform 50% of the recombinations with a normal uniform crossover and the other 50% of the recombinations with the uniform crossover with Gaussian noise, as performed in the last algorithm. Below is described the procedure to perform the exchange of the genetic material.

(6.8)

```
Input: $A \leftarrow$ first selected parent
       $B \leftarrow$ second selected parent
Output: $O$ offspring with 19 genes
 $k \leftarrow 1$
 $decision \leftarrow$ random decision: uniform crossover or noisy crossover
             50% of probability each.

 if ( $decision$ = uniform crossover )
   if (parent A is different to parent B)
     while ( $k \leq nv$ )      $nv = 9$ number of chromosomes
       $O_k \leftarrow$ (select either chromosome $A_k$ or $B_k$, each with 50% of probability)
     end while
   else
     call $CreateNoisyIndividual$
   end if
 else if ( $decision$ = noisy crossover)
   call $CreateNoisyIndividual$
 end if
 return $O$
```

6.8: G19P - Procedure to create a new individual.

One can see in the above procedure that there is a comparison between the parents A and B before performing the uniform crossover. This is done because the roulette wheel may select the same individual twice, and when one tries to exchange the genetic material between two identical parents the result is an offspring equal to the parents. Of course, this is not desired because the algorithm will test again an individual that was already tested. To solve this inconvenience, the algorithm executes a procedure to create a noisy individual.

The procedure 6.9 tries to create offspring with a noisy crossover approximately 50% of the times a new individual is created. A noisy individual uses the parents A and B that were previously selected. From the total number of noisy individuals that are going to be created, 70% of them exchange their genetic material using a similar method to the one used to create individuals in the algorithm that optimized the gait G4P. The other 30% of the parents creating noisy individuals exchange their genetic material in a manner that the new paremeters values can be in all the allowed range. This is done in order to explore other areas of the search space.

The values of $\alpha = 0.25$ and $\beta = 0.025$ were selected after several experiments executed with the simulator. Again, they allow us to calculate the standard deviation σ that will generate a Gaussian noise close to the parameter value of the selected gene. It might happen that the selected chromosome is the same in both parents. When this happens, calculating σ with the genes of the parents will give us a zero stand deviation. To solve this problem, the σ_{min} value is calculated with the help of β and the limits of the parameter. σ_{min} is also used when the value of two parameters are too close to each other. The aforementioned process is executed for all the genes in a chromosome; the number of genes depends on the chromosome, where there are chromosomes with one, two or three genes.

When a chromosome is selected within the 30% of the elements that will be used to generate extreme parameter values, just one of its genes is mutated. The probability of randomly selecting a $gene_i$ is given by $(1/ni)$, where ni is the number of genes in the chromosome. It was decided that just one gene of the chromosome was going to be mutated because this mutation is not as careful as the one performed with the Gaussian noise and it can greatly change the performance of a gait. This radical generation of parameter values is achieved by using a large standard deviation σ, which is calculated with the difference between the limits of the selected $gene_i$ times 0.5. The σ value is big enough to spread the Gaussian bell generating the random values, giving a high probability to all the parameter values in the range.

```
Input: $A \leftarrow$ first selected parent                                        (6.9)
       $B \leftarrow$ second selected parent
Output: $O$ offspring with 19 genes
 $k \leftarrow 1$        $\alpha \leftarrow 0.25$        $\beta \leftarrow 0.025$
 while ( $k \leq nv$ )    where $nv = 9$ Number of chromosomes
 $CH_k \leftarrow$ (select either chromosome $A_k$ or $B_k$, each with 50% of probability)
 $randProbability \leftarrow$ uniform random float between [0,1]
    if ( $randProbability \leq 0.7$)
       for each gene $i$ of the chromosome $CH_k$
          $gene_i \leftarrow CH_{ki}$
          $\mu \leftarrow 0$
          $\sigma \leftarrow \alpha \cdot |A_{ki} - B_{ki}|$    using the gene value of the parents
          $\sigma_{min} \leftarrow \beta \cdot |ub_{ki} - lb_{ki}|$    using the limits of the parameter
          if ( $\sigma < \sigma_{min}$ )
             $\sigma \leftarrow \sigma_{min}$
          end if
          $CH_{ki} \leftarrow gene_i + GaussianNoise(\mu, \sigma)$
          while ( $CH_{ki} < lb_{ki}$ ) or ( $CH_{ki} > ub_{ki}$ )
             $CH_{ki} \leftarrow gene_i + GaussianNoise(\mu, \sigma)$
          end while
       $O_k \leftarrow CH_{ki}$
       end for
    else      − using the other 30% of the selected parents −
       $i \leftarrow$ select a random gene of the current chromosome $CH_k$
            each gene has a equal probability of being selected.
       $\mu \leftarrow 0$
       $\sigma \leftarrow 0.5 \cdot |ub_{ki} - lb_{ki}|$    using the limits of the parameter
       $CH_{ki} \leftarrow 0.5 \cdot |ub_{ki} - lb_{ki}| + GaussianNoise(\mu, \sigma)$
          while ( $CH_{ki} < lb_{ki}$ ) or ( $CH_{ki} > ub_{ki}$ )
             $CH_{ki} \leftarrow 0.5 \cdot |ub_{ki} - lb_{ki}| + GaussianNoise(\mu, \sigma)$
          end while
       $O_k \leftarrow CH_{ki}$
    end if
 $k \leftarrow k + 1$
 end while
 return $O$
```

6.9: G19P - Procedure to create a noisy individual.

The above algorithm exchanges chromosomes with a different number of genes. To ensure the consistency of the chromosomes that contain B and C parameters, every time a B parameter changes, the range of the allowed for the C parameter changes as well. Then, when the crossover is performed, the algorithm exchanges chromosomes with consistent Bs and Cs.

The evolution process of the algorithm that optimizes the gait G19P are presented in the following chapter.

Summary

Chapter 6 introduced the algorithms optimized the gaits G4P and G19P. When presenting the optimization algorithm for the gait G4P, four parameters were shown. These parameters were selected after several experiments with the simulator and with the real robot, as explained in the following chapter. The algorithm that optimized the gait G19P listed all the 19 parameters of the gait.

Chapter 7

Experimental Results

This chapter presents the results obtained with the aforementioned algorithms. The presented results are composed of the evolution of the learning process in both algorithms, as well as selected examples of the analysis of the parameters.

These experiments are divided into simulator experiments and real robot experiments. Clearly, the outcomes with the real robot are the most important ones because they are the ones that demonstrate that the approaches presented here optimized the proposed gaits. Nevertheless, the simulator gave us the possibility to test different genetic algorithms that were then used with the real robot.

7.1 Optimizing the gait G4P

The first experiment that will be presented is one of the early experiments executed with the simulator. At this point, after several other experiments, the learning method was ready but the decision of the parameters to be optimized had not yet taken place. The goal during this experiment was to optimize just three parameters: step frequency, lateral amplitude and the inclination factor. These were selected by observation and experience.

The population size is 15 and the number of generations is 20. The starting individual has the following parameter values:

1. **Step frequency**	$= 3.15$	range[2, 4]
2. **Lateral amplitude**	$= 0.050$	range[0.0, 0.1]
3. **Inclination**	$= 0.0$	range[-0.1, 0.1]

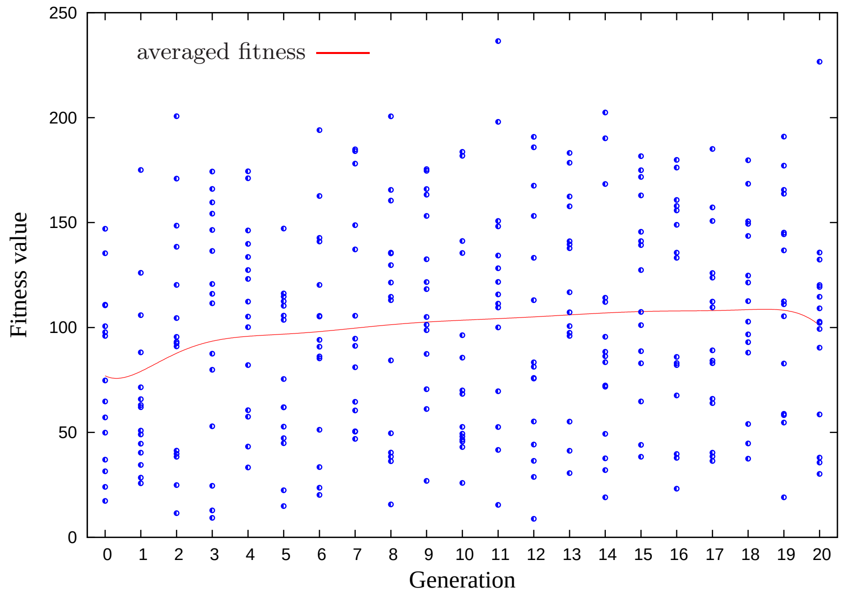


Figure 7.1: Experiment with the simulator trying to optimize 3 parameters

Figure 7.1 shows the fitness evaluation of all the individuals, plotted as single points, during the optimization process. One can see that there are individuals with high fitness values, as well as individuals with low fitness values in each generation. Remember that the fitness value is the x-component of the displacement in *cm*. Therefore, the individuals with a low fitness value are the ones that have parameters that made the robot fall over almost immediately. The curve in the middle of the graphic depicts the evolution of the optimization process. The shape of the this curve was created by drawing an approximation curve of the data points using the smooth Bezier tool of Gnuplot. The Bezier curve is a weighted average of the input points. Therefore, it is possible to use it as an approximation of the average fitness of the entire process.

In the last graphic it is possible to see that the average fitness is increasing during the learning process and that the best individual is not always found in the last generation. Naturally, when selecting which learning algorithms to use, it was only taken into account the slope of the average fitness and not the best individuals found by the algorithm.

In search of higher fitness values, it was decided that a fourth parameter should be integrated in to the optimization process. At first it was not known which parameter should be selected. Again, the selection of this parameter a was done by experimentation with the simulator and observation. Several combinations of four parameters were tested to see if the fitness values showed a larger increase than when optimizing just three. It was also observed during the experiments that the best values of determined parameters sometimes were within an specific area, thus, allowing us to fix the value. For example, the following experiment was performed in the simulator. The analysis of the parameter FootAngleY shows that the best individuals always preferred the value of one of this parameter in a specific area.

The population size used for this experiment was 100 and the number of generations before stopping the evolution process was 40. The starting individual has the following four parameters values:

1. **Step frequency** = 3.0 range[2.5,4]
2. **Lateral amplitude** = 0.050 range[0.03,0.07]
3. **Inclination** = -0.015 range[-1,0.5]
4. **FootAngleY amplitude** = 0.2 range[0,0.4]

Even though the parameter FootAngleY amplitude was later eliminated from the gait G4P, this experiment illustrates how the analysis of the parameters was made. The analysis consists of creating different plots with all the parameters values, looking for consistent behaviors within the data. Figure 7.2 shows a graphic of the step frequency, the FootAngleY amplitude and the fitness values. It possible to visualize that the fittest individuals are concentrated in the right central portion of the graph. Figure 7.3 depicts something similar, although the data belongs to the lateral amplitude, the FootAngleY amplitude and the fitness values. In this case, the area where the fittest individuals are located is bigger. Finally, Figure 7.4 shows the plot of the robot inclination, the FootAngleY amplitude and the fitness values. Again, one can see that the fittest individuals are located in a similar area as that of the two previews graphics.

