Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2008

Hugo Rafael de Brito Picado

Desenvolvimento de Comportamentos para um Robô Humanóide Simulado
Development of Behaviors for a Simulated Humanoid Robot

"You cannot learn to fly by flying. First you must learn to walk, to run, to climb, to dance."

— Nietzsche

Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2008

Hugo Rafael de Brito Picado

Desenvolvimento de Comportamentos para um Robô Humanóide Simulado
Development of Behaviors for a Simulated Humanoid Robot

Thesis presented to University of Aveiro in order to achieve the necessary prerequisites to obtain the Master degree in Engineering of Computers and Telematics, under the scientific supervision of Nuno Lau (PhD, Assistant Professor at University of Aveiro and researcher at IEETA) and Luís Paulo Reis (PhD, Assistant Professor at the University of Porto and researcher and member of the directive board of LIACC)

o júri / the jury

presidente	**Doutor António Rui de Oliveira e Silva Borges** Professor Associado da Universidade de Aveiro
vogais	**Doutor Paulo José Cerqueira Gomes da Costa** Professor Auxiliar da Universidade do Porto
	Doutor José Nuno Panelas Nunes Lau Professor Auxiliar da Universidade de Aveiro
	Doutor Luís Paulo Gonçalves dos Reis Professor Auxiliar da Universidade do Porto

agradecimentos

Esta é uma parte difícil de escrever porque eu não me quero esquecer de ninguém. Agradeço em particular ao meu orientador, Nuno Lau, pela ajuda, amizade, disponibilidade, conhecimento e motivação em momentos críticos. Agradeço também ao meu co-orientador, Luís Paulo Reis, pelo conhecimento transmitido e motivação durante este projecto. Um agradecimento especial pela atenção daqueles que contribuíram para o desenvolvimento desta dissertação, em particular para os professores Filipe Silva e Ana Tomé, e também para os meus grandes amigos João Certo e Marcos Gestal. Um agradecimento especial à minha família, especialmente aos meus pais, Miguel Picado e Augusta Correia, que sofreram durante estes cinco anos para me darem amor e a oportunidade de atingir esta fase da minha vida. Agradeço também ao Jumbo de Aveiro, pela amizade dos meus colegas de trabalho e chefes durante os últimos cinco anos e também por patrocinarem a minha participação no RoboCup 2008 (Suzhou, China). Gostaria também de agradecer aos professores Artur Pereira e António Rui Borges por me motivarem durante os últimos anos. Finalmente, mas não menos importante, um agradecimento especial ao meu melhor amigo, David Campos, e também ao Luis Ribeiro e ao Pedro Alves, pela amizade e motivação mesmo quando eu dizia piadas secas (demasiadas horas acordado). Para quem não está mencionado neste texto: não se preocupem, vocês estão mencionados no meu coração, e isso é o mais importante.

acknowledgements

This is a difficult part to write because I do not want to forget anyone. I particularly thank my supervisor, Nuno Lau, for the help, friendship, availability, knowledge and motivation in critical moments. I also thank to my co-supervisor, Luís Paulo Reis, for the transmitted knowledge and motivation during this project. A special thank for the attention of those who contributed to the development of this thesis, in particular to the professors Filipe Silva and Ana Tomé, and also to my great friends João Certo and Mascos Gestal. A special thank to my family, specially to my parents, Miguel Picado and Augusta Correia, who suffer during these five years to give me love and the opportunity of reaching this phase of my life. I also thank to Jumbo of Aveiro for the friendship of my partners and supervisors during the last five years and also for patrocinating my participation in RoboCup 2008 (Suzhou, China). I would also like to thank to the professors Artur Pereira and António Rui Borges for motivating me during the last years. Finally, but not less important, a special thank to my best friend, David Campos, and also to Luis Ribeiro and Pedro Alves, for the friendship and motivation even when I said dry jokes (too many hours awake). For those who are not mentioned in this text: don't worry, you are mentioned in my heart, and that's the more important.

palavras-chave Locomoção bípede, RoboCup, FC Portugal, Humanóide, Optimização

resumo

Controlar um robô bípede com vários graus de liberdade é um desafio que recebe a atenção de vários investigadores nas áreas da biologia, física, electrotecnia, ciências de computadores e mecânica. Para que um humanóide possa agir em ambientes complexos, são necessários comportamentos rápidos, estáveis e adaptáveis. Esta dissertação está centrada no desenvolvimento de comportamentos robustos para um robô humanóide simulado, no contexto das competições de futebol robótico simulado 3D do RoboCup, para a equipa FCPortugal3D. Desenvolver tais comportamentos exige o desenvolvimento de métodos de planeamento de trajectórias de juntas e controlo de baixo nível. Controladores PID foram implementados para o controlo de baixo nível. Para o planeamento de trajectórias, quatro métodos foram estudados. O primeiro método apresentado foi implementado antes desta dissertação e consiste numa sequência de funções degrau que definem o ângulo desejado para cada junta durante o movimento. Um novo método baseado na interpolação de um seno foi desenvolvido e consiste em gerar uma trajectória sinusoidal durante um determinado tempo, o que resulta em transições suaves entre o ângulo efectivo e o ângulo desejado para cada junta. Um outro método que foi desenvolvido, baseado em séries parciais de Fourier, gera um padrão cíclico para cada junta, podendo ter múltiplas frequências. Com base no trabalho desenvolvido por Sven Behnke, um CPG para locomoção omnidireccional foi estudado em detalhe e implementado. Uma linguagem de definição de comportamentos é também parte deste estudo e tem como objectivo simplificar a definição de comportamentos utilizando os vários métodos propostos. Integrando o controlo de baixo nível e os métodos de planeamento de trajectórias, vários comportamentos foram criados para permitir a uma versão simulada do humanóide NAO andar em diferentes direcções, rodar, chutar a bola, apanhar a bola (guarda-redes) e levantar do chão. Adicionalmente, a optimização e geração automática de comportamentos foi também estudada, utilizado algoritmos de optimização como o Hill Climbing e Algoritmos Genéticos. No final, os resultados são comparados com as equipas de simulação 3D que reflectem o estado da arte. Os resultados obtidos são bons e foram capazes de ultrapassar uma das três melhores equipas simuladas do RoboCup em diversos aspectos como a velocidade a andar, a velocidade de rotação, a distância da bola depois de chutada, o tempo para apanhar a bola e o tempo para levantar do chão.

keywords Biped locomotion, RoboCup, FC Portugal, Humanoid, Optimization

abstract Controlling a biped robot with several degrees of freedom is a challenging task that takes the attention of several researchers in the fields of biology, physics, electronics, computer science and mechanics. For a humanoid robot to perform in complex environments, fast, stable and adaptable behaviors are required. This thesis is concerned with the development of robust behaviors for a simulated humanoid robot, in the scope of the RoboCup 3D Simulated Soccer Competitions, for FCPortugal3D team. Developing such robust behaviors requires the development of methods for joint trajectory planning and low-level control. PID control were implemented to achieve low-level joint control. For trajectory planning, four methods were studied. The first presented method was implemented before this thesis and consists of a sequence of step functions that define the target angle of each joint during the movement. A new method based on the interpolation of a sine function was developed and consists of generating a sinusoidal shape during some amount of time, leading to smooth transitions between the current angle and the target angle of each joint. Another method developed, based on partial Fourier Series, generates a multi-frequency cyclic pattern for each joint. This method is very flexible and allows to completely control the angular positions and velocities of the joints. Based on the work of developed by Sven Behnke, a CPG for omnidirectional locomotion was studied in detail and implemented. A behavior definition language is also part of this study and aims at simplifying the definition of behaviors using the several proposed methods. By integrating the low-level control and the trajectory planning methods, several behaviors were created to allow a simulated version of the humanoid NAO to walk in different directions, turn, kick the ball, catch the ball (goal keeper) and get up from the ground. Furthermore, the automatic generation of gaits, through the use of optimization algorithms such as hill climbing and genetic algorithms, was also studied and tested. In the end, the results are compared with the state of the art teams of the RoboCup 3D simulation league. The achieved results are good and were able to overcome one of the state of the art simulated teams of RoboCup in several aspects such as walking velocity, turning velocity, distance of the ball when kicked, time to catch the ball and the time to get up from the ground.

Contents

List of Figures

List of Tables

List of Acronyms

API	Application Programming Interface
CoM	Center of Mass
CoP	Center of Pressure
CPG	Central Pattern Generator
D-H	Denavit-Hartenberg
DOF	Degree of Freedom
FRP	Force Resistance Perceptor
FSM	Finite State Machine
GA	Genetic Algorithm
GADS	Genetic Algorithm and Direct Search Matlab toolbox
GAlib	C++ Genetic Algorithm library
GCoM	Ground projection of the Center of Mass
HC	Hill Climbing
MDL	Motion Description Language
MSRS	Microsoft Robotics Studio
ODE	Open Dynamics Engine
ODW	Omnidirectional Walking
PD	Proportional-Derivative
PDW	Passive Dynamic Walking
PFS	Partial Fourier Series
PI	Proportional-Integral
PID	Proportional-Integral-Derivative
P	Proportional
SA	Simulated Annealing
SPADES	System for Parallel Agent Discrete Event Simulation
TS	Tabu Search

VMC Virtual Model Control

XML eXtended Markup Language

ZMP Zero Moment Point

Chapter 1

Introduction

This thesis focuses on the development of behaviors for a simulated humanoid robot. Although the work of this thesis has been applied on the RoboCup Humanoid Simulation league, it is not at all restricted to this environment and may be applied to other simulation environments as well as to real robots. However, the transfer from simulation to reality is not always satisfactory [1]. So, efforts are needed to produce accurate simulated models and reliable behaviors.

1.1 Motivation

FC Portugal[1] exists since 2000 and dedicates its research to the development of coordination methodologies applied to the RoboCup Simulation League [2, 3]. In this context, the team won the world championship in 2000 (Melbourne), the European championship in 2000 (Amsterdam) and 2001 (Paderborn) in the 2D simulation league, where the agents are circles that play in a two-dimensional plane. In the 3D simulation league, where the agents are spheres playing in a three-dimensional field, FC Portugal won the world championship in 2006 (Bremen) and the European championships in 2006 (Eindhoven) and 2007 (Hannover). The simulation league, recently (2007) initiated a new 3D soccer simulation league based on humanoid robots and the team is also concerned in researching the necessary techniques in order to make humanoid robots play soccer [2].

Although cooperation and coordination methodologies have been the great challenge of RoboCup simulation leagues for years, the introduction of a simulated humanoid platform opens the doors to a new kind of research, which is particularly concerned with the study of bipedal locomotion techniques. This area includes topics in the area of physics and biology that are mainly focused on maintaining a biped stable, performing some gait and avoiding falling down. This study must drive to the development of stable biped behaviors such as walk, turn, get up and kick. The result of this kind of research may be extended to other domains, such as the use on real humanoid robots, which may be able to perform social tasks such as helping a blind to cross a street or elderly people to perform tasks that became impossible to do alone. The robotic simulated environments are very popular since they allow the developers to make arbitrary or complex tests in the simulator without using the real robot thus avoiding expensive material to get damaged [1].

[1] http://www.ieeta.pt/robocup

1.2 Objectives

The aim of this thesis is to tackle the problem of biped humanoid robot locomotion and control. Humanoids have numerous degrees of freedom[2] requiring complex control in order to achieve stable biped locomotion. The main goal is to develop behaviors for the FC Portugal simulated humanoid team. The specific goals of this thesis include:

- Study of different bipedal locomotion techniques;
- Development of low-level joint control and trajectory planning methods;
- Development of a motion definition language;
- Application of optimization algorithms for generating efficient behaviors;
- Development of different behaviors using the implemented methods.

These behaviors should be developed and tested in the scope of the RoboCup 3D simulation league.

1.3 RoboCup

RoboCup [4] is an international joint project whose objective is to promote research and education in Artificial Intelligence, Robotics and related fields, by providing a standard problem where a wide range of technologies can be integrated and examined. Presently, RoboCup includes several leagues:

- RoboCup Soccer;
  - Simulation League (2D, Coach, 3D and Mixed Reality);
  - Small Size League (SSL);
  - Middle Size League (MDL);
  - Standard Platform League (SPL);
  - Humanoid League.
- RoboCup Rescue;
  - Simulation;
  - Real robots;
- RoboCup Junior;
  - Dance;
  - Soccer;
  - Rescue;
  - Demonstration.
- RoboCup @Home.

[2]Degree of Freedom (DOF): Each independent direction in which the joint can perform a movement.

RoboCup provides an integrated research task which covers the main areas of robotics and artificial intelligence. These include design principles of autonomous agents, multi-agent systems collaboration, strategy acquisition, real-time reasoning, reactive behavior, real-time sensor fusion, machine learning, computer vision, motor control and intelligent robot control. Every year the RoboCup Federation organizes events open to the general public, where different solutions to that problem are compared. The long-term goal of the RoboCup is stated as follows:

> *"By 2050, a team of fully autonomous humanoid robot soccer players shall win a soccer game, complying with the official FIFA*[3] *rules, against the winner of the most recent World Cup of human soccer."*

Although it is a great challenge, soccer is not the only application domain of the RoboCup initiative. The Rescue League [5] consists of an environment representing a city after a big disaster. There are several kinds of agents (Fire Brigades, Police, Ambulances and three respective center agents) acting and cooperating in a dynamic and inaccessible environment in order to rescue victims of the catastrophe so that the loss of human life can be reduced. RoboCup Junior [6] is a project-oriented educational initiative that sponsors local, regional and international robotic events for young students. Finally, the challenge in the @Home league [7] is to build robotic systems capable of navigate through human populated home environments. One of the main complexities present is the human-robot interaction.

RoboCup simulation league

The main drawback of using real robots for experiments is that the robots can easily get damaged. The price to construct and repair real robots can be a limitation for improvements in this field. The idea of a simulation league is to develop a virtual agent capable of thinking and acting so that the acquired knowledge can be transferred to the real robots. To make this possible, it is necessary to construct accurate and reliable models of the real robots.

RoboCup simulation league started with a 2D simulator. In the 2D simulation league two teams of eleven autonomous agents play against each other using the RoboCup soccer server simulator [8]. Each simulated robot player may have its own play strategy and characteristic. In the 2D simulation league the agents are circles which play in a two-dimensional field. Over the years the research on this league has gained an incredible development level [3, 9, 10].

RoboCup also introduced the Online Coach Competition [11]. This sub-league is for automated coaches which are able to work with a variety of teams through the use of the standard coaching language, CLang, which is mainly based on COACH UNILANG [12]. Over the years this research became an issue of pattern recognition, where the goal of the coach is to detect and recognize playing patterns during a game.

In 2004, a new kind of simulation appeared in this league, named 3D simulation league. The circles in the 2D simulation became spheres and the simulation monitor shows a game in 3D. The 2D teams soon adapted their work to this league. The RoboCup simulation league adopted the Simspark simulator which pretends to be a generic simulator capable of simulating anything we want and to be as more realistic as possible [13].

Mixed Reality League was originally defined as Physical Visualization League and appeared in 2007 [14]. This league consists of real miniature robots playing a virtual game in a virtual field.

[3] Fédération Internationale de Football Association

Recently, in 2007, the 3D spheres became simulated models of humanoid robots. One of the main challenges of this league is to improve the biped locomotion behaviors of these agents (e.g. walk, turn, kick the ball, etc). This thesis is precisely about the development of these behaviors for a simulated humanoid robot in the context of the FCPortugal team.

1.4 Thesis outline

The remainder of this document is organized in more 7 chapters.

Chapter 2 shows the biped locomotion as a scientific challenge and presents common concepts and approaches such as Trajectory-based methods, Heuristic-based methods, Passive dynamic walking and Central Pattern Generators. It also presents some of the most popular humanoid robots as well as the most competitive RoboCup simulated humanoid teams in the last two years.

Chapter 3 gives an overview of some optimization algorithms such as Hill Climbing, Tabu Search, Simulated Annealing and Genetic Algorithms. It is also presented a machine learning technique, called Reinforcement Learning, which is very popular in the world of robotics.

Chapter 4 presents the simulation environment used during the experiments for this thesis as well as the simulated humanoid models used. Some alternatives to the used simulation environment are also presented.

Chapter 5 briefly describes the existing low-level control techniques, with a special emphasis on the Proportional-Integral-Derivative (PID) controllers, developed in the scope of this thesis for low-level joint control.

Chapter 6 presents the developed trajectory planning methods for this thesis and, for each one, some advantages, drawbacks and important results achieved are also presented.

Chapter 7 describes the developed behaviors for the simulated humanoid NAO using the methods presented in Chapter 6. These behaviors include several types of walk, turn, kick the ball, catch the ball and get up from the ground.

Finally, in Chapter 8, a conclusion about the developed work is presented as well as some interesting proposals and challenges for future work.

Chapter 2

Biped locomotion

For a long time, wheeled robots were used for research and development in the field of Artificial Intelligence and Robotics and many solutions were proposed [15]. However, wheeled robot locomotion is not adapted to many human environments [16]. This increased the interest in other types of locomotion like biped locomotion and especially in humanoid robotics. This field has been studied over the last years and many different approaches have been presented, although the ability for robots to walk in unknown terrains is still in a young stage. In order to give an overview of the state of the art in biped locomotion, this chapter presents the fundamentals by introducing common terms, concepts and solutions. The most successful control strategies and some humanoid projects developed are also presented.

2.1 Biped locomotion as a scientific challenge

During millions of years, living beings developed some kind of locomotion to ensure their survival inside their environment. It is assumed that some animals share some common locomotion properties even when they do not move in the same way [17, 18]. For a vertebrate to achieve biped locomotion, the neural system generates rhythmic signals that are sent to the body to produce the desired movement. In this context, locomotion can be described in terms of pattern generators to produce the movements and sensory feedback to achieve stability and motion compensation.

The biped locomotion control is a problem that is really difficult to solve since it counts only with two legs to keep stability. There are different approaches, but even the better solutions are far from reaching perfection. Nowadays there are some humanoid robots capable of producing efficient movements (e.g. walk, climb stairs) but the most unexpected small problem, such as a little slope or an unexpected condition in a rough terrain may cause a robot to fall down easily. The locomotion strategy and the robot model are closely related and this connection by itself makes the biped locomotion problem hard to solve. In spite of not being proved, it is assumed that humans use something like a pattern generator [18], which is central at the spinal level[1].

Biped locomotion can be divided into different stages. This is needed since bipeds show very different dynamical properties depending on many conditions. The human walking gait, for example, is generally divided into double support phase and single support phase. The former happens when both feet are in contact with the ground. The latter happens when one

[1] Central Pattern Generators (CPG) will be explained later in this chapter (Section 2.2.4)

foot is swinging forward while the other is doing the support job. For fast walking, the swing phase is usually divided into pre-swing and post-swing phases [19]. During the pre-swing phase, the foot rolls about the toes and during the post-swing phase the swinging foot lands on its heel and rolls about it. Once again this will depend on the robot model since not all the robots have the same degrees of freedom and the same ability to perform some movements. A running gait is even more complex since there is a phase where both feet are out of the ground (which is the so called flight phase).

Keeping the equilibrium of the biped robot is a complicated issue since that will require the control of every degrees of freedom. In order to compensate for disturbances, the gait should be robust enough to be adaptable to each different situation.

2.2 Approaches to biped locomotion

During the last years, several approaches to biped locomotion have been developed. This thesis explains the most common methods. These methods may be broadly divided in three main categories [20, 21]: Trajectory-based, Heuristic-based, Passive-dynamic and Central Pattern Generators. The following sections explain these categories.

2.2.1 Trajectory based methods

Trajectory-based methods consist of finding a set of kinematics trajectories and using a stabilization criterion to ensure that the gait is stable. The most popular stabilization criteria are the Center of Mass (CoM), Center of Pressure (CoP) and Zero Moment Point (ZMP). These stabilization criteria are described in the following sections. The gait is stable when one of these criteria remains inside the support polygon. The support polygon is the convex hull formed by the contact points of the feet with the ground [22]. Figure 2.1 shows the most typical cases for the support polygon for forward walking, assuming that the feet are always parallel with the ground.

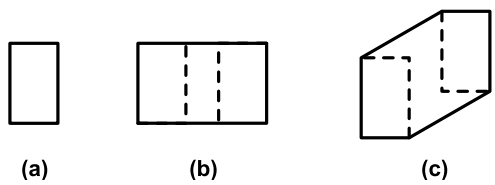

Figure 2.1: Typical cases for the support polygon for forward walking. (a) Single-support phase (support polygon is formed by the support foot) (b) Double support phase, where the feet are aligned (the support polygon is the rect formed by the two feet) (c) Double-support phase, with one foot in front of another (the support polygon is formed by the convex hull formed by the contact points of the feet with the ground.

Some common ways to produce trajectories are by the use of trial-and-error methods and motion capture[2] [23].

[2] A set of wearable motion sensors are read to enable precise tracking of human motions

Center of Mass and its projection on the ground

The CoM of a system of particles is the point at which the mass of the system acts as if it was concentrated [24]. In other words, CoM is defined as the location of the weighted average of the system individual mass particles, as defined by the following equation:

$$p_{CoM} = \frac{\sum_i m_i p_i}{M} \tag{2.1}$$

where $M = \sum_i m_i$ is the total mass of the system, m_i denotes the mass of the ith particle and p_i denotes its centroid. By considering an infinite number of particles in the system, the particle separation is very small and the system can be considered to have a continuous mass distribution thus the sum is replaced by an integral, as follows:

$$p_{CoM} = \frac{1}{M} \int p \; dm \tag{2.2}$$

For a humanoid body, the system is the body and the particles are the several body parts. The orthogonal projection of the CoM on the ground is the so called Ground projection of the Center of Mass (GCoM) [25].

Center of Pressure

Most humanoid robots are equipped with force-torque-sensors at the feet of the robot [22]. The CoP is the result of an evaluation of those sensors and is defined as the point on the ground where the resultant of the ground reaction forces acts [25]:

$$p_{CoP} = \frac{\sum_i p_i F_{N,i}}{\sum_i F_{N,i}} \tag{2.3}$$

where the resultant force $F_R = \sum_i F_{N,i}$ is the vector from the origin to the point of action of force $F_{N,i} = |F_{N,i}|$.

Zero Moment Point

The ZMP is perhaps the most popular stability criterion and was originally proposed by Vukobratovic [26] in 1972. There are some different interpretations of ZMP though they are all equivalent [27, 28]. ZMP can be defined as the point on the ground about which the sum of the moments of all the active forces equals zero [26].

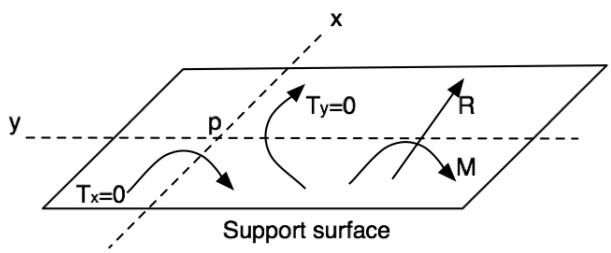

Figure 2.2: Arakawa and Fukuda interpretation of ZMP concept [28].

An alternative, but equivalent, interpretation was given by Arakawa and Fukuda [28] (See Figure 2.2). They define ZMP as the point p, where $T_x = 0$ and $T_y = 0$, where T_x and T_y represent the moments around the x and y axis generated by the reaction force R and reaction torque M, respectively. When p exists within the domain of the support surface, the contact between the ground and the support is stable [28].

Static stability vs. dynamic stability

The static stability criterion prevents the robot from falling down by keeping the GCoM inside the support polygon by adjusting the body posture very slowly thus minimizing the dynamic effects [29, 30] and allowing the robot to pause at any moment of the gait without falling down. Using this criterion will generally lead to more power consumption since the robot has to adjust its posture so that the GCoM is always inside the support polygon.

On the other hand, humans move in a dynamic fashion, a state of constant falling [29], where the GCoM is not always inside the support polygon. While walking, humans fall forward and catch themselves using the swinging foot while continuing to walk forward, which makes the GCoM moves forward without expending energy to adjust the GCoM trajectory. Dynamic stability relies on keeping the ZMP or CoP inside the support polygon and this is a necessary and sufficient condition to achieve stability. Dynamic balance is particularly relevant during the single support phase, which means that the robot is standing in only one foot. This generally leads to more fast and reliable walking gaits.

With the exception of some advanced humanoid projects, which is the case of Honda ASIMO [31] (See Section 2.5.1) and Sony QRIO [32] (See Section 2.5.2), most legged robots today walk using static stability [29].

2.2.2 Heuristic based methods

The most important drawback of ZMP is the use of complex dynamic equations to compute the robot's dynamics. This complexity can be crucial when designing humanoid robots, specially when the programmer wants to minimize the power and memory consumption of the biped. Heuristic based methods are based in heuristics that hide that complexity.

Virtual Model Control

Developed by Jerry Pratt [33], Virtual Model Control (VMC) is a framework that uses virtual components such as springs, dampers or masses to generate the joint torques that control the biped's stability and velocity. The generated joint torques create the same effect that the virtual components would create if they were in fact connected to the real robot. This heuristic makes the design of the controller much easier. First it is necessary to place some virtual components to maintain an upright posture and ensure stability. Using the example provided by Pratt [33], imagine that the goal of the robot is to knock a door. With traditional methods, this would be a very difficult task to implement. However, with VMC it is only needed to place a virtual mass with a specified kinetics energy to the robot's hand using a virtual spring and damper. The robot's hand will then move to strike out and once given the desired impact, the hand will get back due to mass resonating with the virtual component attached to the hand. VMC has the advantage of being less sensitive to external perturbations and unknown environments since these can be compensated by the use of virtual components.

2.2.3 Passive dynamic walking

In the Passive Dynamic Walking (PDW) approach, the biped walks down a slope without using any actuator [34]. The mechanical charecteristics of the legs (e.g. length, mass, foot shape) determine the stability of the generated walking motion.

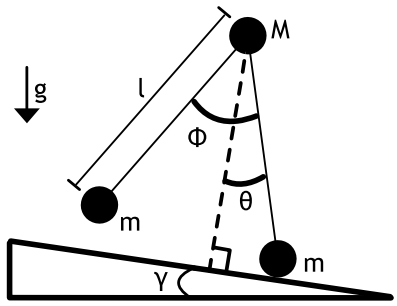

Figure 2.3: Passive dynamic walking model. ϕ is the supporting surface, θ is the angle of the support leg, m is the point mass at the respective foot, M is the mass at the hip, γ is slope inclination and g is the gravity. Both legs have length l. Adapted from: [35].

PDW is based on the inverted pendulum model [34], which assumes that, in the single support phase, human walking can be modeled as an inverted pendulum. Inverted pendulum has been applied for years in several situations [36, 37]. The swinging leg (assuming there is just the hip and the ankles and no knees) is represented by a regular pendulum, while the support leg is represented by an inverted pendulum. The support leg is then controlled by the hip joint's torque. However, in the specific case of PDW the only actuating force is the gravity. Figure 2.3 represents the PDW model.

Tad McGeer, in 1990, was the first to apply this idea to humanoid robotics by developing a 2D bipedal robot with knee joints and curved feet [34]. The developed robot was able to walk down a three degree slope. This work demonstrated that the morphology of the robot might be more important than the control system itself. This method is known for the low power consumption and was defined as a benchmark for walking machine efficiency.

2.2.4 Central pattern generators

It is assumed, by the fields of biology, that vertebrate locomotion is controlled by a spinal central pattern generator [38]. A Central Pattern Generator (CPG) is a set of circuits which aims to produce rhythmic trajectories without the need for any rhythmic input. In legged animals, the CPG often contains several centers that control different limbs. There has been a growing interest in these CPG models in robotics. This trajectory planning method does not need, necessarily, any sensory feedback information to generate oscillatory output for the motor neurons [38]. However, it is possible to integrate the sensory feedback information such as force resistors and gyroscopes to produce motion correction and compensation [39]. By coupling the neural oscillators signals when they are stimulated by some input, they are able to synchronize their frequencies. The smooth properties of coupled oscillators makes it easier to integrate feedback information in the defined mathematical model when generating the trajectories [20, 40, 41]. Figure 2.4 represents a CPG model applied to the Fujitsu HOAP-2 humanoid robot [42] (See section 2.5.3).

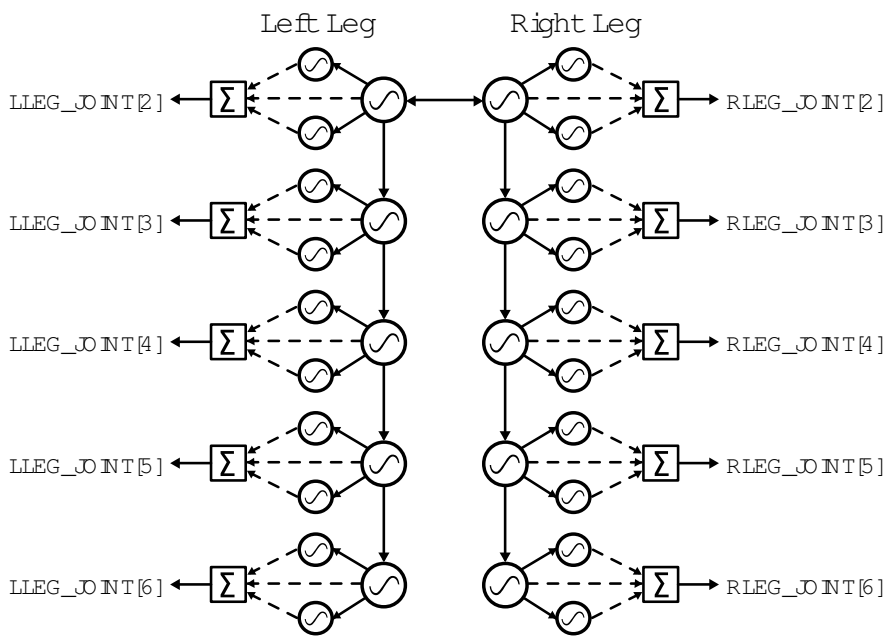

Figure 2.4: Structure of the walking CPG applied to Fujitsu HOAP-2 humanoid. The LLEG_JOINT and RLEG_JOINT arrays describe the set of left and right joints, respectively. Adapted from: [41].

Each Degree of Freedom (DOF) has a CPG. Antisymmetric coupling is used between the two legs through the main oscillator of the first DOF of each leg (which is represented in the figure by the horizontal arrow between the both legs). The trajectories generated for each DOF is the weighted sum of the corresponding three oscillators.

In the field of artificial intelligence and robotics, it is possible to build structures that are similar to the neural oscillators found in animals by the definition of a mathematical model. Sven Behnke [43] proved that it is possible to apply CPGs to generate a omnidirectional walking gait, where the input is simply the walking direction, walking speed and rotational speed.

2.3 Sensory feedback

Sensory feedback information are provided by sensors installed on the robot. Tipically, a humanoid robot has gyroscopes and force/torque sensors under the feet. These sensors are useful for maintaing the stability of the robot given specific constraints.

2.3.1 Gyroscope

A gyroscope is a device for measuring the oscillation rate in different directions. It provides information about the angular velocity of the robot's trunk in the sagittal plane[3] and also in frontal plane[4].

[3]Sagittal plane: Vertical plane running from front to back and dividing the body into left and right sides

[4]Frontal plane: Vertical plane, perpendicular to the sagittal plane, running from side to side and therefore dividing the body into front and back

If this angular velocity becomes too large, the robot is likely to lose balance. Most humanoid robots are equipped with a gyroscope. The use of gyroscope as a feedback control mechanism helps to keep the robot stable by adjusting the necessary angles for motion compensation.

2.3.2 Force/torque sensors

A force/torque sensor is a transducer that measures forces and torques. A six-axis sensor will provide information about three cartesian coordinates for the force (F_x, F_y and F_z) as well as for the torque (T_x, T_y and T_z). These sensors react to the applied forces accordingly to the Newton's third law: *To every action there is an equal and opposite reaction.*

It is common to find this kind of sensors below the feet of the humanoid. These sensors may be helpful to know information such as whether the foot is touching the ground, which is the force applied on the ground and the point where such force is applied. Moreover, they are often used to measure the CoP (See Section 2.2.1). It is commonly assumed that foot contact occurs just after the leg has finished the swinging phase. This may not be the true due to external perturbations, i.e., the foot contact might occur earlier or later than the predicted moment. Phase resetting is a phenomena that allows to reset the phase of the trunk at the moment of foot contact [39] using the information provided by these sensors. This will ensure that the leg will move as intented accordingly to the gait pattern.

2.4 Kinematics

Kinematics is the branch of mechanics that studies the motion of a body or system of bodies without taking into account its mass or the forces acting on it. Two main types of Kinematics are considered in articulated motion control: Forward Kinematics and Inverse Kinematics. The following sections describe both methods.

2.4.1 Forward kinematics

A humanoid robot is composed of a set of bodies interconnected by joints. Forward kinematics allows for computing the position of the joints using its current angles and, consequently the position of the body parts. This strategy is very useful to compute several parameters such as CoM, when the positions of body parts are not known in advance. A very popular way to derive the forward kinematics equations is by using the Denavit-Hartenberg (D-H) convention [44] in order to be systematic in the calculation of coordinates. This approach is based on the use of transformation matrices of homogeneous coordinates. This allows the computation of the coordinates of any point P using the coordinate matrix of any known point, K, and the transformation matrix between P and K. Homogeneous coordinate systems are based on multiplication of transformation matrices. Tipically, in the two-dimensional space, a point P is represented in homogeneous coordinates by $P = [x, y, 1]^T$. The transformations considered are the conventional matrices for translation (T), scale (S) and rotation (R) as stated by the following matrices:

$$T = \begin{bmatrix} 1 & 0 & \Delta_x \\ 0 & 1 & \Delta_y \\ 0 & 0 & 1 \end{bmatrix}, \; R = \begin{bmatrix} \sin(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \; S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Where Δ_x and Δ_y are the displacement parameters with respect to the origin of the coordinate system, θ is the rotation angle and, finally, s_x and s_y are the rates for scaling.

2.4.2 Inverse kinematics

Inverse kinematics is the opposite of forward kinematics [45] and is used to know the angles that the joints should have so that a body (called the end-effector) can reach a predefined position. As an example, if the goal is to put the hand of a robot at a certain position, the end effector is the hand and the inverse kinematics will compute the necessary joint angles for the shoulder and the elbow. An inverse kinematics problem may have several solutions or perhaps no solution. The former happens because, assuming that the end effector is already at the target position, the remaining joint angles can still have other combinations that maintain the end effector at the target position. The latter happens when the target position is outside the scope of the robot and cannot be reached.

When using trajectory-based methods, one major stability criterion for the generation of joint trajectories is the position of ZMP. For stable dynamic locomotion, the necessary and sufficient condition is to have the ZMP within the support polygon at all stages of the locomotion. Several algorithms of trajectory planning are based on ZMP. Most of these algorithms use the ZMP to achieve the desired joint trajectories by using inverse kinematics to compute the necessary angles so that the ZMP can follow the desired trajectory and kept inside the support polygon.

2.5 Humanoid robots

This section present the most important research projects in humanoid robots which are being developed, with emphasis on biped locomotion control.

2.5.1 Honda ASIMO

ASIMO[5] stands for Advanced Step in Innovation and MObility and was presented by Honda Motors Co. in 2002 [31] (See Figure 2.5). It has 26 DOFs, 120 centimeters of height and 52 kilograms. It is designed to operate in the real world so it is trained to reach and pick up things, navigate along different floors and even climb stairs, communicate, recognize people's voices and faces and it is also able to act in response to voice commands.

Figure 2.5: Honda ASIMO.

[5]http://asimo.honda.com/

In the future, Honda intends to make ASIMO help in tasks such as assisting the elderly or a person confined to a bed or a whellchair. In the context of walking behavior, for example, ASIMO has a set of precalculated trajectories and relies on ZMP to ensure dynamic stability. If these trajectories do not match the requirements, new motions are generated online by interpolating between closely matching patterns.

2.5.2 Sony QRIO

QRIO stands for Quest for cuRIOsity, formerly known as Sony Dream Robot (SDR-4X II), was presented in the end 2003 [32] (See Figure 2.6). This robot has 28 DOFs, 58 centimeters of height and 6.5 kilograms. It was designed for entertainment but it was never sold. Major technology includes stable dynamic walking and running and full leg movement allowing the robot to kick a ball. This robot is able to recognize voices and faces, avoid obstacles and communicate. The 2005 QRIO robot shows advances like a "third eye", which is an extra camera which allows the robot to notice and track individuals in a group of people. The QRIO robot has the ability to identify blocks by size and color, lift them using its lower body and stack one on the top of the other.

Figure 2.6: Sony QRIO.

QRIO has a highly advanced equilibrium system which allows it to balance on a moving surf board or skating on roller skates. A special tool has also been created so that humans can teach motion patterns to the robot.

2.5.3 Fujitsu HOAP-2

HOAP stands for Humanoid for Open Architecture Platform [42]. HOAP-2 is a 25-DOF humanoid which has 50cm and weights 7kg was developed by Fujitsu and is a humanoid robot that is small, simple an very versatile. HOAP-2 is able to walk, climb and descend stairs, stand up when it's loosing equilibrium and kick a ball. Additionally it is able to do headstands, Tai chi chuan[6], and write its own name.

"HOAP-2 is very easy to program and very easy to understand" were the words of Fujitsu robotics researcher Riadh Zaier. The technology is based on CPG networks, which simulate the neural oscillator found in animals. This is combined with a numerical perturbation method that quantifies the configuration and weights of the connections of the neural network.

[6]Tai chi chuan is an internal chinese martial art. It is very well known by its relation to health and longevity

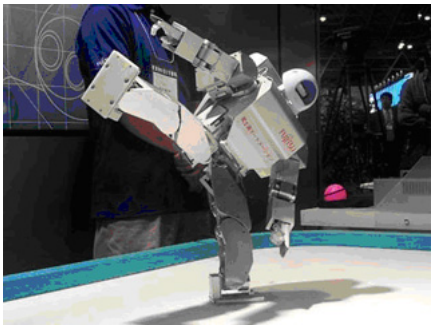

Figure 2.7: Fujitsu HOAP-2.

2.5.4 Aldebaran NAO

Project NAO, launched in early 2005, is a humanoid robot developed by Aldebaran Robotics [46] (See Figure 2.8). The robot has 21 to 25 DOFs (depending on the version), 57 centimeters of height and 4.5 kilograms. The robot was designed for entertainment and has mechanical, electronic, and cognitive features. Additionally, it is based on a Linux platform and scripted with URBI, which is an easy-to-learn programming language. It has been presented early 2007 and it is planned to go to the market in the end of 2008. Its simple and intuitive programming interface will make the entire family enjoy the robot experience. NAO Robot has been used in several RoboCup 2008 competitions[7]: Standard Platform League, 3D Soccer Simulation League, Microsoft Robotics League and Robotstadium Simulation League.

Figure 2.8: Aldebaran NAO.

URBI[8] stands for Universal Real-Time Behavior Interface and is a new scripting language with a C++ like syntax that pretends to do a revolution in the robotics programming. It brings new features such as parallelism, event-based programming and distributed object management with UObject. It also provides interfaces for the most popular languages in robotics, which is the case of C++, Java, Matlab, Python and Ruby.

[7] http://www.robocup-cn.org
[8] http://www.gostai.com/urbitechnology.html

2.5.5 NimbRo robots

Among the several robots used in RoboCup humanoid league, NimbRo[9] robots have been between the most popular (See Figure 2.9). NimbRo is one of the teams which participates in RoboCup championships. Robots have anthropomorphic body and senses, team research includes achieving energy-efficient bipedal locomotion by supporting the system dynamics and intuitive multi-modal communication with humans through analysis and synthesis of body language, gestures, facial expressions, and language. One of the most used techniques is the learning by imitation, where a robot learns new behaviors imitating a human teacher.

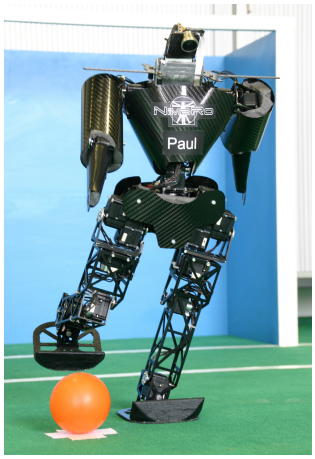

Figure 2.9: Paul, one of the NimbRo humanoid robots.