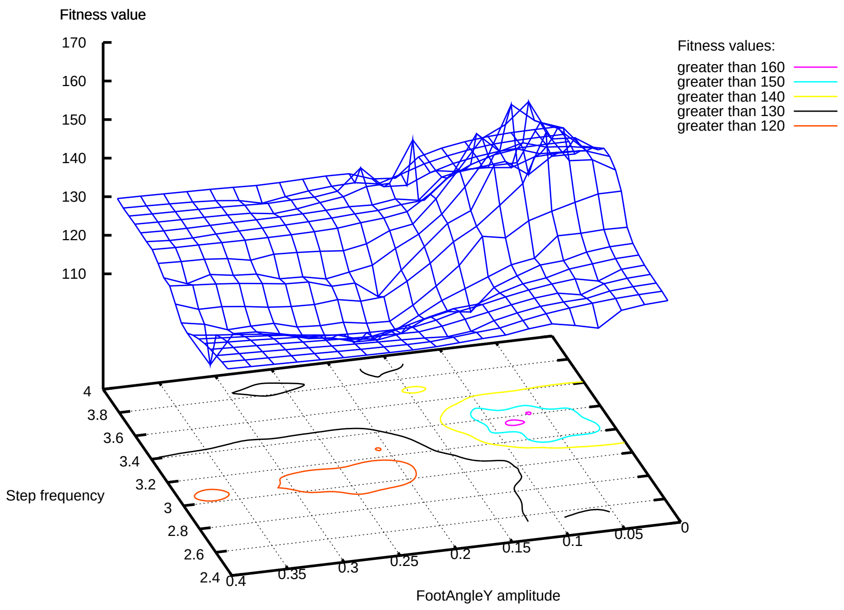


Figure 7.2: Example of parameter analysis

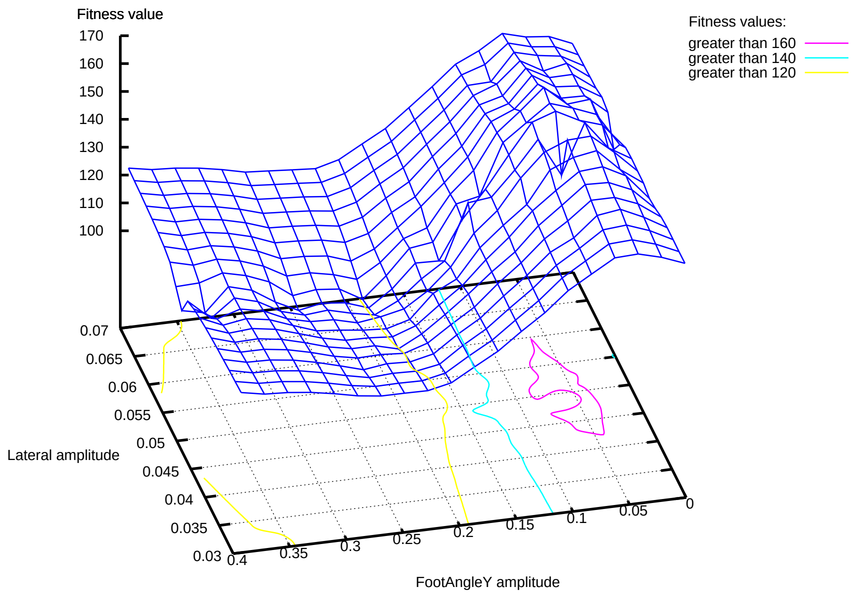


Figure 7.3: Example of parameter analysis

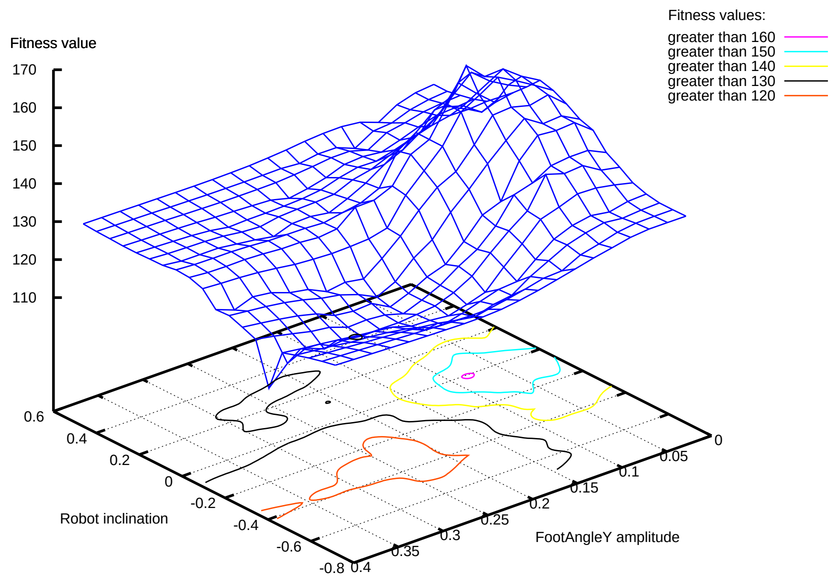


Figure 7.4: Example of parameter analysis

After observing the graphics one can conclude that the values of the FootAngleY amplitude that produce the best individuals are within a specific visual range, specifically the pick area. Therefore, the value of the FootAngleY amplitude was fixed to 0.075, which is included in the area where the best individuals are located in the three graphics.

Several other experiments were executed using different parameters trying to find the meaningful ones before experimenting with the real robot. The next experiment shows how the learning process evolved in the simulator with a population size = 50, a generation number = 31 and the parameter values of the first individual:

1. **Step frequency** = 3.0	range[2,4]
2. **Lateral amplitude** = 0.030	range[0.0,0.06]
3. **Swing start** = 1.2	range[0.7,1.9]
4. **Shortening factor amplitude** = 0.15	range[0,0.3]

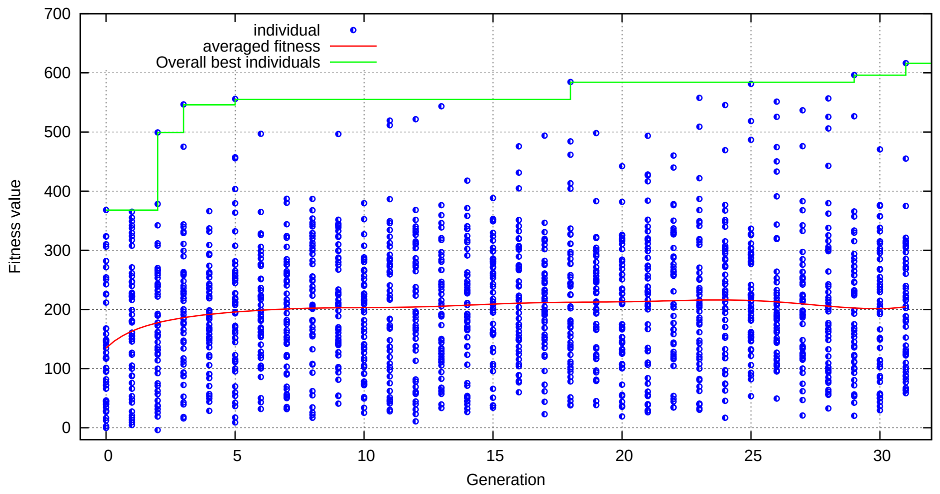


Figure 7.5: Experiment with the simulator trying to optimize 4 parameters

Again, it is possible to visualize in Figure 7.5 that the slope of the averaged fitness curve is always increasing until the last five generations, where the average decreases a little. In Figure 7.5 the overall best individuals of the population are depicted. Moreover, the fitness values are much higher than when just trying to optimize three variables, see Figure 7.1. One can see better fitness values because experimenting with four variables allow a better optimization.

Other important data plots visualize the evolution of the variables over time. The following image (Figure 7.6) shows the evolution of the parameter *Swing start*.

Here it is possible to see that the algorithm produces something similar to a branching effect, where most of the values in the population follow a big trunk and little branches try to search new parameter values. These little branches die out when the parameter value does not produce good individuals.

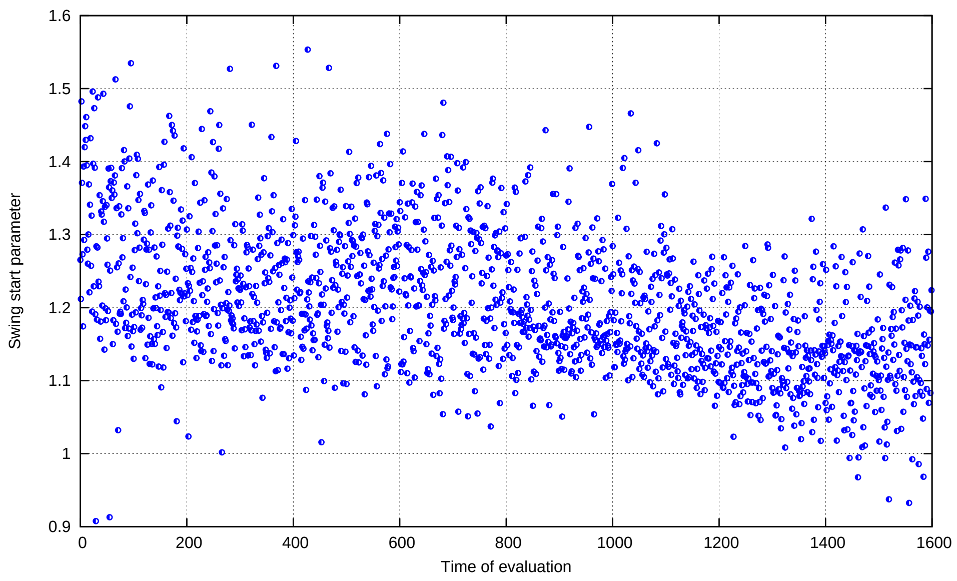


Figure 7.6: Evolution of one parameter over time

After several experiments with different combinations of parameters were run in the simulator, we had a relatively a good idea of which parameters could be fixed and which parameters be suitable for optimization. Therefore, the following experiments were performed with the real robot.

In the first attempt to optimize the gait G4P with the real robot, the selected parameters were:

1. **Step frequency** = 3.0	range[2.5,3.5]
2. **Lateral amplitude** = 0.041	range[0.038, 0.045]
3. **Inclination** = -0.07	range[-0.10, -0.04]
4. **Shortening factor amplitude** = 0.15	range[0.1 0.2]

This initial individual produces the forward motion of the robot with a maximum speed of $5cm/s$ and a fitness value of $160.5cm$. The population size is 12 and the number of generations is 12.