Sven Behnke[10] is the principal investigator inside NimbRo team. He developed an online gait generation method based on Central Pattern Generators that allows a humanoid to perform omnidirectional walking [43]. The engine was implemented and proved to be extremely advantageous for locomotion in dynamic environments, which is the case of RoboCup soccer leagues.

2.6 RoboCup simulated humanoid soccer teams

This section describes the three best simulated humanoid teams both in 2007 and 2008 RoboCup world competitions. The main concern of these teams was to develop stable and fast gaits for their humanoid agents in order to be able to build a competitive 3D simulation soccer team. To achieve this goal, it is important to develop the base skills such as walking without falling, getting up from the ground, turning around a spot, and kicking the ball. This is essential for the humanoid to play a reasonable soccer game. Given the particular focus of this thesis in biped locomotion, this section will emphasize the developed gaits and not the soccer strategy of these teams.

[9] http://www.nimbro.org/
[10] http://www.informatik.uni-freiburg.de/ behnke/

2.6.1 Little Green BATS

The main research focus of Little Green Bats[11] team is concerned with hierarchical behavior models. In these models, the agent's intelligence is constructed as a tree of behaviors, where each behavior controls the lower level behaviors. This allow the agent to have a form of abstraction which will permit the agent to act and response to difficult situations in real-time [47, 48]. A behavior consists of a sequence of steps. Each step has a subgoal and a set of sub-behaviors can be used to achieve these goals. This leads to a tree where the highest abstraction level is at the root and the most primitive behaviors are at the leafs. The latter does not select more behaviors, since they are primitive, but perform real world actions like applying a certain angular velocity to a joint.

The joint trajectory planning are defined using one of two methods [48]. In the first method a very simple scripting language define the trajectories for the joints and the agent run that script sequentially in runtime, setting joint angles and eventually waiting certain amount of time before going to the next line. The second approach is a mathematical model, based on Partial Fourier Series, to generate the trajectories for each joint:

$$\alpha_i^{'}(t) = \sum_{j=1}^{N} A_j sin(\omega_j t + \theta_j) + C_j \tag{2.4}$$

where $\alpha_i^{'}(t)$ is the angular position of the joint i at time t, N is the number of frequencies of the Fourier pattern and A_j, w_j, θ_j and C_j are, respectively, the amplitude, the frequency, the phase and the offset of the jth term. Some behaviors were defined by setting these values manually. The running behavior, in particular, were defined using a Genetic Algorithm (See Section 3.1.4) where the genotype includes the parameters A_j, ω_j, θ_j and C_j with $N = 1$.

The research of this team has been successfully applied, and they achieved the second place in the RoboCup 2007 world championship (Atlanta, USA) and the third place in the RoboCup 2008 world championship (Suzhou, China).

2.6.2 SEU-RedSun

SEU-RedSun was a dominant team during 2007 and 2008 RoboCup world competitions. In 2007 they proved that it is possible to implement a omnidirectional walking engine [49, 50] even using the unstable model of HOPE-1 simulated robot (See section 4.3.1). They implemented one of the most stable walking gaits of the competition. In 2008, they improved their work by increasing the speed and the agility of the walking gait and they became really fast. They were even able to score by kicking backwards or kicking sidewards [50]. This team focused its research on developing an efficient real-time method to generate humanoid behaviors to be integrated with an adversarial and dynamic environment, which is the case of Robocup [49, 50]. In this context, they developed an omnidirectional biped walking controller architecture (See Figure 2.10). Multiple layers that run on different time scales contain tasks of different complexity. The walking path planner receives the desired position and direction and passes the needed movement and rotation to the gait primitive generator, which will generate the next gait primitive. At the next layer, the limb controller will determine the desired joint angles, which will be the input for the joint motor controller.

[11] http://www.littlegreenbats.nl

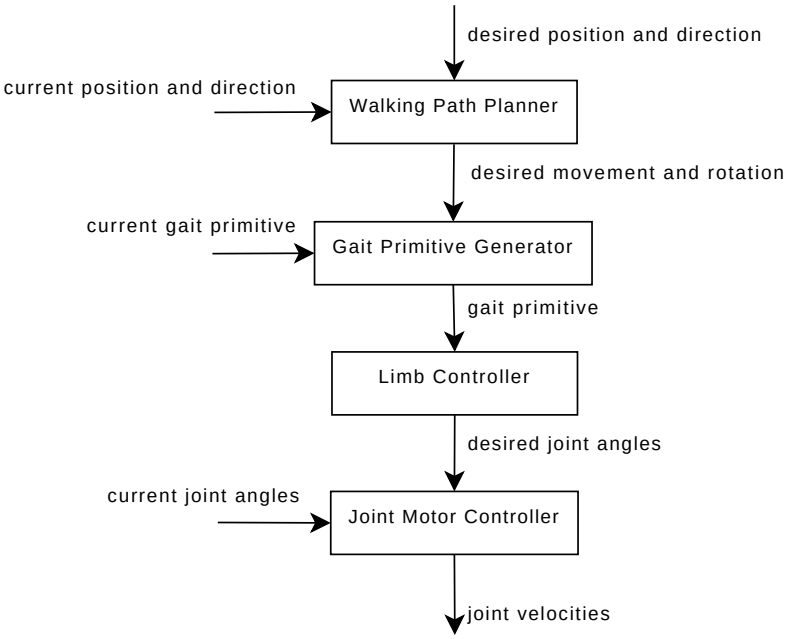

Figure 2.10: SEU-RedSun architecture of layered controller for omnidirectional walking. Adapted from: [49].

The desired joint angles and the current joint angles will make the joint motor controller determine the velocity needed for each joint to produce the desired behavior. The goal is to develop a stable walking pattern that does not need to be stopped before changing direction for a subsequent turning action. This improves the velocity since it is possible to change the walking direction without delaying the motion since everything can be done smoothly.

The research of this team was successfully applied, and they achieved the third place in the RoboCup 2007 world championship (Atlanta, USA) and the first place in the RoboCup 2008 world championship (Suzhou, China).

2.6.3 Wright Eagle 3D

Wright Eagle[12] has been a powerful team in many leagues of RoboCup competition. In their walking procedure, the state of each foot is divided into three phases: Raise, Land and Support [51]. So, the whole walking gait is divided in four phases as we can see in Table 2.1.

Left foot phase	**Right foot phase**	**Gait phase**
Raise	Support	RS
Land	Support	LS
Support	Raise	SR
Support	Land	SL

Table 2.1: Walking gait phases of Wright Eagle 2008 team [51].

[12]http://www.wrighteagle.org/.

When a walk command is sent to the action controller, Walk sets the default state as RS and then when RS is finished, it will transmit the control to the next state and so on (See Figure 2.11). The walking gait will automatically adjust the walking speed and slow down when the defined target is near.

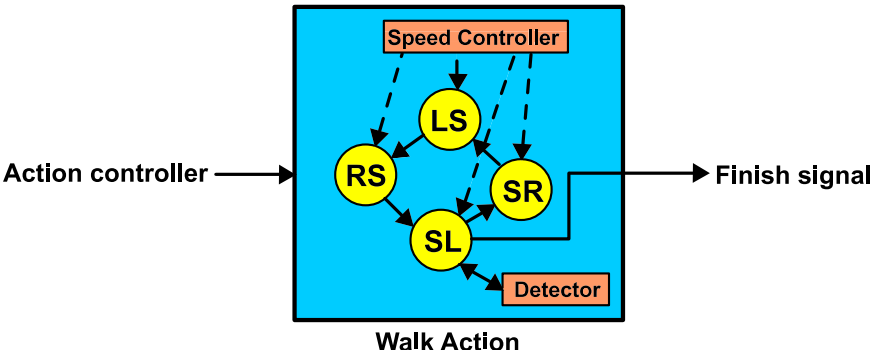

Figure 2.11: Wright Eagle 2008: Walk states transfer. Adapted from [51].

The research of this team was successfully applied, and they achieved the first place in the RoboCup 2007 world championship (Atlanta, USA) and the second place in the RoboCup 2008 world championship (Suzhou, China). We should refer that almost all the Wright Eagle matches matches on RoboCup 2008 were played with just one agent due to unpredictable problems they had during the competition.

2.7 Summary

In this chapter it was presented the state of the art of the humanoid robotics research field. After reading this chapter, one should be familiar with the actual humanoids and actual approaches to biped locomotion control and trajectory planning methods. Some real humanoids and teams of the RoboCup simulated humanoid league and their main features were presented. Common concepts in humanoid robotics were described in detail such as robot dynamics and kinematics, joint trajectory generation methods, equilibrium, common physical constraints used such as Center of Mass and its projection on the ground, Center of Pressure, Zero Moment Point, and also sensory feedback information and its applications.

Chapter 3

Optimization and Machine Learning

3.1 Optimization

Optimization problems aim at determining the minimal point of a functional on a non-empty subset of a real linear space [52]. A more formal definition is: Let X be a real linear space, let S be a non-empty subset of X, and let $f : S \rightarrow \Re$. An element, $s' \in S$, is called the minimal point of f on S if:

$$f(s') \leq f(s), \forall s \in S \tag{3.1}$$

Hence, the goal of optimization problems is to find the best configuration of a set of parameters of some problem that minimize[1] some measure. This measure is obtained through a function, f, of one or more parameters, which is commonly known as the *objective function* or *fitness function*. A configuration of the set of input parameters are known as an *individual* and refers to a possible solution of the problem to be solved. Moreover, one or more constraints may also be defined to restrict the values of the input parameters. The optimization problems explore the *search space*, S, which consists of a set of *candidate solutions*, aiming at finding a feasible solution, $s \in S$, which maximizes (or minimizes) the fitness function.

Optimization problems may be broadly divided into two main categories: individual-based methods and population-based methods [53]. Individual-based methods deal with only one current solution. Conversely, in population-based methods counts with a set of individuals (population) that are handled simultaneously. The following sections explain some optimization algorithms. Hill Climbing, Tabu Search and Simulated Annealing are individual-based methods, whereas a Genetic Algorithm is a population-based method.

[1] A maximization problem may be obtained by symmetry of the function f.

3.1.1 Hill climbing

The Hill Climbing (HC) algorithm [54, 55] is simple to implement and performs well in several situations, achieving reasonable results. Given an initial solution and a neighborhood relation, the hill climbing strategy runs over a graph whose states are threated as candidate solutions. Algorithm 1 shows the pseudo code of a generic HC.

Algorithm 1 Hill Climbing

```
CS ← GetInitialSolution()
E1 ← Evaluate(CS)
while TerminationConditionsNotMet() do
  SS ← GetNeighborhood(SS)
  for i = 1 to Length(SS) do
    E2 ← Evaluate(SS[i])
    if E2 < E1 then
      CS ← SS[i]
      E1 ← E2
    end if
  end for
end while
return CS
```

The procedure *GetInitialSolution* initializes the current solution either with a manually defined solution or a randomly generated one. Iteratively, if the procedure *TerminationConditionsNotMet* returns true, the algorithm keeps the iterative process. Otherwise, it stops the search. *GetNeighborhood* generates the neighbors by applying random variations to each parameter of the individual. Each neighbor is evaluated and its score is compared with the score of the current solution. The algorithm chooses as the next state, the one which as a better score than the current solution.

This algorithm easily gets lost in a local optima solution, i.e., none of the neighbors has a better evaluation than the current solution, which may be a poor quality solution. A possible strategy to solve this problem is to accept worst solutions. Another problem is the effect of cycling through solutions. This may be solved by introducing memories to remember the nodes already visited. Simulated Annealing [56] and Tabu Search [57] aim at solving the problems inherent to Hill Climbing. These algorithms are explained in the following sections.

3.1.2 Simulated annealing

In 1983, Kirkpatrick and coworkers [56] proposed a method of using a solution to find the lowest energy (most stable) orientation of a system. Their method is based upon the procedure used to make the strongest possible glass. This procedure hits the glass to a high temperature so that the glass is a liquid and the atoms can move freely. The temperature of the glass is then slowly decreased so that, at each temperature, the atoms can move enough to begin adopting the most stable orientation. If the glass is cooled slowly enough, the atoms are able to achieve the most stable orientation. This slow cooling process is known as annealing and was the origin of the name Simulated Annealing (SA).

SA was one of the first algorithms incorporating an explicit mechanism to escape from local optima. Analogous to the glass annealing problem, the candidate solutions of the optimization problem have correspondence with the physical states of the matter, where the ground state corresponds to the global minimum. The fitness function corresponds to the energy of the solid at a given state. The temperature is initialized to a high value and then decreased during the search process, which corresponds to the cooling schedule. The SA avoids local optima by accepting a solution worse than the current one with a probability that decreases along the optimization progress. Algorithm 2 shows the pseudo code the Simulated Annealing algorithm.

Algorithm 2 Simulated Annealing

```
CS ← GetInitialSolution()
T ← T_MAX
while TerminationConditionsNotMet() do
  NS ← GetNeighbor(CS)
  if Evaluate(NS) < Evaluate(CS) then
    CS ← NS
  else
    CS ← AcceptanceCriterion(CS, NS, T)
  end if
  T = CoolingSchedule(T)
end while
return CS
```

The chance of getting a good solution is a trade off between the computation time and the cooling schedule. The slower is the cooling, the higher will be the chance of finding the optimal solution, but higher will be the time needed for optimization. The neighbor is evaluated and will be subject to acceptance if it gets a score worse than the score of the current solution. The acceptance criterion accepts the new solution with a probability, $p_{acceptance}$, which defined as follows:

$$p_{acceptance} = e^{\frac{Evaluate(CS) - Evaluate(NS)}{T}}$$

This probability follows the Boltzmann distribution [58]. It depends on the temperature, T, and the difference of energy, $Evaluate(CS) - Evaluate(NS)$. The procedure *CoolingSchedule* decides how the cooling schedule is updated. A common approach follows the following rule:

$$T_{i+1} = \alpha * T_i, \qquad \alpha \in (0, 1) \tag{3.2}$$

where T_i is the current temperature level and T_{i+1} is the scheduled temperature level for the next iteration.

3.1.3 Tabu search

In 1986, Fred Glover proposed the Tabu Search (TS) algorithm [57]. TS improves the efficiency of the exploration process since it not only uses the local information (the result of the evaluation function), but also some information related to the exploration process (nodes already visited). In this way, TS may prefer to choose states with an inferior evaluation score instead of the already visited ones.

The main characteristic of TS is the systematic use of memory. While most exploration methods keep in memory essentially the value of the evaluation function of best solution visited so far, TS also keeps information of the itinerary through the last solutions visited. In its simplest form, TS declares each node already visited as a *tabu*. *Tabus* are stored in a list, the *tabu list*, and the search in the neighborhood is restricted to the neighbors that are not present in the *tabu list*. Algorithm 3 shows the pseudo code of a simple TS algorithm.

Algorithm 3 Simple Tabu Search

```
CS ← GetInitialSolution()
E1 ← Evaluate(CS)
TabuList ← Empty
while TerminationConditionNotMet() do
  SS ← GetNeighborhood(CS)
  for i = 1 to Length(SS) do
    if Not Member(SS[i],TabuList) then
      Add(TabuList, SS[i])
      E2 ← Evaluate(SS[i])
      if E2 < E1 then
        CS ← SS[i]
        E1 ← E2
      end if
    end if
  end for
end while
return CS
```

TS will evaluate each member and will pick up the best one. During the search process, TS prefers a solution worse than the current one, instead of an already tested solution. This way the algorithm tries to escape from cycling through solutions and also avoids the local optima since it accepts worst solutions.

3.1.4 Genetic algorithms

Proposed by the mathematician John Holland in 1975 [59], A Genetic Algorithm (GA) an optimization method inspired by the evolution of biological systems and based on global search heuristics. GA belongs to the class of evolutionary algorithms. In spite of being different, evolutionary algorithms share common properties since they are all based on the biological process of evolution. Given an initial population of individuals (also called chromosomes), the environmental pressure causes the best fitted individuals to survive and reproduce more. Each individual (chromosome) is a set of parameters (genes) and represents a possible solution to the optimization problem.

The algorithm starts by creating a new population of individuals. Typically, this population is created randomly but any other creation function should be acceptable. The genes of each individual should be inside a range of acceptable values that is defined for each gene. The algorithm then starts the evolution which consists of creating a sequence of new populations. At each step, the algorithm uses the individuals in the current population to create the next population by applying several operators.

These genetic operators are described as follows [60]:

- **Selection:** Specifies how the GA chooses parents for the next generation. The most common option is the roulette option which consists of choosing parents by simulating a roulette wheel, in which the area of the section corresponding to an individual is proportional to its fitness value;

- **Elitism:** Defines the number of individuals in the current generation that are guaranteed to survive in the next generation;

- **Crossover:** A crossover function performs the crossover of two parents to generate a new child. The most common is the scattered function which creates a random binary vector and selects the genes where the vector is a 1 from the first parent, and the genes where the vector is a 0 from the second parent. Moreover, the crossover function receives a parameter named as crossover fraction, p_c, which corresponds to the fraction of the population that is created by crossover;

- **Mutation:** The mutation function produces the mutation children. The most common is the uniform mutation which applies random variations to the children using an uniform distribution. Uniform mutation receives a parameter, p_m, which corresponds to the probability that an individual entry has of being mutated.

In the end of the optimization process, the individual in the current population that have the best fitness value is chosen as the best individual. Algorithm 4 shows the pseudo code of a generic GA.

Algorithm 4 Generic Genetic Algorithm

```
Population ← CreateInitialPopulation()
Evaluate(Population)
while TerminationConditionNotMet() do
    [Selection] Parents ← Selection(Population)
    [Elistism] Elite ← Elitism(Population)
    [Crossover] Children ← Crossover(Parents, p_c)
    [Mutation] Mutants ← Mutation(Children, p_m)
    Population ← Elite + Mutants
    Evaluate(Population)
end while
return Best(Population)
```

There exist several tools that facilitate the work with GA by providing several configuration options as well as user-friendly interfaces. Between the most popular are the C++ Genetic Algorithm library (GAlib), by Matthew Wall [61], and the Genetic Algorithm and Direct Search Matlab toolbox (GADS), from MathWorks [62]. GADS was used in the scope of this thesis for biped gait optimization. A more complete description of the GADS configuration options may be found in the GADS tutorial [62].

3.2 Machine learning

Learning can be viewed as the process of modifying the learner's knowledge by exploring the learner's experience [63]. Machine learning is a branch of artificial intelligence that deals with the design of algorithms that allow a machine to learn by itself thus reducing (perhaps eliminating) the need for human intervention in the learning process.

Machine learning has several applications such as natural language processing, pattern recognition, search engines, bioinformatics and robotics [64]. There are many types of learning algorithms, the most popular are supervised learning algorithms and unsupervised learning algorithms [65]. In supervised learning the algorithm keeps an example set that maps inputs to desired outputs. This type of algorithms are often used in classification problems (e.g. face detection, text recognition, etc). In unsupervised learning there are no labeled examples and the set of inputs is totally modeled by an agent.

A popular unsupervised learning algorithm is reinforcement learning [65]. In reiforcement learning problems [10, 66, 67], reactive or deliberative agents have knowledge about the state where they are and have to pick the next action (or perhaps a sequence of actions) to execute from a set of possible actions, with the objective of maximizing a measure called reward. In spite of being used in other fields, reinforcement learning has been receiving increased attention as a method of robot learning with a little or no *a priori* knowledge and a higher capability for reactive and adaptive behaviors. Figure 3.1 shows a basic model of the humanoid-environment interaction on which the agent and the environment are modeled by two synchronized finite state machines interacting in a discrete time process. The humanoid senses the current state of the environment and selects an action. Based on the state and the action, the environment transits to a new state and generates a reward that is sent back to the robot. Trough these action-reward method, the robot is able to learn autonomously.

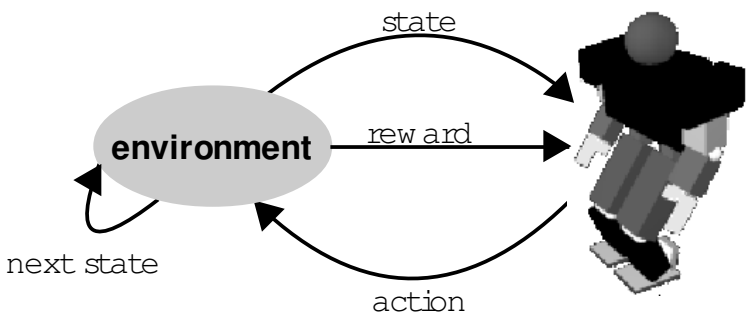

Figure 3.1: Reinforcement Learning Model.

The environment should be modeled as a Markov Decision Process[2], in discrete time, which can be described by the tuple (S, A, P, R) that consists of a set of states (S), a set of actions (A), an expected reward ($R^a_{s,s'}$), received due to the transition from state s to state s' with the execution of the action a and the state transition probabilities (P) [67]. An important property of Markov Decision Processes is that the state transition probabilities do not depend on past actions and states which simplifies the reinforcement learning algorithm [66].

[2]Markov assumption stated that each state depends on a finite number of past states.

The agent's behavior is to pick a policy $\pi : S \rightarrow A$. $\pi(s)$ corresponding to the action picked on the state s. The agent will receive an immediate reward $r(s, \pi(s))$ and the environment will change to a state *s'* with a probability $p^{\pi(s)}_{s,s'}$. If the agent always uses the same policy, what will be observed is a set of states and rewards. The goal of reinforcement learning is to find an optimal policy which maximizes the expected sum of rewards throughout time:

$$maximize_{\pi} E[\sum_{t=0}^{\infty} \gamma r_t | s_0 = s, \pi] \tag{3.3}$$

where r_t represents the received reward at instant t and $\gamma \in [0, 1)$ represents the discount factor which is introduced to guarantee convergence of the algorithm. A small discount factor gives more importance to near states instead of future states which is a behavior similar to the human learning behavior.

Q-Learning [68] is a recent form of reinforcement learning that does not need a model of the environment. The general idea is to introduce value functions Q^* which depend on the actual state and the action which should be the potential choice of the agent. This function models the expected reward for an agent:

$$Q^*(s, a) = r(s, a) + \gamma \sum_{s' \in S} (p^a_{s,s'} * Q^*(s', a')) \tag{3.4}$$

Finally, the optimal policy is to select the largest estimated Q-value and is computed as follows:

$$\pi^* : s \mapsto argmax_{a \in A} Q^*(s, a) \tag{3.5}$$

3.3 Summary

After reading this chapter, one should be familiar with the most popular optimization techniques. These algorithms were broadly divided into individual-based methods and population-based methods. Hill Climbing, Simulated Annealing and Tabu Search are classified as individual-based methods. Hill Climbing is the most simple algorithms and consists of consecutively iterating through a graph of solutions, by choosing as the next state the neighbor with a best fitness value than the current solution. This search over the neighborhood causes the algorithm to be sensitive to local optima. Simulated Annealing and Tabu Search try to avoid this problem by proposing different solutions. Simulated Annealing accepts a solution worse than the current one with a probability that decreases during the search process. Tabu Search uses a short-term memory to store the solutions already visited. This way, it prefers to chose a worst solution instead of an already visited one. A Genetic Algorithm is a population-based algorithm inspired on the biological evolution process. The algorithm imitates the natural reproduction of species by consecutively applying several operators to improve the fitness of the population. These operators are selection, elitism, crossover and mutation. In the end of the chapter it is present a popular machine learning technique, called reinforcement learning, which is a type of unsupervised learning. This machine learning technique choses the next action based on a measure called reward. In the scope of this thesis, Hill Climbing and Genetic Algorithm were implemented and tested.

Chapter 4

Simulation Environment

For all the project development, a simulation environment was used as the main platform for testing the developed behaviors. This chapter describes the simulation enviornment used in the experiments of this thesis.

4.1 Advantages of the simulation

The robotic platforms (either the most simple articulated arms or the most complex humanoid robots) are usually very expensive. The use of simulation environments for research, development and test in robotics provides many advantages over the use of real robots [21]. The main advantages of the simulation are:

- Less expensive than real robots;
- Easy development and testing of new models of robots;
- Easy testing of new algorithms;
- Less development and testing time;
- The problem can be studied at several different levels of abstraction;
- Possibility to easily add, remove, and test different components;
- Facilitates study of multi-agent coordination methods;
- All tests can be done without damaging the real robot;
- For repetitive tests (e.g. optimization processes), the use of a virtual model is better because the robot will not need assistance to reinitialize every iteration;
- It is possible to retrieve very detailed information from the simulation. It is possible to easily monitor physical measures such as CoM, CoP and ZMP;
- With the quality of simulation environments that exist today, is is possible to use the results obtained by simulation in the real robots with just a few changes;
- Control over the simulation time.

4.2 Simspark

The Simspark Simulator is a generic simulation platform for physical multi-agent simulations and it is currently used in the RoboCup 3D simulation league. This simulator was developed over a flexible application framework (Zeitgeist) and pretends to be a generic simulator, capable of simulating anything, since the launch of a projectile for academic purposes to a big soccer game for scientific research purposes. The framework facilitates exchanging single modules and extending the simulator [69]. The simulation consists of three important parts [13]: the server, the monitor and the agents.

4.2.1 Server

The server is responsible to handle connections from the agents, receive and process messages and send reply messages to the agent. The server architecture is illustrated in Figure 4.1 and is described as follows:

- Open Dynamics Engine (ODE)[1]: This is the physical simulation engine. It allows to simulate the system's dynamics and the physical properties of the simulated objects by providing advanced joint types and integrated collision detection with friction. ODE is particularly useful for simulating objects in virtual reality environments. ODE is cross-platform and provides a user-friendly C/C++ Application Programming Interface (API).
- Zeitgeist: This is a framework for handling data objects and functional components of a system in a uniform way [69] which strictly follows the object-oriented paradigm using C++ programming language.
- Simulation Engine: This represents the core of the simulator, it receives the messages with actions from the agents, performs the simulation operations and sends a reply message back to the agent with the environment information.
- System for Parallel Agent Discrete Event Simulation (SPADES) [70]: Provides a middleware layer that may be present between the agent and the simulation engine to handle the distribution of the simulation across machines and it is robust enough to variations in the network and machine load. It does not require the agents to be written in any particular programming language.

The agents may interact directly with the simulation engine or through the System for Parallel Agent Discrete Event Simulation (SPADES) middleware layer. The simulation engine acts as a server, handling the messages of the agents and replying back with other messages. In the case of the humanoid soccer simulation, on each simulation cycle the agent sends a message to the server containing information about the effectors[2] (e.g. joints). The message from the server to the agent contains temporal information and information specific from the application domain (which is soccer). This information includes game state (play mode, time and current result), and information of the perceptors of the robot[3] (e.g. joints, gyroscopes, foot sensors, vision information). The messages are constructed using a LISP-like format.

[1] http://www.ode.org/.
[2] Effector is a common term used in robotics to represent an actuator of the robot.
[3] Perceptor is a common term used in robotics to represent an input sensor of the robot.

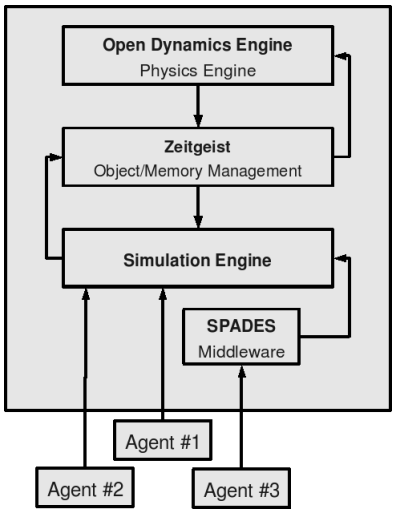

Figure 4.1: Simspark server architecture.

4.2.2 Monitor

The monitor provides a simple graphical interface that allows the user to watch a simulation. The simulations may be watched in real-time, but it is also possible to play simulation log files. In the particular case of humanoid soccer simulation, it provides additional information such as the team names and the game time, play mode and result. Several shortcut keys may be used to change camera views, to drop the ball, and other useful operations. Figure 4.2 represents the soccer monitor.

Figure 4.2: Simspark monitor screenshot.

4.3 The simulated agent

The simulator provides some agent models modeled in a LISP-based scripting language so it is possible to change the agent's configuration without recompiling all the code. In the first experiments related to this thesis, the HOPE-1 humanoid was used. With the introduction of a model of the Aldebaran NAO robot (See section 2.5.4), the agent model used for tests was changed not only because the popularity of NAO, but also because the defined model is more realistic. Despite their physiognomy (e.g. dimensions, number and type of joints) both humanoids have a gyroscope at the torso, two foot sensors that provide information about the force applied by the feet on the ground and also allow to know whether a foot is touching the ground or not. Additionally, they have a perceptor and an effector for each joint. The perceptors provide feedback information about the angular position of the joint. The effectors allow for changing the angular position of the joints and affect the environment.

4.3.1 HOPE-1

HOPE-1 stands for Humanoid Open Platform Experiment and it was the first model of a humanoid robot used for RoboCup 3D Soccer Simulation League [13]. It was loosely based on Fujitsu HOAP-2 (Section 2.5.3). Loosely because the model is not realistic at all since the relation height-weight is not real (3.75 meters of height and 5.50 kilograms). This was an obstacle to the development of behaviors since the equilibrium was unlike to be really achieved and this is one of the main reasons why the NAO simulated model (explained in the next section) became so popular among the RoboCup teams.

Figure 4.3: RoboCup simulated HOPE-1 humanoid.

Description

HOPE-1 has 3.75 meters of height and 5.50 kilograms and contains 20 DOFs, 4 on each arm and 6 on each leg. There is no DOF for neck movements but since HOPE-1 had an omnidirectional vision camera, this was not a problem. Additionally, there is no joint limits so unnatural movements are possible. This humanoid is also equipped with one gyroscope at the torso (that provides pitch, roll and yaw information about the oscillation rates of the robot) and Force Resistance Perceptors (FRP) which provide information about the force applied on each foot, whenever the foot is in contact with the ground. The FRP also provides a point which is the average of all contact points where the force is applied.

Joint configuration

Figure 4.4 represents the configuration of the HOPE-1 joints. It shows the hinge joints[4] and the Universal Joints[5].

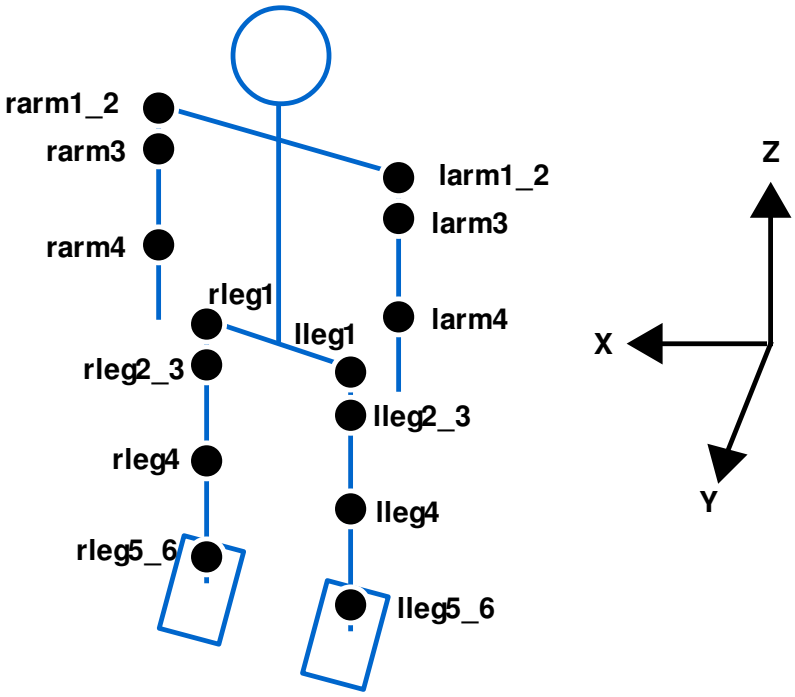

Figure 4.4: HOPE-1 joint configuration.

A more detailed configuration is presented in the Table 4.1. The table presents, for each joint, its name, its type, its parent body part and the axis it can perform a movement relative to the referential illustrated in the Figure 4.3.

Joint name	Joint type	Parent	Rotation axis (X,Y,Z)
arm1_2	Universal	Shoulder	(1,0,0) and (0,1,0)
arm3	Hinge	Shoulder	(0,0,1)
arm4	Hinge	Elbow	(1,0,0)
leg1	Hinge	Hip	(0,0,1)
leg2_3	Universal	Thigh	(1,0,0) and (0,1,0)
leg4	Hinge	Knee	(1,0,0)
leg5_6	Universal	Foot	(1,0,0) and (0,1,0)

Table 4.1: HOPE-1 joint configuration. The l and r prefixes of leg and arm joints represent the left and right side, respectively, and were omitted for readability.

[4] A hinge joint is a simple joint with one DOF.
[5] An universal is a joint composed by two DOFs.

4.3.2 Simulated NAO

RoboCup humanoid soccer league recently created a virtual model of NAO (See Figure 4.5) with the same physical characteristics and it is more likely to be used in future competitions. It is really similar to the real NAO, not only in the body measures but also in the body look.

Figure 4.5: RoboCup simulated NAO humanoid.

Description

The simulated model of NAO pretends to be as much as possible near to the real NAO (See Section 2.5.4). It has 57cm of height, its weight is about 4.5kg and contains 22 DOFs, 4 on each arm, 6 on each leg and two on the neck. Joints are limited so unnatural movements should not be possible, though this feature was deactivated in the 2008 competition. This humanoid is also equipped with one gyroscope at the torso (that provides pitch, roll and yaw information about the oscillation rates of the robot) and FRPs which provides information about the force applied on each foot, whenever the foot is in contact with the ground. The FRP also provides a point which is the average of all contact points where the force is applied.

Joint configuration

NAO contains only hinge joints which means that the number of joints is the same as the number of DOFs. Figure 4.6 represents the configuration of the simulated NAO body.

A more detailed configuration is presented in the Table 4.2. The table presents, for each joint, its name, its parent body part and the axis it can perform a movement relative to the referential illustrated in the Figure 4.6.

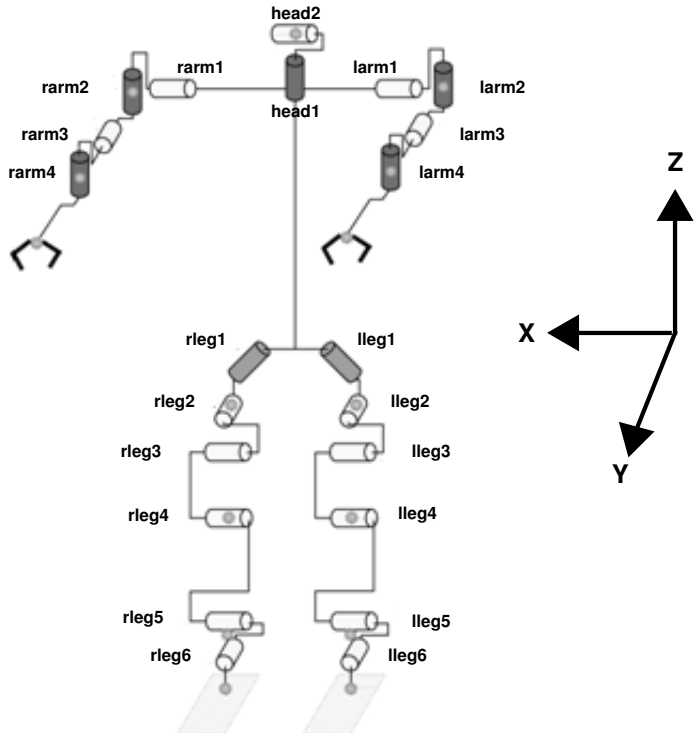

Figure 4.6: Simulated NAO joint configuration. Hinge joints are represented on blue. The used referential is also represented. Adapted from: [46].

Joint name	**Joint type**	**Parent**	**Rotation axis (X,Y,Z)**
head1	Hinge	Neck	(0,0,1)
head2	Hinge	Neck	(1,0,0)
arm1	Hinge	Shoulder	(1,0,0)
arm2	Hinge	Shoulder	(0,1,0)
arm3	Hinge	Shoulder	(0,0,1)
arm4	Hinge	Elbow	(1,0,0)
leg1	Hinge	Hip	$(-\sin(\frac{\pi}{4}),\ 0,\ \sin(\frac{\pi}{4}))$
leg2	Hinge	Thigh	(0,1,0)
leg3	Hinge	Thigh	(1,0,0)
leg4	Hinge	Knee	(1,0,0)
leg5	Hinge	Foot	(1,0,0)
leg6	Hinge	Foot	(0,1,0)

Table 4.2: Simulated NAO joint configuration. The l and r prefixes of leg and arm joints represent the left and right side, respectively, and were omitted for readability.

4.4 Alternatives

Many simulators are available that can be used to develop biped locomotion strategies. However, in this section the most similar to Simspark are briefly presented:

- **Microsoft Robotics Studio (MSRS)[6]:** Freeware Windows-based environment to create robotics applications for academic or commercial purposes. A huge variety of hardware platforms are supported. It includes a lightweight REST-style, service-oriented runtime, a set of visual authoring and simulation a complete documentation support.
- **Webots[7]:** Open-source environment powered by Cyberbotics. It allows the user to model, program and simulate mobile robots. The included robot libraries allows the user to transfer the control programs to several commercially available real mobile robots.
- **Player/Stage/Gazebo[8]:** Open-source project that runs under UNIX-like environments such as Linux. Player is a network server for robot control. It provides a cleaning and simple interface to the robot's sensors and actuators over the IP network. Stage and Gazebo provide 2D and 3D simulation environments, respectively, and are capable of simulating a population of robots moving in and sensing the environment. Moreover, Gazebo is capable of simulating the physical properties of the simulated objects.

4.5 Summary

This chapter presented the simulation environment used for experiments related to this thesis. A set of advantages of the simulation were also discussed. The simulation environment used is called Simspark and it is currently being used at the RoboCup 3D Simulation League. After reading this chapter, one should be familiar with the architecture and components of the simulator and how the communication between the agents and the simulator takes place. Additionally, two simulated humanoid agents were presented: HOPE-1 and simulated NAO. Since both these models were used and tested in the scope of this thesis, a brief description about its structure was presented.

[6] http://msdn.microsoft.com/en-us/robotics/default.aspx.
[7] http://www.cyberbotics.com/.
[8] http://playerstage.sourceforge.net/.

Chapter 5

Low-level control

Control theory is a branch of engineering that is concerned with controlling a dynamical system by influencing its inputs [71]. In robotics, one of the applications of these controllers is to control servo motors[1]. In the particular case of humanoid robotics, servo motors can be found in the joints and that's why the biped as the ability to move an arm or a leg or even its head. One of the major challenges in this field is to build low-cost and effective controllers [72]. The cost is measured in energy consumption and effectiveness is measured by the error between the desired and the effective behavior of the controlled system. In engineering, a control system has at least two main modules: the controller itself and the system to be controlled (the plant) [73].

5.1 Open-loop control

An open-loop control does not provide any feedback to verify if the system really reached the desired output. Hence, using an open-loop system there is no way to correct the difference between the desired output and the effective output (called the output error) since no information is available about this difference. Figure 5.1 shows the generic architecture of an open-loop controller.

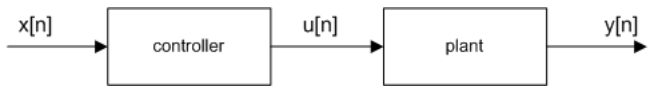

Figure 5.1: Open-loop control. For a discrete temporal value of n, x[n] represents the input (the setpoint), u[n] the output of the controller and y[n] the output of the controlled system

For a correct use of this type of control, a very good knowledge about the plant is needed. Open-loop control is more common in systems where the input-output relation can be modeled by a mathematical equation. If the plant behavior is predictable, the open-loop control is enough to control the system. Otherwise the system may need to be fed back. If K_c is the controller proportional gain and K_p is the plant proportional gain, the output of the system will be given by:

$$y[n] = K_p u[n] = K_c(K_p x[n]) \tag{5.1}$$

[1]Servo motors are widely used in robotics due to its size and effectiveness and because they are not expensive

Assuming that the plant corresponds to a linear system (which may not be the case [74]), any external disturbance will produce an error in the plant and the controller will not be able to notice this disturbance and correct the error. If the mathematical model of the plant is not precise, the consequences may be catastrophic, depending on the needs of the system. For nonlinear systems, this problem becomes even more complex than this, since the nonlinear systems are difficult to model and the nonlinear equations are difficult to solve [74].

5.1.1 Advantages

The main advantages of open-loop control are:

- Simple to understand;
- Simple to implement;
- Low-cost implementation and maintenance.

5.1.2 Drawbacks

The main drawbacks of open-loop control are:

- An accurate knowledge of the plant is needed;
- The system cannot be fed back;
- Sensitive to external disturbances.

5.2 Closed-loop control

As an alternative to open-loop control, the closed-loop control can be used. Usually, it is not possible to model a precise mathematical formula of the plant. In this case, it would be better a controller capable to adapt the system's input, taking into consideration not only the desired output, but also the effective output of the system. When the output of the system is fed back into the system as part of its input, it is called the feedback control or closed-loop control. Figure 5.2 represents the generic architecture of a closed-loop control system.

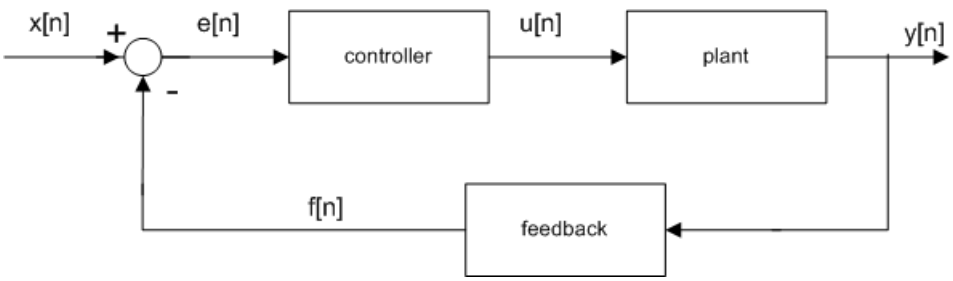

Figure 5.2: Closed-loop control.

For a discrete temporal value of n, x[n] represents the input (the setpoint), u[n] the output of the controller and y[n] the output of the controlled system. Aditionally, f[n] represents the feedback information provided by the feedback sensor.

Assuming a negative feedback control [71, 74], this information will be subtracted from the desired output to generate the so called error, which is represented in the figure by e[n]. This error, which is the difference between the desired output and the effective output, will be the new input of the controller. Let K_c, K_p and K_f be the proportional gains of the controller, plant and feedback sensor, respectively. The output of the system is based on the following system of equations [73].

$$\begin{cases} y[n] = K_p u[n] \\ u[n] = K_c(x[n] - f[n]) \\ f[n] = K_f y[n] \end{cases} \tag{5.2}$$

By solving these equation system, the output y[n] is given by:

$$\begin{aligned} y[n] &= K_p u[n] \\ &= K_p(K_c(x[n] - f[n])) \\ &= K_p(K_c(x[n] - (K_f y[n]))) \\ &= K_p K_c x[n] - K_p K_c K_f y[n] \end{aligned} \tag{5.3}$$

Isolating y[n], we obtain:

$$y[n] = \frac{K_p K_c}{1 + K_p K_c K_f} x[n] \tag{5.4}$$

By dividing the numerator and the denominator of the above equation by $K_p K_c$ we get:

$$y[n] = \frac{1}{\frac{1}{K_p K_c} + K_f} x[n] \tag{5.5}$$

Assuming the controller gain, K_c, is large enough to discard the first term of the denominator, a simplified view of the equation can be obtained:

$$y[n] \approx \frac{1}{K_f} x[n] \tag{5.6}$$

The Equation 5.6 means that, if the controller gain is large enough, the output becomes less sensitive to external disturbances in the plant. Since real systems are always affected by some noise, with a particular focus to robotics, this type of control has many advantages over the open-loop control described in the previous section.

5.2.1 Advantages

The main advantages of closed-loop control are:

- The system is fed back for error correctness;
- Less sensitive to external disturbances;
- There is no need for an accurate knowledge of the plant.

5.2.2 Drawbacks

The main drawbacks of closed-loop control are:

- The architecture is more complex than in the open-loop control;
- Need for sensory feedback;
- Its inherent reactive structure is not suited for acting in antecipation.

5.2.3 PID control

PID stands for Proportional-Integrative-Derivative and is a particular implementation of a closed-loop control, widely used in industrial processes and robotics [71, 74, 75, 76]. It is also known by the three-term controller since it uses three gains to generate the output. The controller's output as a continuous time signal is described by:

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau)d\tau + K_D \frac{de(t)}{dt} \tag{5.7}$$

The equivalent discrete equation is given by the following system of equations [73]:

$$\begin{cases} u[n] = K_P P[n] + K_I I[n] + K_D D[n] \\ P[n] = e[n] \\ I[n] = I[n-1] + Te[n] \\ D[n] = \frac{e[n]-e[n-1]}{T} \end{cases} \tag{5.8}$$

where T is the interval between discrete samples. The generic architecture is the same as shown in the Figure 5.2. What makes this control special is the internal configuration of the controller block. Figure 5.3 represents a detailed view of the controller block in the case of PID control:

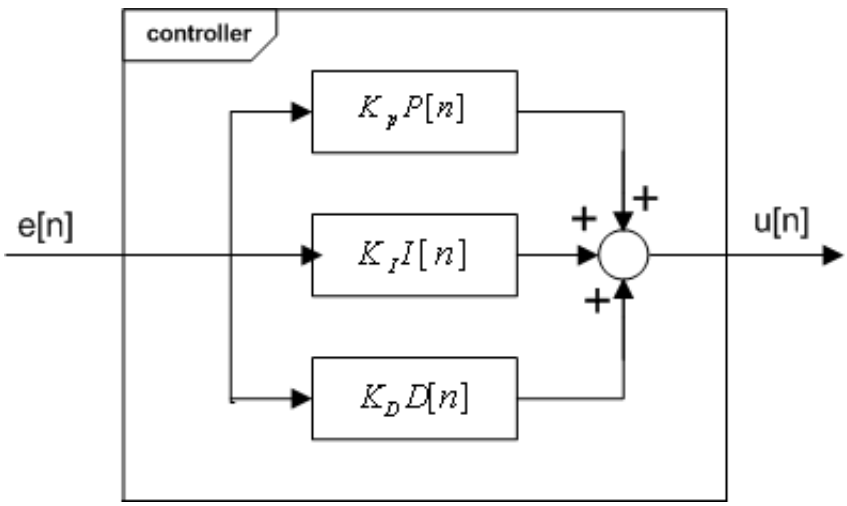

Figure 5.3: PID closed-loop control: Detailed view of the controller block.

The controller may appear in different variants: Proportional (P) controller (the most simple, uses only the proportional term), Proportional-Integral (PI) Proportional-Derivative (PD) controller and finally PID-controllers (all the terms are used). Taking the example of a humanoid joint, if we want to move the joint 1 radian, the joint response will not be immediate, but there will be some rise time. The goal of the joint controller is to increase the rise as much as possible without producing instabilities. Figure 5.4 represents four situations of the controlled joint.

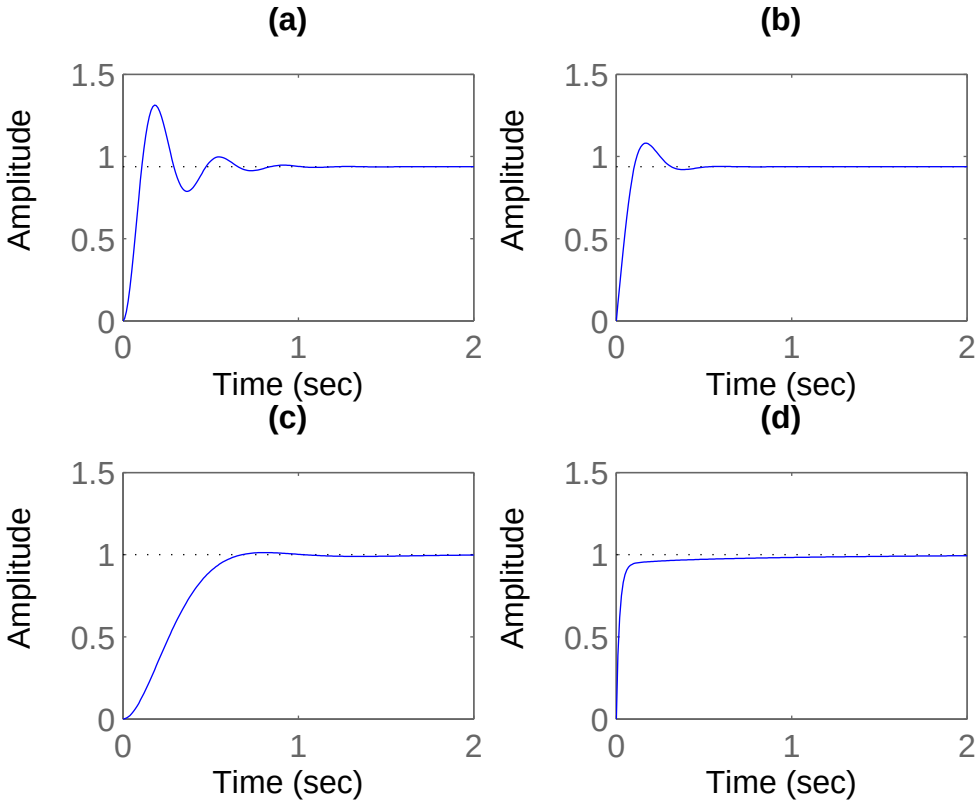

Figure 5.4: Example: Joint control using PID controllers **(a)** P-controller **(b)** PD-controller **(c)** PI-controller **(d)** PID-controller

Figure 5.4 shows some consequences of the different terms. A large proportional gain may produce overshoot[2]. The integral term reduces (usually eliminates) the steady-state error[3] but, like the proportional term, reduces the rise time and may produce overshoot, so it may be needed to decrease the proportional gain in order to add an integral term. The derivative term reduces the overshoot and has a small effect in the steady-state error. A correct definition of the three terms will reduce the rise time, reduce the overshoot and reduce (probably eliminating) the steady-state error. There are some approaches to define the PID gains though the most popular is Ziegler-Nichols method [77].

[2]Overshoot refers to an output exceeding its final value
[3]The steady-state error is defined as the different between the input and the output of a system

5.3 Summary

This chapter presented the two main types of control, open-loop and closed-loop, as well as its main differences. PID control is a particular case of closed-loop control which deserves a particular attention due to its popularity in robotics. Due to that popularity and results already demonstrated, a PID control interface was implemented in the scope of this thesis to control the humanoid joints at a lower level and proved to be extremely useful when generating the trajectories for the angular velocities of the different joints.

Chapter 6

Trajectory Planning

Trajectory can be defined as the set of points followed by a mobile object over the time. In the case of robotic joints, trajectory planning consists of breaking the joint space into many start and end points during some amount of time. A gait generator (or behavior generator) is a system capable of generating gaits (e.g. walk, turn, get up) by computing different joint target trajectories. This chapter describes the several trajectory planning methods implemented in terms of joint trajectory equations, implementation, supporting configuration language (when applicable) and some interesting results. The behaviors developed using the implemented methods will be presented later in Chapter 7.

A GaitGenerator class was developed to simplify the integration of the different generators (Figure 6.1).

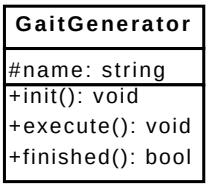

Figure 6.1: GaitGenerator class. It has a protected attribute, which is the name of the gait, and three main methods that must be implemented by derived classes.

The class GaitGenerator provides three virtual methods that must be implemented by all derived classes:

- **init**: Initializes the gait by resetting gait phases and other control variables.
- **execute**: Schedules the gait for execution by sending the desired trajectory to the joint control module.
- **finished**: Checks whether the gait is finished. A gait is finished when it completes an entire gait cycle.

Figure 6.2 represents the life-cycle of a behavior. The gait generator provides the angular positions for all joints to the joint controller. The joint controller then applies some kind of low-level control (e.g. PID) and generates the corresponding angular velocities. Finally, the angular velocities are collected by the Actions module that creates the message and then sends it to the server so that the behavior can be performed by the humanoid.

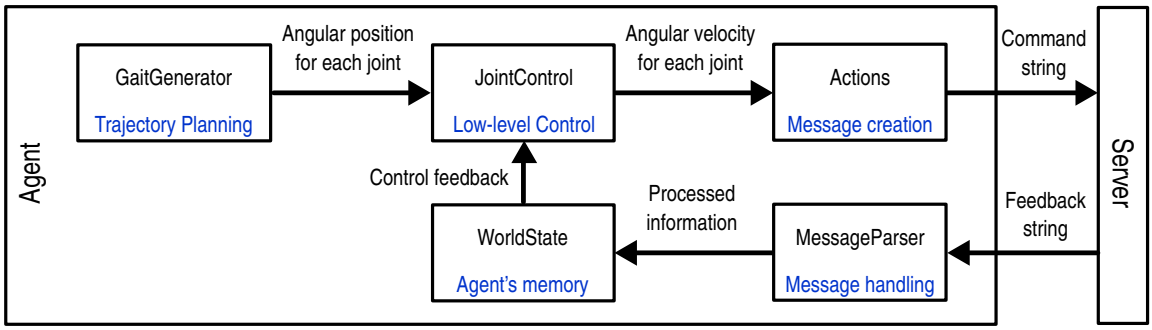

Figure 6.2: Behavior life-cycle.

The server performs the action and sends the effective joint angular positions back to the agent. The message parser handles the feedback message and processes this information to deliver it to the World State. This information is then kept in the agent's memory but it is also sent to the joint controller so a closed-loop low-level control is possible.

6.1 Step-based method

This section presents the method used by the FCPortugal3D team before this thesis. The step-based method generates trajectories using step functions. A step function is a discontinuous function consisting of a series of constant functions, each one defined in some interval of time.

6.1.1 Joint trajectory generation

The method used for the generation of the joint trajectories is very simple. It corresponds to a step function whose amplitude is the desired target angle for the joint on each interval of motion. The trajectory equation for each joint is described by the function *f(t)*, defined at time t, as follows:

$$f(t) = \sum_{i=0}^{n} \theta_i \cdot u_{A_i}(t), \forall t \in \Re \tag{6.1}$$

where n is the number of intervals, A_i is the interval i, and θ_i is the desired target angle for the joint at the interval i. $u_{A_i}(t)$ is called the indicator function of A and is defined as follows:

$$u_{A_i}(t) = \begin{cases} 1, & \text{if } t \in A_i \\ 0, & otherwise \end{cases} \tag{6.2}$$

The rise time of the step response will depend on the controller used (See Chapter 5), as shown in the Figure 6.3. For this method a simple proportional controller was defined. Hence, the joint angular velocities are computed based on the following equation:

$$\omega(t) = \gamma(\theta_{target} - \theta_{current}) \tag{6.3}$$

The parameter γ is the proportional controller gain, θ_{target} is the desired angle and $\theta_{current}$ is the effective angle of the joint. It is not possible to control the exact time that the controller will take to reach the desired angle so a tolerance value is associated with each angle. The angle tolerance means that, if the desired angle is 30 degrees and the tolerance is 2 degrees, 28 degrees will be acceptable. Hence, the step is considered finished when the current angle has an acceptable value.

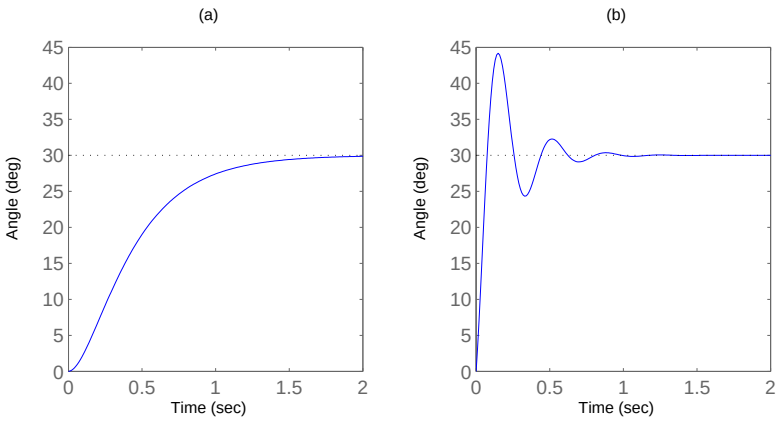

Figure 6.3: Step response. In the example the joint will try to go from 0 degrees to 30 degrees. **(a)** Small proportional gain results on long rise time **(b)** Large proportional gain reduces rise time but may result on overshooting the target.

6.1.2 Implementation

A step-based behavior consists of a sequence of joint moves and each sequence moves a set of joints in parallel by sending the corresponding target angles to the controller. A behavior finishes with the end of the last sequence of the behavior, which happens when all joints of that sequence finish their movements. A joint movement is finished when its current angle has an acceptable value taking into consideration the tolerance value.

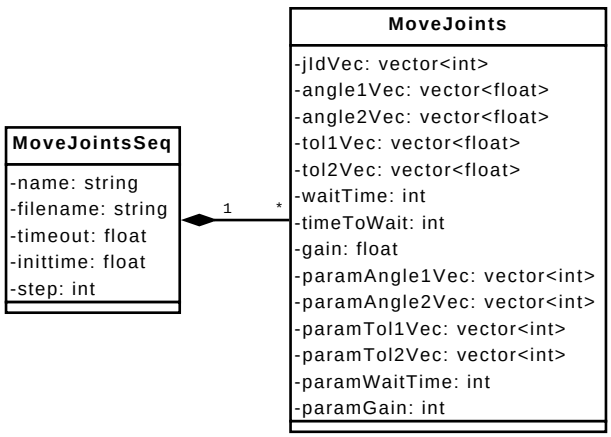

Figure 6.4: Class diagram for step based behaviors.

A simple scripting language is used to define the behavior[1]. The configuration file provides some flexibilities to define the movements by providing extra options. Specifically, it is possible to divide a sequence into several steps (A movement from $\theta_{current}$ to θ_{target} may execute in N steps instead of a single one). Moreover, the user can define the proportional gain that will be used by the low-level controller, which will be used for all joints in a sequence. Finally, the language allows for the definition of parameters instead of real numbers, which allows for the use of online generated values (e.g. allowing for taking into account sensory feedback information). Figure 6.4 represents the class diagram of the step-based method. Table 6.1 describe the classes MoveJointsSeq and MoveJoints in more detail.

Class	**Attribute**	**Description**
MoveJointsSeq	filename	Name of the configuration file
	timeout	Forces a sequence to finish when the behavior enters in a loop state
	initTime	Initialization time
	step	Current sequence being executed
MoveJoints	jIdVec	Identifiers of the joints
	angle1Vec, angle2Vec	Angles of the first and second DOFs of the joints
	tol1Vec, tol2Vec	Tolerances of the first and second DOFs of the joints
	waitTime	Number of cycles to wait
	timeToWait	Number of cycles remaining until finish the wait period
	gain	Proportional controller gain

Table 6.1: Description of the the MoveJointsSeq and MoveJoints classes.

Each attribute described above for the MoveJoints class has a corresponding attribute with the same name with the prefix *param*. These additional attributes define, for each joint, the index of the parameter vector where the value to use is stored, when using parameters instead of constant real values in the configuration file. The variables *paramsUse* and *paramsSet* are used to know if the parameters are being used and also if they are already defined.

6.1.3 Results

Figure 6.5 represents the trajectory of the knee for a simple bend-stretch movement, i.e., the robot bends the knees and then stretches the knees again. It is not possible to directly define the exact duration of the several sequences of movements. However it is possible to approximate it by changing the proportional gains. Although a large gain is needed to produce the stretch movement in a quarter of a second, it will result on an overshoot that is not possible to eliminate without increasing the rise time since only a proportional gain is available. The generated trajectory is neither smooth nor precise leading to a non-stable gait.

[1]See Appendix C for more details

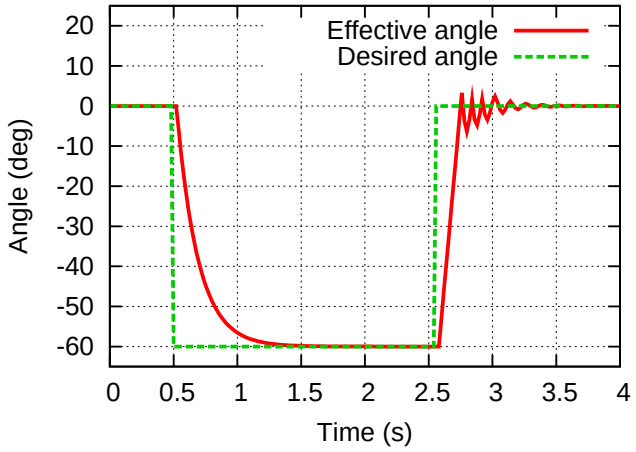

Figure 6.5: Evolution of a Step-based trajectory

6.1.4 Advantages

The main advantages of the step-based method are:

- Simple to understand;
- Simple to implement;
- Simple to define target trajectories (target angles and tolerances).

6.1.5 Drawbacks

The main drawbacks of the step-based method are:

- Time from current angle to target angle is unpredictable;
- No control over the angular velocity trajectory;
- The same gain is used for all joints;
- Sensitive to overshoot reactions at the control level;
- The syntax of the configuration file is not user-friendly;
- The model is not flexible.

6.2 Sine interpolation

The goal of sine interpolation is to give the user the opportunity to have a better control over each joint trajectory by defining an interpolation of some smooth function over a specific amount of time between the current angle and the target angle. Using this strategy, it is possible to define not only the target angles, but also the time in which those angles should be achieved, as well as control the initial and final angular velocities. This method is a very simple version of the method proposed in [72] and was the first method implemented in the scope of this thesis.

6.2.1 Joint trajectory generation

This method is based on the concept of slot, which corresponds to an interval of time from 0 to δ, where several joints are moved in parallel. In each slot, the controller will interpolate between the current angle and the desired angle by performing a sine-like trajectory in a specified amount of time. Each joint follows a trajectory generated by the following expression:

$$f(t) = A * \sin\left(\frac{\phi_f - \phi_i}{\delta} t + \phi_i\right) + \alpha, \forall t \in [0, \delta] \tag{6.4}$$

where $f(t)$ is the trajectory function, δ is duration of the slot in milliseconds, ϕ_i is the initial phase (which will influence the initial angular velocity), ϕ_f is the final phase (which will influence the final angular velocity), A is the amplitude and α is the offset. In order to interpolate between the current angle and the desired angle, taking into account the initial and final angular velocities, A and α must be calculated carefully. This is done using the following expressions:

$$A = \frac{\theta_f - \theta_i}{sin(\phi_f) - sin(\phi_i)} \tag{6.5}$$

$$\alpha = \theta_i - A * sin(\phi_i) \tag{6.6}$$

where θ_i and θ_f are the initial and final angles, respectively, and should be defined between $-\pi$ and π. Figure 6.6 shows examples of smooth generated trajectories based on the equations described. As an example, assuming $\phi_i = -\pi/2$ and $\phi_f = \pi/2$, the initial and final angular velocities will be zero.

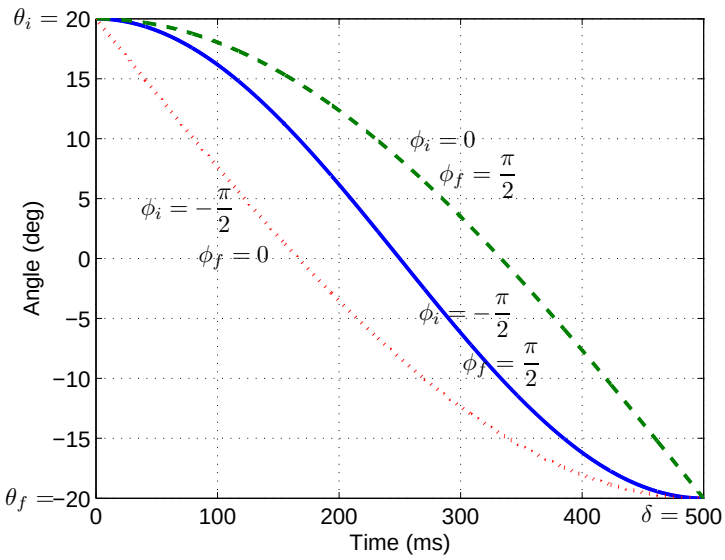

Figure 6.6: Possible shapes with the Sine Interpolation method.

6.2.2 Implementation

The slot-based behavior is a set of slots describing the desired key poses of the biped. It uses a PID controller for each joint. A slot is a set of move commands that will be executed in parallel during some interval of time, δ. Each move defines the identifier of the joint and the corresponding target angle. Optionally, it is possible to define other control parameters such as initial phase and final phase of the sine trajectory (ϕ_i and ϕ_f) and the PID control parameters. Figure 6.7 shows the class diagram of a slot-based behavior.

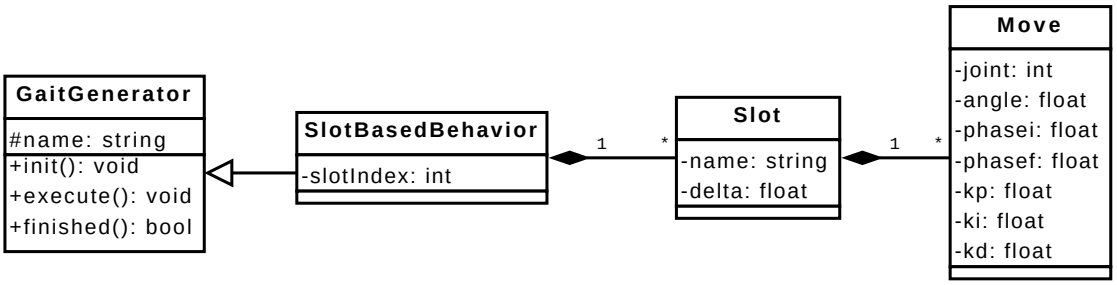

Figure 6.7: Class diagram for slot based behaviors.

The step-based method, explained in the Section 6.1, gives support for universal joints because it was entirely developed for experiments on HOPE-1 humanoid, which makes use of universal joints. However, the simulated NAO humanoid has only hinge joints so the universal joints supporting was omitted for clear readability. However, this generator and the others presented in the remain of this chapter are flexible enough to easily add this support with just a few lines of code. Table 6.2 describes the attributes of the classes SlotBasedBehavior, Slot and Move in more detail.

Class	**Attribute**	**Description**
SlotBasedBehavior	slotIndex	Current slot being executed
Slot	name	Name of the slot. This name can be used to access the slot to change the values online.
	delta	Duration of the slot, δ
Move	joint	Identifier of the joint
	angle	Desired target angle for the joint
	phasei, phasef	Initial and final phases of the sine trajectory (ϕ_i and ϕ_f)
	kp, ki, kd	PID proportional, integral and derivative gains

Table 6.2: Description of the classes SlotBasedBehavior, Slot and Move

For significant values of δ, only a P controller is needed. However, when this time becomes very small (near to the simulation cycle duration, which is about 20 milliseconds), the integral and derivative gains become important to produce fast transitions without overshoot reactions. A slot without any moves corresponds to wait δ milliseconds, which makes the controller maintain the same values for the joint angles.

6.2.3 Results

The example used for the previous generator is useful to check the advantage of using PID controllers. Remembering the example, the robot bends the knees and then stretches the knees again. In fact, without increasing the gain it is possible control the rise time using the slot duration. However, this quick change in the joint produces overshoot (Figure 6.8a).

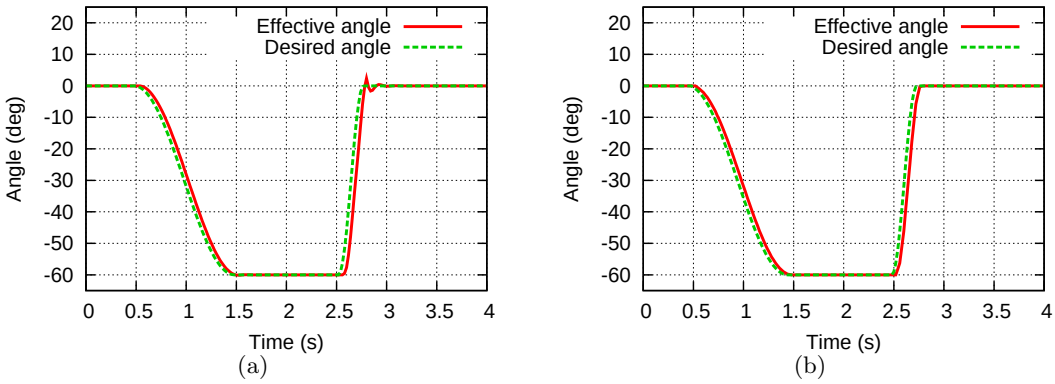

Figure 6.8: Evolution of a Sine Interpolation trajectory. **(a)** Using only a proportional gain **(b)** Using the proportional and the derivative gains.

By adjusting the derivative gain[2], a new trajectory for the knee was obtained (Figure 6.8b). It is possible no note a smooth and precise trajectory, that only differs from the target by a simulation cycle, that is imposed by the server for realistic behavior purposes.

6.2.4 Advantages

The main advantages of the sine interpolation method are:

- Simple to understand and implement;
- Time from current angle to target angle is controlled;
- Some control over the angular velocities trajectories;
- The model is flexible;
- PID control allows for controlling the overshoot reactions;
- The motion description language[3] is user-friendly and well structured.

6.2.5 Drawbacks

The main drawbacks of the sine interpolation method are:

- It is not possible to define more complex sinusoidal shapes;
- The angular velocities trajectories are not completely controlled.

[2]Derivative gain has the effect of reducing the overshoot (See Section 5.2.3)
[3]See Appendix C for more details

6.3 Partial Fourier series

The method of sine interpolation is restricted since it is not possible to define more complex shapes with so few parameters. Most of humanoid movements show complex cyclic patterns, which cannot be achieved using a simple sine interpolation. To overcome such restrictions in the previous method, a new kind of target generation method, based on Partial Fourier Series (PFS), was developed.

6.3.1 Joint trajectory generation

Some human-like movements are inherently periodic and repeat the same set of steps several times (e.g. walk, turn, etc). Multi-frequency shapes can be achieved by PFS. The principle of PFS consists of the decomposition of a periodic function into a sum of simple oscillators (e.g. sines or cosines) as represented by the following expression:

$$\begin{aligned} f(t) &= A_0 + A_1 \sin(\omega t + \phi_1) + A_2 \sin(2\omega t + \phi_2) + \ldots + A_N \sin(N\omega t + \phi_N) \\ &= A_0 + \sum_{n=1}^{N} A_n \sin(n\omega t + \phi_n), \forall t \in \Re \end{aligned} \tag{6.7}$$

where N is the number of frequencies, A_n is the amplitude of the nth term, ω is the angular frequency and ϕ_n is the phase of the nth term. Figure 6.9 shows examples of trajectories generated by this method for N=1 and N=2.

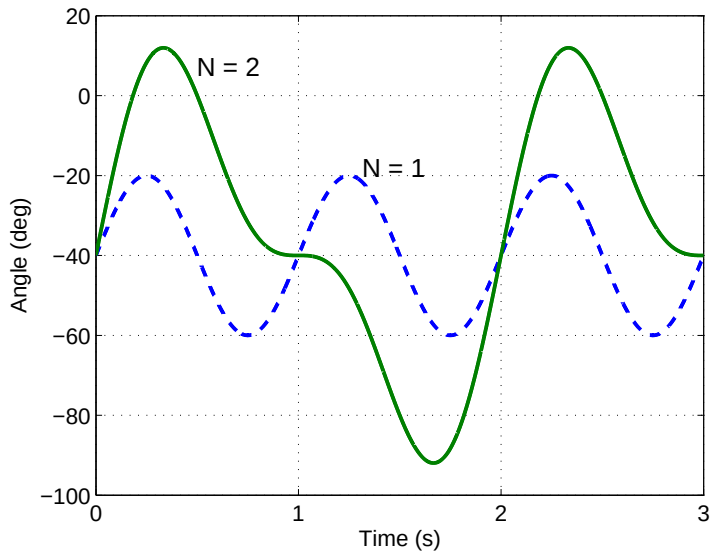

Figure 6.9: Examples of trajectories obtained with the PFS method.

This is a solution many times applied to humanoid robotics [20, 48, 78]. The described parameters can be adjusted to obtain different shapes. Tipically, evolutionary algorithms such as Genetic Algorithms (See section 3.1.4) are used to find values to these parameters [48, 78].

6.3.2 Implementation

The implementation of this model is similar to the previous one, except that it generates a different shape for the joint trajectory. It also uses a PID controller for each joint. The gait generator has an attribute δ, aiming at controlling the duration of a complete gait cycle. The initialization time is defined on each call of the init method with the current simulation time.

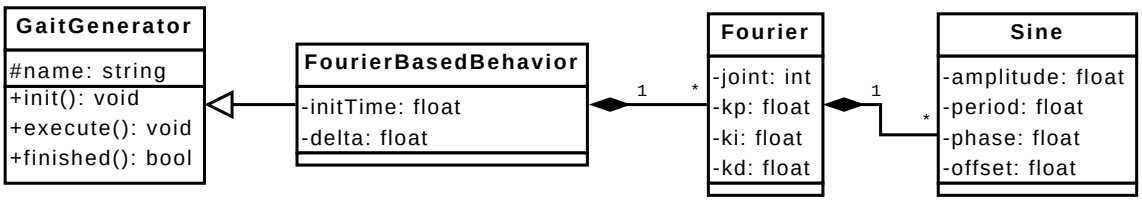

Figure 6.10: Class diagram for fourier based behaviors.

Figure 6.10 represents the class diagram of the fourier based behavior. Table 6.3 describes the classes FourierBasedBehavior, Fourier and Sine in more detail.

Class	Attribute	Description
FourierBasedBehavior	initTime	Initialization time
	delta	Duration of gait cycle
Fourier	joint	Identifier of the joint
	kp, ki, kd	PID controller gains
Sine	amplitude	Amplitude
	period	Period
	phase	Phase
	offset	Offset

Table 6.3: Description of the classes FourierBasedBehavior, Fourier and Sine.

6.3.3 Results

Figure 6.11 represents the trajectory of the knee when it is subject to the following trajectory equation: $f(t) = -40 + 20\sin(2\pi t) + 20\sin(4\pi t)$.

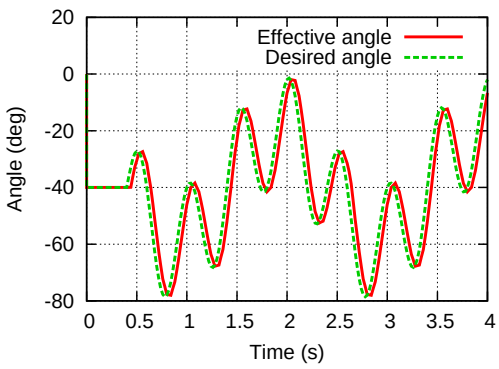

Figure 6.11: Evolution of a PFS trajectory.

6.3.4 Advantages

The main advantages of the PFS method are:

- More complex shapes are possible;
- More control over the angular velocities trajectories;
- The model is flexible;
- The motion description language[4] is user-friendly and well structured.

6.3.5 Drawbacks

The main drawbacks of the PFS method are:

- Requires some more complex knowledge about human behaviors;
- It is not easy to define trajectories manually.

6.4 Omnidirectional walking CPG

The ability to change the direction while moving has proved to have advantages in dynamic environments. Cornell introduced the omnidirectional drive for locomotion in 2000 for wheeled robots [79]. However, omnidirectional locomotion can be applied in many other situations [80, 81]. Based on the work of Sven Behnke [43, 39] an Omnidirectional Walking (ODW) was implemented. Sven Behnke describes this method for the humanoid robot Jupp (NimbRo humanoid team) [43].

Note: With the exception of some few changes for the adaptation to the NAO robot, all the trajectory generation equations here described are part of the study of Sven Behnke [43].

The engine consists of a CPG that generates trajectories of the legs. Three important things are essential when generating omnidirectional walking gait [43]: To shift the center of mass of the robot to the support foot, to short the non-support leg, and to move the non-support leg into the walking direction and the support leg against the walking direction. Using the same generator with different parameters, it is possible to generate forward walk, backward walk, side walk, curved walk and the turn motion.

6.4.1 Joint trajectory generation

The input parameters are described by the vector $a = (a_r, a_p, a_y)$ which corresponds to the lateral swing amplitude, forward swing amplitude and rotational amplitude, respectively. A gait phase, ϕ_{gait}, varies between $-\pi$ and π. ϕ_{leg} represents the phase of a leg and will correspond to $\phi_{gait} - \pi/2$ for the left leg and $\phi_{gait} + \pi/2$ for the right leg. The step of one leg during the whole gait is divided into five stages (Shifting, Shortening, Swinging, Loading and Balance), each of them with a special purpose. These stages are described in detail in the following sections.

[4]See Appendix C for more details

Shifting

The shifting stage consists of a lateral shift of the CoM over the support foot so the robot can stand in one leg without falling. This stage is essential to change the walking direction without falling. The function must produce a trajectory between $-a_{shift}$ and a_{shift} accordingly to the leg phase, where a_{shift} is the shifting amplitude. The following equation is used to produce such a trajectory:

$$\theta_{shift} = a_{shift} * \sin(\phi_{leg}) \tag{6.8}$$

To compute a_{shift}, the roll and pitch amplitudes should be taken into account. The following equation is used: $a_{shift} = 0.12 + 0.08 * \|(a_r, a_p)\| + 0.7 * |a_r|$. To shift the leg, both leg and foot roll angles are needed. The trajectories for leg and foot are both calculated based on the θ_{shift} computed above [43]:

$$\theta_{legshift} = \theta_{shift} \tag{6.9}$$

$$\theta_{footshift} = -0.5 * \theta_{shift} \tag{6.10}$$

Shortening

Since the robot shifts to be supported by only one foot, the non-support foot is shortened to be prepared to follow the walking direction without scrape on the ground [43]. The shortening phase, ϕ_{short}, determines the time course of the shortening [43]: $\phi_{short} = v_{short} * (\phi_{leg} + \pi/2 + o_{short})$, where v_{short} is the shortening duration and o_{short} is the phase shift relative to the shifting stage. To produce a smooth trajectory between the fully extended leg and the shortened leg, a shortening factor is computed using the following equation:

$$\gamma_{short} = \begin{cases} -a_{short} * 0.5 * (\cos(\phi_{short}) + 1), & if -\pi \leq \phi_{short} < \pi \\ 0, & otherwise \end{cases} \tag{6.11}$$

where $a_{short} = 0.2 + 2 * \|(a_r, a_p)\|$ is the shortening amplitude. This dependency on swing amplitudes makes the shortening amplitude increase with the gait speed. To lift the foot off while pointing into the walking direction the following equation is used.

$$\theta_{footshort} = \begin{cases} a_p * 0.5 * (\cos(\phi_{short}) + 1), & if -\pi \leq \phi_{short} < \pi \\ 0, & otherwise \end{cases} \tag{6.12}$$

The trajectory equation for $-\pi \leq \phi_{short} < \pi$ was originally presented in [43] as $-a_p * 0.125 * (cos(\phi_{short} + 1))$. This equation was generating incorrect movements for the NAO humanoid so it has to be adjusted to the one presented in Equation 6.12.

Swinging

The swinging stage should be planned carefully. It is the reason for most of the instability problems in biped walking gait generators, since the biped must drive its entire body into the walking direction supported by only one leg. The swing phase, ϕ_{swing}, determines the time course of the swing [43] and is defined as follows:

$$\phi_{swing} = v_{swing} * (\phi_{leg} + \pi/2 + o_{swing}) \tag{6.13}$$

where v_{swing} and o_{swing} are, respectively, the duration of the swing and the phase shift of the swing relative to the current gait phase, ϕ_{gait}. The swing is followed by a slow reverse swing during the rest of the walking cycle. The swinging is sinusoidal but the reverse motion is linear [43]. The original swinging equation, as stated in [43], was defined as:

$$\theta_{swing} = \begin{cases} \sin(\phi_{swing}), & if -\pi/2 \leq \phi_{swing} < \pi/2 \\ b * (\phi_{swing} - \pi/2) - 1, & if \pi/2 \leq \phi_{swing} \\ b * (\phi_{swing} + \pi/2) + 1, & otherwise \end{cases} \tag{6.14}$$

where $b = -2/(2 * \pi * v_{swing} - \pi)$ represents the reverse motion speed. The swing is performed using the leg joints and then partially balanced using the foot angles. The swing equations are stated as follows. This equation has discontinuities at for $\phi_{swing} = -\pi/2$ and $\phi_{swing} = \pi/2$. This problem is illustrated in Figure 6.12a.