Figure 7.7 depicts the fitness of each individual as individual points, the red curve that represents the average fitness of the entire process, and the green line depicts the overall best individuals. One can see that none of the

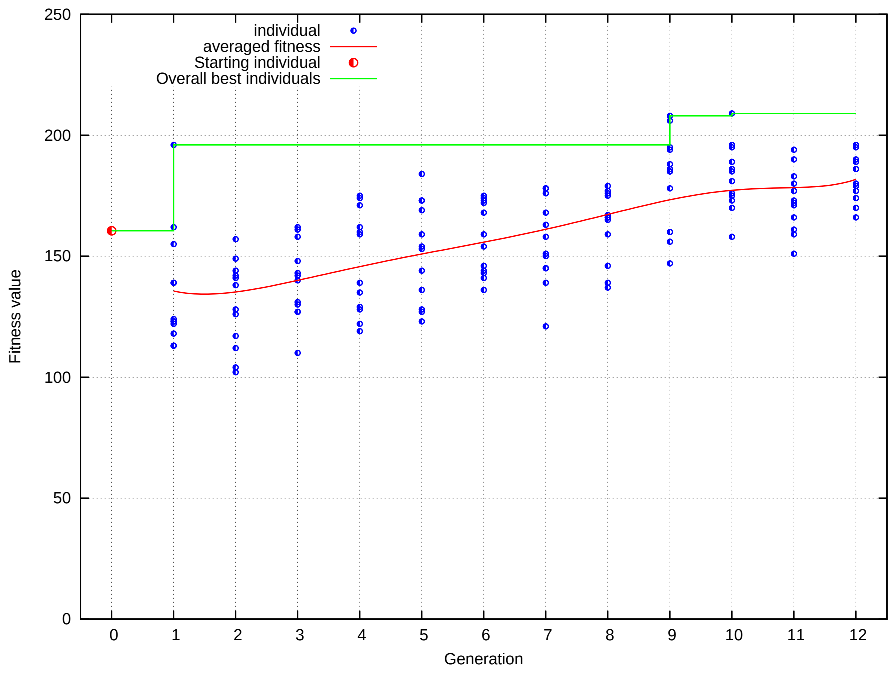


Figure 7.7: Learning process of the first experiment with the real robot

individuals made the robot fall over immediately, they all made the robot walk for more than one meter. Even though the results of the experiment look promising, when testing the best individuals found during the whole process, the robot was able to reach a maximum stable speed of only $7cm/s$.

The maximum speed reached by robot was not satisfactory. Therefore, it was decided to look for other parameters that would positively influence the gait speed of the robot.

The new parameter that would be included in the optimization process is the $LegExtension$ (see Section 5.1). The decision of optimizing this parameter was made because when testing values different than 1, it was realized that the speed of the robot increased considerably.

In the new experiment, the initial individual was able to reach a maximum speed of $11.5cm/s$, clearly much more than $7cm/s$. After testing several values, the parameter Shortening factor amplitude, optimized in the last experiment, was fixed to 0.3. In the end the following parameters were the ones that were used for gait G4P.

The parameter values of the initial individual are:

1. **Step frequency** = 3.0	range[2.5,3.5]
2. **Lateral amplitude** = 0.041	range[0.03, 0.08]
3. **Inclination** = -0.07	range[-0.10, -0.04]
4. **Leg extension K** = 0.96	range[0.92,1.0]

Again, the population size was set to 12 and the number of generations was set to 12. Making a total of 144 individuals to be evaluated plus the initial one.

Figure 7.8 shows the evolution process when optimizing the individual that was able to reach 11.5cm/s. The slope of the learning curve is almost unchanging, and sometimes the average fitness of one generation decreases below the average fitness of the starting generation. This is due to the fact that the starting individual has already a very good fitness value, therefore improvement is more difficult.

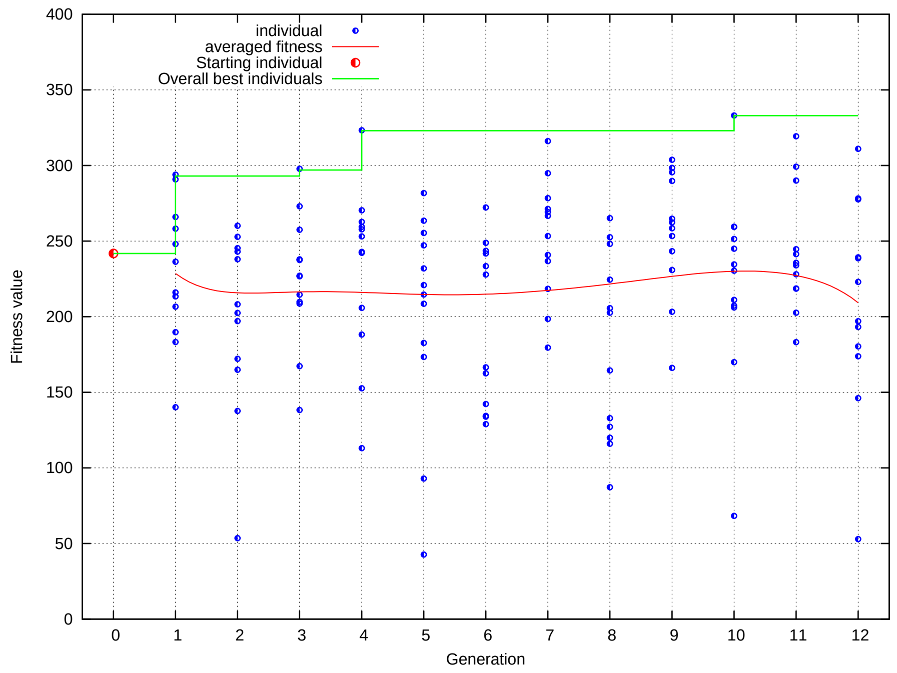


Figure 7.8: Gait G4P optimization experiment with the real robot

Despite the difficulty of optimizing an already fit individual, the algorithm found fitter individuals. This is depicted by the green line that shows the overall best individuals in Figure 7.8. The parameter values of the best individual found are:

1. **Step frequency** = 2.8802
2. **Lateral amplitude** = 0.0537
3. **Inclination** = -0.0177
4. **Leg extension K** = 0.9682

The above parameters make the robot reach a maximum speed of $14cm/s$. Naturally, one might find individuals with better parameters if the evolution process is continued, but it was decided that the number of evaluations of the fitness function should be kept low because the amount of time to evaluate a generation of 12-individuals is 45 minutes. Clearly, this long evaluation time is a drawback, which was brought about because during this investigation the evaluation of the fitness function was completely manual.

Now, let us analyze some of the data output of this experiment. The first graphic shows the data plot of the behavior of the leg extension variable over time (Figure 7.9).

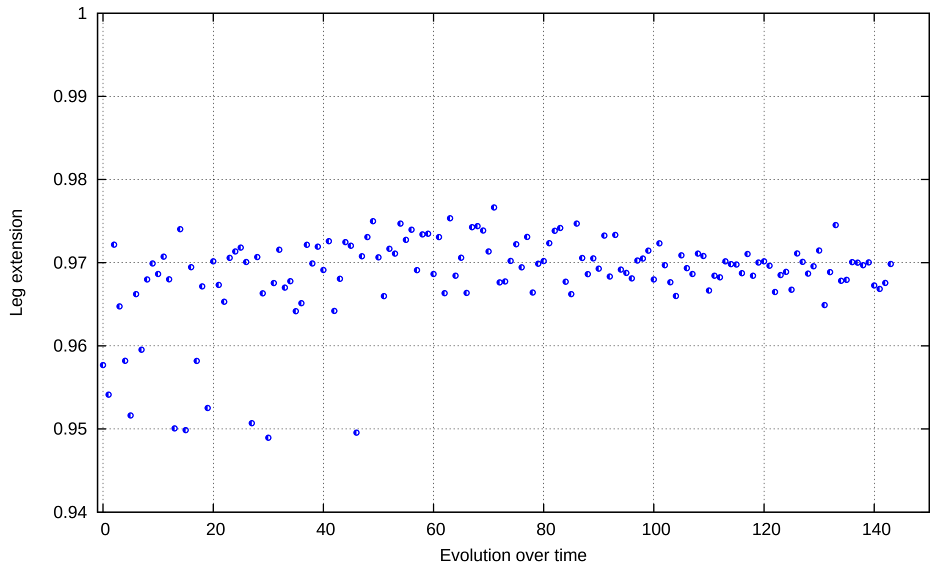


Figure 7.9: Evolution of the Leg extension parameter

One can see that the algorithm preferred values above 0.96, which was the starting value. When a branch of the trunk was getting close to 0.98, the algorithm realized that the fittest individuals were below this value. In order to see this situation, a new graphic will be presented (Figure 7.10).

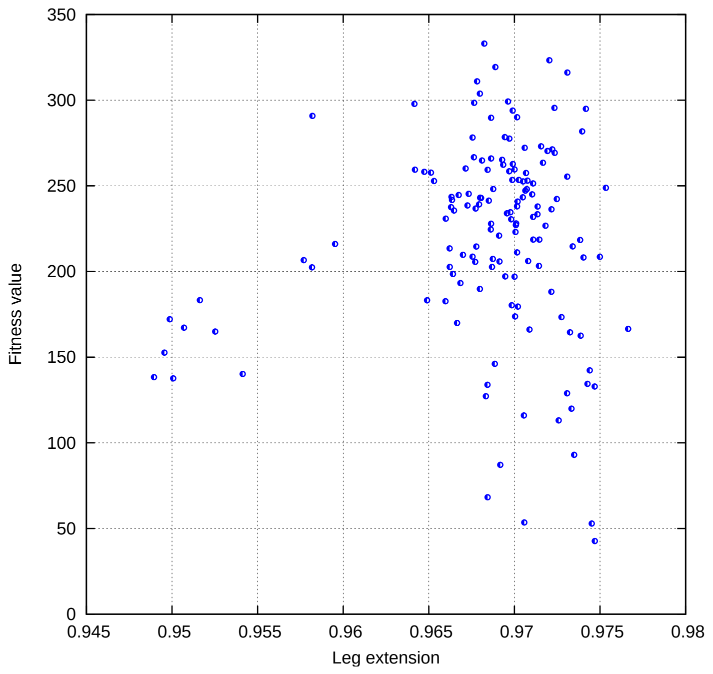


Figure 7.10: Fitness value and Leg extension

From the above graph, one can realize that the good values for the parameter $LegExtension$ are between 0.965 and 0.975.

Figure 7.11 shows the plots of the other three parameters (step frequency, lateral amplitude and inclination) against the fitness value obtained during the optimization process. (a) Shows that the best values for the step frequency are between 2.85Hz to 3Hz. (b) Shows that the best values of the lateral amplitude are between 0.05 to 0.055 radians. (c) Shows that the best values for the inclination parameter are between -0.014 to -0.018 radians.

As afore mentioned, there might be better parameter values than the ones presented, if the evolution process was continued. Nevertheless, $14 cm/s$ for such a small robot that is also carrying a pocket pc and a camera is a very good result in comparison with the initial speed.