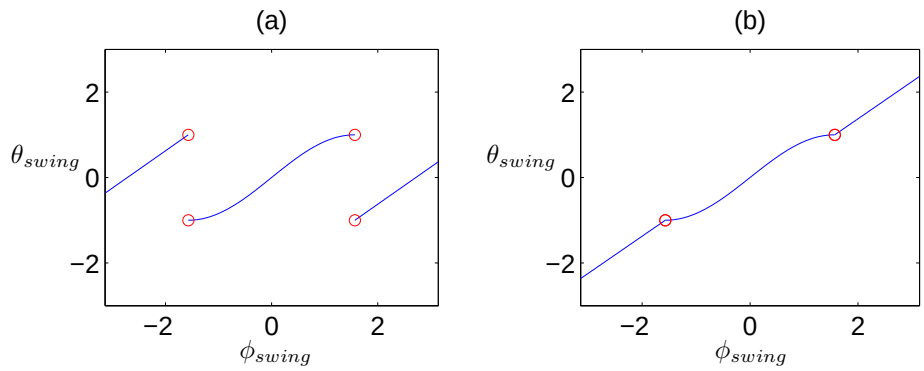

Figure 6.12: Swing trajectory. **(a)** Discontinuities of the original equation [43] **(b)** Corrected trajectory.

The trajectory was corrected using the following equation and is illustrated in Figure 6.12b:

$$\theta_{swing} = \begin{cases} \sin(\phi_{swing}), & if -\pi/2 \leq \phi_{swing} < \pi/2 \\ b * (\phi_{swing} - \pi/2) + 1, & if \pi/2 \leq \phi_{swing} \\ b * (\phi_{swing} + \pi/2) - 1, & otherwise \end{cases} \tag{6.15}$$

Finally, the final swing trajectories, both for the leg and for the foot, are described using the equations 6.16 to 6.20.

$$\theta^r_{legswing} = \lambda * a_r * \theta_{swing} \tag{6.16}$$
$$\theta^p_{legswing} = a_p * \theta_{swing} \tag{6.17}$$
$$\theta^y_{legswing} = \lambda * a_y * \theta_{swing} \tag{6.18}$$
$$\theta^r_{footswing} = \lambda * 0.25 * a_r * \theta_{swing} \tag{6.19}$$
$$\theta^p_{footswing} = 0.25 * a_p * \theta_{swing} \tag{6.20}$$

where λ represents the leg side (-1 for left leg and 1 for right leg). The foot swing trajectories were also changed from its original form [43] so that the leg side, λ, is multiplied by the roll angle, $\theta^p_{footswing}$, instead of the pitch angle, $\theta^r_{footswing}$, because the both legs perform the same movement in the pitch direction thus the leg side is just considered for roll and yaw rotations.

Loading

When a human performs a walking motion, the swing phase extends the non-support leg and it lands on the ground. After this, the non-support leg must be shortened once again to help the other leg to perform its movement correctly. Additionally, at this stage, the lateral shifting trajectory (which corresponds to a sine between $-a_{shift}$ and a_{shift}) is passing through its inflection point and the robot will now shift to the other side. This second shortening was called by Sven Behnke as the Loading phase [43]. The phase of this second shortening is determined by:

$$\phi_{load} = v_{load} * GetNormalizedAngleRad(\phi_{leg} + \pi/2 - \pi/v_{short} + o_{short}) - \pi \tag{6.21}$$

where v_{load} is the duration of the second shortening. The function *GetNormalizedAngleRad* normalizes its input argument angle (in radians) to an angle between $-\pi$ and π. Once again a shortening factor, γ_{load} will be needed:

$$\gamma_{load} = \begin{cases} -a_{load} * 0.5 * (\cos(\phi_{load}) + 1), & if -\pi \leq \phi_{load} < \pi \\ 0, & otherwise \end{cases} \tag{6.22}$$

where $a_{load} = 0.025 + 0.5 * (1 + \cos(|a_p|))$ represents the amplitude of the second shortening.

Balance

To ensure a stable gait, the robot is balanced every step, which leads the body to tilt and keep the upright posture [43]. For this stage, roll and pitch angles for the foot are determined, as well as an additional leg roll angle to avoid collisions between the legs during side or turn movements:

$$\theta^r_{footbalance} = 0.5 * \lambda * a_r * \cos(\phi_{leg} + 0.35) \tag{6.23}$$

$$\theta^p_{footbalance} = 0.02 + 0.08 * a_p - 0.04 * a_p * \cos(2\phi_{leg} + 0.7) \tag{6.24}$$

$$\theta^r_{legbalance} = 0.01 + \lambda * a_r + |a_r| + 0.1 * a_y \tag{6.25}$$

Output of the walking engine

The output of the engine, for each leg, will be $\theta_{leg} = (\theta^r_{leg}, \theta^p_{leg}, \theta^y_{leg})$, $\theta_{foot} = (\theta^r_{foot}, \theta^p_{foot})$ and the leg extension factor, γ ($-1 \geq \gamma \leq 0$). The leg extension corresponds to the distance between the pelvis plate and the foot plate (See Figure 6.13).

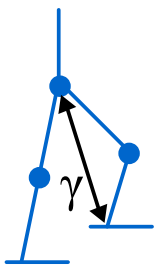

Figure 6.13: Representation of the leg extension factor (γ).

It is assumed that $\gamma = 0$ when the leg is fully extended and $\gamma = -1$ when the leg is shortened to $\eta_{min} = 0.775$ of its original length. The final output is computed by integrating the several patterns using the following equations:

$$\theta^r_{leg} = \theta^r_{legswing} + \theta_{legshift} + \theta^r_{legbalance} \tag{6.26}$$
$$\theta^p_{leg} = \theta^p_{legswing} \tag{6.27}$$
$$\theta^y_{leg} = \theta^y_{legswing} \tag{6.28}$$
$$\theta^r_{foot} = \theta^r_{footswing} + \theta_{footshift} + \theta^r_{footbalance} \tag{6.29}$$
$$\theta^p_{foot} = \theta^p_{footswing} + \theta_{footshort} + \theta^p_{footbalance} \tag{6.30}$$
$$\gamma = \gamma_{short} + \gamma_{load} \tag{6.31}$$

Leg kinematics interface

The leg kinematics interface handles the output produced by the engine and generates the joint target trajectories. The target relative leg length is based on the leg extension factor, γ, and is computed as follows:

$$\eta = 1 + (1 - \eta_{min}) * \gamma \tag{6.32}$$

The target relative leg length allows for the calculation of the knee joint, which is not directly generated:

$$\theta_{knee} = -2 * acos(\eta) \tag{6.33}$$

The yaw trajectory of the leg is taken directly from the θ^y_{leg} output. The trajectory of the knee will short the leg, but will also affect the leg and foot angles. Assuming that the thigh and the shank has the same length, if the leg is not twisted ($\theta^y_{leg} = 0$) it is enough to subtract $0.5 * \theta_{knee}$ from the leg and from the ankle to compensate this effect. For a twisted leg ($\theta^y_{leg} \neq 0$), the knee angle must be rotated before subtracting it from the leg and from the ankle [43]. The final trajectories of the several leg joints are determined as follows:

$$f(t) = \begin{cases} -2 * acos(\nu), & \text{for knee joint} \\ \theta^y_{leg}, & \text{for leg yaw joint} \\ \begin{pmatrix} \theta^r_{leg} \\ \theta^p_{leg} \end{pmatrix} + rotate_{\theta^y_{leg}} \begin{pmatrix} 0 \\ -0.5 * \theta_{knee} \end{pmatrix}, & \text{for leg roll and pitch joints} \\ \begin{pmatrix} 0 \\ -0.5 * \theta_{knee} \end{pmatrix} + rotate^{-1}_{\theta^y_{leg}} \left(\theta_{foot} - \begin{pmatrix} \theta^r_{leg} \\ \theta^p_{leg} \end{pmatrix} \right), & \text{for foot roll and pitch joints} \end{cases}$$

where, $rotate_X(\overrightarrow{Y})$ rotates the vector $\overrightarrow{Y}$, by an angle defined by X.

6.4.2 Implementation

This generator was implemented based on the class GaitGenerator. There is no supporting language since these trajectories are completely generated online. The implemented gait generator is internally divided in two modules: The ODW module itself and a Kinematics interface for the leg that handles the output of the ODW module and produces the target trajectories for each joint. Figure 6.14 shows the class diagram of the omnidirectional walk generator. Besides the input, $a = (a_r, a_p, a_y)$, some of the constants defined in the trajectory generation equations (e.g. shortening duration, shortening phase shift) are also included as input parameters to make the model more flexible.

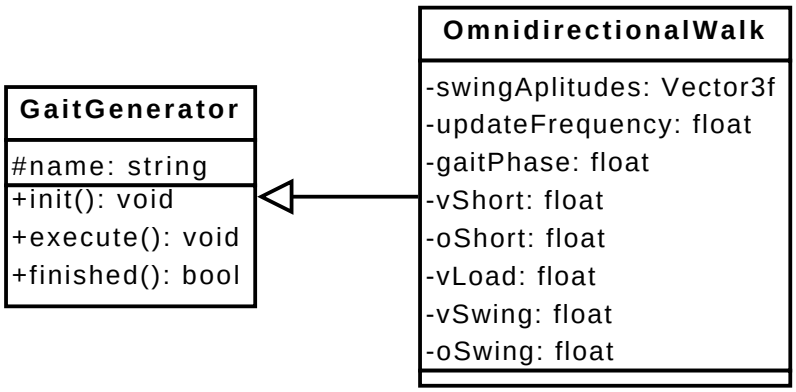

Figure 6.14: Class diagram for omnidirectional walk generator.

Table 6.4 describes the attributes of the class OmniDirectionalWalk in more detail. The gait phase, ϕ_{gait}, is incremented by the update frequency, Ψ, on each cycle. The higher the update frequency, the higher the gait speed but increasing this value may lead to instabilities.

Class	**Attribute**	**Description**
OmniDirectionalWalk	swingAmplitudes	Three-float vector containing the values for a_r, a_p and a_y.
	gaitPhase	Gait phase, ϕ_{gait}
	updateFrequency	Determines the increment of the gait phase on every cycle, Ψ.
	vShort	Shortening duration, v_{short}
	oShort	Shortening phase shift, o_{short}
	vLoad	Loading duration, v_{load}
	vSwing	Swinging duration, v_{swing}
	oSwing	Swinging phase shift, o_{swing}

Table 6.4: Description of the class OmniDirectionalWalk.

6.4.3 Results

The behavior generated by this CPG will be presented in detail in the Chapter 7.

6.4.4 Advantages

The main advantages of the implemented Omnidirectional Walking CPG are:

- Complex shapes are possible;
- Angular velocities trajectories are completely controlled;
- Allows several gaits with just one generator (forward walk, backward walk, sided walk, curved walk and turn on the spot);
- The generated gait is very similar to the natural human behavior.

6.4.5 Drawbacks

The main drawbacks of the implemented Omnidirectional Walking CPG are:

- There is no supporting language;
- Requires complex knowledge about human behaviors;
- The shifting stage leads to slower movements.

6.5 Summary

In this chapter four gait generation methods were presented. For each of them it was described how the joint trajectories are generated, the implementation structure and the main advantages and drawbacks. The step-based gait generator was developed out of the scope of this thesis but it is also presented to give an overview of the state of the agent before this thesis. It is possible to notice that as more complex are the shapes, more control is possible over the gait but harder is to define the parameters. Low-level control issues were successfully handled by the PID controllers.

The Slot Interpolation method was the first method developed and proved to be very effective. It describes an interpolation of a sine function between the current and the desired angle for each joint. This method allows for the control of the initial and final angular positions of each joint as well as the duration of the movement. Optionally, it is also possible to control the initial and final angular velocities for the joints.

The Partial Fourier Series (PFS) extends the previous method by allowing for more control over the joint position trajectory and also over the angular velocities trajectories. For each joint, it is possible to define a N-frequency PFS. This results on more complex shapes, harder to define, but also results on more natural, stable and fast gaits.

An Omnidirectional Walking CPG was implemented based on the work of Sven Behnke [43]. This method was studied in detail and implemented in the scope of this thesis and applied with success FC Portugal 3D simulated team that participated in on RoboCup 2008 (Suzhou, China). In spite of not being faster than other gaits in the competition, it has proved to be one of the most stable gaits.

Chapter 7

Developed behaviors

This chapter presents the developed behaviors and the tests performed with the simulated humanoid NAO, which employ the proposed trajectory planning methods, explained in the previous chapter. These behaviors were developed in the scope of RoboCup 3D Soccer competition using the Simspark Simulation Environment (Chapter 4). Some behaviors were initially developed for HOPE-1 simulated humanoid (Section 4.3.1) and used in RoboCup German Open 2008. With the introduction of the simulated model of NAO (Section 4.3.2), new behaviors were developed and tested for the new platform.

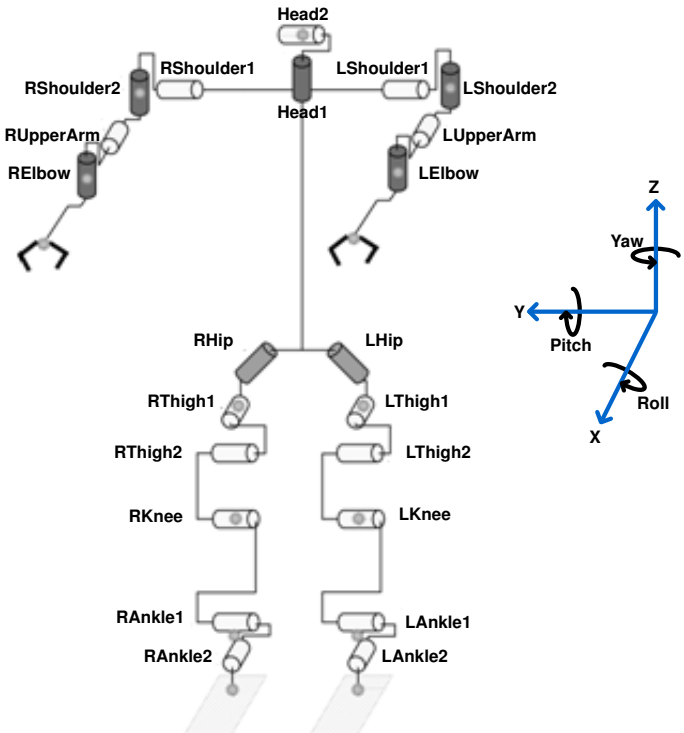

Figure 7.1: Humanoid structure and global referential. The arrows around the axes represent the positive direction of the pitch, roll and yaw rotations. Adapted from [46].

Figure 7.1 shows the humanoid structure and the referential axis considered in the remainder of this chapter.

7.1 Four-phase forward walking

The four phase forward walking behavior can be seen as a Finite State Machine (FSM) with four states, each one representing a pose of the biped. All states form a complete walking cycle: raise left leg (RL), land left leg (LL), raise right leg (RR) and land right leg (LR) (See Figure 7.2).

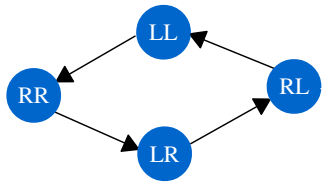

Figure 7.2: Four-phase walking cycle.

The four-phase walking behavior was implemented using the Sine Interpolation method and consists of four slots, each one representing one state. Repeating these states several times will result on a periodic motion, where the CoM is kept at a constant height and at a constant lateral direction. Using this strategy the CoM is not always inside the support polygon, as illustrated in Figure 7.3. Hence, the steps should be small and fast enough to avoid falling.

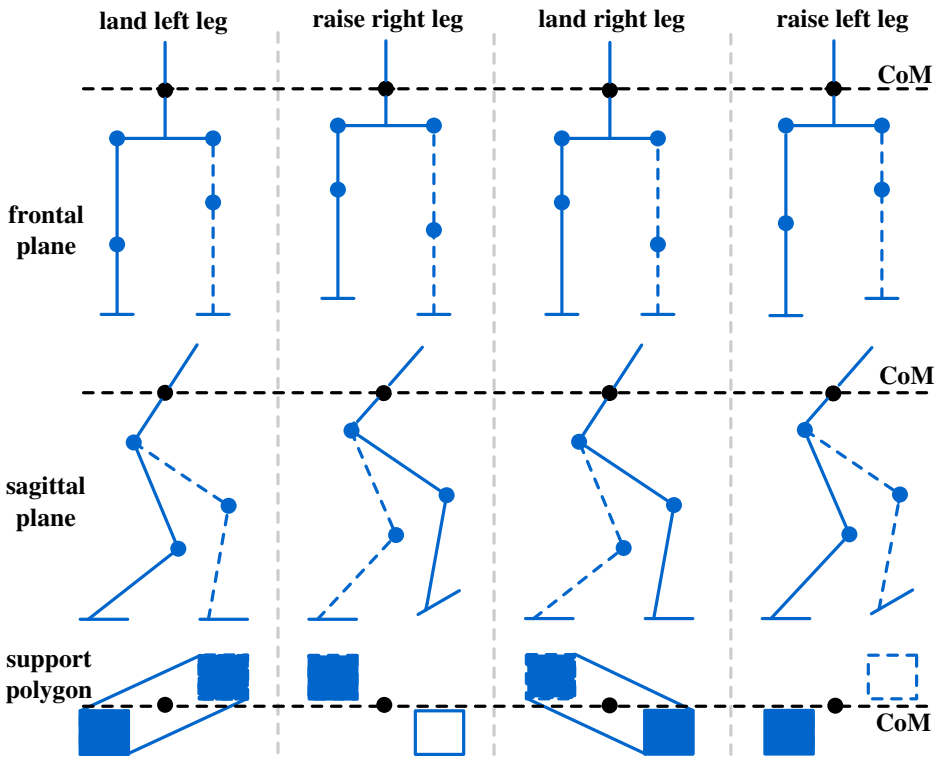

Figure 7.3: Four-phase walking structure.

7.1.1 Reducing the parameter space

The first approach was to produce the trajectories manually for the 3 joints of each leg that perform a pitch rotation (LThigh2, RThigh2, LKnee, RKnee, LAnkle1 and RAnkle1). Since there are four slots to define, this results on 24 parameters. Additionally, the duration of the slots (δ_1 to δ_4) should also be defined, increasing the size of the parameter space to a total of 28 parameters.

The forward walking can be considered a symmetric motion, i.e., the right leg produces the same movement of the left leg but shifted by half period. This reduces the parameter space by half (12 parameters for the joints and the duration of only two slots). Taking a more closer look, it is possible to define the gait using only 7 parameters (A to G) for the joints, as illustrated in the Figure 7.4.

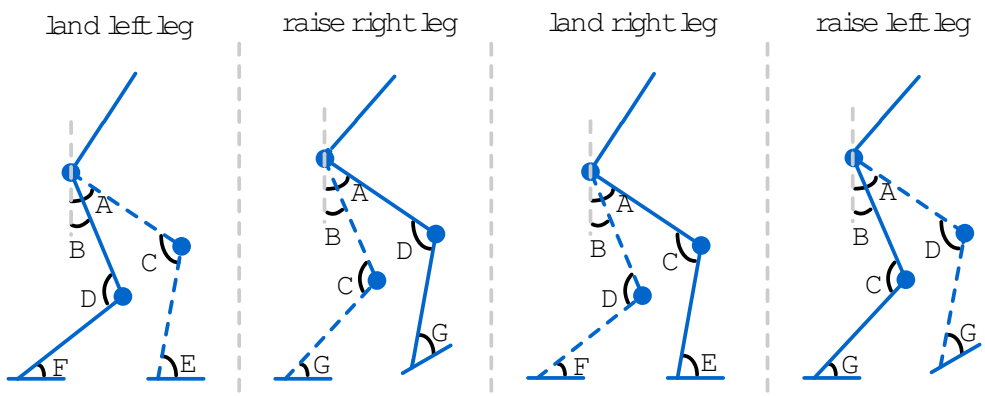

Figure 7.4: Four-phase walking: Reduced parameter space.

Defining 7 parameters for the joints and the duration of two slots (totalling 9 parameters) is easier than define the 28 original parameters. This reduction of the parameter space is favorable not only for the manual definition of parameters but also shortens the time that the optimization process will take to complete, as will be seen later.

7.1.2 Manual definition of parameters

The parameters were initially defined by a manual trial-and-error process. The values obtained manually for the parameters are present in the following table:

Parameter	Value	Parameter	Value
A	47.0	F	55.0
B	43.0	G	50.0
C	-83.0	δ_1	20.0
D	-87.0	δ_2	20.0
E	47.0		

Table 7.1: Four-phase walking: Values for the manually defined parameters.

The duration of the slots was set to the minimum[1] in order to produce a faster gait. However, a slot of 20 milliseconds is equivalent to a step function since the controller has no possibility to follow a sine in one cycle since it can only read a single value. Using this values for the slot duration results on a faster gait, but the controller is not always capable to follow the trajectories correctly, even when adjusting the PID controller gains. Since the steps are very small and the feet are too near from the ground, any small noise added by the simulator can make the robot scrape the feet, resulting on an unpredictable behavior. Figure 7.5 shows the generated joint trajectories. In this figure it is possible to note the symmetry between the left and the right legs.

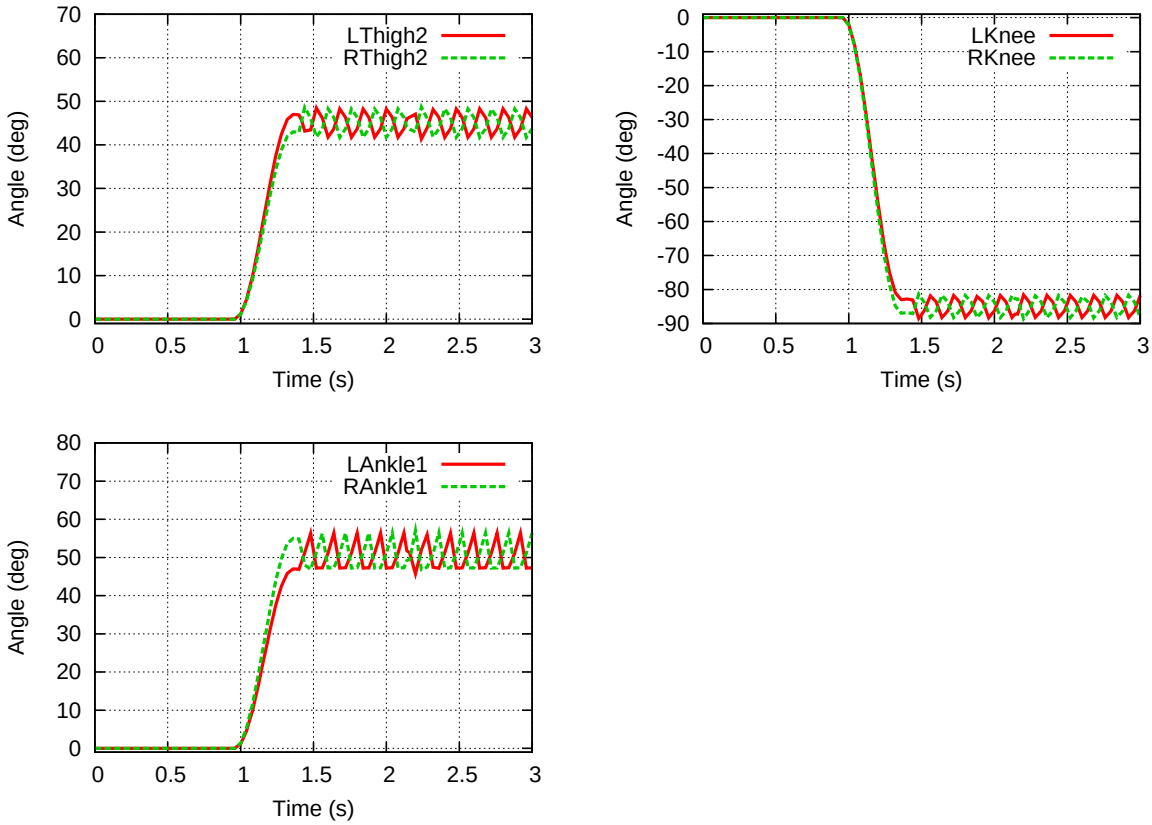

Figure 7.5: Four-phase walking: Joint trajectories. Only the joints that change are shown.

The trajectories are not very smooth due to the duration of the slots but most of the times the robot performs well. However, a more predictable behavior is desirable. Keep the non-support foot higher while swinging the leg would reduce the impact of the slightly variations of the joints from the desired trajectories. A better walking quality is reached by the employed optimization as will be explained later in this section.

[1]The duration of one simulation cycle is 20 milliseconds.

The CoM monitoring helps to evaluate the quality of the gait. Figure 7.6 shows the evolution of the CoM. The robot keeps a constant height during the movement but there is a problem with this behavior. Theoretically, the CoM would follow a linear trajectory between the feet and the biped would perform an exact forward walking, as illustrated earlier in Figure 7.3. Looking at the CoM trajectory it is possible to see that the biped does not follow exactly a forward trajectory, i.e., it tends to deviate from the target (Note the evolution of the dashed line, which shows the Y component). Since the gait is completely symmetric, a possible reason for this unnatural behavior is the noise associated with the simulation environment.

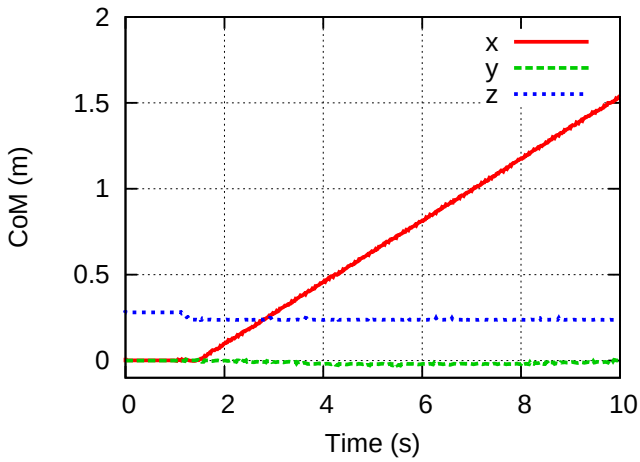

Figure 7.6: Four-phase walking: Evolution of CoM over time.

Figure 7.7 focus the problem by showing the evolution of the CoM and the placement of the feet in the XY plane. The CoM keeps between the both feet but the feet does not perform the correct trajectory. Another characteristic that can be seen is the small size of the steps. The robot gives 13 steps to travel 40 centimeters which means that the steps are very short.

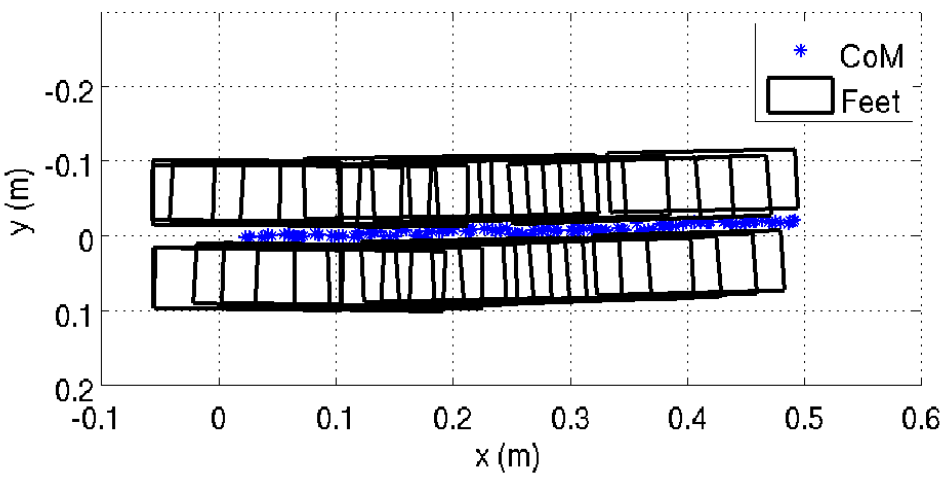

Figure 7.7: Four-phase walking: Evolution of CoM in the XY plane.

Another important measure to evaluate the quality of the walking gait is the evolution of the average velocity over time (Figure 7.8). With this simple walking gait, the robot is able to reach more than 15 centimeters per second.

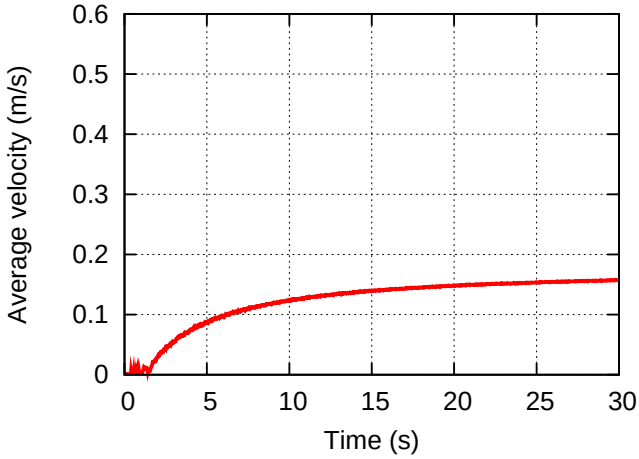

Figure 7.8: Four-phase walking: Average velocity over time.

The torso average oscillation is a measure taken from the gyroscope readings[2] and represents how much the torso oscillates over time in degrees per second. It is desirable to minimize this measure. Figure 7.9 shows the evolution of the torso average oscillation over time.

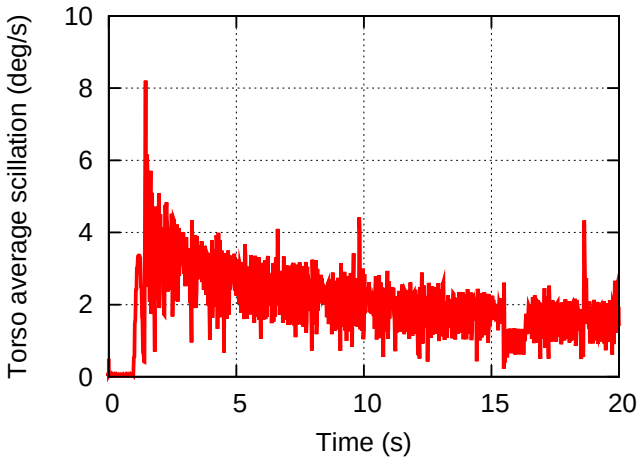

Figure 7.9: Four-phase walking: Torso average oscillation over time.

It is possible to note some disturbance during the load of the gait but soon the humanoid tends to stabilize the torso over time. The initial disturbance happens because the robot is completely stopped and then starts an abrupt movement.

Figure 7.10 shows NAO demonstrating the walking motion. It should be noticed the particular characteristic of having the knees bent to keep a constant height in the torso. Once again, it is possible to see the steps of small height and swing amplitude.

[2]See Appendix B for more details

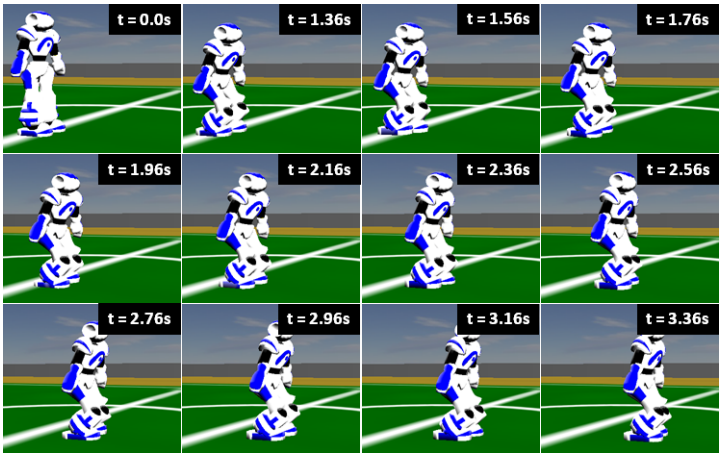

Figure 7.10: Four-phase walking: Screenshot.

From t=0.0s to t=1.36s the robot prepares the gait by bending the knees and placing the left foot slightly in front of the right foot. At t=1.56s it raises the right leg (the height of the step is very small and it is difficult to note in the screenshot) and then lands the right leg at t=1.76s. The same movement is then repeated for the left leg.

7.1.3 Optimization

Two different optimization algorithms were tested for the optimization of the four-phase forward walking: Hill Climbing (Hill Climbing (HC)) and Genetic Algorithm (GA)[3]. Both algorithms optimize the same parameters and use the same fitness function for evaluation so that a comparison could be possible. The parameters are the same that were defined manually. The optimization algorithm tests each individual and assigns a score (fitness value) to that individual. The individual with the minimum score at the end of the optimization process will be chosen as the best individual. In the case of the forward walking, a simple but effective fitness function to minimize can be the distance to the ball, assuming that the robot is placed far enough from it.

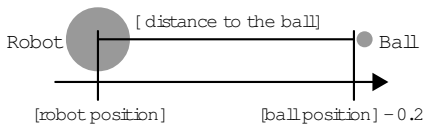

Figure 7.11: Distance to the ball as a possible fitness measure.

Additionally, the torso average oscillation is also used in order to obtain more stable gaits. The test of each individual is time-bounded but it also stops when the robot is 20 centimeters away from the ball to avoid touching it, as illustrated in the Figure 7.11, so the minimum ideal value for the fitness will would be 0.2. Two individuals capable of reaching that point at the same time will be distinguished by the value of the torso average oscillation.

[3]See Sections 3.1.1 and 3.1.4)

The final version of the fitness function is stated as follows:

$$fitness = d_{Ball} + \overline{\theta} \tag{7.1}$$

where d_{Ball} is the distance to the ball (in meters) and $\overline{\theta}$ is the average oscillation of the torso (in radians per second). For each parameter there is an associated range of values representing the definition domain. Accordingly to the previous knowledge of the expected values, the range for each parameter was set by hand to reduce the solution space (Table 7.2).

Parameter	**Range**	**Parameter**	**Range**
A	[37,57]	F	[45,65]
B	[33,53]	G	[40,60]
C	[-93,-73]	δ_1	[20,100]
D	[-97,-77]	δ_2	[20,100]
E	[37,57]		

Table 7.2: Four-phase walking: Definition domain of the parameters

Hill climbing results

HC is a very simple optimization algorithm that allows for a rough adjustment of the parameters, achieving a better quality in a reasonable time. The initial individual is considered the best so far and it consists of the manually defined parameters. The algorithm tests the current best individual and then uses a neighborwood function to find the neighbors. Each individual is a possible solution composed by the parameters from A to G, δ_1 and δ_2. The neighbors are the result of random variations applied to the current individual (Each parameter is changed with an uniform distribution between -0.10 and 0.10). After testing the current best individual and its neighbors, the one with the minimum fitness is choosen as the best individual. Table 7.3 shows the configuration of the hill climbing algorithm used to optimize the four-phase walking gait.

Size of an individual	9
Initial individual	Manually defined
Neighbors selected	20 (Random Uniform)
Tests per individual	10
Termination	Manual

Table 7.3: Four-phase walking: HC settings.

The simulation is non-deterministic so if the gait is not planned carefully, the same individual may produce very different results under the same conditions. This problem can be avoided by testing the same individual several times and assign it the worst score obtained, favoring the behaviors that are less sensitive to the non-determistic nature of the simulation. The obtained fitness evolution after about two days of optimization is represented in Figure 7.12.

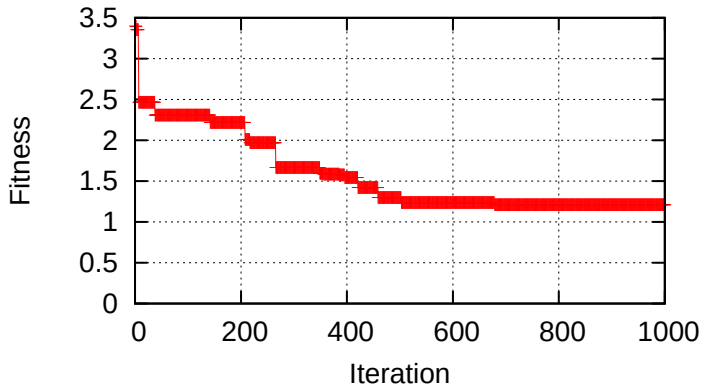

Figure 7.12: Four-phase walking with HC: Evolution of the fitness

The fitness value decreases during the first 600 iterations but after that it starts taking a stable value. In fact, the value at the 1000 iteration is 1.21. This value is yet far from the desired fitness value, which is 0.2. Table 7.4 shows the exact values of the best individual.

Parameter	**Value**	**Parameter**	**Value**
A	47.62	F	54.69
B	42.67	G	50.05
C	-82.61	δ_1	20.00
D	-87.41	δ_2	20.00
E	47.26		

Table 7.4: Four-phase walking with HC: Values for the parameters.

The values of δ_1 and δ_2 shows that the algorithm reached a local optima where it considers the value 20 the best value for that parameters and did not improved beyond that. Figure 7.13 shows the evolution of the CoM in the XY plane as well as the placement of the feet during the walking with the parameters optimized by the HC.

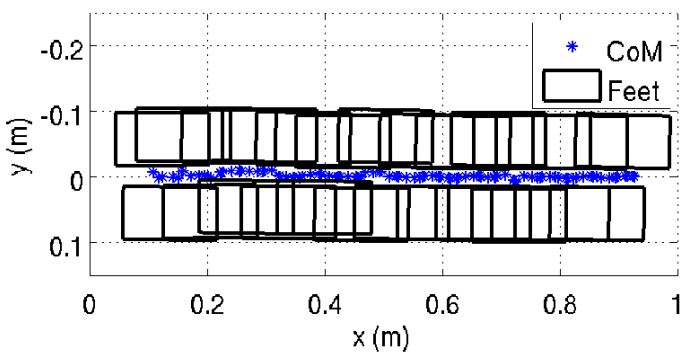

Figure 7.13: Four-phase walking with HC: Evolution of CoM in the XY plane.

Some improvements are seen in the direction followed by the robot. After travelling one meter the robot keeps its original orientation and lateral direction. The steps are larger than the ones obtained with the manually defined parameters but yet small, i.e., the robot gives 25 steps and travels about 1 meter.

The average velocity shows visible improvements, i.e., the robot exceeded the 20 centimeters per second, which is better than using the manually defined parameters (Figure 7.14).

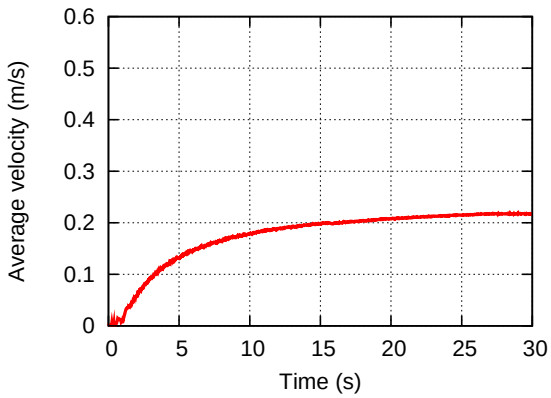

Figure 7.14: Four-phase walking with HC: Average velocity over time.

The torso average oscillation increased a little bit during the load, but tends to stabilize around a small value (Figure 7.15).

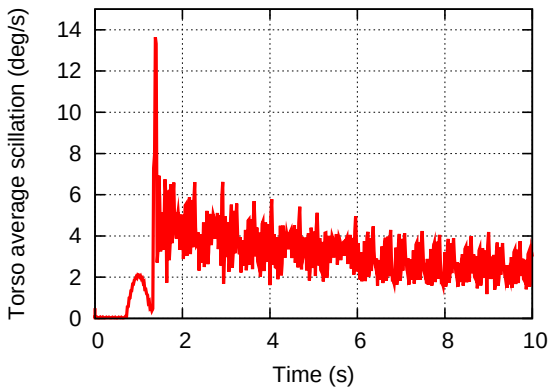

Figure 7.15: Four-phase walking with HC: Torso average oscillation over time.

In spite of being far from the desired fitness value, HC proved that can achieve good results in a reasonable time (about two days). This time is reduced by testing each individual only once but this will increase the sensibility of the gait to the non-determinism of the simulator.

Genetic algorithm results

GA imitates the biologic evolution of species by applying mutation and crossover operations in the individuals. This biological inspired method proved to achieve very good results in several situations. Table 7.5 shows the configuration of the GA used to optimize the four-phase walking gait.

Size of an individual	9
Population type	Real numbers
Population size	100
Initial population	Random Uniform
Selection	Roulette
Mutation	Uniform ($p_m = 0.5$)
Crossover	Scattered ($p_c = 0.8$)
Elite Count	10
Migration interval	5
Migration fraction	0.1
Termination	Manual

Table 7.5: Four-phase walking: GA settings.

The non-deterministic problem is now avoided by increasing the elitism parameter (Elite count). By setting this parameter, the 10 best individuals are kept on each successive generation and tested again Figure 7.16 shows the evolution of the fitness during the optimization process. The generation took almost 6 days to complete.

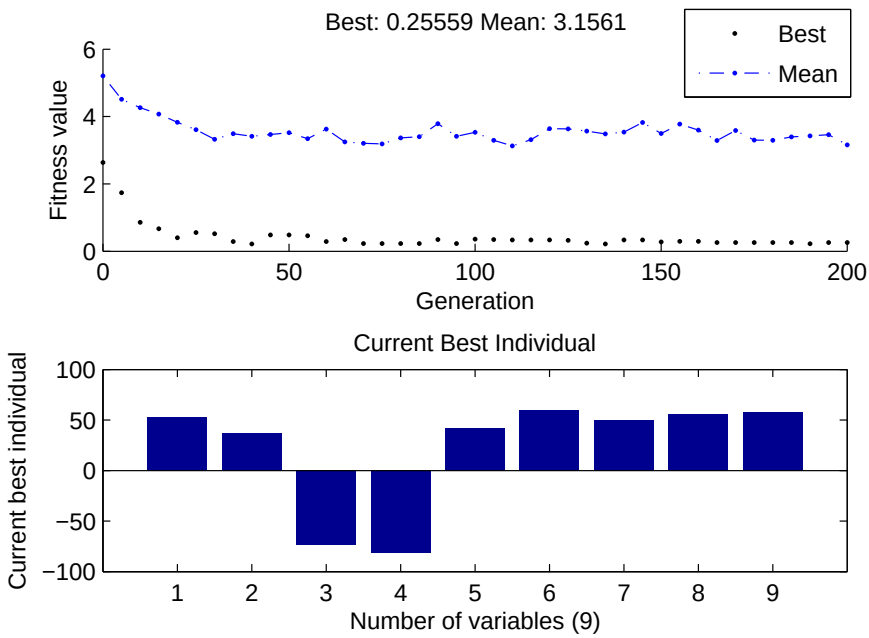

Figure 7.16: Four-phase walking with GA: Evolution of the fitness.

The fitness decreases fast but takes a long time to stabilize in some value. On the other hand, the mean fitness also decreases fast after reaching its minimum it oscillates over that value and does not tend to the minimum fitness value. This means that the algorithm is not converging. Ideally, the mean fitness would be near to minimum fitness and all the individuals of the final population would be good individuals. Decreasing the population size may solve this problem. However, small population sizes may lead to a prematurely convergence of the population on a inaccurante genetic form [82]. The minimum fitness is 0.25559, which is a good result and the best individual is also presented in the figure. Table 7.6 shows the exact values of the best individual.

Parameter	**Value**	**Parameter**	**Value**
A	52.8328	F	60.1334
B	36.574	G	49.4642
C	-73.9008	δ_1	55.7978
D	-81.4432	δ_2	57.9796
E	41.9147		

Table 7.6: Four-phase walking with GA: Values for the parameters

It is possible to note that GA was capable of achieving different values for δ_1 and δ_2, which means that it avoided the local minima reached by HC, where the value for that parameters is 20. Figure 7.17 shows the evolution of the CoM in the XY plane as well as the placement of the feet during the walking with the parameters generated by the GA.

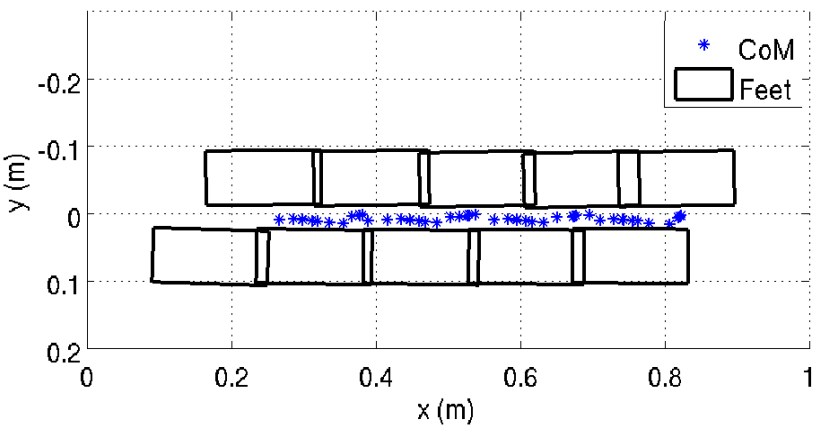

Figure 7.17: Four-phase walking with GA: Evolution of CoM in the XY plane.

It should be noticed that the biped follows a more correct forward trajectory, despite of small oscillations present. Moreover, the steps given by the biped are longer, i.e., the robot gives 5 steps and travels about 1 meter.

The average velocity also shows improvements. Now the robot is capable of reaching more than 40 centimeters per second which is the double of the speed reached using the manually defined parameters. Figure 7.18 shows the average velocity over time, when using the new parameters.

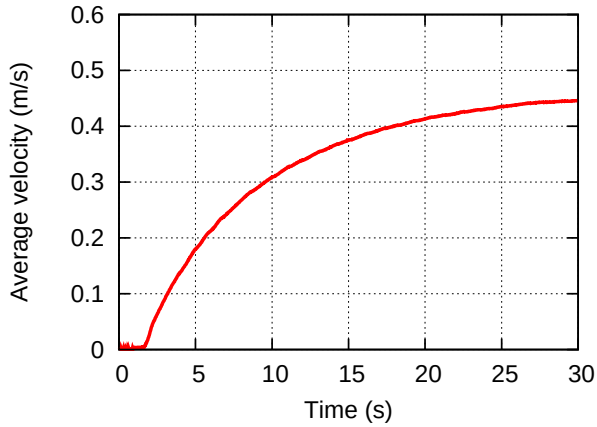

Figure 7.18: Four-phase walking with GA: Average velocity over time.

On the other hand, the torso average oscillation increased a little bit, as shown in Figure 7.19. Once again the torso average oscillation is more during the load of the gait but this time it stabilizes over time. This is not a bad result but it is more than the one achieved with the manually defined parameters. This behavior is expected since the speed increased too much causing an increase of the torso oscillation.

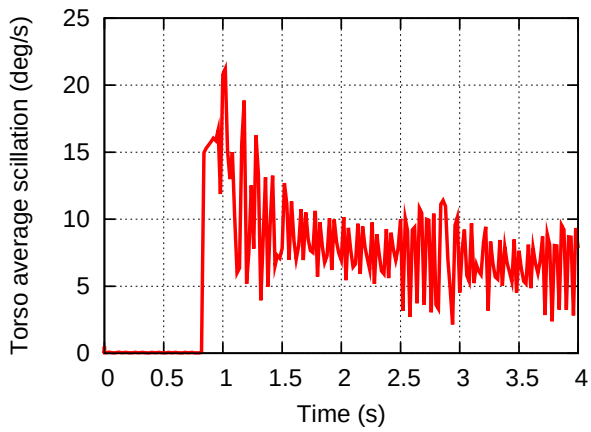

Figure 7.19: Four-phase walking with GA: Torso average oscillation over time

As expected, GA provided a way to improve the quality of the gait in a great scale but this can be improved even more by increasing the size of the initial population so the algorithm can converge faster. The fitness function chosen proved to be very effective.

7.2 Omnidirectional walking

Based on the work of Sven Behnke[43], an Omnidirectional Walking CPG was implemented. The definition of the CPG was already presented in the Section 6.4, on the subject of trajectory planning methods. This section presents the results obtained with the CPG. In the description of the CPG (Section 6.4) some variables were not assigned. Table 7.7 describe these variables.

Variable	Description
a_r	Lateral swing amplitude (roll rotation)
a_p	Forward swing amplitude (pitch rotation)
a_y	Forward swing amplitude (yaw rotation)
Ψ	Update frequency
v_{short}	Duration of the shortening stage
o_{short}	Phase shift of the shortening stage
o_{load}	Duration of the loading stage
v_{swing}	Duration of the swinging stage
o_{swing}	Phase shift of the swinging stage

Table 7.7: Omnidirectional Walking CPG control variables

The first 3 variables (a_r, a_p, a_y) are the key variables used to control the direction of the walking gait. The variable Ψ states the update frequency, i.e., on each cycle the gait phase, ϕ_{gait}, is increased by Ψ. The values for these variables will depend on what is pretended from the gait. By setting these values, it is possible to make the robot walk in different directions (e.g. forward walk, side walk, backward walk, curved walk) and turn on the spot. The remaining variables are internal to the CPG and were defined by Sven Benhke [43] as follows:

Variable	Value
v_{short}	3.0
o_{short}	-0.05
v_{load}	3.0
v_{swing}	2.0
o_{swing}	-0.15

Table 7.8: Omnidirectional Walking CPG: Values for the control variables defined by Sven Behnke [43]

This gait was developed to ensure static stability, which means that it keeps the CoM inside the support polygon. As explained in the Section 2.2.1, *the static stability criterion prevents the robot from falling down by keeping the CoM inside the support polygon by adjusting the body posture very slowly thus minimizing the dynamic effects.* Thus, it should be predictable that the generated gait will be slower than the previous ones. However, this gait has the advantage of being capable of performing different behaviors using the same CPG. The following sections will show the results when using the values presented in the Table 7.8.

7.2.1 Forward walking mode

The values of a_r, a_p, a_y and Ψ should be planned carefully so that the robot can perform the desired behavior as determistic as possible. For forward walking, only the pitch swing amplitude, a_p, should have a value different from zero. Assuming $a_r = 0$ and $a_y = 0$, there are three main possibilities:

- $a_p = 0$: The robot stands still
- $a_p > 0$: The robot walks forward
- $a_p < 0$: The robot walks backwards

The value of the update frequency, Ψ, will vary the walk velocity but also its stability. The slower the value of Ψ, the slower will be the velocity and the robot remains more stable. As the value of Ψ increases, the velocity also increases thus generating instabilities, because the trajectories will not be followed properly. A suitable set of values for forward walking are $(a_p, \Psi) = (0.25, 0.1)$. Figure 7.20 shows the generated joint trajectories.

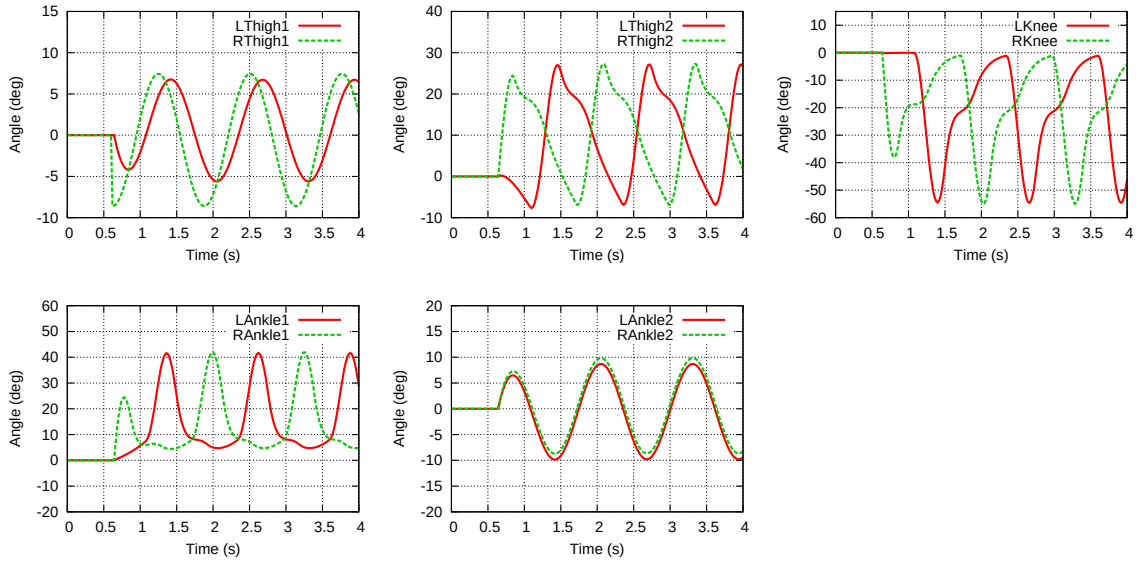

Figure 7.20: ODW Forward Walking: Joint trajectories. Only the joints that change are shown.

The generated trajectories are very smooth. It is possible to note double-frequency shapes. For example, the two peaks that appear for the knee joints (LKnee and RKnee) clearly show the shortening and the loading phases, respectively. The symmetry between the both legs is also clear. Moreover, all the joints show a smooth transition from its initial position to the desired trajectory.

The Figure 7.21 shows the evolution of the CoM in the XY plane. The CoM is shifted to the supported foot and keeps inside the support polygon everytime.

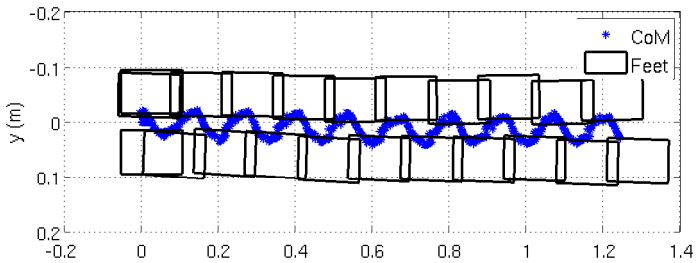

Figure 7.21: ODW Forward Walking: Evolution of CoM in the XY plane

This results on slower movements which can be depicted from the Figure 7.22, which shows the evolution of the average velocity of the robot over time. The robot reaches about 10 centimeters per second, which is a low velocity when compared to the other developed walking gaits.

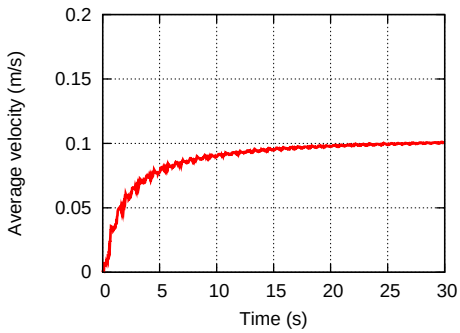

Figure 7.22: ODW Forward Walking: Average velocity over time

The generated behavior is, in fact, slower, but the torso average oscillation tends to stabilize on a slower value than the other walking gaits, as can be seen in Figure 7.23.

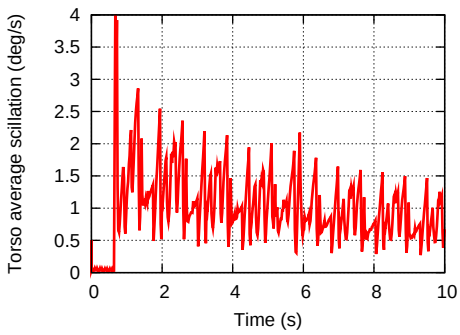

Figure 7.23: ODW Forward Walking: Torso average oscillation over time

These results are useful to show a powerful advantage of this walking gait when relative to the previous ones: The online control of direction. It is possible to correct the direction of the robot online, by adjusting the value of the input parameter a_y, which corresponds to the yaw swing amplitude. By setting the value of a_y to be, on each cycle, the direction to a point that is placed some meters away in the forward direction relative to the vision referential, the robot is able to adjust the angles of the hip joints to correct its direction. The hips follow a non-periodic trajectory, since it will be adjusted only when needed, as shown in Figure 7.24.

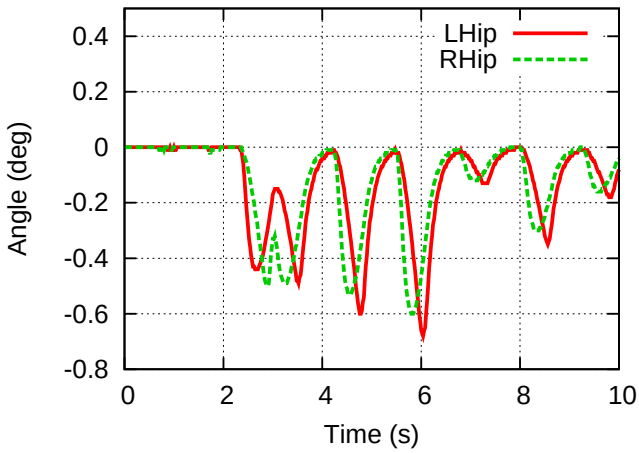

Figure 7.24: Corrected ODW Forward Walking: Hip trajectory

The evolution of CoM in the XY plane (Figure 7.25 also shows that the robot corrects itself to walk in the forward direction.

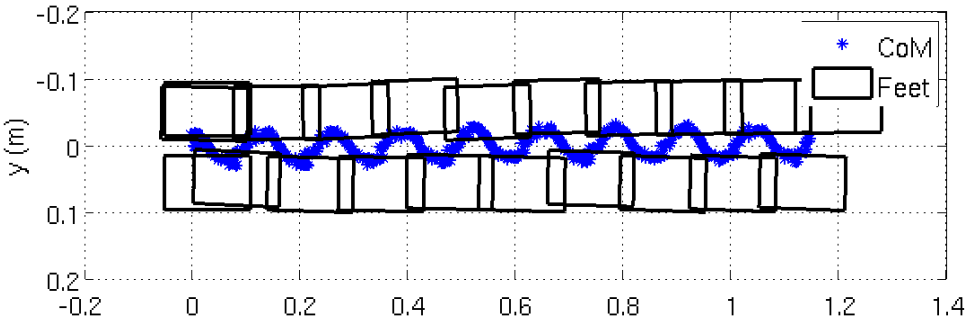

Figure 7.25: Corrected ODW Forward Walking: Evolution of CoM in the XY plane

This shows the great advantage of an omnidirectional walking gait. Moreover, this gait can be configured for side walking and turning on the spot, as will be explained in the following sections.

7.2.2 Side walking mode

For side walking, only the roll swing amplitude, a_r, should have a value different from zero. Assuming $a_p = 0$ and $a_y = 0$, there are three main possibilities:

- $a_r = 0$: The robot stands still
- $a_r > 0$: The robot walks to the right
- $a_r < 0$: The robot walks to the left

A suitable set of values for left side walking are $(a_r, \Psi) = (-0.07, 0.15)$. Figure 7.26 shows the generated joint trajectories using these values.

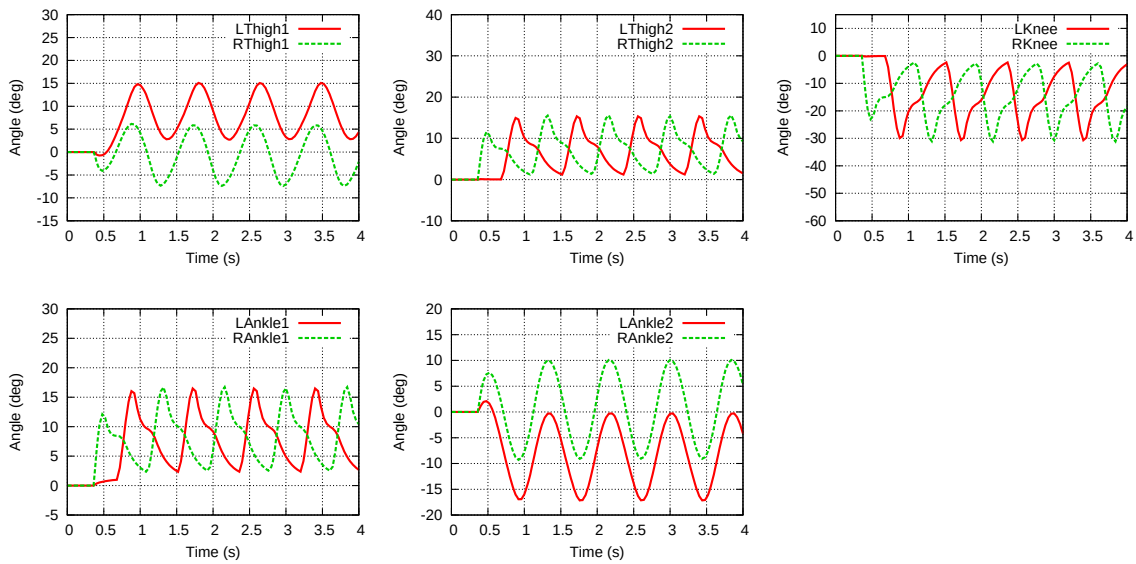

Figure 7.26: ODW Side Walking: Joint trajectories. Only the joints that change are shown.

The Figure 7.27 shows the evolution of the CoM in the XY plane. The figure shows that there is not deviation from the target direction.

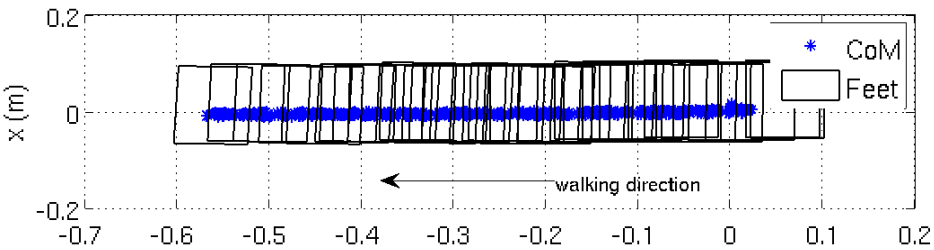

Figure 7.27: ODW Side Walking: Evolution of CoM in the XY plane

The side walking motion achieved with this parameters is not very fast. The average velocity only reaches a little bit more than 3 centimeters per second (Figure 7.28).

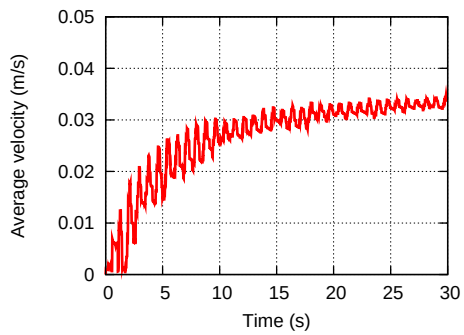

Figure 7.28: ODW Side Walking: Average velocity over time

Once again, the torso average oscillation stabilizes in a very low value, as can be seen in Figure 7.29.

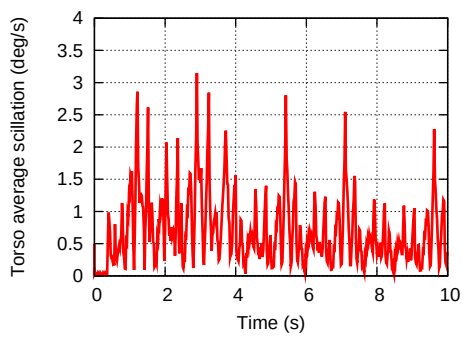

Figure 7.29: ODW Side Walking: Torso average oscillation over time

7.2.3 Turning mode

Besides being capable of walking into different directions, the Omnidirectional Walking CPG is also capable for generating trajectories to make the robot turn on the spot. This can be done by adjusting the value of a_y, which is the yaw swing amplitude. Assuming that a_p and a_y are zero, there are three main possibilities:

- $a_y = 0$: The robot stands still
- $a_y > 0$: The robot turns to its left
- $a_y < 0$: The robot turns to its right

A suitable set of values for turning right are $(a_y, \Psi) = (-0.25, 0.15)$. Figure 7.30 shows the generated joint trajectories using these values. Once again smooth trajectories were achieved. The hip joint plays an important role since it is the main responsible for the turning behavior.

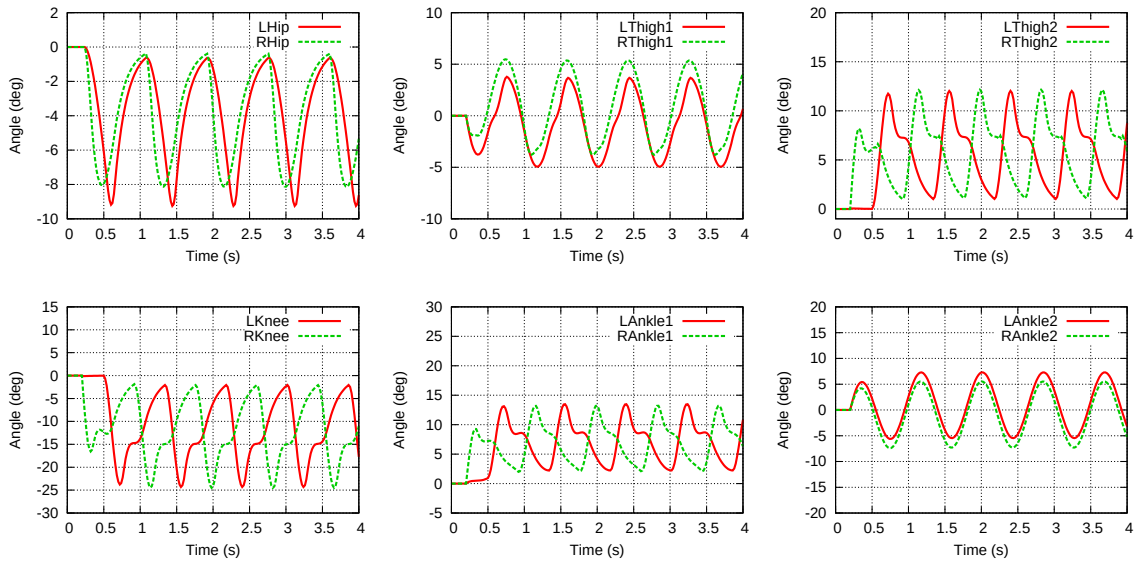

Figure 7.30: ODW Turning: Joint trajectories. Only the joints that change are shown.

The Figure 7.31 shows the evolution of the CoM in the XY plane. As can be seen, the robot performs a regular circle, finishing in the same place it started.

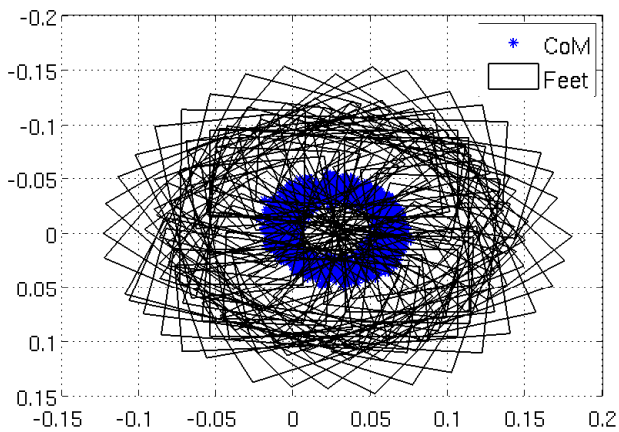

Figure 7.31: ODW Turning: Evolution of CoM in the XY plane

The turning motion is not very fast. A complete turn takes about 30 seconds to complete, as can be seen in the Figure 7.32, which results on an angular velocity of about 12 degrees per second.

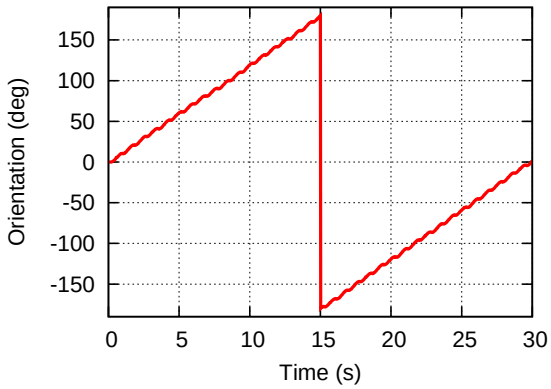

Figure 7.32: ODW Turning: Orientation of the body over time

Moreover, the torso average oscillation has in a low value, even during the load, as can be seen in Figure 7.33.

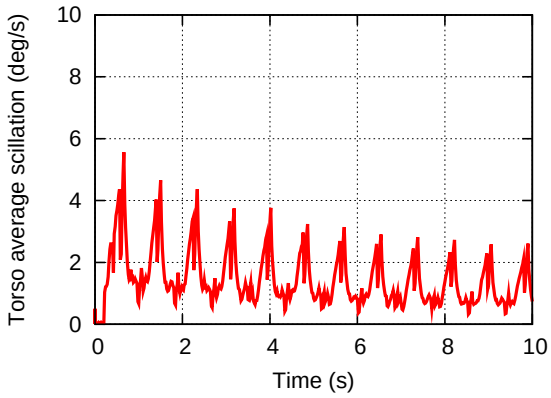

Figure 7.33: ODW Turning: Torso average oscillation over time

This was the main locomotion strategy of FC Portugal during the RoboCup 2008 (Suzhou, China). It was clearly one of the most stable walking gaits in the competition and not so sensitive to the machine load and network disturbances. However, the achieved velocities were not able to compete with the best teams, as will be seen in Section 7.9.

7.3 Forward walking based on PFS

After the competition in Suzhou, several improvements were made aiming at reaching the quality of the best teams. One of the improvements was a walking gait based on PFS, which is presented in this section. Due to the advantages of this trajectory planning method, it was predicted that the achieved results would be better.

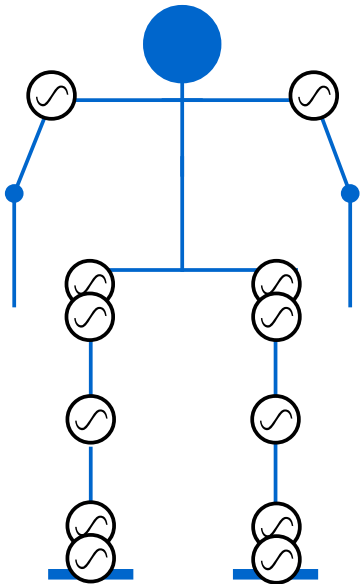

Figure 7.34: Forward Walking based on PFS. The figure represents the oscillators placed on the joints considered for the motion.

The main idea behind the definition of this gait is to place an oscillator on each joint we pretend to move, as illustrated in Figure 7.34. The oscillators are placed on the following joints: LShoulder1, RShoulder1, LThigh1, RThigh1, LThigh2, RThigh2, LKnee, RKnee, LAnkle1, RAnkle1, LAnkle2 and RAnkle2. The shoulders are used to help on stabilizing the robot while walking. The joints that perform roll rotations (LThigh1, RThigh1, LAnkle2 and RAnkle2) are used to shift the CoM over the support foot before raising the non-support foot, aiming at producing a more stable gait. For the definition of this gait, 12 single-frequency oscillators are used.

7.3.1 Reducing the parameter space

Since each single-frequency oscillator will have 4 parameters to define, $12 * 4 = 48$ parameters are needed to completely define the gait. As explained before, it is possible to assume a sagittal symmetry, which states same movements between corresponding left and right sided joints with a half-period phase shift. Hence, it is possible to reduce the number of parameters by half of the original size, resulting on $6 * 4 = 24$ parameters. Additionally, the period of all oscillators should be the same to keep all the joints synchronized by a single frequency clock. This consideration reduce the number of parameters to $6 * 3 + 1 = 19$ parameters.

7.3.2 Defining the oscillators

After reducing the number of parameters, the next step is to define the gait. These include defining the equations of the several oscillators. Let $f_X(t)$ be the trajectory equation for the joint X. The gait trajectories can be generally defined as follows:

$$f_{LShoulder1}(t) = A_1 \sin\left(\frac{2\pi}{T}t + \phi_1\right) + \alpha_1 \tag{7.2}$$

$$f_{RShoulder1}(t) = A_1 \sin\left(\frac{2\pi}{T}t + \phi_1 + \pi\right) + \alpha_1 \tag{7.3}$$

$$f_{LThigh1}(t) = A_2 \sin\left(\frac{2\pi}{T}t + \phi_2\right) + \alpha_2 \tag{7.4}$$

$$f_{RThigh1}(t) = A_2 \sin\left(\frac{2\pi}{T}t + \phi_2\right) + \alpha_2 \tag{7.5}$$

$$f_{LThigh2}(t) = A_3 \sin\left(\frac{2\pi}{T}t + \phi_3\right) + \alpha_3 \tag{7.6}$$

$$f_{RThigh2}(t) = A_3 \sin\left(\frac{2\pi}{T}t + \phi_3 + \pi\right) + \alpha_3 \tag{7.7}$$

$$f_{LKnee}(t) = A_4 \sin\left(\frac{2\pi}{T}t + \phi_4\right) + \alpha_4 \tag{7.8}$$

$$f_{RKnee}(t) = A_4 \sin\left(\frac{2\pi}{T}t + \phi_4 + \pi\right) + \alpha_4 \tag{7.9}$$

$$f_{LAnkle1}(t) = A_5 \sin\left(\frac{2\pi}{T}t + \phi_5\right) + \alpha_5 \tag{7.10}$$

$$f_{RAnkle1}(t) = A_5 \sin\left(\frac{2\pi}{T}t + \phi_5 + \pi\right) + \alpha_5 \tag{7.11}$$

$$f_{LAnkle2}(t) = A_6 \sin\left(\frac{2\pi}{T}t + \phi_6\right) + \alpha_6 \tag{7.12}$$

$$f_{RAnkle2}(t) = A_6 \sin\left(\frac{2\pi}{T}t + \phi_6\right) + \alpha_6 \tag{7.13}$$

where $A_{i=1..6}$ are amplitudes, T is the period, $\phi_{i=1..6}$ are phases and $\alpha_{i=1..6}$ are offsets. The parameters for this gait were not defined manually because a great number of parameters were available. Since the genetic algorithm was implemented, it was more practical to configure and extend its capabilities to generate parameters for any arbitrary gait. The next section describes the process for automatic generation of these 19 parameters.

7.3.3 Automatic generation of parameters

The parameters described in the previous section were defined by a GA. The option for using GA is because it proved to be very good in achieving results. A range of possible values for each parameter is used to generate the initial population of individuals. These ranges are described in the Table 7.9. The ranges presented in the table were carefully planned with base on the experience obtained from the definition of the other gaits, aiming at reducing the time for the genetic algorithm. Some values are not wanted for some parameters so these values can be excluded before starting the generation process.

Parameter	Range	Parameter	Range	Parameter	Range
A_1	[45,60]	ϕ_1	$[-\pi,\pi]$	α_1	[-100,-80]
A_2	[0,7]	ϕ_2	$[-\pi,\pi]$	α_2	[-7,7]
A_3	[40,60]	ϕ_3	$[-\pi,\pi]$	α_3	[20,40]
A_4	[30,40]	ϕ_4	$[-\pi,\pi]$	α_4	[-40,30]
A_5	[45,60]	ϕ_5	$[-\pi,\pi]$	α_5	[20,40]
A_6	[0,7]	ϕ_6	$[-\pi,\pi]$	α_6	[-7,7]
T	[0.02,1.0]				

Table 7.9: Forward Walking based on PFS: Range for the parameters

The configuration of the GA had to be redefined to handle this new gait. Table 7.10 shows the configuration of the GA used to generate the Forward Walking based on PFS.

Size of an individual	19
Population type	Real numbers
Population size	100
Initial population	Random Uniform
Selection	Roulette
Mutation	Uniform ($p_m = 0.5$)
Crossover	Scattered ($p_c = 0.8$)
Elite Count	10
Migration interval	5
Migration fraction	0.1
Termination	300 generations reached

Table 7.10: Forward Walking based on PFS: Genetic Algorithm settings

The fitness function is the same used for the four-phase walking gait. Remembering the fitness function, it is defined by the following expression:

$$fitness = d_{Ball} + \overline{\theta} \tag{7.14}$$

where d_{Ball} is the distance to the ball (in meters) and $\overline{\theta}$ is the average oscillation of the torso (in radians per second).

Since each test is time-bounded but also terminated when the robot is 20 centimeters away from the ball, the ideal value for the fitness will be $d_{Ball} = 0.2$ and $\overline{\theta} = 0$, thus resulting on a fitness value of 0.2. The generation process took five entire days to complete. Figure 7.35 shows the evolution of the fitness during the optimization process.

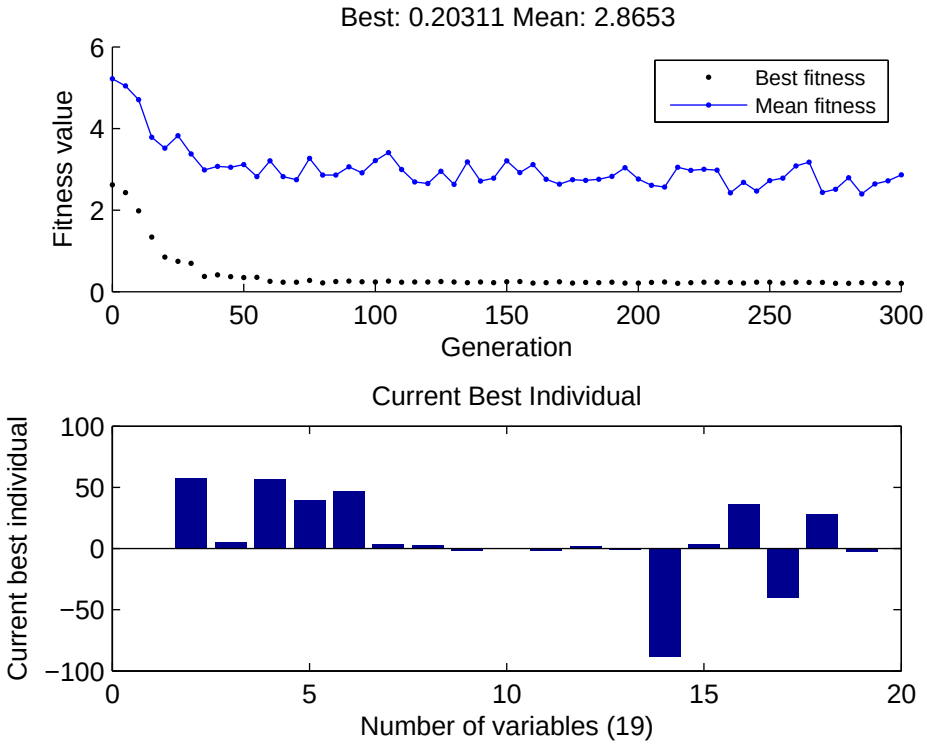

Figure 7.35: Forward Walking based on PFS: Evolution of the fitness.

The fitness decreases fast and stabilizes in a hundred of generations. However, it is possible to note that the mean fitness is decreasing but not very fast. This means that the algorithm is converging very slowly. Ideally, the mean fitness would be near to minimum fitness and all the individuals of the final population would be good individuals. Decreasing the population size may solve this problem. However, small population sizes may lead to a prematurely convergence of the population on a inaccurante genetic form [82]. The minimum fitness is 0.20311, which is a very good result and the best individual is also presented in the figure. Table 7.11 shows the exact values of the best individual.

Param.	**Value**	**Param.**	**Value**	**Param.**	**Value**
A_1	57.1841664300830	ϕ_1	2.9594374142694	α_1	-88.4624498858096
A_2	5.6445494158487	ϕ_2	-2.2855532147247	α_2	3.6390431719035
A_3	57.1211279163714	ϕ_3	0.0886662311291	α_3	35.9535879997734
A_4	39.6205376170462	ϕ_4	-1.8292118427537	α_4	-39.9481259274544
A_5	46.6315429218987	ϕ_5	1.7640169998412	α_5	28.5095204716035
A_6	3.7946795761163	ϕ_6	-1.2066624249361	α_6	-2.9360040558212
T	0.3711045452651				

Table 7.11: Forward Walking based on PFS: Values for the parameters

Figure 7.36 show the trajectories of the joints over time. Smooth trajectories were achieved. A PD controller was used for each joint. The robot performs well and it becomes less sensitive to disturbances due to the larger and higher steps.