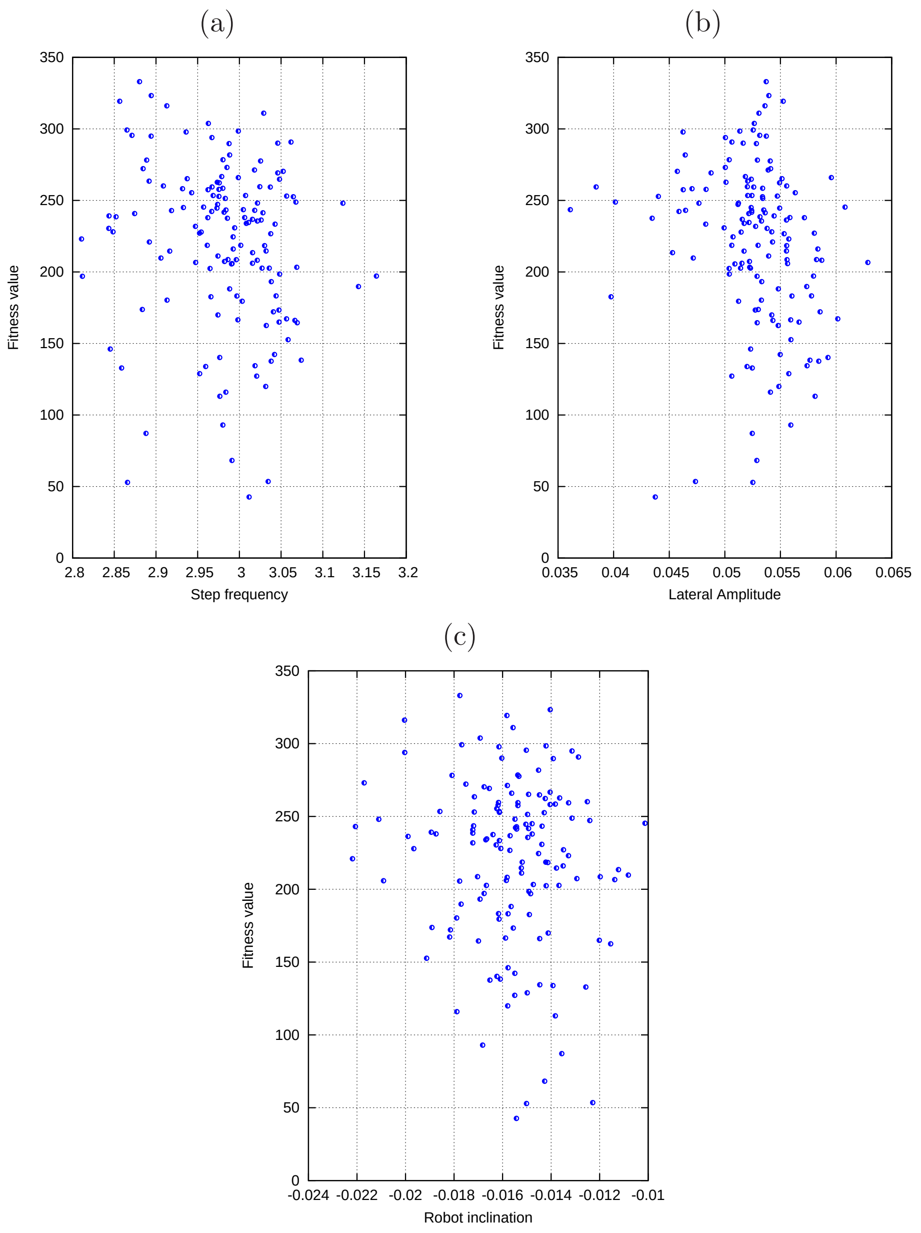


Figure 7.11: Step frequency, lateral amplitude and robot inclination

7.2 Optimizing the gait G19P

When optimizing the gait G19P, the problem was not selecting which variables to train, because all the parameters are selected to be optimized, but instead it was creating the right algorithm after several experiments with the simulator that would be able to handle 19 variables at the same time. Also in this case, the optimization had to be done in relatively few evaluations of the fitness function due to the large amount of time needed to carry out the experiments.

In order to test the effectiveness of a determined genetic algorithm, the simulator was used. Many experiments were executed in order to test different values of the algorithm's variables. Each change improved or worsened the performance of the optimization process. Nevertheless, the method proposed here is the one with the best performance.

Other important characteristic of the simulator is that one is able to set the value of the friction with the ground. It was realized that low values affect the learning process in an interesting way. When learning to walk, the genetic algorithm realized that lifting the foot increases the possibility of falling. Therefore, if the friction value is low, the algorithm will not see the necessity of lifting the foot during the swing phase. This was the case in one of the early experiments, where the robot learned to walk without lifting the foot by using a skating or shuffling movement. After increasing the value of the friction, a second experiment was executed. The value was still too low, because the algorithm found the way of not lifting the foot by keeping a small lateral part of the foot plate in contact with the ground, thus minimizing the friction.

These results were a clear sign that the learning method was working. Nevertheless, it was also a problem because the manner of walking was not the desired one. To solve the problem, a considerably large friction value was set in the simulator; pushing the learning method to lift the foot and not leaving space for unwanted solutions.

The following experiment shows the performance in the simulator of the best trained algorithm. The population size is 12 and the number of generations is also 12. The starting individual had the following parameter values:

1. Step frequency = 3.262
2. Leg Extension = 0.960

Support footAngleX

3. amplitude (A) = 0.140
4. frequency (B) = 2.000
5. time shift (C) = 0.000

Support hipAngleX

6. amplitude (A) = 0.000
7. frequency (B) = 2.000
8. time shift (C) = 0.130

Support motion

9. frequency (B) = 1.000
10. time shift (C) = 0.500

11. Shortening Factor = 0.075

Swing footAngleX

12. amplitude (A) = 0.0
13. frequency (B) = 2.50
14. time shift (C) = 0.0

Swing hipAngleX

15. amplitude (A) = 0.075
16. frequency (B) = 3.000
17. time shift (C) = 1.00

Swing motion

18. frequency (B) = 2.600
19. time shift (C) = 2.600

The range for the parameters are the following ones:

Gene	*range*	*Gene*	*range*	*Gene*	*range*
1	$[2.5, 4.5]$	9	$[1, 3]$	18	$[1, 3]$
2	$[0.9, 1.0]$	11	$[0.05, 0.2]$		
3	$[0.0, 0.35]$	12	$[0, 0.175]$		
4	$[2, 3.5]$	13	$[2, 3.5]$		
6	$[0, 0.0875]$	15	$[0, 0.175]$		
7	$[2, 3.5]$	16	$[2, 3.5]$		

One must remember that the ranges of the time shift variables depend on the corresponding frequency parameter. For more detail see section 6.2.2.

The following image depicts the evolution of the optimization process (Figure 7.12). It is possible to see that the first generation did not produce fitter individuals than the starting one. But in successive generations the algorithm found fitter individuals. If one sees the slope of the learning curve, one can realize that the average fitness is always increasing. An other important point to mention is that the algorithm produces individuals that make the robot fall almost immediately, this effect is a consequence of trying to optimize many variables and also because some individuals are created with the extreme Gaussian noise. Clearly, this notably affects the average fitness of the generation, but nevertheless the algorithm was able to handle these inconveniences.

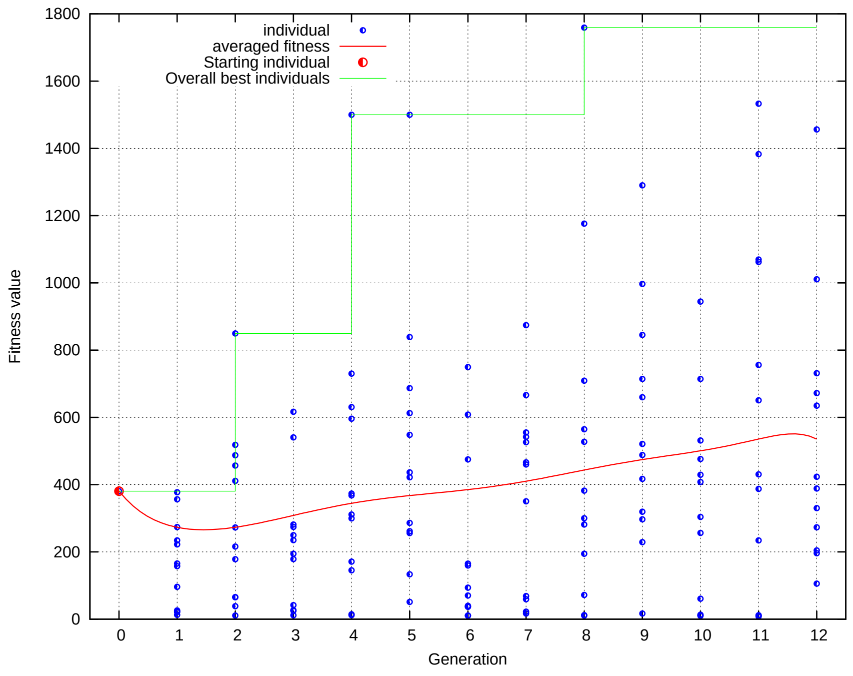


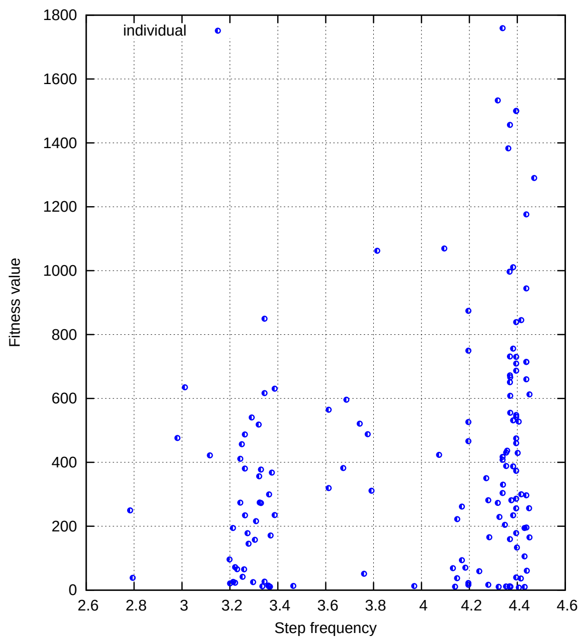
Figure 7.12: Evolution of the optimization process in the simulator

The fitness values are different from those values obtained when training the gait G4P because the fitness function is the total displacement plus the x-component of the displacement, for more detail see section 6.2.2.

Figure 7.12 shows a green line that represents the overall best individuals of the evolution process. One can observe that at the end of the process, the best individual has a much higher fitness than the starting individual (almost 5 times). Of course, this effect is desired but not expected when performing experiments with the real robot. One must take into account that the evolution process starts with a fit individual, from which it is difficult to find fitter individuals.

Following is an explanation of the evolution of some of the parameters during this experiment in the simulator. First, let us present the plot of the step frequency against fitness value, see Figure 7.13(a). In this graph it is possible to see that the parameter values that produce the best evaluations are close to 4.4Hz. The second graph, Figure 7.13(b), plots the evolution over time of this parameter. One can observe that a branching effect is also present, although the first branch dies out after some evaluations of the fitness function. A second branch starts growing in another place most likely because the extreme Gaussian noise produced a parameter value that generated fitter individuals.

(a)



(b)

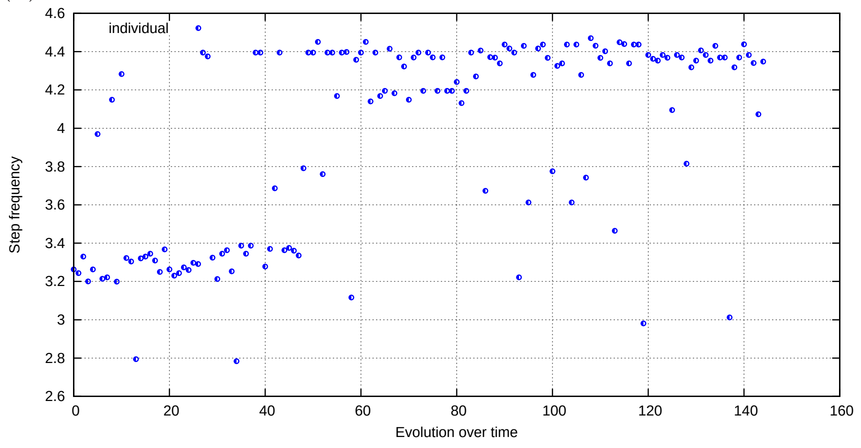
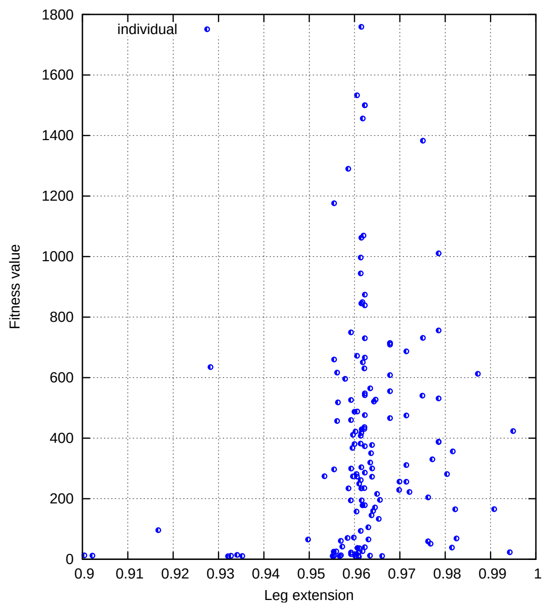


Figure 7.13: Step frequency data plots

The second parameter to be analyzed is the leg extension. Two plots are presented in order to inspect the evolution of this parameter in a similar way to the step frequency.

(a)



(b)

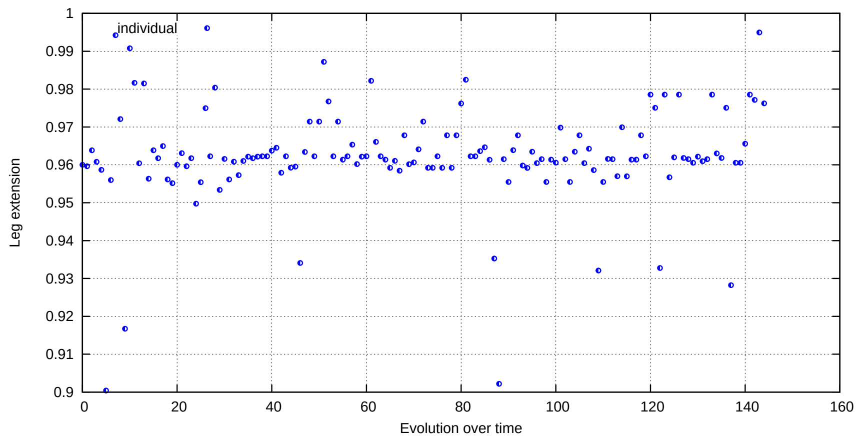


Figure 7.14: Leg extension data plots

In this case, there is just one branch during the evolution of this parameter, as can be seen in Figure 7.14(b). Figure 7.14(a) shows that the best individuals have a leg extension close to 0.96.

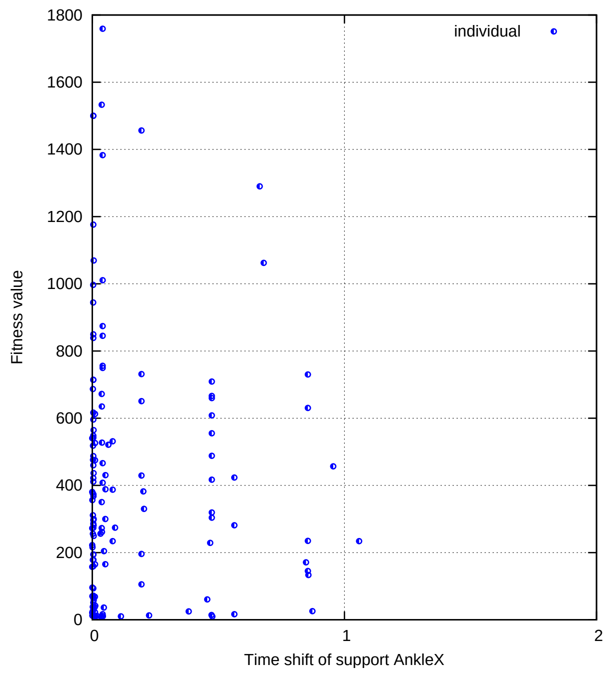


Figure 7.15: Parameter C of the support AnkleX joint

The plot in Figure 7.15 presents the time shift parameter at the support foot. This plot shows that the best individuals prefer a time shift close to zero, meaning that the motion of the ankle starts when the time dedicated to the support phase starts.

In a similar way, it is possible to analyze the rest of the parameters in order to get some idea of what is going to happen when experimenting with the real robot.

Following is an experiment executed with the real robot, although this time the number of evaluations is under a 100. To be precise, the population size is 12 and the number of generations is 8. The starting individual is able to reach a maximum speed of $8.5cm/s$. The parameters corresponding to this individual, as well as the limits, are the ones mentioned above.

First of all, let us present the plot of the the individuals in a generation against their corresponding fitness. Figure 7.16 depicts this data plot.

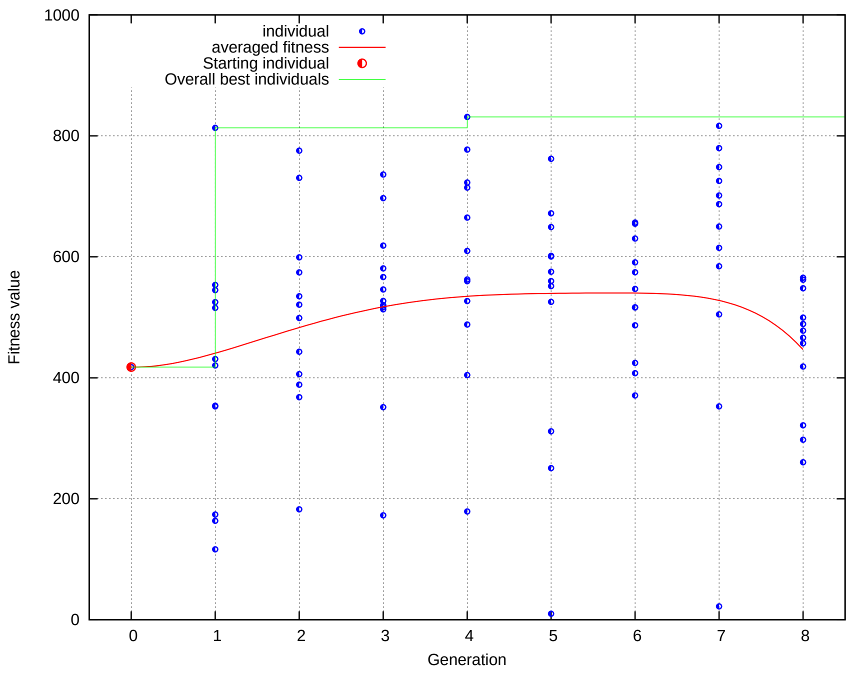


Figure 7.16: Averaged fitness and overall best individual during the optimization experiment.

In the above graph, one can see that the curve of the average fitness tends to improve until the last generation, where the fitness average drastically dropped. The algorithm found a very fit individual during the first generation, which was very difficult to improve. In order to see the overall best individuals on needs to look at the green line in Figure 7.16.

After testing the best individual found by the algorithm, it was realized that the maximum stable speed was $14cm/s$. Amazingly, this speed was also reached by the gait G4P. One can think of $14cm/s$ as the maximum stable speed reachable by the robot with the two proposed gaits G4P and G19P.

As done with the other experiments, let us analyze some of the parameters with the data outputted from the experiment.

It is interesting to see the behavior of the time shift parameter of the support and swing motions in Figure 7.17. This plot shows that the time shift parameter of the inverted pendulum motion has to be close to zero and smaller than the time shift parameter of normal pendulum motion. This result agrees with the fact that the robot has to move the body weight to the support leg, send the body weight forwards and finally proceed to lift the swinging leg.

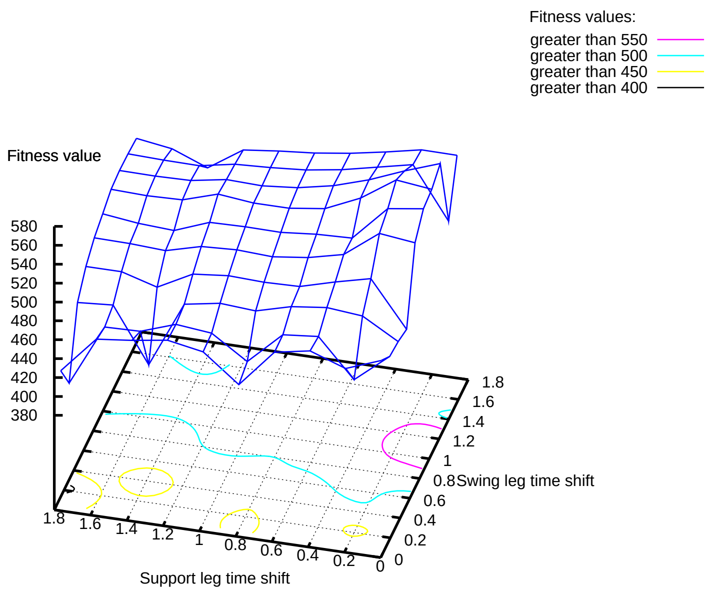


Figure 7.17: Time shift parameters of pendular motions against fitness

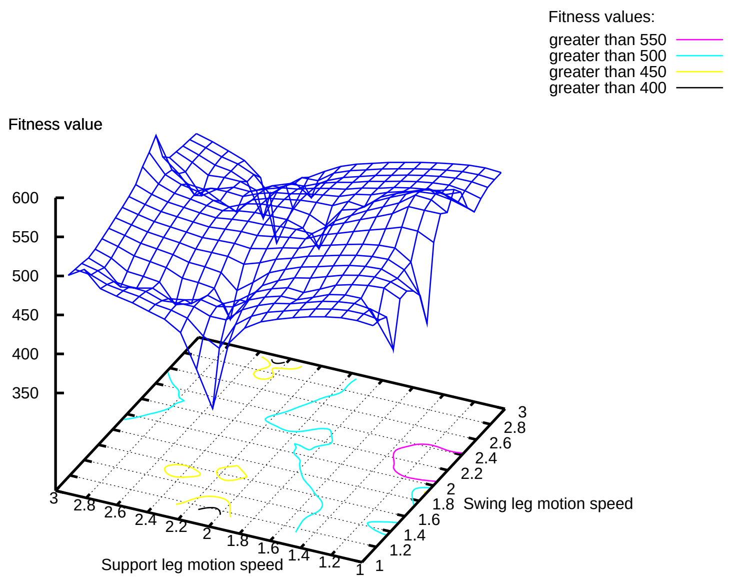


Figure 7.18: Speed parameters of pendular motions against fitness

If one thinks about the speed of the pendular motions, one might think that the swinging motion has to be faster than the support motion because this is what humans do when walking. In fact, the algorithm found that the best individuals have this characteristic, which is depicted in Figure 7.18.