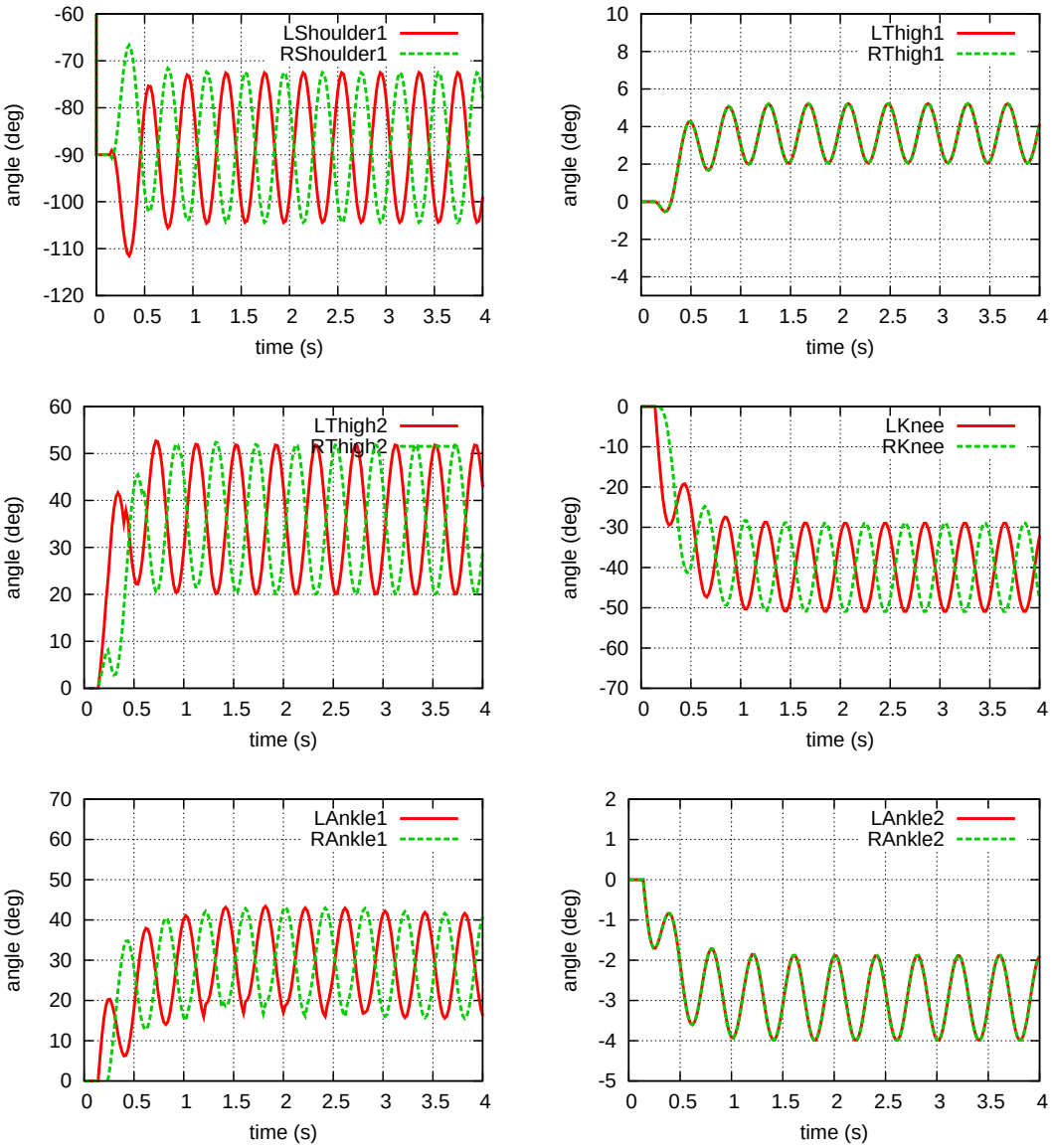

Figure 7.36: Forward Walking based on PFS: Joint trajectories. Only the joints that change are shown.

Figure 7.37 shows the evolution of the CoM over time. It should be noticed here that the robot keeps a constant height (Z component) and does not deviate from the forward direction (Y component).

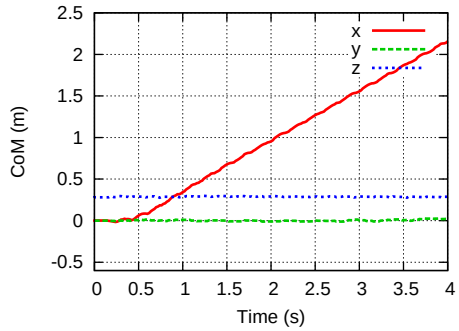

Figure 7.37: Forward Walking based on PFS: CoM trajectory over time

Figure 7.38 shows the evolution of the CoM in the XY plane. It also shows the placement of the feet. The graphic shows that the CoM follows a linear trajectory in the forward direction, which is the main characteristic of the walk. Another characteristic shown by the same graphic is the size of the steps, which is big when compared with the four-phase walking gait. The robot gives 5 steps and travels more than 1 meter.

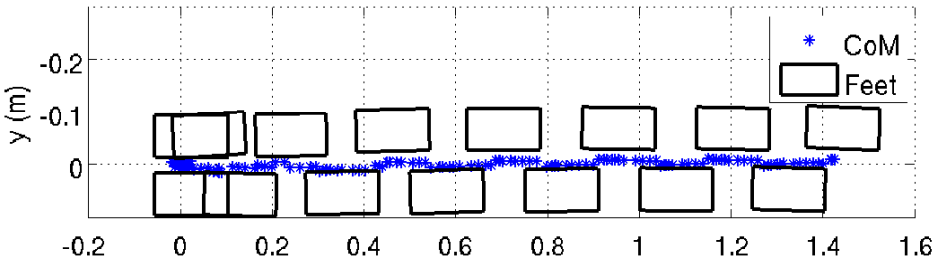

Figure 7.38: Forward Walking based on PFS: The CoM and the feet in the XY plane

The average velocity (Figure 7.39) shows very good results. More than 50 centimeters per second were achieved. This is a good velocity taking into account the torso average oscillation, that is represented in the Figure 7.40.

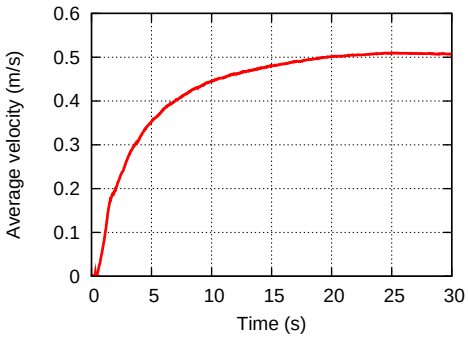

Figure 7.39: Forward Walking based on PFS: Average velocity over time

The torso average oscillation is, in general, less than the same measure obtained for the four-phase walking gait optimized with GA.

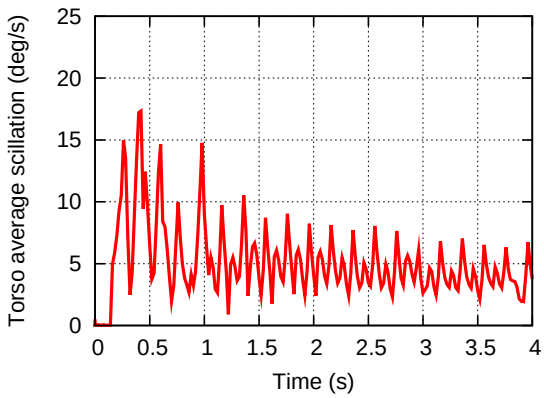

Figure 7.40: Forward Walking based on PFS: Torso average oscillation over time

Figure 7.41 shows NAO walking forward using the proposed solution. At t = 1.64 the biped already covered a great distance. It is possible to note the large steps and also the height of the steps. Due to these two properties, this walking gait is less sensitive to the non-deterministic problems that came from the simulation.

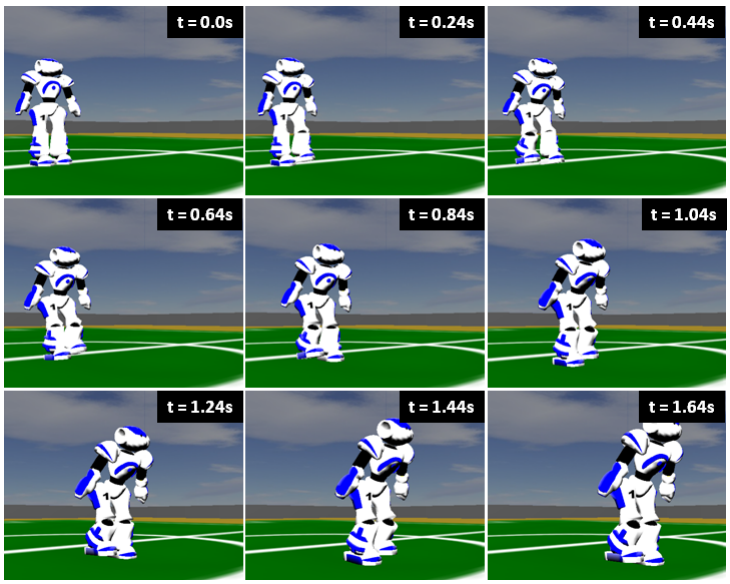

Figure 7.41: Forward Walking based on PFS: Screenshot

The PFS method proved to be very effective to generate periodic gaits, which is the case of the walking gait.

7.4 Side walking

The ability to walk sideward is very useful to perform adjustments of the position when the robot is near to the target position or even when walking sideward takes less time to place the robot on the right place, at the right position and with the desired target orientation. In the scope of a robotic soccer competition, this might be very useful to approach the ball quickly to be ready to kick it without having to walk around it. The side walking motion moves the CoM sideward without changing the absolute position in the forward direction. Figure 7.42 shows the stages of the side walking.

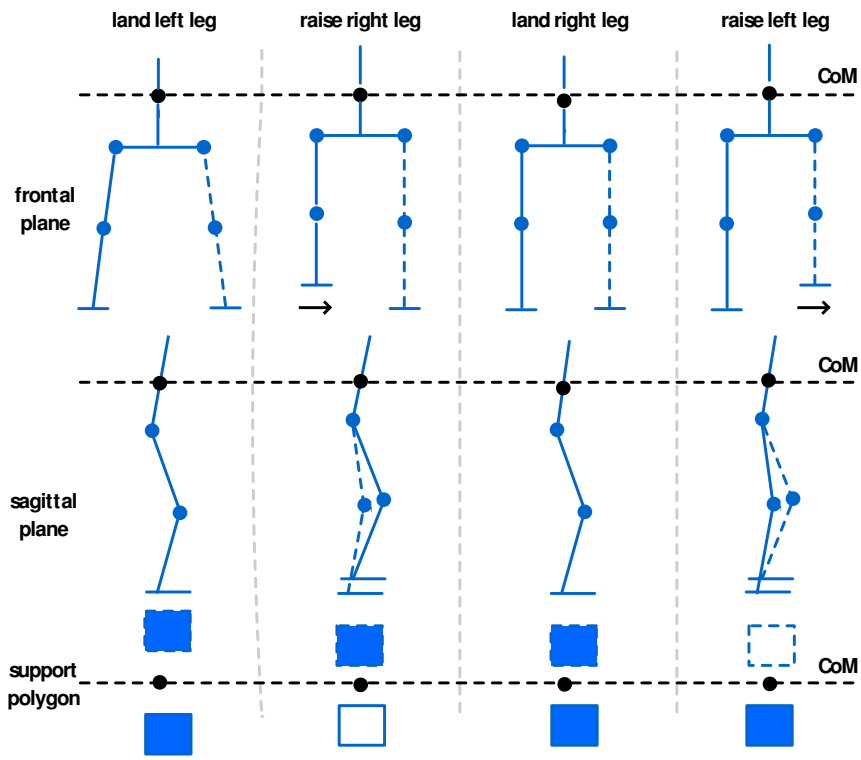

Figure 7.42: Side walking

Due to its periodic nature, this method was also developed using the Partial Fourier Series method. The placement of the oscillators is similar as described in the Figure 7.34, with the exception that the arms oscillators are not defined.

7.4.1 Defining the oscillators

Thinking of how to define the oscillators took some days. This is because too many parameters are involved. Due to the experience acquired when defining parameters manually, it was possible to define a set of oscillators. An important thing to notice first is that the offset parameters define the gait when the oscillators keep a constant value (no oscillation). A sine will oscillate with some amplitude around the value stated by the offset. For fast and stable movements, it was already seen that it is better to keep the knees bent while moving. The offsets will define this bent position.

Some drafts of sine graphics were made in order to reach a final decision of how to define the trajectories. The final trajectories are described from Equation 7.15 to Equation 7.24. Basically, it is desirable to follow the phases represented in Figure 7.42, i.e, when the legs are opened the knees are stretched to its maximum (considering the offset) and when the legs are closed the non-support leg is shortened. It is possible to create this effect by differing the sagittal plane and the frontal plane by a quarter of the period, $\pi/2$. The trajectories for the thigh and the ankle in the sagittal plane are considered to be symmetric thus symmetric amplitudes are used. On the other hand, in the frontal plane, the thigh and the ankle are assumed to follow the same trajectory thus the same amplitudes are used.

$$f_{LThigh1}(t) = A_1 \sin\left(\frac{2\pi}{T}t + \frac{\pi}{2}\right) \tag{7.15}$$

$$f_{RThigh1}(t) = -A_1 \sin\left(\frac{2\pi}{T}t + \pi + \frac{\pi}{2}\right) \tag{7.16}$$

$$f_{LThigh2}(t) = A_2 \sin\left(\frac{2\pi}{T}t\right) + \alpha_1 \tag{7.17}$$

$$f_{RThigh2}(t) = -A_2 \sin\left(\frac{2\pi}{T}t + \pi\right) + \alpha_1 \tag{7.18}$$

$$f_{LKnee}(t) = A_3 \sin\left(\frac{2\pi}{T}t\right) + \alpha_2 \tag{7.19}$$

$$f_{RKnee}(t) = -A_3 \sin\left(\frac{2\pi}{T}t + \pi\right) + \alpha_2 \tag{7.20}$$

$$f_{LAnkle1}(t) = A_2 \sin\left(\frac{2\pi}{T}t\right) + \alpha_3 \tag{7.21}$$

$$f_{RAnkle1}(t) = -A_2 \sin\left(\frac{2\pi}{T}t + \pi\right) + \alpha_3 \tag{7.22}$$

$$f_{LAnkle2}(t) = -A_1 \sin\left(\frac{2\pi}{T}t + \frac{\pi}{2}\right) \tag{7.23}$$

$$f_{RAnkle2}(t) = A_1 \sin\left(\frac{2\pi}{T}t + \pi + \frac{\pi}{2}\right) \tag{7.24}$$

7.4.2 Manual definition of parameters

The PFS parameters were tuned manually during some days and good results were achieved. The values found for the parameters described in the previous section are represented in Table 7.12.

Parameter	**Value**	**Parameter**	**Value**
T	0.4	α_1	30.0
A_1	6.0	α_2	-50.0
A_2	4.8	α_3	20.0
A_3	-9.6		

Table 7.12: Side walking: Values for the manually defined parameters

For the right-side walking, it is enough to change the parameter A_1 to -6. The trajectories generated during the turn motion are described in Figure 7.51. It should be noticed a smooth sinusoidal pattern on each joint. The robot automatically adapts itself to the inherent pose, i.e., in spite of the need to keep the knees bent, the robot does not need to use another gait to prepare the side walking when it starts from a completely upright position because the PFS method provides a smooth transition from its original pose.

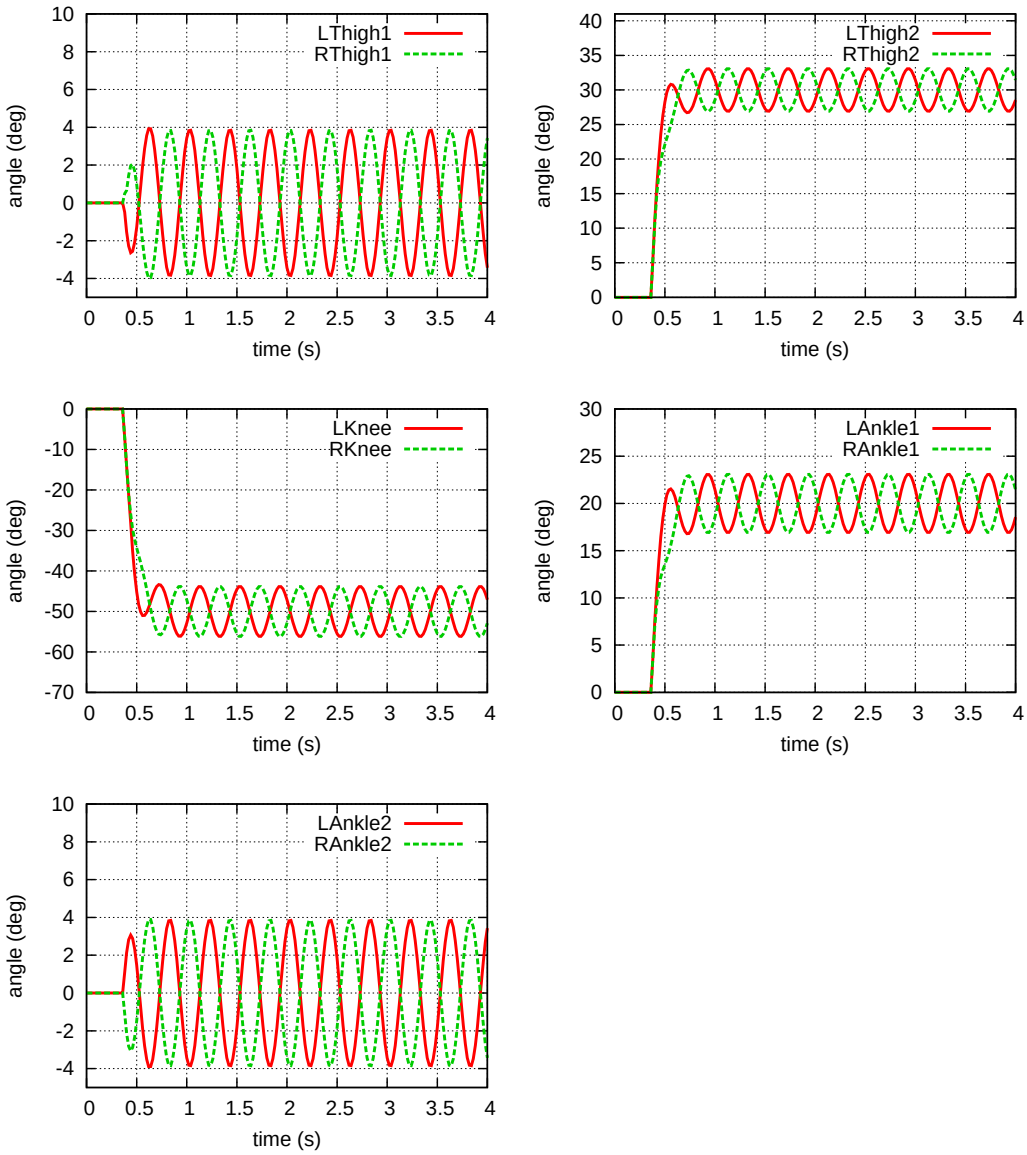

Figure 7.43: Side walking: Joint trajectories. Only the joints that change are shown.

Figure 7.44 shows the evolution of the CoM over time. The Z component of the CoM trajectory shows that the height of the body is kept at a constant height and that there almost no deviation in the forward direction, which is desirable.

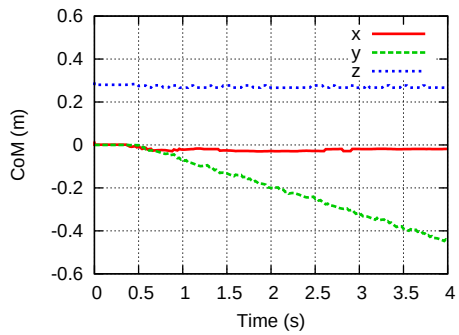

Figure 7.44: Side walking: Evolution of the CoM over time

To take a more closer look of the CoM, Figure 7.53 shows the evolution of the CoM in the XY plane during a complete turn. It also represents the placement of the feet.

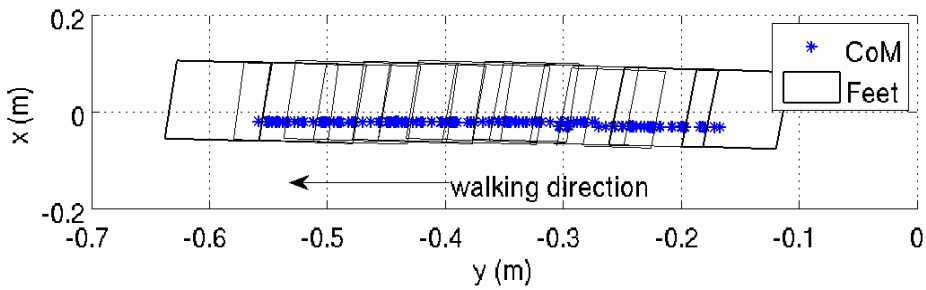

Figure 7.45: Side walking: The CoM and the feet in the XY plane

The trajectory of the CoM describes a line in the Y component. This predictable behavior is essential to ensure good results during a competition. By analyzing the Figure 7.46 it is possible to note that the robot can reach more than 10 centimeters per second. The torso average oscillation (Figure 7.47) keeps very small over time.

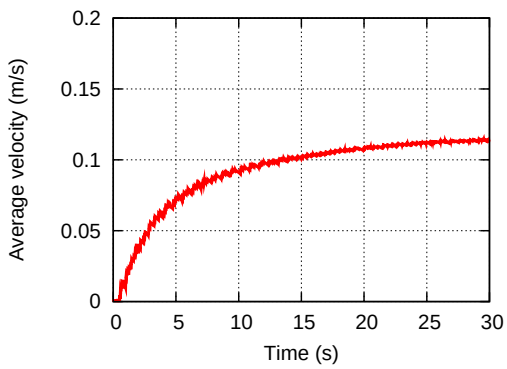

Figure 7.46: Side walking: Average velocity over time

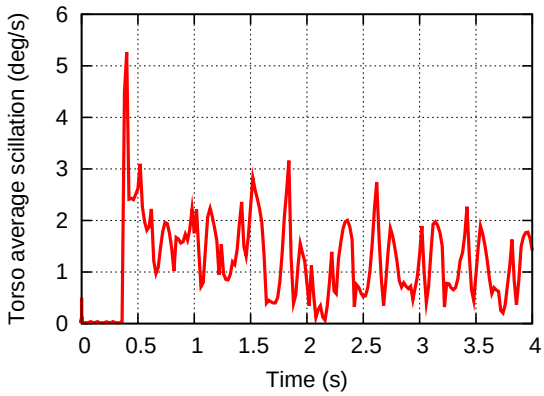

Figure 7.47: Side walking: Torso average oscillation over time

Figure 7.48 shows NAO demonstrating the side walking motion. The natural position of the side walk consists of bent knees and the legs slightly opened. Using the PFS method, there is no need to previously prepare the gait since the robot will adapt to it smoothly, starting from the position where all joints are set to zero.

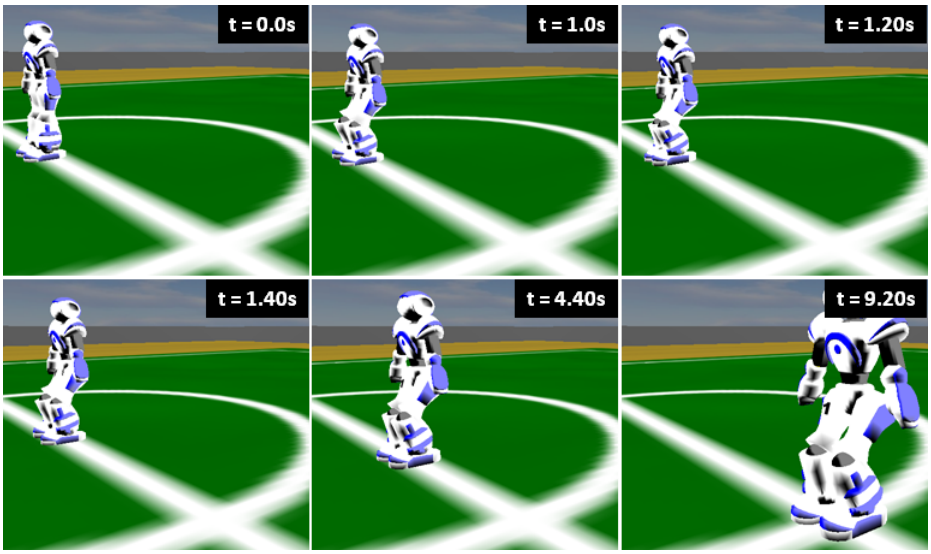

Figure 7.48: Side walking: Screenshot

It can be seen from t = 0.0 to t = 0.94, where the robot performs the first lateral step and finishes with in a double support phase with the knees bent and the legs slightly open. After this time, the robot alternately raises and lands a leg producing a lateral walking motion.

7.5 Turn around the spot

Turning the body consists of changing the orientation of the body around a spot (typically the spot is its current position). The robot can turn to the left or to the right. The side to turn is chosen by means of the target direction, i.e., if the target direction is between -180 and 0 degrees, the turn left movement is the best choice. Otherwise, the turn to right is better. To keep the equilibrium, the biped should turn θ degrees periodically until reach the desired direction (Figure 7.49).

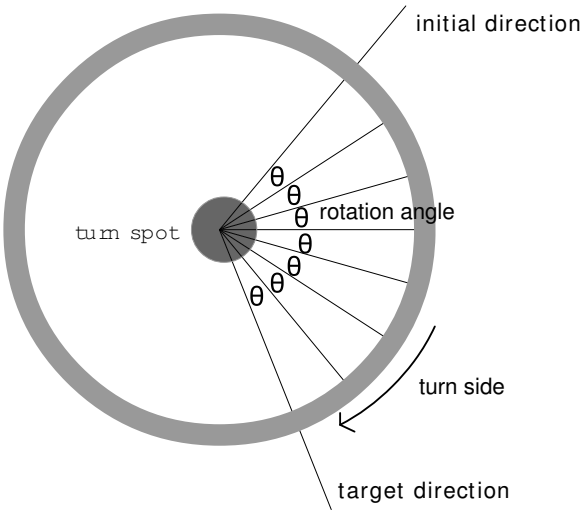

Figure 7.49: Turn motion is performed by rotating the legs by small angle, θ periodically

After each step, the biped should be with the two feet aligned on the ground and rotated θ degrees from its previous position. With respect to the target direction, a small tolerance should be given since several turns of θ degrees may not make the robot face exactly the target direction. The smaller the value of θ, the smaller is the tolerance needed, but slower will be the gait. The turn motion is very similar to the side walk, as illustrated in Figure 7.50, i.e., instead of a roll rotation of the leg, resulting on a lateral translation, d, of the foot. The turn motion performs an additional yaw rotation of θ degrees.

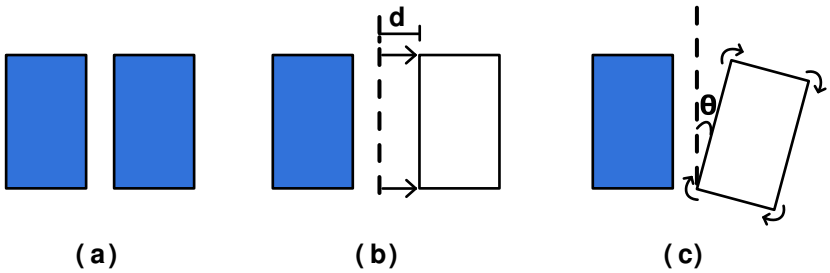

Figure 7.50: Similarities between the side walk and the turn motion: **(a)** Normal position of the feet **(b)** Displacement due to a side walk step **(c)** Rotation of a turn step

7.5.1 Defining the oscillators

The same base constructed for the side walk was used and two new oscillators for the hip were defined. Since there is no swing movement of the legs in the lateral direction during a turn motion, the amplitudes of the joints LThigh1 and RThigh1 are defined with the same values used for the joints LThigh2 and RThigh2. This reduces the original parameter space to 5 parameters. The addition of the amplitude and the offset for the hips results on a total of 7 parameters. The trajectory planning equations are stated as follows:

$$f_{LThigh1}(t) = A_2 \sin\left(\frac{2\pi}{T}t + \frac{\pi}{2}\right) \tag{7.25}$$

$$f_{RThigh1}(t) = -A_2 \sin\left(\frac{2\pi}{T}t + \pi + \frac{\pi}{2}\right) \tag{7.26}$$

$$f_{LThigh2}(t) = A_2 \sin\left(\frac{2\pi}{T}t\right) + \alpha_1 \tag{7.27}$$

$$f_{RThigh2}(t) = -A_2 \sin\left(\frac{2\pi}{T}t + \pi\right) + \alpha_1 \tag{7.28}$$

$$f_{LKnee}(t) = A_3 \sin\left(\frac{2\pi}{T}t\right) + \alpha_2 \tag{7.29}$$

$$f_{RKnee}(t) = -A_3 \sin\left(\frac{2\pi}{T}t + \pi\right) + \alpha_2 \tag{7.30}$$

$$f_{LAnkle1}(t) = A_2 \sin\left(\frac{2\pi}{T}t\right) + \alpha_3 \tag{7.31}$$

$$f_{RAnkle1}(t) = -A_2 \sin\left(\frac{2\pi}{T}t + \pi\right) + \alpha_3 \tag{7.32}$$

$$f_{LAnkle2}(t) = -A_2 \sin\left(\frac{2\pi}{T}t + \frac{\pi}{2}\right) \tag{7.33}$$

$$f_{RAnkle2}(t) = A_2 \sin\left(\frac{2\pi}{T}t + \pi + \frac{\pi}{2}\right) \tag{7.34}$$

$$f_{LHip}(t) = A_4 \sin\left(\frac{2\pi}{T}t + \frac{\pi}{2}\right) + \alpha_4 \tag{7.35}$$

$$f_{RHip}(t) = A_4 \sin\left(\frac{2\pi}{T}t + \frac{\pi}{2}\right) + \alpha_4 \tag{7.36}$$

7.5.2 Manual definition of parameters

Most of the parameters of the side walking oscillators were reused and the remaining parameters were defined manually. Table 7.13 shows the values found for the parameters.

Parameter	**Value**	**Parameter**	**Value**
T	0.4	α_1	30.0
A_2	4.8	α_2	-50.0
A_3	-9.6	α_3	20.0
A_4	-9.6	α_4	-10.0

Table 7.13: Turn around the spot: Values for the manually defined parameters

The trajectories generated during the turn motion are described in Figure 7.51. It should be noticed a smooth sinusoidal pattern on each joint. The robot automatically adapts itself to the inherent pose, i.e., in spite of need to keep the knees bent, the robot does not need to use another gait to prepare the turn when it starts from a completely upright position because the PFS method provides a smooth transition from its original pose.

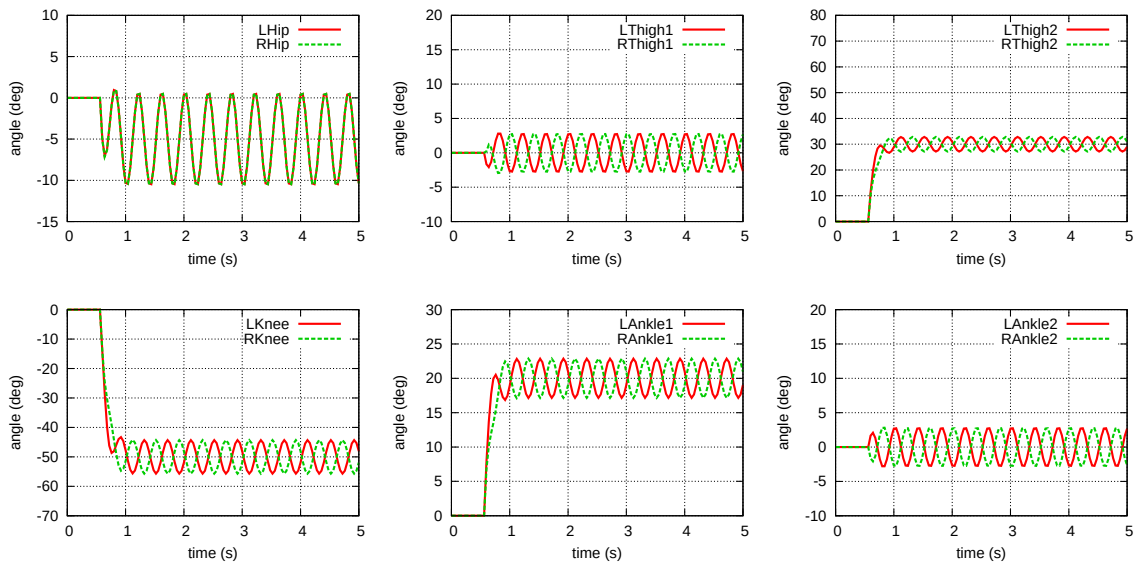

Figure 7.51: Turn around the spot: Joint trajectories. Only the joints that change are shown.

Figure 7.52 shows the evolution of the CoM (on the left) and the body orientation (on the right) over time.

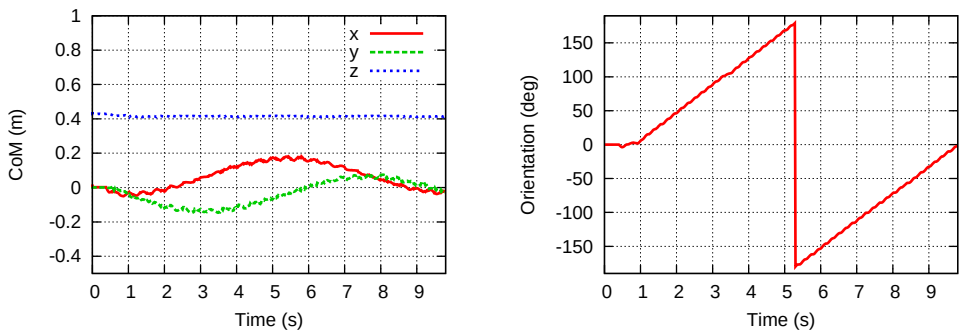

Figure 7.52: Omnidirectional Walking: Evolution of CoM (left) and body orientation (right) over time

The Z component of the CoM trajectory shows that the height of the body is kept at a constant height. Both CoM (X and Y components) and body orientation (from -180 to 180 degrees) show that the biped can perform a complete turn in about 9.8 seconds, which corresponds to an angular velocity of 36.7347. Figure 7.53 shows the evolution of the CoM in the XY plane during a complete turn. It also represents the placement of the feet.

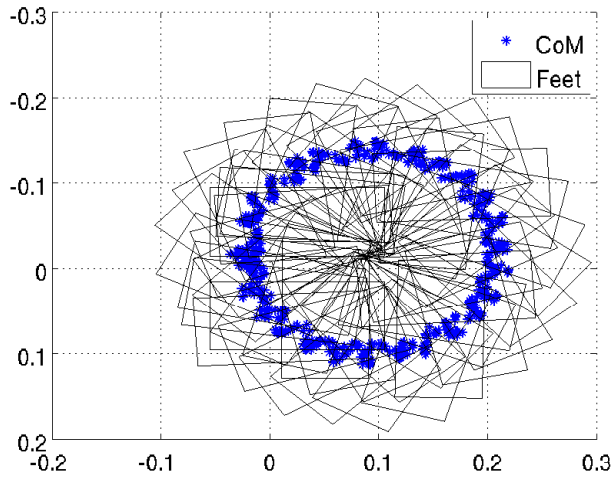

Figure 7.53: Turn around the spot: The CoM and the feet in the XY plane

The trajectory of the CoM describes a regular circle around a spot that is placed just in front of the robot. After a complete turn the biped is back to its initial position. This predictable behavior is essential to ensure good results during a competition. Figure 7.54 shows NAO demonstrating the turn motion.

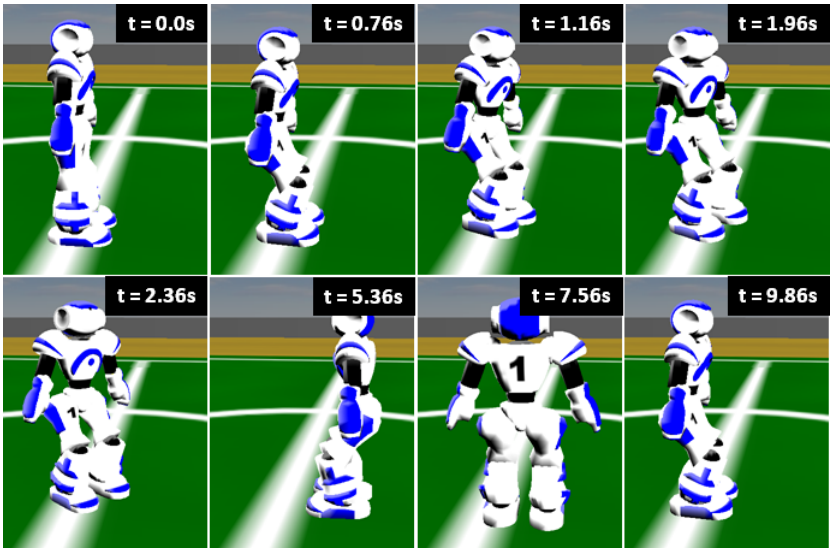

Figure 7.54: Turn around the spot: NAO performing a complete turn

The motion could be described using PFS with almost the same parameters as the side walk. This led to the definition of a multi-purpose set of parameters that might be used in the future to define other similar gaits.

7.6 Kick the ball

To kick a ball is an essential skill for a robot that plays soccer. It is not very easy to control the trajectory of the ball after the kick (aiming at producing an omnidirectional kick) since there is no rough control over the position where the ball will be touched, unless when using inverse kinematics (See Section 2.4.2). Inverse Kinematics allows to control the position of the kicking foot (end effector) by guiding it to wherever point in the ball it should be kicked. This would allow for flexible control over the direction and the speed of the ball. The Inverse Kinematics module is out of the scope of this thesis so it was not possible to apply it in time.

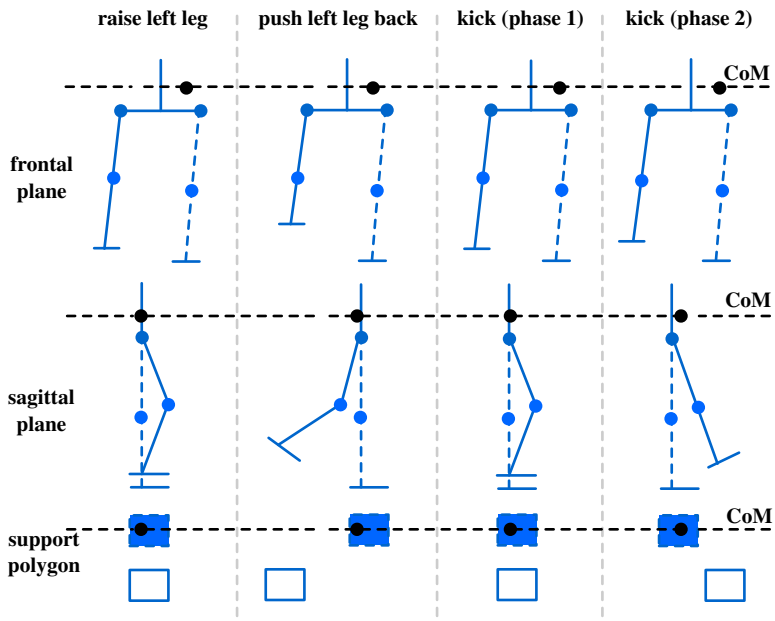

Figure 7.55: Kick the ball. The figure shows a biped kicking in the forward direction with the right leg

The Slot Interpolation method was used to define the kick motion. The parameters were tuned manually in a trial-and-error process. Figure 7.55 shows the structure of the kick motion. Several tests proved that better results can be achieved by dividing the kicking phase into two phases. The first phase consists in place the leg aligned with the other leg (in the sagittal plane) quickly. The second phase consists of straight the leg forward to kick the ball. Two phases are necessary because moving the leg from back to front quickly may produce overshoot in some joint trajectories. Since the kicking foot is very near to the ground when it is aligned with the other leg (See the first phase of the kick in the Figure 7.55), this overshoot can be enough for the foot to scrape on the ground, completely ruining the movement. Due to the shifting phase, the CoM is always inside the support polygon (which is formed by the non-kicking foot), with the assumption that the movement is not too abrupt to generate instabilities.