Figure 7.18 shows that the best parameter values for the speed of the support motion are close to 1 (the slowest frequency) and that the parameter values for the swing motion are twice as fast. With the gait G4P, a similar behavior was intended by using a gentle motion of the support leg and a faster swing motion.

Let us do a similar analysis using the time shift parameters of the ankle at the support leg and the swing motion. If one remembers the rocking motion of the robot, one can easily intuit that the motion at the ankle must start before the swing motion. If the robot tries to swing the leg before it has the weight of the body on the support leg, it will immediately fall. The algorithm realizes this situation and tries to use parameter values that start the motion of ankle before swinging and lifting the leg. Figure 7.19 shows that when the value of the time shift at the ankle is greater than the time shift of the swing motion, the individuals have a bad fitness value. This is visible in the area that has a black circle.

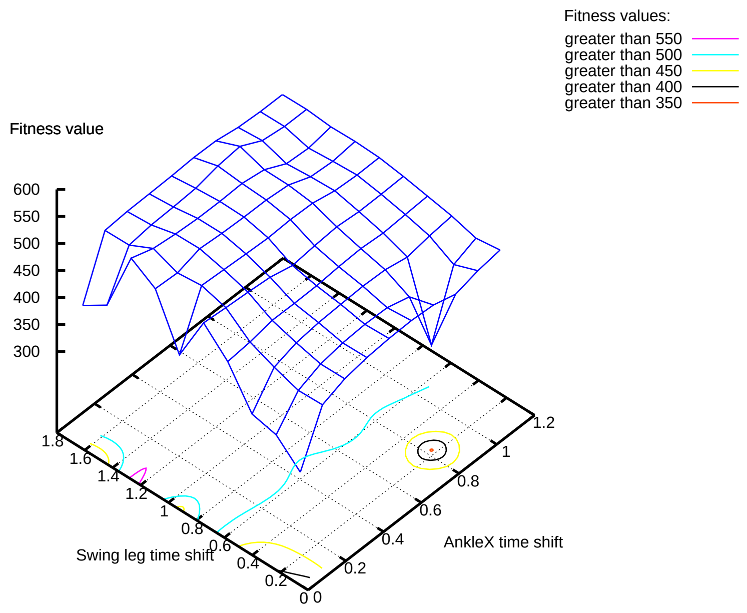


Figure 7.19: Time shift parameters of the support ankle and leg swing motion against fitness

The next parameter to be presented is the step frequency. Contrary to the results in the simulator, which have step frequencies higher than $4Hz$, the algorithm found that the best frequencies are between $3Hz$ and $3.4Hz$. This is similar to the results obtained when experimenting with the gait G4P. Figure 7.20 depicts this situation.

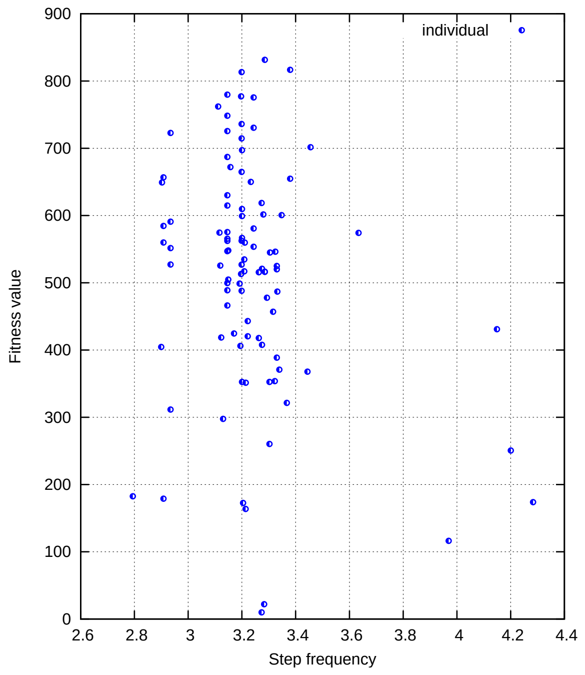


Figure 7.20: Behavior of the step frequency

Figure 7.21 shows that the algorithm found values for the leg extension parameter different from 1 (100% full leg extension) that allowed the robot to walk faster. In this case, there are two visible areas where the algorithm found fit individuals, one between 0.915-0.93 and the other close to 0.96.

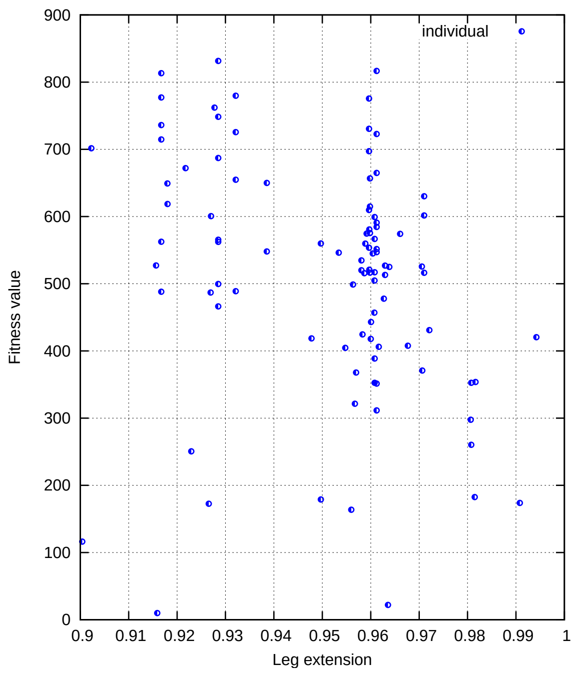


Figure 7.21: Behavior of the leg extension

The values of the 19 optimized parameters from the best individual are:

1. Step frequency = 3.2852
2. Leg Extension = 0.92850

Support footAngleX

3. amplitude $(A_1) = 0.11773$
4. frequency $(B_1) = 2.1250$
5. time shift $(C_1) = 0.0112$

Support hipAngleX

6. amplitude $(A_3) = 0.0054$
7. frequency $(B_3) = 2.0641$
8. time shift $(C_3) = 0.1265$

Support motion

9. frequency $(B_5) = 1.0678$
10. time shift $(C_5) = 0.0126$

11. Shortening Factor = 0.11234

Swing footAngleX

12. amplitude $(A_2) = 0.00670$
13. frequency $(B_2) = 2.4125$
14. time shift $(C_2) = 0.06249$

Swing hipAngleX

15. amplitude $(A_4) = 0.0681$
16. frequency $(B_4) = 2.5670$
17. time shift $(C_4) = 1.0605$

Swing motion

18. frequency $(B_6) = 2.3404$
19. time shift $(C_6) = 2.4800$

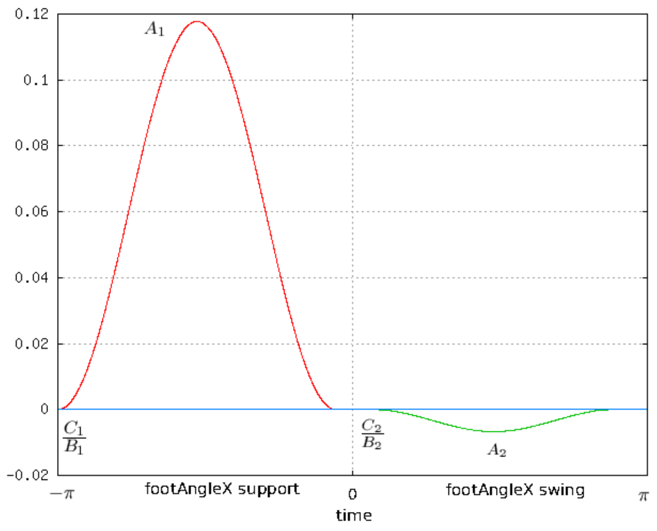


Figure 7.22: Motion of the ankleX joints

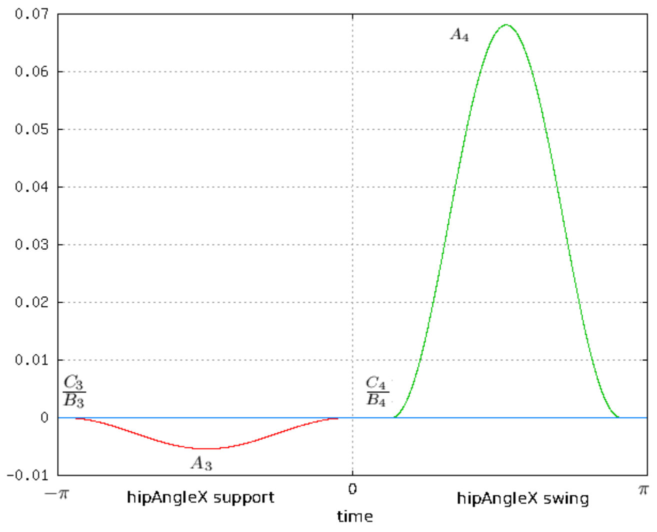


Figure 7.23: Motion of the hipX joints

Figures 7.22-7.23 depict the trajectories of the servos at the ankles and the hip. If one looks at the parameter values of the best individual, there are two values that are really small that can be set to zero (amplitude Support hipAngleX = 0.0054, amplitude Swing footAngleX = 0.00670). Since these parameters are amplitudes, one can realize that these two servo joints are not moving during their corresponding phases. The effect of not moving these two joints is that the joint in charge of changing the body weight to the support leg is principally the ankle joint of the support leg. This is different from the assumption of the gait G4P that the robot has to move four joints when changing the body weight to the support leg. The following data plots show these amplitude values against fitness.

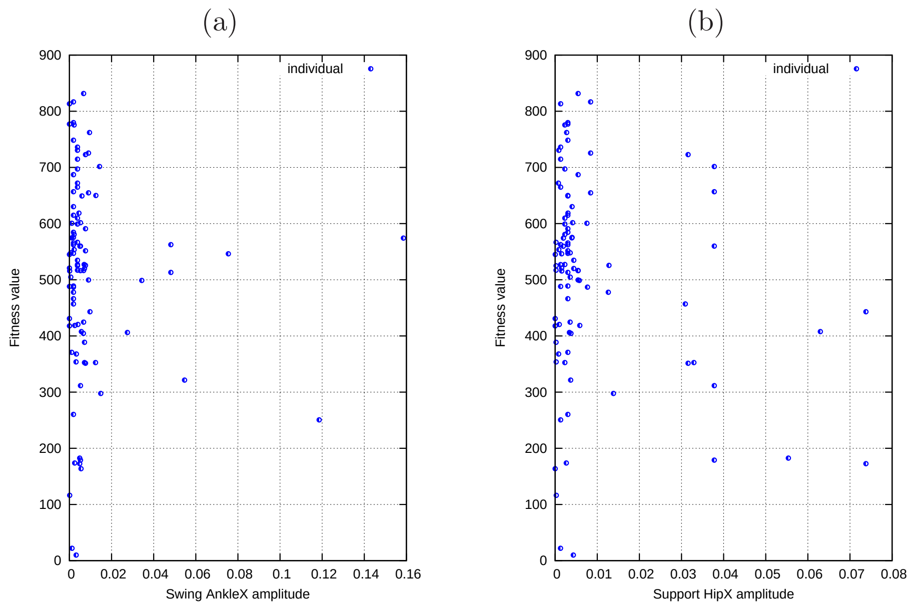


Figure 7.24: Amplitude values of swing AnkleX and support HipX

Figures 7.24(a) shows that the fittest individuals found by the algorithm have the amplitude value of the swing ankleX close to zero. Figures 7.24(b) shows something similar for the amplitude value of the support hipX, although, this plot also shows that there are fit individuals which prefer some motion at the hipX joint to help reduce the vertical displacement of the center of mass.

Figure 7.25 shows the trajectories of the support and swing phases. One can see that the swing motion is faster than the support motion. The frequency of the support motion is 1.0678 and the frequency of the swing motion is 2.3404. The phase shift of the support motion is almost zero, whereas the phase shift of the swing motion is much larger, meaning that the support motion starts before the swing motion.

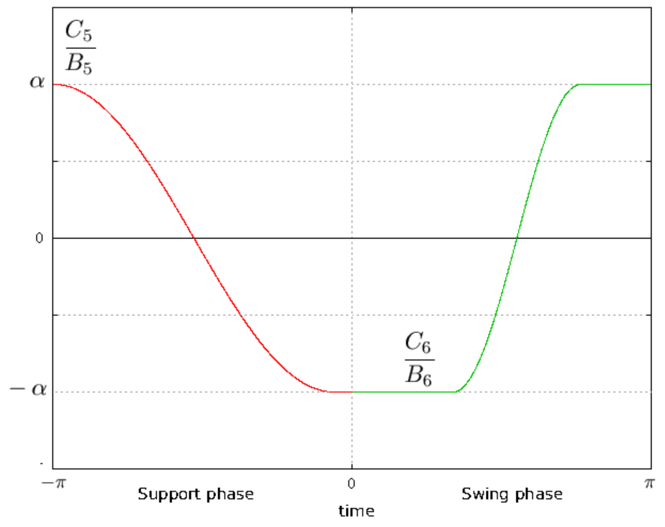


Figure 7.25: Support and swing motions

Naturally, one can think of analyzing many other parameters. Nevertheless, the parameters here presented are the ones that are thought to be the most important ones.

7.3 Comparison of the gait engines

The last two sections presented the results obtained when optimizing each of the proposed gait engines. During this section the pros and cons of both gaits are presented.

Let us start with the number of parameters that were optimized in each of the gait engines. The gait G4P: 4 parameters, see Table 6.1. The gait G19P: 19 parameters, see Table 6.2.

Optimizing 4 parameters is computationally simpler because the search space is smaller. However, the problem arises with the selection of these parameters, if the selection is not correct the optimization can be very poor. This means that more time is taken up by having to pick the parameters by trial and error. When optimizing 19 parameters, the problem of selecting parameters does not exist because they are all selected to be optimized. However, the disadvantage of gait G19P comes with the large size of the search space.

The gait G4P assumes that the time of execution and the speed of the pendular motions are the same for all the individuals in the optimization process. The gait G19P allows different speeds and different time shifts for each individual. See Sections 6.2.1 and 6.2.2. It was shown in Figures 7.17 and 7.18 that assuming that the inverted pendular motion of the support leg is slower and has to start before the normal pendular motion of the swing leg is correct. This means that one can fix the pendular motions without affecting the performance of the optimization of the gait.

During the experiments, both gaits concurred that walking with the legs not fully extended permits walking faster. See Figures 7.10 and 7.21.

Another concordance between the two gaits is that the best step frequencies are close to $3Hz$. See Figures 7.20 and 7.11.

After examining the parameter values of the gait G19P, it was realized that two of the servo joints do not move. One is the hipX joint when performing the support phase, the other joint is the ankleX when executing the swing phase. This means that the gait G19P does not uses all four joints of the rocking motion, explained in Section 5.1, to change the weight of the body to the support leg; it just uses the ankleX joint at the support leg, see Figures 7.24(a)(b). One benefit of not moving two servo joints for a period of time is that it helps to save some energy of the batteries. On the other hand gait G4P uses all four servos to execute the rocking motion. Since these values in gait G4P were fixed, there was no chance that the genetic algorithm could have found this variation used by gait G19P.

If one compares the two algorithms used to optimize both gaits, it can be seen that new parameter values are being tested at all times, specifically with the algorithm optimizing the gait G4P. Given that the search space is small, one can afford testing new values with every new individual, see Section 6.2.1. On the other hand, the algorithm that optimizes the gait G19P also tries new parameters values and also reuses the best parameter values found in the new individuals. Due to the number of parameters (19), one can not afford losing good parameter values in each recombination, thus the carrying over of the best parameters into the new individuals, see Section 6.2.2.

Even though the creation of new parameter values is very similar in both algorithms, the algorithm used to train the gait G19P has a method for creating parameters not too close to each other. This has the effect of creating new branches when good individuals are found in different areas of the seach space, see Figure 7.13 (a) and (b). Besides searching in different areas, the creation of new parameter values without caution, introduces the danger of finding not desired values that might negatively affect the performance of the gait engine. For this reason, the creation of these extreme values is sporadic, see Section 6.2.2.

One more difference between the optimization algorithms is the fitness evaluation function, see Equations 6.5 and 6.7. The fitness evaluation for the individuals corresponding to the gait G4P is the x-component of the displacement, whereas the fitness evaluation for the gait G19P is the total displacement plus the x-component of the displacement. The fitness function of gait G4P is more rigorous when evaluating individuals that did not walk in a straight direction, however, optimizing just four parameters allows one to be more strict with this type of individual. When optimizing 19 parameters, one can not afford such a rigorous penalization of individuals. Therefore, instead of penalizing, a reward is given to the individuals that walked in a straight direction.

Optimizing individuals that have already a good fitness value is very difficult, however, both algorithms were able to achieve this. The optimization of the gait G4P started with an individual that was able to reach $11.5cm/s$, and the optimization of the gait G19P started with an individual that was capable of walking at $8.5cm/s$. The final stable speed for both gaits is $14cm/s$.

At the end of the experimental phase, both algorithms found individuals that make the robot walk stably at the same maximum speed. However, when observing the performance of both gaits over the carpet, the gait G4P seams to be sensible to the bulges over the walking surface. Whereas the gait G19P is not as sensible.

Finally, a set of images that depict the robot KHR-1 performing a walking cycle are presented. These images were taken when the robot was walking at $14cm/s$. One can see that the robot does not need to lift much the foot plate in order to perform a step.

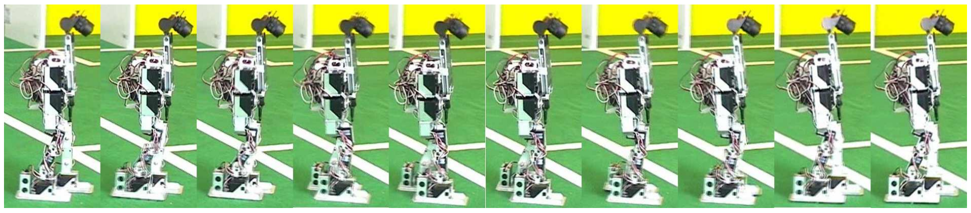


Figure 7.26: KHR-1 walking sequence

Summary

During this chapter, the results of the experiments performed with the real and simulated robot were presented. The optimization of both gaits, G4P and G19P, demonstrate the successful outcome of $14cm/s$, as the maximum speed achievable by humanoid robot KHR-1.

Several data plots were presented in order to describe the evolution of the optimization processes, as well as the behavior of some parameters.

At the end of the chapter a comparision between the two gait engines is elaborated, describing their pros and cons.

Chapter 8

Conclusions

The problem of optimizing the gait of a humanoid robot is not as easy as one may think. There are numerous parameters to take into account, each of which influences in a different manner the performance of the gait. During this research, two gait engines were implemented in order to make the robot walk. Nevertheless, the majority of the attention was focused in trying to find the best parameter values that would make the robot walk as fast as possible.

It has been shown in Chapter 7 that genetic algorithms are an effective way to optimize the parameters of a gait engine, not only because the robot is now able to walk fast and stable, but also because the solutions were found in relatively few evaluations of the fitness function.

The above solutions were achieved by training parameter optimization machines with the help of a simulator. Even though it was never expected that the parameters from the simulator worked in reality, it was expected that the optimization methods were transferable from the simulated world to the real world.

It has also been shown in Chapter 7 that the original postulation, which assumes that there is one set of parameter values that would make the robot accelerate through a finite number of discrete speeds before reaching a maximum high-speed, is a good way of reducing the complexity of the bipedal walk.

The fact that the humanoid robot is able to accelerate from a zero speed to a seemingly high walking speed and vice versa, is a very useful feature that can be applied in the RoboCup humanoid soccer league [30], where the bipedal robots need to stably walk at different speeds. In fact, these two gaits have already demonstrated at the RoboCup German Open 2005 [31] that they are much faster and more stable than other aproaches [32].

During this work, the only parts of the body taken into account were the ones belonging to the locomotive aparatus i.e. the legs. As future work the movement of the arms could be introduced in order to see their influence in the performance of the gait.

Another area for improvement is the evaluation of the fitness when experimenting the real robot because at this point the measurements were done by hand, consuming a large amount of time, as well as introducing a certain level of inacuracy to the measurements. This process could be improved by using a camera positinioned over the walking surface and then automatically taking the measurements from the image.

The addition of sensors to the robot, for example a gyro for balance and servos that give feedback of their current possition, would allow the implementation of similar learning processes where the robot could detect when it is loosing its balance and correct itself before falling.

Appendix

Home position of KHR-1

Upper body		Lower body	
Servo Channel	Home position (degrees)	Servo Channel	Home position (degrees)
$CH1$	6	$CH13$	133
$CH2$	2	$CH14$	114
$CH3$	90	$CH15$	116
$CH4$	0	$CH16$	93
$CH5$	0	$CH17$	89
$CH6$	90	$CH18$	48
$CH7$	174	$CH19$	86
$CH8$	180	$CH20$	65
$CH9$	90	$CH21$	65
$CH10$	0	$CH22$	91
$CH11$	0	$CH23$	90
$CH12$	0	$CH24$	131

The servo channels marked with red color are the ones that are not used.