Figure 7.56 show the evolution of the joint trajectories over time. The produced trajectories are, in general, very smooth. The main problems arise when the robot quickly moves the leg in the forward direction. This could not be avoided since the quick motion is needed so the ball can reach a more far position.

For the joints that move in the sagittal plane (LThigh2, RThigh2, LKnee, RKnee, LAnkle1 and RAnkle1) the trajectories are not so smooth due to the fast transition during the kick phases.

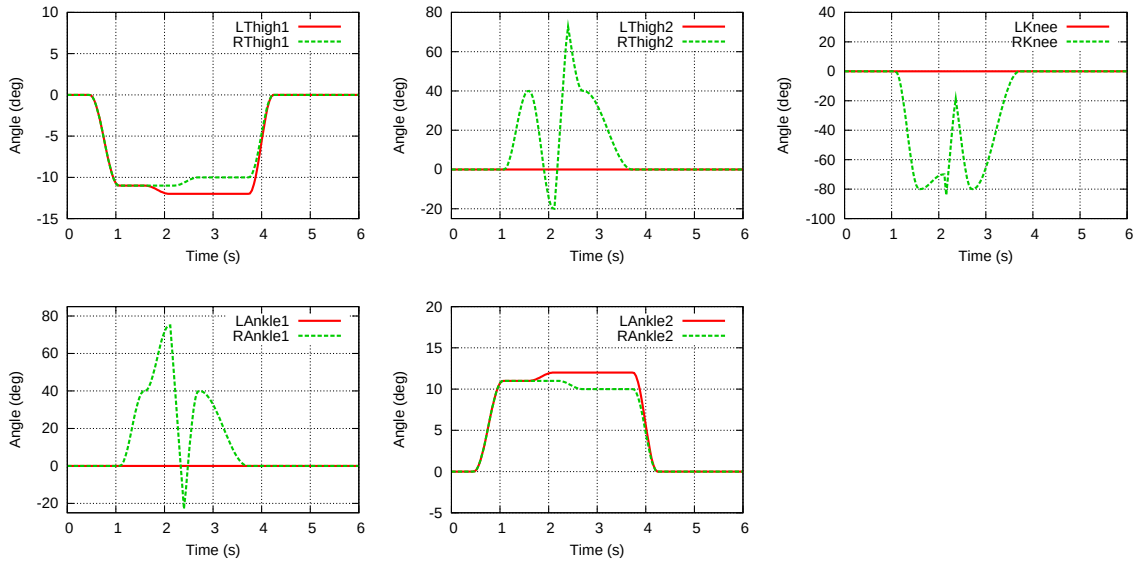

Figure 7.56: Kick the ball: Joint trajectories. Only the joints that change are shown.

The trajectory of the CoM in the XY plane (Figure 7.57) shows that the CoM is always kept inside the support foot.

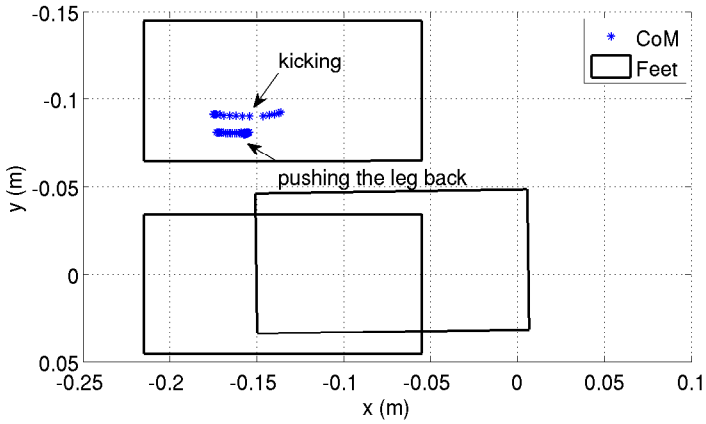

Figure 7.57: Kick the ball: The CoM and the feet in the XY plane

The graphic shows the CoM trajectory and the placement of the feet during the phases represented earlier in Figure 7.55. There is a sligtly change of the CoM in the lateral direction but since it is kept inside the support foot this does not affect the quality of the kick.

An important measure to evaluate the quality of the kick motion is the trajectory of the ball after the kick has effect. Figure 7.58 shows the position of the ball over time.

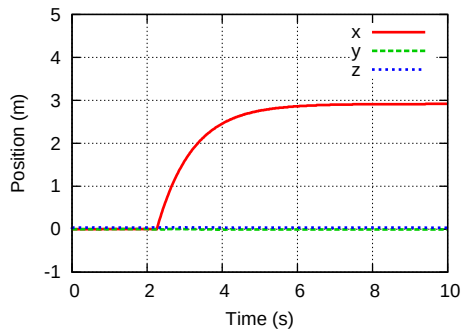

Figure 7.58: Kick the ball: Position of the ball over time

The ball keeps a constant height and follows a completely linear trajectory in the forward direction. This shows that the kick result in a very precise trajectory of the ball. The ball travels about 3 meters away of its original position. Additionally, the torso oscillates a little bit more during the kick phases but in general it keeps a low average oscillation value, as represented in Figure 7.59.

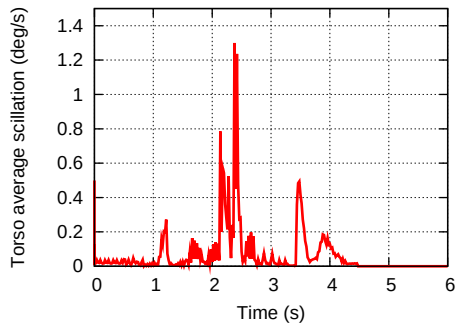

Figure 7.59: Kick the ball: Torso average oscillation over time

Figure 7.60 shows NAO demonstrating the kick motion. It is possible to note the several phases of the kick motion. The robot starts the motion at t=1.12 seconds and backs to the initial position at t=4.32 seconds, completely stable.

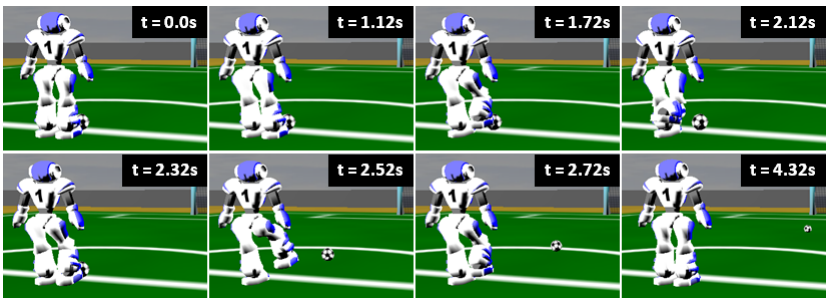

Figure 7.60: Kick the ball: Screenshot

Moreover, at t=4.32 seconds, the ball movement is almost over. The figure also shows that the trajectory of the ball is completely in the forward direction and this always happens. If we are able to ensure this deterministic behavior of the ball after the kick, the success is more guaranteed.

Taking into consideration the strength characteristics of the simulated NAO model and the stability of the motion, this kick produces very good results. It is possible to produce a stronger kick by balance the robot forward while pushing the leg back and balance it backwards again while kicking. The ball now travels more than 5 meters but not completely forward. Figures 7.61 and 7.62 shows the results for the position of the ball and torso average oscillation during the called Super Kick motion.

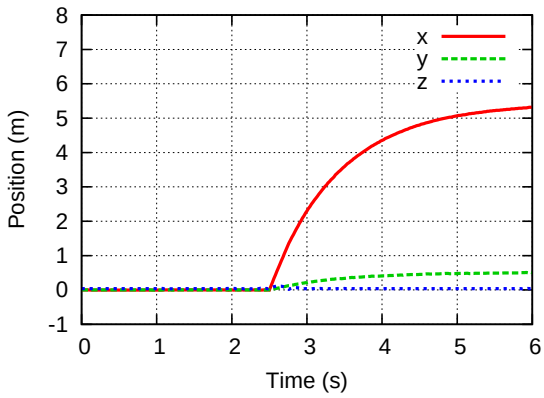

Figure 7.61: Super Kick: Position of the ball over time

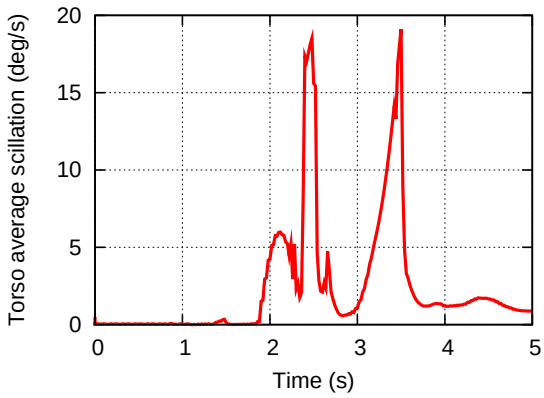

Figure 7.62: Super Kick: Torso average oscillation over time

There are two main drawbacks of this gait. The trajectory of the ball is not predictable and the robot falls on the ground. In the Figure 7.61 it is possible to note a considerable deviation of the ball from the target trajectory (forward direction). Figure 7.62 shows the torso average oscillation reaching almost 20 degrees per second (At t=4 seconds, the biped is already fallen, resulting on a low torso oscillation). For simulated competitions, this may be useful when we are sure that the robot will score using the Super Kick. However, for real robots, this is not acceptable since this will definitely damage the robot.

7.7 Catch the ball

For a robot to catch a ball, acting as a goal keeper, a possible solution is to fall on purpose to the side where it predicts the ball will be. The robot should be fast enough to be effective on catching the ball. The gait was developed using the Sine Interpolation method and consists of one single slot shifting the robot to the side quickly by adjusting the joints LThigh1, RThigh1, LAnkle2 and RAnkle2. In order to cover a larger part of the goal, the robot also raises the arms (Figure 7.63).

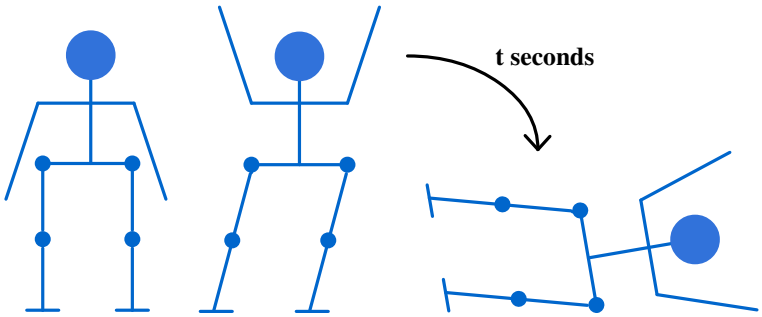

Figure 7.63: Catch the ball. The shifting movement results on a fall with a duration of t seconds (represented by the arrow).

The duration of the fall, t, should be as small as possible so the catch can be effective. Figure 7.64 shows the trajectories of the joints during the simulation of the catch motion.

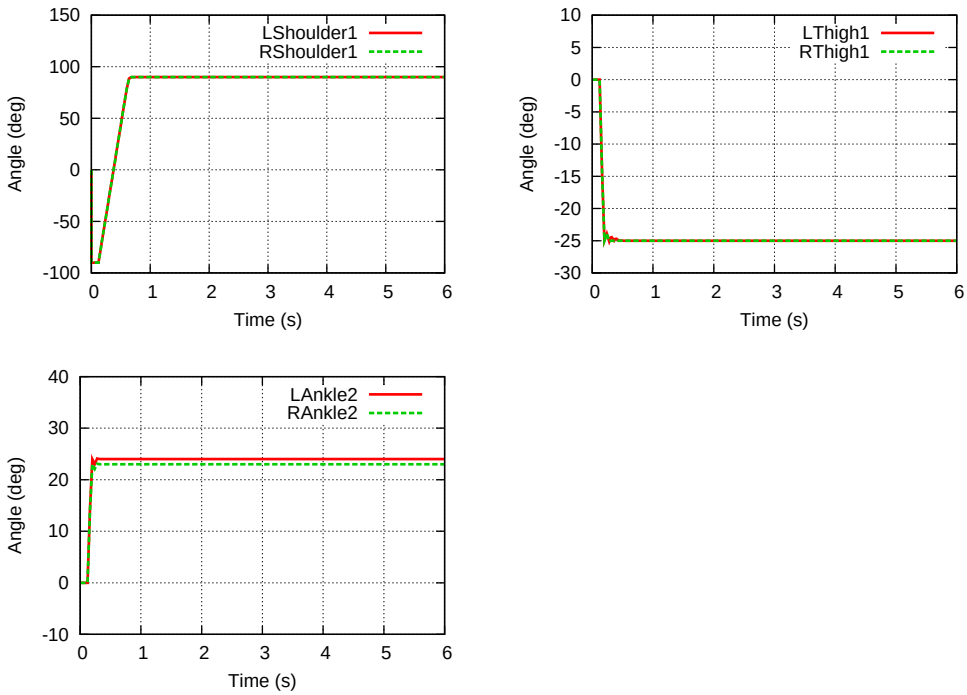

Figure 7.64: Catch the ball: Joint trajectories. Only the joints that change are shown.

Due to the fast transition of the leg joints, the controller was not able to avoid the overshoot, even after setting the PID parameters. The arms were able to follow the correct trajectory using a simple PD controller.

Figure 7.65 shows the evolution of the CoM over time. The CoM trajectory is useful to show the displacement in the lateral direction and that there is no deviation in the forward direction.

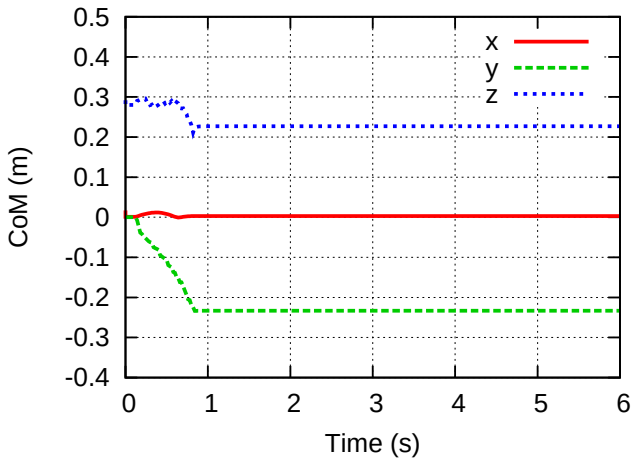

Figure 7.65: Catch the ball: Evolution of CoM over time

By analyzing the Y and Z components of the CoM, we see that the robot finishes the gait in less than a second. Figure 7.66 shows NAO falling on purpose to the left to demonstrate the Catch motion.

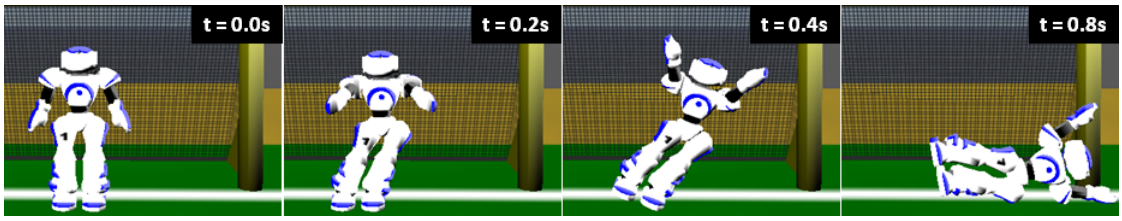

Figure 7.66: Catch the ball: Screenshot

It should be noticed here that the robot starts at $t = 0.92s$ and finishes at $t = 1.52s$. Hence, the duration of the catch motion is less than a second (0.6s). In the final position the robot is able to cover a half of the goal width, which is good if the prediction about the ball position is accurate.

7.8 Get up from the ground

A humanoid will always be subject to complex situations that will make it fall sometimes. This situations are related not only with the quality of the locomotion but also with the interaction with obstacles present in the environment. The robot may also fall on purpose (e.g. to catch the ball). This led the researchers to develop new gaits that would make the robot get up autonomously after the fall. Two situations are common to describe a fall:

- The humanoid falls on its face
- The humanoid falls on its back

Before starting with the trajectory planning, the robot needs to have the necessary high-level knowledge to recognize the both situations above. The method used consists of determining the normal to the field with respect to the vision coordinate system (head referential). Let $\hat{N} = (N_x, N_y, N_z)$ be the normal vector in vision coordinates (Figure 7.67).

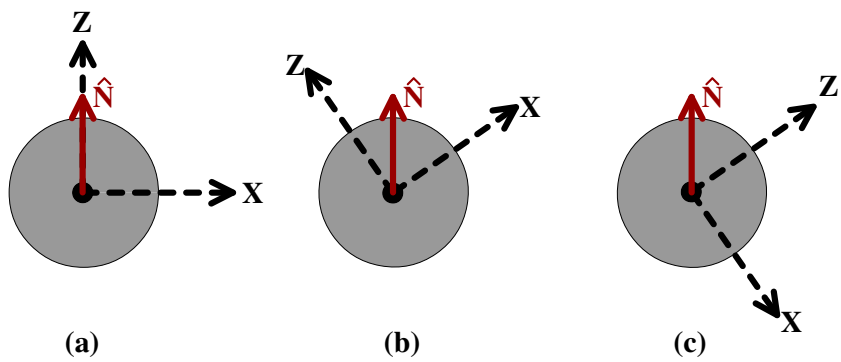

Figure 7.67: Representation of the normal to the field with respect to the coordinate system of the head (Figure shows a side view of the head). **(a)** $N_x = 0$: Upright position **(b)** $N_x > 0$: Inclined backward **(c)** $N_x < 0$: Inclined forward

Figure 7.67 shows three possible situations. Using the normal to the field and the height of the robot it is possible to know if the robot actually fell, as represented by the following decision rule:

$$\begin{cases} Height < 0.15 \text{ and } N_x < 0 \Rightarrow \text{robot fell on its face} \\ Height < 0.15 \text{ and } N_x > 0 \Rightarrow \text{robot fell on its back} \end{cases} \tag{7.37}$$

Get up after falling forward

To get up after falling forward, the robot follows a set of key poses to get back in the upright position. The Sine Interpolation is used in this case and each key pose of the robot is described by one slot. Figure 7.68 shows the joint trajectories over time. The robot moves several joints to get up from the ground. It is possible no note smooth trajectories both for the arms and for the legs. All the joints are controlled by a PD controller to help on avoiding the overshoot reactions. The figure shows the robot falling on purpose (before t = 2 seconds) and starting getting up (after t = 2 seconds).

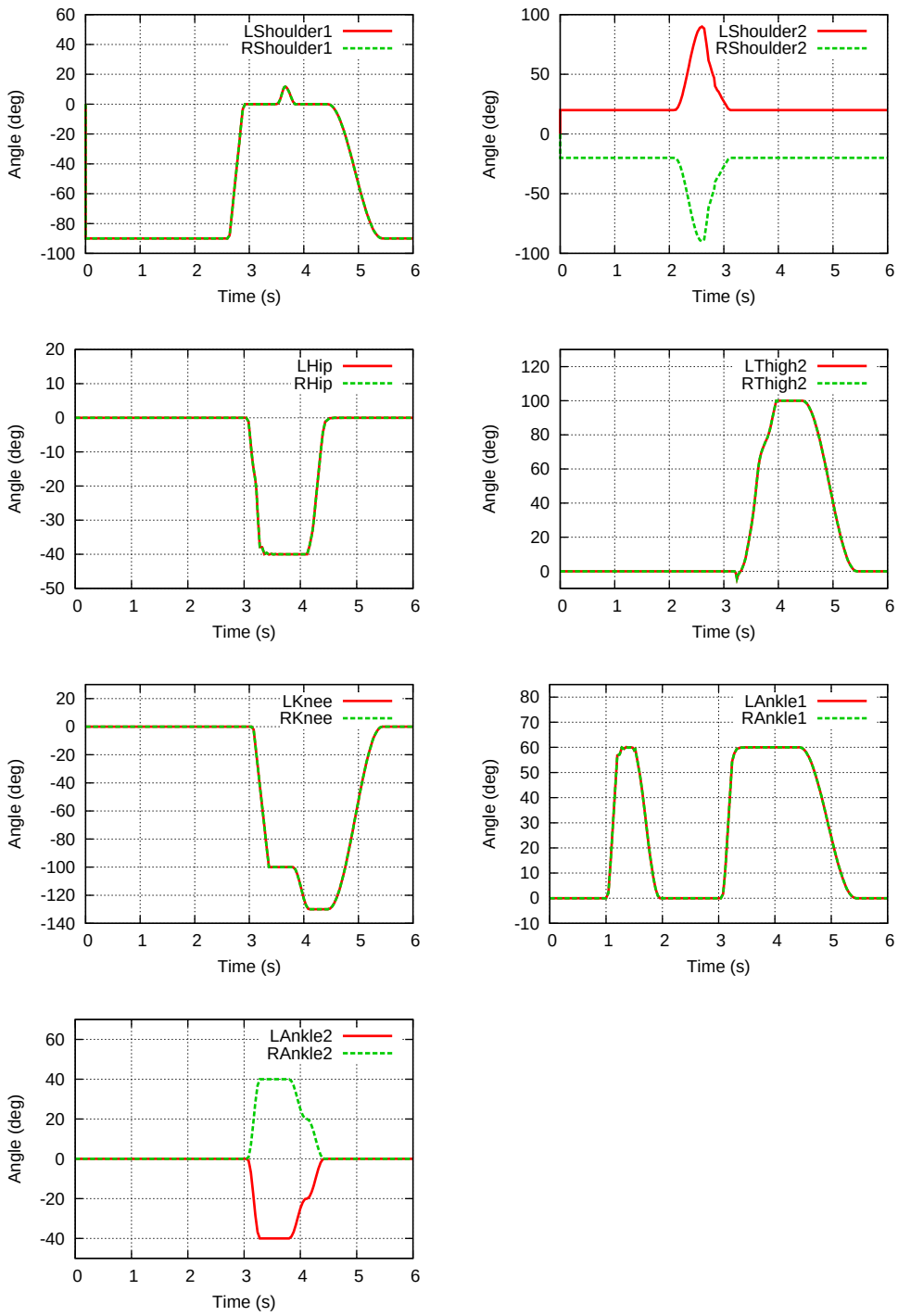

Figure 7.68: Get up after falling forward: Joint trajectories. Only the joints that change are shown.

Once again, monitoring the CoM is useful to show the quality of the gait. Figure 7.69 shows the evolution of the CoM over time.

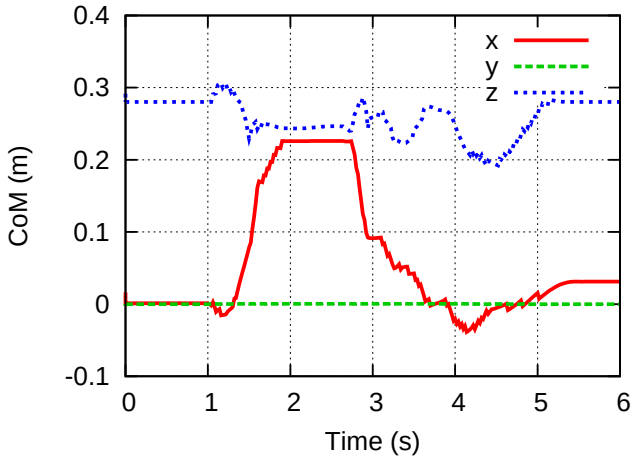

Figure 7.69: Get up after falling forward: Evolution of the CoM over time

In about 3.2 seconds, the robot is back to the upright position, with a small displacement in the forward direction (See the X component of CoM). The Y component of the CoM shows that there is no deviation to the side while getting up. This means that the robot falls and gets back in the upright position, keeping its previous orientation, which is typically the desired case.

Another useful measure is the torso oscillation. Figure 7.70 shows a peak that is reached while the robot falls on purpose to try to get up next. It can show how the torso oscillates during the execution of the gait.

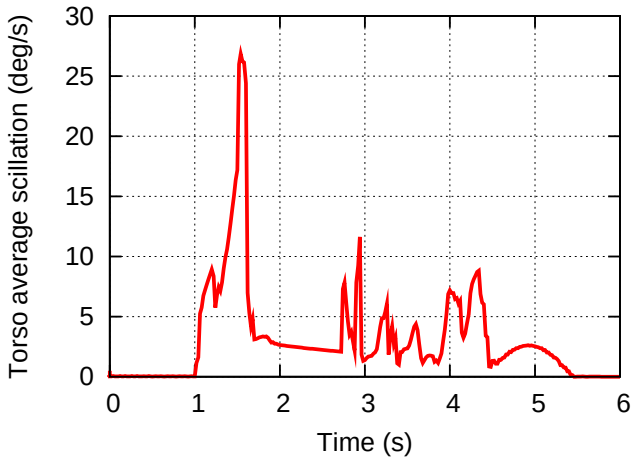

Figure 7.70: Get up after falling forward: Torso average oscillation over time

The robot gets up keeping a small torso oscillation and then it becomes zero when the gait is finished. During the execution of the gait, a small oscillation is present.

A total of 8 poses were needed to get the robot upright. Figure 7.71 shows NAO getting up after falling on face. The eight poses are represented.

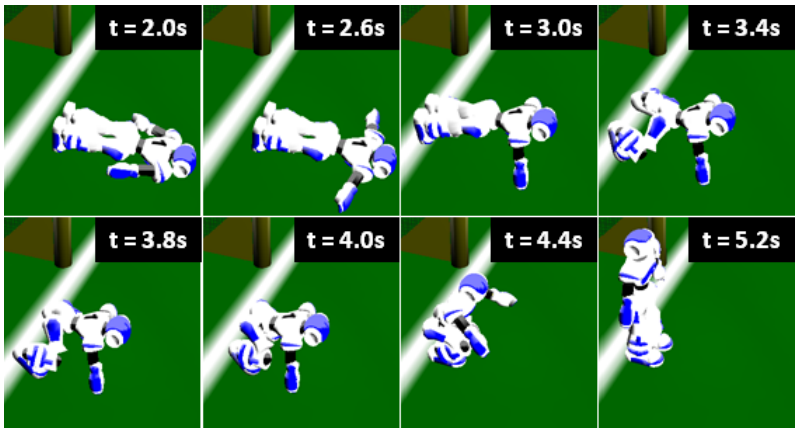

Figure 7.71: Get up after falling forward: Screenshot

The robot takes about 3.2 seconds to get up (from t =2.0s to t=5.2s) which is a very good result. This gait proved to be very successful, i.e., after a fall on its face, the robot always gets up with success. This gait proved to be very effective during the competition in Suzhou (RoboCup 2008).

Getting up after falling backwards

Sometimes the robot will also falls on its back. In order for a biped to get up after falling backwards, two different approaches are commonly used:

- Turn the body on the ground and then use the gait described in the previous section
- Use a new gait to get up directly after falling backwards

Both approaches have advantages and disadvantages. The first approach is easier since the hard work is already done, i.e., a gait to turn the robot is easier to develop than a gait to get up directly. However, if the latter is correctly developed, the robot can get up faster than using the first approach. For this thesis the first approach was used at the beginning, but soon a gait to get up directly was also developed and it is currently being used. The Sine Interpolation was once again used and each key pose of the robot is described by one slot.

Figure 7.72 shows the joint trajectories over time. Once again, the trajectories are the same for the left and the right joints (arms and legs). The robot moves several joints to get up. Both arms and legs use a PD controller aiming at avoiding the overshoot reactions. It is possible to note smooth trajectories, meaning that the trajectory planning and the low-level control are completely synchronized. The figure shows the robot falling on purpose (before t = 1 second) and starting getting up again (after t = 1 second).

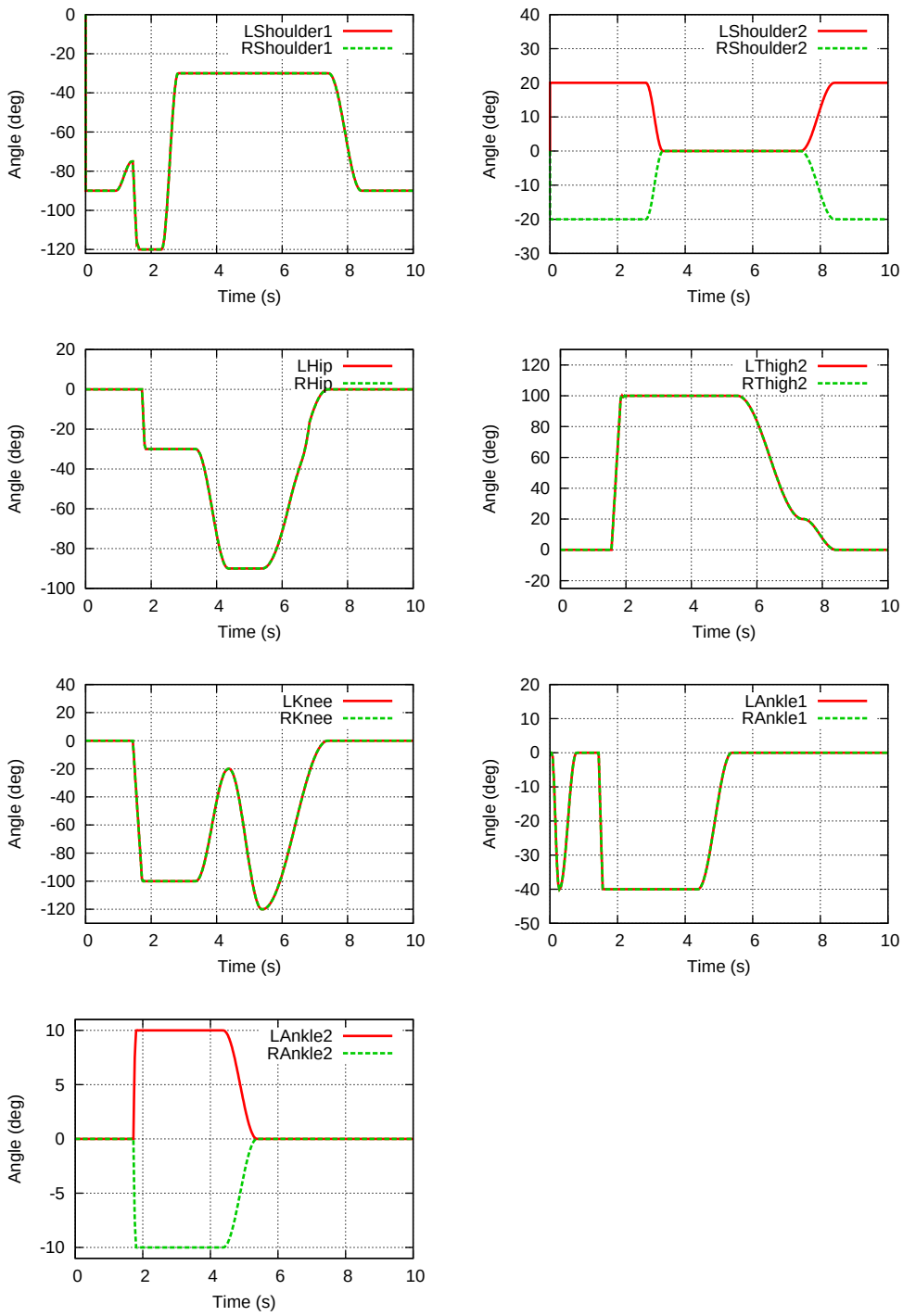

Figure 7.72: Get up after falling backwards: Joint trajectories. Only the joints that change are shown.

CoM trajectory is very useful to monitor the quality of the developed gait. Figure 7.73 shows the evolution of the CoM over time. The figure shows clearly that the movement shows at approximately 8.4 seconds. Since the robot starts the movement at t = 1 seconds. The movement takes about 7.4 seconds to complete. The CoM trajectory shows no deviation in the lateral direction (Y component).

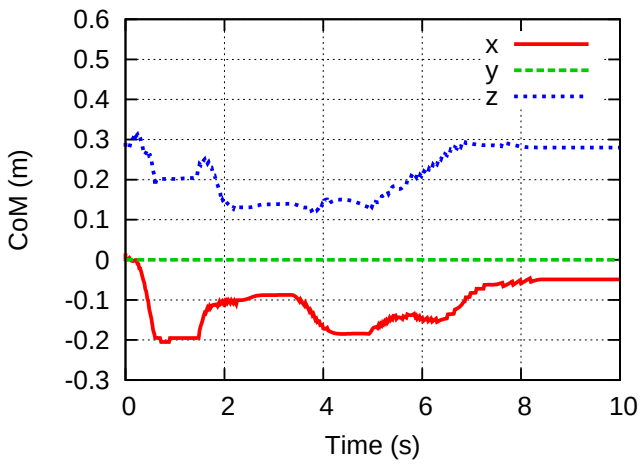

Figure 7.73: Get up after falling backwards: Evolution of the CoM over time.

Figure 7.74 shows the torso average oscillation during the execution of the gait. It is clear that this gait leads to more oscillations on the torso than the gait presented in the previous section. This happens because getting up NAO after it falls backwards is a complex task that involves unusual movements, due to its anthropomorphic characteristics.

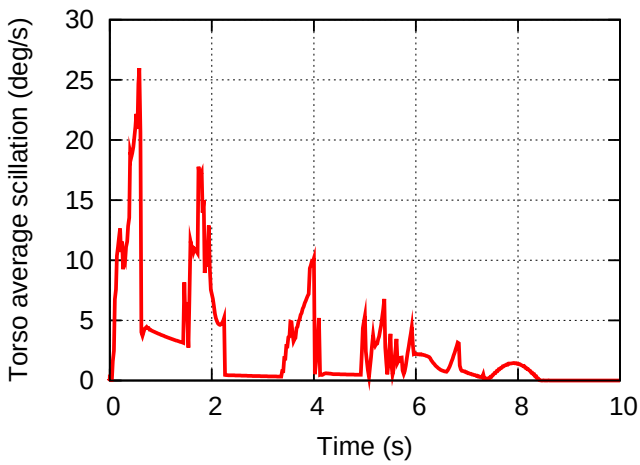

Figure 7.74: Get up after falling backwards: Torso average oscillation over time.

A total of ten key poses are followed by the robot so that it can get back in the upright position. Figure 7.75 shows NAO getting up from the ground after falling backwards. It is possible to see some unusual movevents, as illustrated in the figure at t = 4.4 seconds.

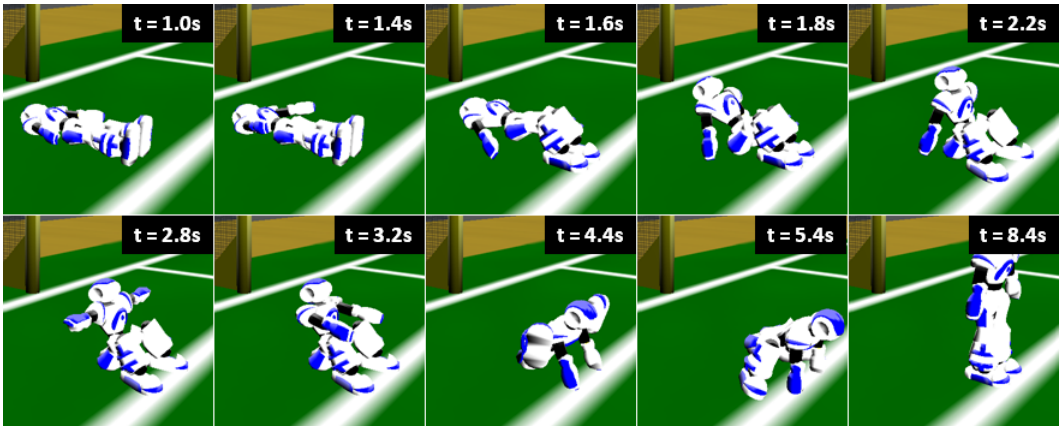

Figure 7.75: Get up after falling backwards: Screenshot

NAO takes about 7.4 seconds to complete its task of getting up from the ground. This gait proved to be very effective during the competition in Suzhou (RoboCup 2008).

7.9 Analysis of results

Some of the results presented in the previous sections have no meaning if not compared with the state of the art teams. This is the case of the following measures:

- Average linear velocity for forward walking
- Average linear velocity for side walking
- Average angular velocity for turning
- Distance achieved by the ball when kicked
- Time to fall to catch the ball or to get up from the ground

Section 2.6 presented the three best teams of RoboCup 3D simulation league in 2007 and 2008. This section aims to compare the FCPortugal3D with those teams. Table 7.14 shows the average linear velocity for each forward walking behavior.

Behavior	**Definition**	**Average linear velocity (m/s)**
4-phase forward walking	Manual	0.1600
	HC	0.2281
	GA	0.4468
Forward walking based on PFS	GA	0.5055
ODW-CPG: forward walking mode	Manual	0.1008

Table 7.14: Average linear velocity for the different forward walking behaviors

The presented results correspond to the average linear velocity after reaching a stable value. It is possible to see that the forward walking based on PFS achieved the best result so it will be used for the comparison with the forward walking of the other teams. The side walking has also two behaviors. One is based on PFS and the other is one of the modes of the Omnidirectional Walking CPG. Table 7.15 shows the stable values for the average linear velocity for the different side walking behaviors.

Behavior	**Definition**	**Average linear velocity (deg/s)**
Side walking based on PFS	Manual	0.1140
ODW-CPG: side walking mode	Manual	0.0345

Table 7.15: Average linear velocity for the different side walking behaviors

Once again, the PFS method proved to achieve the best result. Similarly to the side walking, the turning motion has two different behaviors. One is defined using the PFS method and the other one is part of the Omnidirectional Walking CPG. Table 7.16 shows the stable value of the average angular velocity for the turning behaviors.

Behavior	**Definition**	**Average angular velocity (deg/s)**
Turning based on PFS	Manual	36.7347
ODW-CPG: turning mode	Manual	12.0000

Table 7.16: Average angular velocity for the different turning behaviors

The turning based on PFS has the best result so it will used to compare the turning motion with the other teams. Another measure that has no meaning if not compared with the state of the art teams, is the distance covered by the ball when the robot executes the kick behavior. In the scope of this thesis, two different kick behaviors were developed. The first is very stable and the robot keeps upright after kicking. The second is not so stable, the robot falls for sure, but the ball covers a greater distance. Table 7.17 shows the distance of the ball when it stops moving after the execution of the kick.

Behavior	**Definition**	**Ball distance (m)**
Stable kick	Manual	2.9204
Super kick	Manual	5.4687

Table 7.17: Distances achieved by the ball when kicked

Finally, it will be compared the time to fall of the catch motion and also the time to get up from the ground. Table 7.18 shows the catch falling time and the time to get up from the ground (after falling forward and after falling backward).

Behavior	**Definition**	**Time (s)**
Catch falling	Manual	0.6000
Get up after falling forward	Manual	3.2000
Get up after falling backward	Manual	7.4000

Table 7.18: Times for catching and getting up

Comparing the results

Once the synthesis of results is complete, the best results of each behavior are chosen for the comparison with the other teams. Table 7.19 now presents the comparison of the best results with the state of the art teams.

Behavior	**Measure**	**Unit**	**Team**			
			FCPortugal	**SEU**	**WrightEagle**	**Bats**
Forward walking	Average linear velocity	m/s	0.5055	1.2000	0.6700	0.4300
Side walking	Average linear velocity	m/s	0.1140	0.6000	0.5900	0.0600
Turn	Average angular velocity	deg/s	36.7347	25.0000	100.0000	32.1000
Kick	Ball distance	m	2.9204	3.0000	N/A	2.1000
Super kick	Ball distance	m	5.4687	6.0000	N/A	N/A
Catch	Time to fall	s	0.6000	1.0000	N/A	1.4000
Get up	Time to get up	s	3.2000	2.0000	2.4000	5.2000

Table 7.19: Comparing the results with the other teams

The results from the SEU-RedSun team were provided by the team itself (thanks to them). Some of the WrightEagle results were obtained from the *.perform files provided with the binary. The results for getting up were obtained from the log files of the RoboCup 3D 2008 competition[4]. As stated by the team, the Kick motion were unstable so it was not used in competition and the Catch motion was not defined in time, thus these values are not available. The information of Little Green Bats were retrieved from the log files.

[4]http://www.robocup-cn.org

As it is possible to see in Table 7.19, the achieved results are reasonable:

- Forward walking: Better than Bats but worst than the other teams.
- Side walking: Better than Bats but worst than the other teams.
- Turning: Better than SEU and Bats but far from reaching the result of WrightEagle.
- Distance covered by the ball when kicked: Near to SEU and better than Bats. For WrightEagle this information is not available.
- Catch falling time: Better than SEU and Bats. For WrightEagle there this information is not available.
- Time to get up: Near to all the other teams when getting up after falling forward.