Robot planes

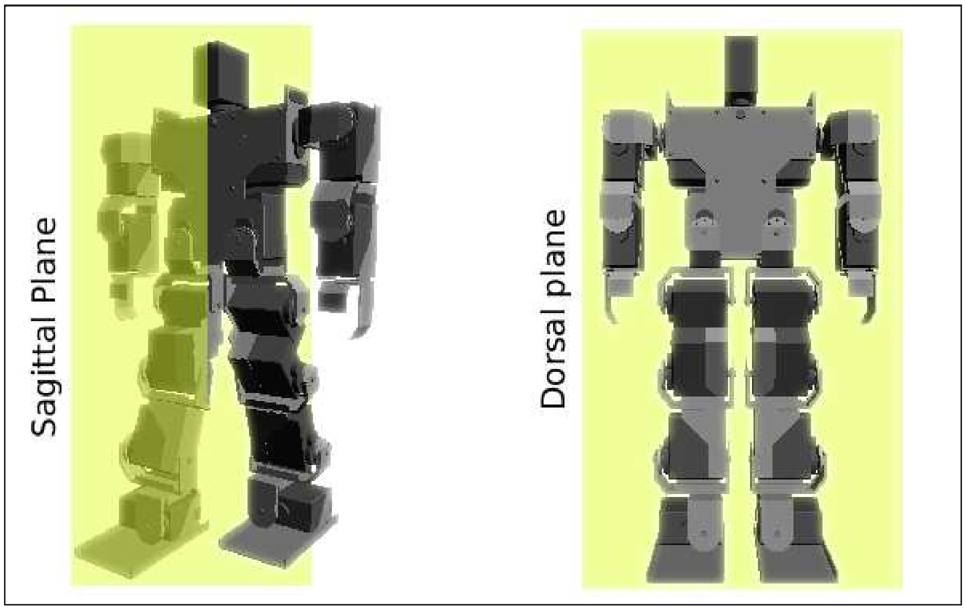


Figure 8.1: KHR-1 planes

- The sagittal plane divides the head, body or limb longitudinally into symmetrical right and left "halves."
- The dorsal plane divides the body into front and back parts. In humans it is also called the frontal plane.

Gaussian Noise

Function in C to generate random numbers following a Gaussian distribution.

```
double randomGaussian (double stdev, double mean)
{
     if ( stdev < 0.){
        fprintf(stderr, "Error: standard deviation < 0");
        exit(0);
     }

double u,v,x;
double a_random, b_random;

   do {

        a_random = ((double)rand())/((double)RAND_MAX);
        b_random = ((double)rand())/((double)RAND_MAX);

        u = 2.0 * a_random - 1.0;
        v = 2.0 * b_random - 1.0;
        x = u * u + v * v;

    }while ( x >= 1.0 || x==0);

 return mean + stdev * u * sqrt( -2. * log( x ) / x);
}
```

This algorithm is the polar form of the **Box-Muller Method** to generate white Gaussian noise [33].

Acknowledgments

First of all, I would like to thank my parents, whose unconditional love, help and support made this and many other dreams come true. Thank you so much Mom and Dad for always being there for me.

I would also like to thank my sister Genoveva and my brother Juan Carlos. Thank you so much for always helping the little boy of the house. I am always thinking about you *hermanitos*, and more in days like today because I got an enormous cramp in my leg, probably an effect of the multiple kicks you gave me when I was a child. I also remember the times when you rescued me when I started the primary school. Probably you remember this more than me *"Juncalo/Geno me ..."*.

I would like to thank my best friend, companion, soul mate, confidante, girlfriend, fiancée Erica *mi gordolfita*. Thank you for always being so supportive and for tolerating me. Thank you for waking me up every day, for encouraging me to keep on writing, for watching movies with me, for dancing with me, etc. I promise I will try to wash the dishes more often. Thank you for helping me with the experiments, for reading this thesis thousands of times and for answering all my questions about the English grammar.

I am deeply indebted to my advisor, Dr. Sven Behnke, for his constant support. Without his help, this work would not be possible.

Thanks also to my friends Maren Bennewitz and Cyrill Stachniss, they helped me reading this thesis. Let me tell you that every time I hear you say "corrections" I start shaking.

Many thanks to Jörg Stueckler for creating an excellent simulator, thanks also to Jürgen Mueller for helping me when I had problems with the pocket PC.

Finally, I would like to thank the rest of the NimbRo team: Michael Schreiber, Felix Faber, Dominik Joho, Tobias Latzke, Joachim Strach for alway trying to understand my lousy German.

Bibliography

[1] Charles Darwin. *The Origin of Species by Means of Natural Selection.* Penguin: London, 1859.

[2] Gregor Mendel. *Experiments in Plant Hybridization.* 1865.

[3] Stein and Rowe. *Physical Anthropology, 3rd. Ed., Los Angeles Pierce College.* McGraw-Hill Books, 1982.

[4] Holland J.H. *Adaptation in natural and artificial system.* The University of Michigan Press, 1975.

[5] David E. Golberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison Wesley Publishing Company, The University of Alabama, 1989.

[6] Brad l. Miller and David F. Goldberg. Genetic algorithms, selection schemes and the varying effects of noise. *Evolutionary Computation*, 4(2):113–131, 1996.

[7] David E. Goldberg. A note on boltzmann tournament selection for genetic algorithms and population–oriented simulated annealing. *Complex Systems*, 4(4):445–460, 1990.

[8] Melanie Mitchell. *An introduction to Genetic Algorithms.* MIT Press, Massachusetts Institute of Technology, USA, 1999.

[9] Michalewicz Zbigniew. *Genetic algorithms + data structures = evolution programs 3rd ed.* Springer, Charlotte, NC 28223, USA, 1999.

[10] William M. Spears and Kenneth A. De Jong. On the virtues of parameterized uniform crossover, 1991. Naval Research Laboratory, George Mason University.

[11] Dennis O'Neil. Darwin and natural selection, 2005. Available from: http://cs.nyu.edu/cs/faculty/yap/allpapers.html/. Accessed: May, 5th 2005.

[12] Wikipedia: The free encyclopedia, 2005. Available from: http://en.wikipedia.org/. Accessed: May, 10th 2005.

[13] Edmond Ayyappa. Normal human locomotion, part1: Basic concepts and terminology. *Journal of Prosthetics and Orthotics*, 9 Num 1:10–17, 1997.

[14] Edmond Ayyappa. Normal human locomotion, part 2: Motion, ground reaction force and muscle activity. *Journal of Prosthetics and Orthotics*, 9 Num 2:42–57, 1997.

[15] Saunders JB, Inman VT, and Eberhart HD. The major determinants in normal and pathological gait. *Journal of Bone and Joint Surgery*, 35 A:543–558, 1953.

[16] Michael S. Orendurff et al. The effect of walking speed on center of mass displacement. *Journal of Rehabilitation Research and Development*, 41 6A:829–834, 2004.

[17] Robert Ducroquet, Jean Ducroquet, and Pierre Ducroquet. *Walking and Limping: A Study of Normal and Pathological Walking.* Lippincott, 1968.

[18] Michael Hardt and Oskar von Stryk. Increasing stability in dynamic gaits using numerical optimizations. *15TH IFAC World Congress on Automatic Control, Barcelona, Spain.*, 2002.

[19] Bertenthal Bennett I. *Dynamical Systems: Its about Time!* Perception and Action Laboratory, University of Chicago, 2005. Chapter to appear in Data Analytic Techniques for Dynamical Systems.

[20] Tad McGeer. Passive dynamic walking. *I. J. Robotic Res.*, 9(2):62–82, 1990.

[21] Tad McGeer. Passive walking with knees. *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1640–1645, 1990.

[22] Arthur D. Kuo, J. Maxwell Donelan, and Andy Ruina. Energetic consequences of walking like an inverted pendulum: Step-to-step transitions. *Exercise and Sport Sciences Reviews*, 33(2):88–97, 2005.

[23] Andy Ruina. Research & laboratory homepage, 2005. Available from: http://ruina.tam.cornell.edu/index.html. Accessed: May, 25th 2005.

[24] M. Vukobratovic and D. Juricie. Contribution to the synthesis of biped gait. *IEEE Tran. On Bio-Medical Engineering*, 16(1):1–6, 1969.

[25] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kiyoshi Fujiwara, Kensuke Harada, Kazuhito Yokoi, and Hirohisa Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1620–1626, 2003.

[26] Kensuke Harada, Shuuji Kajita, Kenji Kaneko, and Hirohisa Hirukawa. Zmp analysis for arm/leg coordination. *Proceedings of the 2003 IEEE/RSJ. Conference on Intelligent Robots and Systems*, pages 75–81, 2003.

[27] Open Dynamics Engine (ODE). Available from: http://ode.org.

[28] OpenGL. Available from: http://www.opengl.org.

[29] Sven Behnke and Raul Rojas. A hierarchy of reactive behaviors handles complexity . *In M. Hannebauer, J. Wendler, and E. Pagello*, pages 125–136, Springer 2001.

[30] RoboCup. Available from: http://www.robocup.org.

[31] Robocup german open 2005.
Available from: http://www.ais.fraunhofer.de/GO/2005/.

[32] Martin Friedmann et al. Darmstadt Dribblers 2005: Humanoid Robots. 2005. Technische Universitaet Darmstadt, Germany.

[33] G.E.P Box and M.E. Muller. A note on the generation of random normal deviates. *Annals Math. Stat*, 29:610–611, 1958.

[34] R. Herrejon. Control of a semi-passive walking robot with torso. Master's thesis, 2004.

[35] Mariano Garcia, Anindya Chatterjee, and Andy Ruina. Speed, efficiency and stability of small-slope 2-d passive dynamic bipedal walking. In *ICRA*, pages 2351–2356, 1998.

[36] S.H. Collins, M. Wisse, and A. Ruina. A 3-D passive-dynamic walking robot with two legs and knees. *The International Journal of Robotics Research*, 20(7):607–615, 2001.

[37] M. Garcia, A. Chatterjee, A. Ruina, and M. Coleman. The simplest walking model: Stability, complexity, and scaling. *Journal of Biomechanical Engineering — Transactions of the ASME*, 120(2):281–288, 1998.

[38] W. I. Sellers, L. A. Dennis, W.-J. Wang, and R. H. Crompton. Evaluating alternative gait strategies using evolutionary robotics. *Journal of Anatomy*, 204:343–351, 2004.

[39] J. Roberts, D. Kee, and G. Wyeth. Improved joint control using a genetic algorithm for a humanoid robot. School of Information Technology and Electrical Engineering. University of Queensland, St. Lucia, Qld 4072, Australia, 2003.

[40] Krister Wolff and Peter Nordin. Learning biped locomotion from first principles on a simulated humanoid robot using linear genetic programming. In E. Cantú-Paz, J. A. Foster, K. Deb, D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. Dowsland, N. Jonoska, and J. Miller, editors, *Genetic and Evolutionary Computation – GECCO-2003*, volume 2723 of *LNCS*, pages 495–506, Chicago, 12-16 July 2003. Springer-Verlag.

[41] Thomas Röfer. Evolutionary gait-optimization using a fitness function based on proprioception. In *RobuCup*, pages 310–322, 2004.

[42] Nate Kohl and Peter Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 3, pages 2619–2624, May 2004.