A very important thing to consider is that all the behaviors developed in the scope of this thesis do not violate the joint restrictions. This was not imposed by the simulator during the competition so some teams did not care about these restrictions. Violating the restrictions may result on better results but, for sure, the same behaviors cannot be applied to a real robot. In this way, FCPortugal decided to generate values inside the corresponding ranges.

7.10 Summary

This chapter presented the several behaviors developed in the scope of this thesis. These behaviors were developed using different methods, which were explained in the Chapter 6. Some behaviors are more sensitive to the non-deterministic behavior of the simulation, which may cause them to fail their goal.

After reading this chapter, it is possible to conclude that Sine Interpolation method is adequate for non-periodic behaviors (e.g. kicking, getting up, and catching the ball). This method is very simple to use and provides a very simple configuration language. Additionally, the possibility to configure the PID controller gains was very useful to correct some trajectories.

The Partial Fourier Series (PFS) method was an important method for the generation of periodic gaits, such as walking and turning. It was possible to define a common base for turning and side walking, which differ by two oscillators and two amplitudes. This multi-purpose base may be extended in the future two produce more movements.

The Omnidirectional Walking CPG is less sensitive to disturbances but it takes longer to perform the movements because it is based on static stability, which is achieved by moving the CoM into the support foot to keep thus generating slower movements.

Optimization was used to optimize the Four-phase walking and the forward walking based on PFS. Hill Climbing proved that it is capable to improve a solution in a short period of time. On the other hand, Genetic Algorithms proved to be an essential element for automatic generation of gaits and, despite taking a long time to complete, they can achieve very good results.

Chapter 8

Conclusion and future work

8.1 Conclusion

The main focus of this thesis was on the development trajectory planning methods for biped locomotion and its application to a simulated humanoid platform. The step-based behavior was the method used by the FCPortugal3D team before this thesis and consists of sequentially generating a step function whose amplitude is the desired target angle for the corresponding joint. This method has several drawbacks since there is no flexible control over the step response. To overcome such drawbacks, a simple trajectory planning method based on the interpolation of a sine function was developed. This method aims at interpolating a sine function from the current angle to the desired angle during a configurable amount of time. Besides the duration of the trajectory, it is possible to control the initial and final angular velocities which make this model more flexible. Moreover, the generated trajectories are very smooth. The third method presented generates trajectories based on Partial Fourier Series (PFS). A PFS is a continuous function, consisting of a sum of sines and cosines allowing for approximating a huge number of continuous functions. This method provides full control over the trajectories and it is the most appropriate to generate periodic behaviors such as walking and turning. Finally, a method developed by Sven Benhke [43] was studied and implemented in the scope of this thesis. The method consists of a Central Pattern Generator (CPG) capable of generating an omnidirectional walking behavior online. In spite of being slower due to the adjustment of the Center of Mass (CoM) to keep over the support foot, it is capable of changing the walking direction online, which is extremely useful in dynamic environments, which is the case of RoboCup soccer. Besides the trajectory planning methods, low-level control is also a challenge in biped locomotion control. PID controllers were very useful to generate smooth trajectories. These controllers were most of the times capable to completely eliminate the overshoot reactions of the joints, as well as the steady-state error.

A motion description language based on XML was developed to give support to the trajectory planning methods based on Sine Interpolation and Partial Fourier Series. This language offers many advantages which include the possibility of generating movements using a simple configuration language, changing the behaviors without recompiling all the agent's code and using user-friendly names instead of complex identifiers. Additionally, an arithmetic expression parser was developed to use arithmetic expressions instead of real numbers. These expressions may include variable that may refer to the name of the sensors aiming at integrating the feedback sensory information easily.

By integrating the MDL, the trajectory planning methods and the low-level controllers, it was possible to define several behaviors for a simulated model of the humanoid NAO. The four-phase forward walking were planned using the Sine Interpolation and consists of four stages (raise right leg, land right leg, raise left leg and land left leg). The CoM is not kept inside the support polygon so it is based on small and low steps to avoid falling. These low steps make this behavior very sensitive to the non-determinism of the simulation environment, since any small disturbance may modify the behavior results. The four-phase walking was optimized using Hill Climbing and Genetic Algorithms (GA). GA proved to be better on achieving good results though it is slower. A different forward walking style based on PFS was completely generated using GA. This walking style is faster and it is also stable and has the great advantage of being less sensitive to disturbances. The Omnidirectional Walking CPG proved to be capable of generating forward walking, side walking and turning motions using the same CPG. This CPG generates slower movements due to the positioning of the CoM but has the advantage of being capable of changing its trajectory very quickly and easily. Using the PFS method, two behaviors for side walking and turning were developed. These behaviors use the same base of parameters with the difference that the turning behavior uses the hip joints to rotate the legs. This proved the similarities between the structures of both behaviors. Finally, using the Sine Interpolation method, a set of useful behaviors were also developed, which consist of kicking the ball, catching the ball and getting up from the ground after falling forward and after falling backwards.

8.2 Future Work

Despite of the work in this thesis covers a great part of the biped locomotion control, several improvements are possible and needed. This thesis is mainly based on the trajectory of CoM to monitor the quality of the gait. However, the calculation and monitoring of the ZMP trajectory is essential for achieving dynamic stability. Additionally, the use of Inverse Kinematics to compute the trajectories of end effectors instead of controlling the joints directly is very useful to have more flexibility among the generation of behaviors. Moreover, motion capturing, which consists of monitoring the human behaviors, provides a great way to define the humanoid behaviors, due to the anthropomorphic characteristics between the both.

A Motion Description Language was developed in the scope of this thesis. Although this language allows for the definition of the movements at a joint level, it would be better to provide support for high-level behaviors. The development of a generic language capable of integrating reflexive, reactive and deliberative behaviors would provide an easy way to organize the different skills so that the robot would be capable of deciding what to do, when to do and how to do its actions.

As future work, it would be better to invest in the use of biological inspired optimization algorithms such as Genetic Algorithms and machine learning methods such as Reinforcement Learning. These methods provide great advantages for the automatic generation of behaviors which reduce, and possibly eliminate, the human intervention during the optimization or learning process.

Appendix A

Simspark installation guide

This section pretends to give a brief explanation of how to install the simulator (currently on version 0.6) from the source code. The package names and commands are related to Ubuntu Hardy Linux but similar packages and commands might be found on other distributions.

Required packages

Some development libraries are required to perform a successful installation. The package names for Ubuntu are listed below:

- build-essential, libdevil-dev, libmng-dev, libtiff4-dev, libjpeg-dev, libpng12-dev, libsdl-dev, libfreetype6-dev, freeglut3, libslang2-dev and libboost-thread-dev.

These packages can be installed through the use of Synaptics front-end or using the following command on the terminal:

```
$ sudo apt-get install <package-name>
```

Installing Ruby

```
$ wget ftp://ftp.ruby-lang.org/pub/ruby/1.8/ruby-1.8.7-p72.tar.gz
$ tar xvfz ruby-1.8.7-p72.tar.gz
$ cd ruby-1.8.7-p72
$ ./configure --enable-shared
$ make
$ sudo make install
```

Installing ODE

```
$ wget http://downloads.sourceforge.net/opende/ode-0.10.1.tar.gz
$ tar xvfz ode-0.10.1.tar.gz
$ cd ode-0.10.1
$ ./configure --enable-double-precision
$ make
$ sudo make install
```

Installing the Simulator

```
$ wget http://downloads.sourceforge.net/sserver/rcssserver3d-0.6.tar.gz
$ tar xvfz rcssserver3d-0.6.tar.gz
$ cd rcssserver3d-0.6
$ ./configure
$ make
$ sudo make install
```

Running the simulator and the test agent

```
$ simspark &
$ agentspark &
```

Appendix B

The AgentBody class

The body of a humanoid is composed by a set of body parts, joints and perceptors. It would be useful the possibility to group all these entities and refer to them as, generically, body objects, by grouping its main characteristics in a single main class and creating inheritance relations to define the specific characteristics for each one. The BodyObject class was developed with that purpose, as illustrated on Figure B.1. The BodyObject class describes a generic object of the body. This object will have an identification number, a user-friendly name (e.g. LeftFoot) to help on debugging, and also a position since an object will be spacially located somewhere.

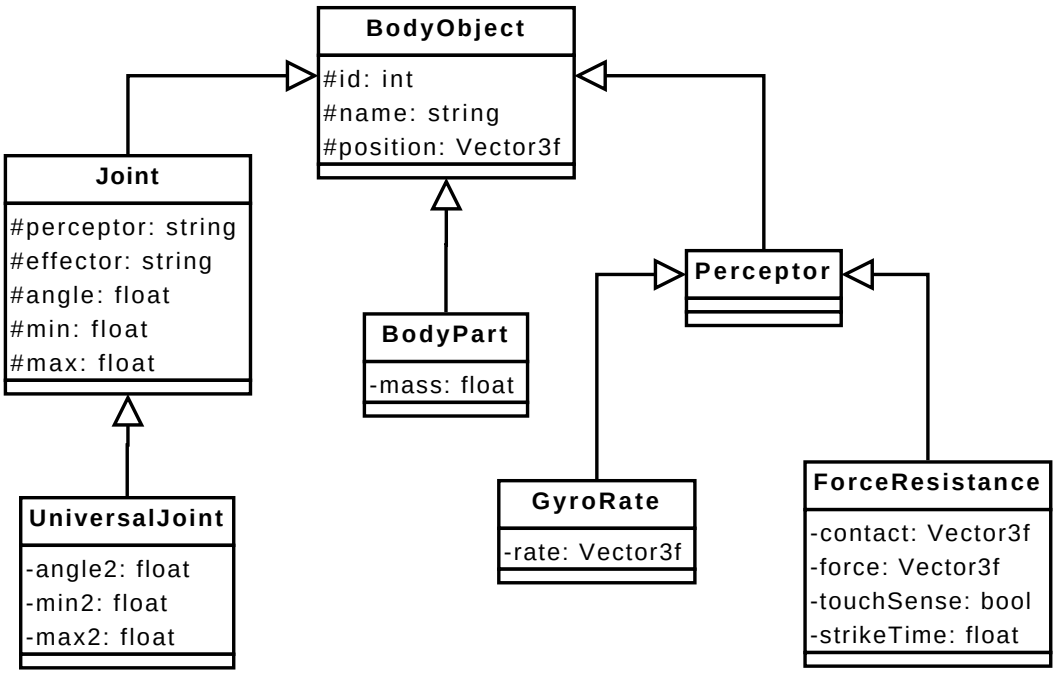

Figure B.1: The BodyObject class

A body part refers to any visible part of the body. Body parts include the head, the neck, upper arms, elbows, lower arms, hands, hips, thighs, knees, shanks, ankles and the feet.

The joints are used to describe the articulations of the body. An hinge joint is a single-DOF joint and is described by the class Joint. It stores the name of the perceptor and the name of the effector (See section 4.2.1), the current angle and the minimum and maximum limits. An universal joint extends an hinge joint by including the information about the second DOF. Examples of joints are the articulations of the neck, shoulders, elbows, hips, thighs, knees and ankles.

The class Perceptor has no attributes and is used to group the several perceptors installed on the humanoid. The GyroRate class represents the gyroscope and stores the angular rate read from the gyroscope on each cycle. The ForceResistance class refers to the foot force sensors. Each foot has its own istance of this class to keep track of the current force applied on the ground and the corresponding contact point. Two additional variables are used to know whether the foot is effectively in touch with the ground and also the last strike time of the corresponding foot.

A class named *Types* provides standard identifiers for joints and body parts that can be used anywhere in the whole code of the agent. This is possible using the following public enumerators:

```
enum BodyParts {
    ilHead,
    ilNeck,
    ilTorso,
    ilLShoulder,
    ilRShoulder,
    ilLUpperArm,
    ilRUpperArm,
    ilLElbow,
    ilRElbow,
    ilLLowerArm,
    ilRLowerArm,
    ilLHip1,
    ilRHip1,
    ilLHip2,
    ilRHip2,
    ilLThigh,
    ilRThigh,
    ilLShank,
    ilRShank,
    ilLAnkle,
    ilRAnkle,
    ilLFoot,
    ilRFoot,

    NBODYPARTS,
};
```

```
enum Joints
{
    ijHead1,
    ijHead2,
    ijLHip,
    ijRHip,
    ijLThigh1,
    ijRThigh1,
    ijLThigh2,
    ijRThigh2,
    ijLKnee,
    ijRKnee,
    ijLAnkle1,
    ijRAnkle1,
    ijLAnkle2,
    ijRAnkle2,
    ijLShoulder1,
    ijRShoulder1,
    ijLShoulder2,
    ijRShoulder2,
    ijLUpperArm,
    ijRUpperArm,
    ijLElbow,
    ijRElbow,

    NJOINTS,
};
```

The AgentBody class

The AgentBody class is responsible to handle all the information related with the body of the NAO agent and provides public methods so that this information is accessible externally. This class keeps a list of joints, body parts and perceptors. The private members of the class are defined as follows:

```
std::map<int, BodyPart*> parts;    // list of body parts
std::map<int, Joint*>    joints;   // list of joints

GyroRate*                gyro;     // gyroscope
ForceResistance*         lfrp;     // left foot sensor
ForceResistance*         rfrp;     // right foot sensor
```

Additionally, the class provides a set of public methods that provide access to the information of the body. The following list describes those methods:

- **void** updatePosture();

  This method should be called on each cycle to update the state information with base on the values received from the server (e.g. position of joints and body parts). The Forward Kinematics module is used to help on computing the position of joints and body parts, as described by the following code:

  ```
  for(int i = 0; i < Types::NJOINTS; i++)
      joints[i]->setPosition(fkin->JointRelField(i));

  for(int i = 0; i < Types::NBODYPARTS; i++)
      parts[i]->setPosition(fkin->BodyPartRelField(i));
  ```

  where *fkin* is the Forward Kinematics module and *JointRelField* and *BodyPartRelField* are public methods that provide information about the position of a joint or a body part, respectively.

- BodyPart* getBodyPart(**int** id);

  Gets a pointer to a body part with base on the corresponding identification number. This function also appears overloaded to perform the same operation using the name instead of the identification number.

- Joint* getJoint(**int** id);

  Gets a pointer to a joint with base on the corresponding identification number. This function also appears overloaded to perform the same operation using the name instead of the identification number.

- Perceptor* getPerceptor(**int** id);

  Gets a pointer to a perceptor with base on the corresponding identification number. This function also appears overloaded to perform the same operation using the name instead of the identification number.

- Vector3f getCoM();

  Gets a three-float vector with the three components of the Center of Mass (X, Y and Z). The body of the method is defined as follows:

```
Vector3f sum(0,0,0);

for(unsigned i = 0; i < parts.size(); i++)
    sum += parts[i]->getPosition() * parts[i]->getMass();

return (sum / totalMass);
```

- Vector getAVel();

  Gets a two-float vector containing the two components of the average velocity (forward speed and lateral speed). The average velocity is a relation between the displacement of the humanoid and the elapsed time since the beginning of the gait and is calculated using the following equation:

$$\Delta v = \frac{\Delta x}{\Delta t} \tag{B.1}$$

  where Δx is the displacement related to the initial position of the body and Δt is the elapsed time since the beginning of the gait simulation.

- **float** getTOsc();

  Gets the value of the torso average oscillation. The torso average oscillation, $\overline{\theta}$ is a measure calculated with base on the values received from the gyroscope installed on the torso. It is calculated using the following equation [83]:

$$\overline{\theta} = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \overline{x})^2 + \sum_{i=1}^{N}(y_i - \overline{y})^2 + \sum_{i=1}^{N}(z_i - \overline{z})^2}{N}} \tag{B.2}$$

  where N represents the number of simulation cycles, x_i, y_i and z_i are the values received from the gyroscope in the ith cycle and $\overline{x}$, $\overline{y}$ and $\overline{z}$ are the mean of gyroscope readings over the time and are calculated as follows:

$$\overline{x} = \frac{\sum_{i=1}^{N} x_i}{N}, \overline{y} = \frac{\sum_{i=1}^{N} y_i}{N}, \overline{z} = \frac{\sum_{i=1}^{N} z_i}{N} \tag{B.3}$$

- Polygon getSupportPolygon();

  Computes and returns the support polygon with base on the contact of the feet with the ground. The sensors are used to know whether the foot is in touch with the ground and the contact position. The size of the foot (provided by the documentation of the simulator) are then used to compute the polygon.

```
Polygon AgentBody::getSupportPolygon() {
    Vector lp[4], rp[4];

    /*
     *  Feet model:
     *      lp1    lp2         rp1   rp2
     *        ____              ____
     *       |    |            |    |
     *       |    |            |    |
     *       |----|            |----|
     *      lp0    lp3         rp0   rp3
     */

    fkin->getLeftFootExtremes(lp);
    fkin->getRightFootExtremes(rp);

    /* situations for support polygon:
     *
     * - both feet on the ground...
     *   ...aligned: polygon(lp0 lp1 lp2 rp1 rp2 rp3 rp0 lp3)
     *   ...left in front: polygon(lp0 lp1 lp2 rp2 rp3 rp0)
     *   ...right in front: polygon(lp0 lp1 rp1 rp2 rp3 lp3)
     * - only the left foot on the ground
     *   polygon(lp0 lp1 lp2 lp3)
     * - only the right foot on the ground
     *   polygon(rp0 rp1 rp2 rp3)
     */

    Vector3f lf = lfrp->getForceVector();
    Vector3f rf = rfrp->getForceVector();
    Vector3f lc = lfrp->getContactPoint();
    Vector3f rc = rfrp->getContactPoint();

    Polygon sp;
    if (lfrp->senseTouch() && rfrp->senseTouch())
    { // both feet on the ground
        if (fabs(lfoot.getX() - rfoot.getX()) < 0.01)
        { // aligned
            sp.addVertex(lp[0]); sp.addVertex(lp[1]);
            sp.addVertex(lp[2]); sp.addVertex(rp[1]);
```

```
                sp.addVertex(rp[2]); sp.addVertex(rp[3]);
                sp.addVertex(rp[0]); sp.addVertex(lp[3]);
            }
            else
            if (lfoot.getX() > rfoot.getX())
            { // left in front
                sp.addVertex(lp[0]); sp.addVertex(lp[1]);
                sp.addVertex(lp[2]); sp.addVertex(rp[2]);
                sp.addVertex(rp[3]); sp.addVertex(rp[0]);
            }
            else
            { // right in front
                sp.addVertex(lp[0]); sp.addVertex(lp[1]);
                sp.addVertex(rp[1]); sp.addVertex(rp[2]);
                sp.addVertex(rp[3]); sp.addVertex(lp[3]);
            }
        }
        else if (lfrp->senseTouch())
        { // only the left foot on the ground
            sp.addVertex(lp[0]); sp.addVertex(lp[1]);
            sp.addVertex(lp[2]); sp.addVertex(lp[3]);
        }
        else if (rfrp->senseTouch())
        { // only the right foot on the ground
            sp.addVertex(rp[0]); sp.addVertex(rp[1]);
            sp.addVertex(rp[2]); sp.addVertex(rp[3]);
        }
        else { // flying - no support polygon }

        return sp;
    }
```

Appendix C

Motion description language

C.1 Step-based method scripting language

Each step-based behavior has to be defined using a configuration file. The configuration file is based on a simple scripting language and gives support to the implementation described in the previous section, so that it is possible to define a set of joint move sequences, each of them containing joint moves. The language is very compact and is based on a simple script with a particular structure, which respects the following BNF[1] grammar:

```
<behavior>    ::= <sequence> | <sequence> EOL <behavior>

<sequence>    ::= <wait> | <move-set> | <move-set> " " <seq-options>

<move-set>    ::= <move-cmd> | <move-cmd> <move-set>

<move-cmd>    ::= "j" <integer> " " <joint-move>

<joint-move>  ::= <hinge-move> | <univ-move>

<hinge-move>  ::= <arg> " " <arg>

<univ-move>   ::= <arg> " " <arg> " " <arg> " " <arg>

<seq-options> ::= "g" <arg> | "N" <arg> | "g" <arg> " N" <arg>

<wait>        ::= "w" <arg>

<arg>         ::= <float> | " p" <integer>
```

[1] Backus-Naur Form

Lines which do not start with a "j" or a "w" are treated as comments. The rules presented above describe the grammar of the step-based generator configuration file. These rules are briefly described below:

- <behavior>: Sequence or a set of sequences defined on each line
- <sequence>: It can be just a wait command or a set of move commands or a set of move commands with extra options
- <move-set>: It can be a base move command eventually followed by another move set
- <move-cmd>: Corresponds to the character "j" followed by a joint id and move parameters, that will depend on the joint type
- <joint-move>: Decides between a hinge joint and an universal joint
- <hinge-move>: Two arguments that refer to the target angle and corresponding tolerance
- <univ-move>: Two first arguments refer for the target angle and corresponding tolerance of the first DOF and two aditional arguments for the target angle and corresponding tolerance of the second DOF
- <seq-options>: Options of a sequence: It can be the character "g" followed by an argument that represents the controller gain (common for all joints of the sequence), the character "N" followed by an argument that represents the number of steps on which the sequence will be devided (if applicable)
- <wait>: The "w" character followed by an argument that represents a time delay in simulation cycles
- <arg>: It can be a floating point value or the character "p" followed by an integer that represents the index inside the parameter vector

C.2 Sine interpolation MDL

A Motion Description Language (MDL) based on the eXtended Markup Language (XML) standard was developed to give support to the Sine Interpolation method (Section 6.2). The option for XML is because it is a very popular standard meta-language and its inherent hierarchical format is perfect for structured behaviors. This will also facilitate the future work on the construction of graphical front-end for the definition of the behaviors.

XML Schema

A XML schema describes the structure of a XML document and is also known as XML Schema Definition (XSD). The best way to show how the Sine Interpolation MDL is structured is through the use of its XSD, which is defined as follows:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="behavior">
<xs:complexType>
  <xs:sequence>
     <xs:element name="slot" minOccurs="1">
     <xs:complexType>
       <xs:sequence>
         <xs:element name="move" minOccurs="0">
         <xs:complexType>
           <xs:attribute name="id" type="xs:integer" />
           <xs:attribute name="angle" type="ArithExpression" />
           <xs:attribute name="phasei" type="ArithExpression" />
           <xs:attribute name="phasef" type="ArithExpression" />
           <xs:attribute name="kp" type="ArithExpression" />
           <xs:attribute name="ki" type="ArithExpression" />
           <xs:attribute name="kd" type="ArithExpression" />
         </xs:complexType>
         </xs:element>
       </xs:sequence>

       <xs:attribute name="name" type="xs:string" />
       <xs:attribute name="delta" type="xs:float" />
     </xs:complexType>
     </xs:element>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" />
  <xs:attribute name="type" type="xs:string" />
</xs:complexType>
</xs:element>

</xs:schema>
```

XML Entities

The XML allows for the definition of entities. In the particular case of the MDL, this makes possible the use of user-friendly names instead of numbers to refer to the joint identifiers. The entities, in this case, define a mapping Name-Id for each DOF of the agent, which will depend on the body structure of the agent. For the NAO simulated humanoid, the following entities were used:

```
<!DOCTYPE joints [
<!ENTITY Head1      "0" >
<!ENTITY Head2      "1" >
<!ENTITY LHip       "2" >
<!ENTITY RHip       "3" >
<!ENTITY LThigh1    "4" >
<!ENTITY RThigh1    "5" >
<!ENTITY LThigh2    "6" >
<!ENTITY RThigh2    "7" >
<!ENTITY LKnee      "8" >
<!ENTITY RKnee      "9" >
<!ENTITY LAnkle1    "10">
<!ENTITY RAnkle1    "11">
<!ENTITY LAnkle2    "12">
<!ENTITY RAnkle2    "13">
<!ENTITY LShoulder1 "14">
<!ENTITY RShoulder1 "15">
<!ENTITY LShoulder2 "16">
<!ENTITY RShoulder2 "17">
<!ENTITY LUpperArm  "18">
<!ENTITY RUpperArm  "19">
<!ENTITY LElbow     "20">
<!ENTITY RElbow     "21">
]>
```

The ArithExpression type

The ArithExpression type was developed in the scope of this thesis to allow for the use of any arithmetic expression instead of simple real numbers on fields such as angle, kp, ki, kd, phasei and phasef. It was developed using C++, Flex and Bison. The Flex performs the lexical analysis and delivers the tokens as input to Bison. Bison will then check the syntax of the sequence of tokens received and create an expression tree that is handled by the ArithExpression class. The expression tree provides the following nodes (derived from the ArithExpression main class):

- Abs(ArithExpression∗ expr); *//absolute value*
- UnaryMinus(ArithExpression ∗expr); *//symmetric value*
- Sum(ArithExpression∗ left, ArithExpression∗ right); *// sum*

- Sub(ArithExpression* left, ArithExpression* right); *// subtraction*
- Mul(ArithExpression* left, ArithExpression* right); *// multiplication*
- Div(ArithExpression* left, ArithExpression* right); *// division*
- Pow(ArithExpression* base, ArithExpression* exp); *//power*
- Float(**float** val); *// real number*
- Variable(**const** std::string& name); *//variable*

The Float and Variable nodes are threated as the possible leafs of the tree since they have no ArithExpression argument. The ArithExpression class is written in C++ and defined as follows:

```
typedef std::map<std::string, float> VarList;
class ArithExpression
{
    public:
        virtual ~ArithExpression();

        virtual ArithExpression* clone() = 0;
        virtual float eval(VarList vars = VarList()) = 0;
        virtual std::string toString() = 0;

        static ArithExpression* parse(const std::string& exprstr);
};
```

All the described nodes must implement the virtual methods *clone*, *eval* and *toString*. The method *clone* is used to clone an expression to another. It is used to redefine the equality operator and the copy constructor on each node. The method *toString* prints the tree in a LISP-like format. Thus, the expression string |(1 - 2^4)*5| will be printed as follows:

```
abs(mul(sub 1 (pow (Float(2) Float(4)) 5)))
```

The method *eval* is a recursive method (except on the leafs) whose function is to evaluate the expression. Its argument is a mapping Variable-Value containing the definition of variables. If a variable appears in the expression but does not appear in this list, it will be threated as zero. The following examples are the *eval* methods of the Sum and the Variable classes:

```
float Sum::eval(VarList vars) {
    return left->eval(vars) + right->eval(vars);
}

float Variable::eval(VarList vars) {
    if (vars.find(m_name) != vars.end()) { // variable is defined
        return vars[m_name];
    }
    else return 0.0;
}
```

The method *parse* is static and provides an object-independent way to parse an expression stored as a string. It delivers all the work to Flex and Bison and then simply returns the final expression tree:

```
ArithExpression* ArithExpression::parse(const string& exprstr)
{
    YY_BUFFER_STATE mybuffer =
                    Expression_scan_string(exprstr.c_str());
    ArithExpression *expr;

    if (Expressionparse(&expr))
    {
        cerr << "Error parsing the expression " << exprstr << endl;
        exit(1);
    }

    Expression_delete_buffer(mybuffer);
    return expr->clone();
}
```

Example of usage

To perform a bend/stretch sequence the robot must use the joints that perform pitch rotations on each leg. The absolute value of the gyroscope pitch rate (a variable named *gyro.x*) may be useful to stabilize the robot at the ankles. This can be done using the following configuration:

```
<behavior name="Example1" type="slot">

  <slot name="bend" delta="200">
        <move id="&LThigh1;" angle="30" kd="0.2" />
        <move id="&RThigh1;" angle="30" kd="0.2" />
        <move id="&LKnee;" angle="-60" />
        <move id="&RKnee;" angle="-60" />
        <move id="&LAnkle1;" angle="30 + |gyro.x|" />
        <move id="&RAnkle1;" angle="30 + |gyro.x|" />
  </slot>

  <slot name="stretch" delta="200">
        <move id="&LThigh1;" angle="0" />
        <move id="&RThigh1;" angle="0" />
        <move id="&LKnee;" angle="0" />
        <move id="&RKnee;" angle="0" />
        <move id="&LAnkle1;" angle="0" />
        <move id="&RAnkle1;" angle="0" />
  </slot>

</behavior>
```

C.3 Partial Fourier series MDL

After the creation of the PFS method (See Section 6.3), the MDL was extended to give support to this trajectory planning method. The XML schema will be used once again to describe the structure of the language:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="behavior">
<xs:complexType>
  <xs:sequence>
     <xs:element name="fourier" minOccurs="1">
     <xs:complexType>
       <xs:sequence>
         <xs:element name="delta" type="xs:float" maxOccurs="1" />
         <xs:element name="sine" minOccurs="1">
         <xs:complexType>
           <xs:attribute name="amplitude" type="ArithExpression" />
           <xs:attribute name="period" type="ArithExpression" />
           <xs:attribute name="phase" type="ArithExpression" />
           <xs:attribute name="offset" type="ArithExpression" />
         </xs:complexType>
         </xs:element>
       </xs:sequence>

       <xs:attribute name="id" type="xs:integer" />
     </xs:complexType>
     </xs:element>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" />
  <xs:attribute name="type" type="xs:string" />
</xs:complexType>
</xs:element>
```

Example of usage

Lets suppose that the goal is to define the following oscillators:

- $f_{LThigh2}(t) = 6.0 * sin(\frac{2\pi}{0.4}t) + 30.0$
- $f_{RThigh2}(t) = -6.0 * sin(\frac{2\pi}{0.4}t + \pi) + 30.0$
- $f_{LKnee}(t) = -9.6 * sin(\frac{2\pi}{0.4}t) - 50.0 - sin(\frac{4\pi}{0.4})$
- $f_{RKnee}(t) = -9.6 * sin(\frac{2\pi}{0.4}t + \pi) - 50.0 - sin(\frac{4\pi}{0.4})$

It is possible to define this oscillators using the PFS method with the following configuration file:

```
<behavior name="Example2" type="fourier">

        <fourier joint="&LThigh2;" kd="0.2">
                <sine amplitude="6" period="0.4"
                                phase="0" offset="30" />
        </fourier>

        <fourier joint="&LThigh2;" kd="0.2">
                <sine amplitude="-6" period="0.4"
                                phase="pi" offset="30" />
        </fourier>

        <fourier joint="&LKnee;">
                <sine amplitude="-9.6" period="0.4"
                                phase="0" offset="-50" />
                <sine amplitude="-1" period="0.2" />
        </fourier>

        <fourier joint="&RKnee;">
                <sine amplitude="-9.6" period="0.4"
                                phase="pi" offset="-50" />
                <sine amplitude="-1" period="0.2" />
        </fourier>

</behavior>
```

Appendix D

Optimization process

Independently of the optimization algorithm chosen, there is a generic flow during the optimization process. The basic optimization process will consist of generating the parameters offline, running the agent to test those parameters and evaluating the generated gait using some evaluation criteria.

The agent code is constructed so that it can be initialized in several modes, as listed below:

- **fcpagent -unum <number>**: Game mode: A strategy module [84] will attempt to give some role to the agent based on its uniform number.
- **fcpagent -move <file> [-prep <file2>]**: File test mode: The agent runs the gait defined by the motion description file, <*file*>. Optionally, it is also possible to run an additional file that contains the preparation gait.
- **fcpagent -f <gait>**: Optimization mode: The *f* argument stands for fitness and it is followed by the name of the gait to optimize (e.g. walk, turn).

For the optimization process, the optimization mode is used. It consists of reading the parameters from a file, create the gait based on those parameters, test the gait and write the fitness value to some other file. The files are used as a way to perform communication between the agent and the optimization algorithm.

Figure D.1 shows the generic flow of the optimization process, both for the agent (left side) and for the optimization algorithm (right side). The agent blocks on the file *parameters.dat* waiting for the parameters to be generated and the optimization algorithm blocks on the file *results.dat* waiting for the fitness to be calculated. After reading the file, both the agent and the optimization algorithm will delete it so that the other can block again and the process can continue.

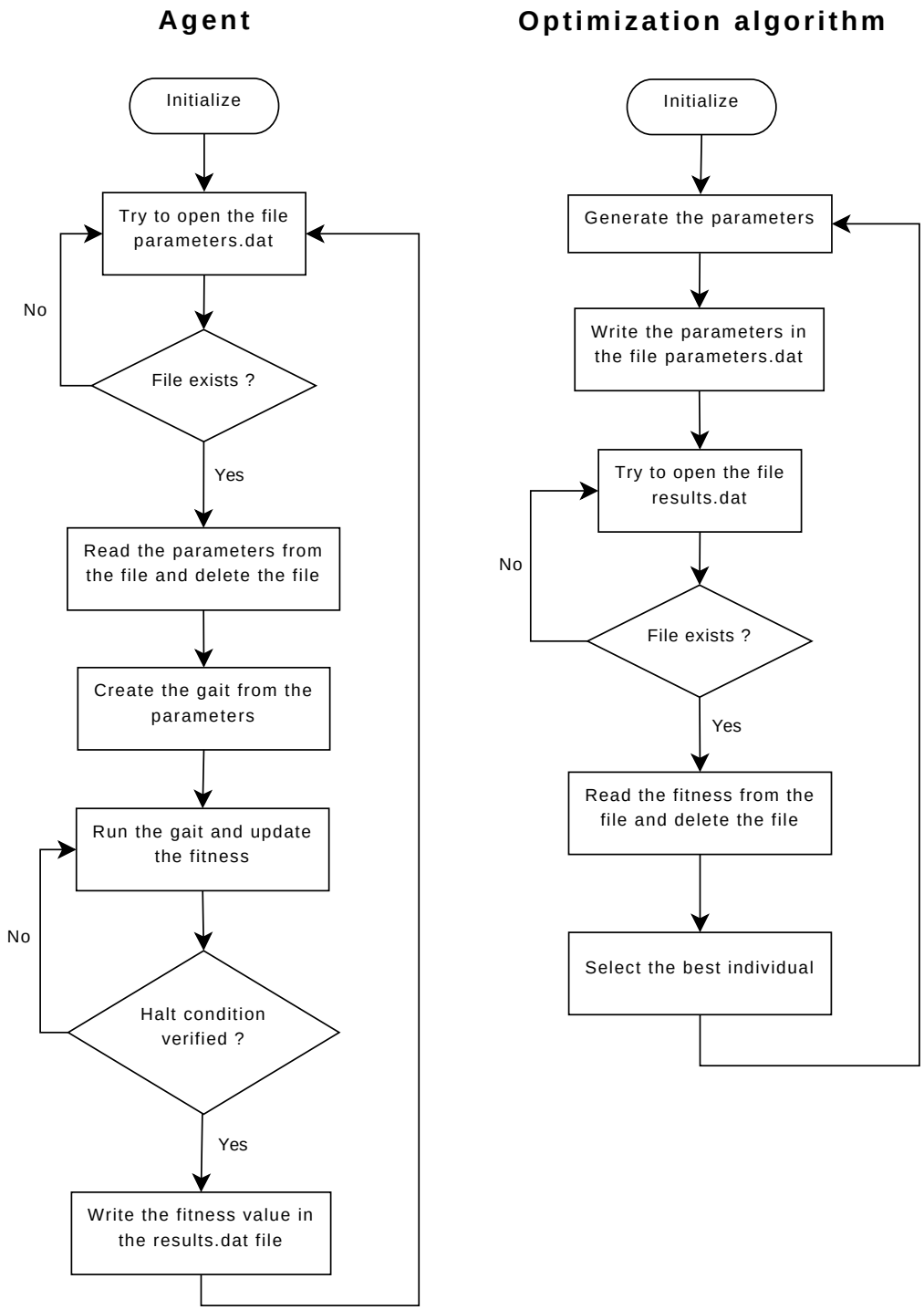

Figure D.1: Optimization process

Some of the operations performed both by the agent and the optimization algorithm will depend on the gait being optimized: The generation of the gait from the parameters, the halt condition that will stop the test of the same parameters and how the parameters should be generated by the optimization algorithm are example of such operations. A class generically called by *Evaluator* was to contain gait-independent information. The gait-specific evaluators must derive from this class and implement its virtual methods (See Figure D.2).

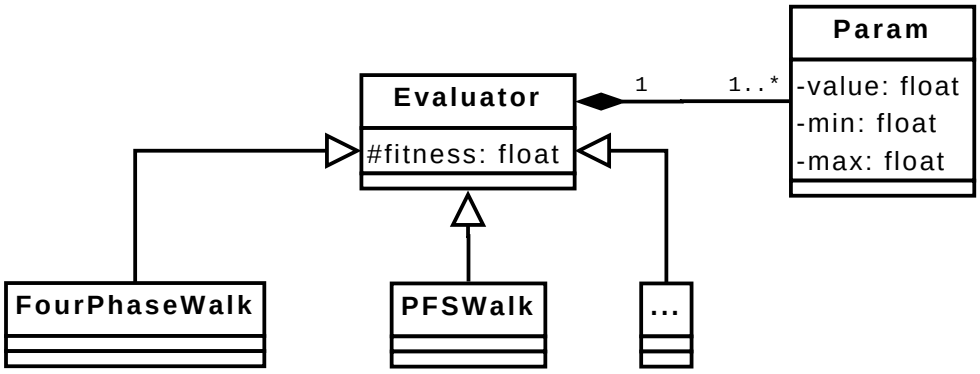

Figure D.2: The Evaluator class

The class *Evaluator* stores the current parameters and the corresponding definition domains in a list and also the fitness value assigned to those parameters. The gait-independent methods are listed below:

- **static** Evaluator∗ getEvaluatorByName(**const** std::string& name);
  Static method that allows to get a pointer to a specific evaluator based on its name. The main class is used for abstraction. This is used by the optimization process to retrieve the particular characteristics of the gait being optimized.

- **static void** writeParams(ParamList params);
  Used by the optimization algorithm to write the parameters into the file *parameters.dat*.

- **static** ParamList readParams();
  Used by the agent to read the parameters from the file *parameters.dat*.

- **static void** writeFitness(**double** fitness);
  Used by the agent to write the parameters into the file *results.dat*.

- **static double** readFitness();
  Used by the optimization algorithm to read the parameters from the file *results.dat*.

- **void** setParams(ParamList params);
  Set a list of parameters in the object's state.

- ParamList getParams();
  Gets the parameters stored in the object's state.

- **double** getFitness();
  Gets the fitness value from the object's state.

ParamList is a list of objects of the class *Param*, represented in Figure D.2. The remaining methods are *virtual* and must be implemented by the derived classes that implement the gait-specific evaluator.

- **virtual void** init ();
  Used by the agent to initialize the fitness value.
- **virtual void** update();
  Used by the agent to update the fitness value with base on same fitness function.
- **virtual** Vector3f getStartPosition ();
  Gets a three-float vector containing the position where the agent should start the test.
- **virtual** GaitGenerator∗ generateGait();
  Used by the agent to generate the gait from the parameters. This strongly depends on the gait being optimized.
- **virtual** GaitGenerator∗ generatePrepGait();
  Used by the agent to generate an eventual preparation gait. It returns *NULL* if no preparation is needed.
- **virtual** checkLocalConstraints();
  Checks whether the local constraints are verified (e.g. are the feet parallel with the ground?)
- **virtual** checkHaltCondition();
  Checks if the test should be stoped (e.g. timeout reached or desired distance traveled).

Bibliography

[1] Henrik Hautop Lund and Orazio Miglino. From simulated to real robots. In *International Conference on Evolutionary Computation*, pages 362–365, 1996.

[2] Nuno Lau, Luis Paulo Reis, Hugo Picado, and Nuno Almeida. FCPortugal: Simulated humanoid robot team description proposal for RoboCup 2008. In *Proceedings CD of RoboCup 2008*, 2008.

[3] Nuno Lau, Luís Paulo Reis, Hugo Picado, and Nuno Almeida. Paper contributions, results and work for the simulation community of FCPortugal. In *Proceedings CD of RoboCup 2008*, 2008.

[4] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawa, and Hitoshi Matsubara. RoboCup: A challenge problem for ai and robotics. In Hiroaki Kitano, editor, *RoboCup*, volume 1395 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 1997.

[5] Hiroaki Kitano, Satoshi Tadokoro, Itsuki Noda, Hitoshi Matsubara, Tomoichi Takahashi, Atsuhi Shinjou, and Susumu Shimada. RoboCup Rescue: search and rescue in large-scale disasters as a domain for autonomous agents research. In *Systems, Man, and Cybernetics, 1999. IEEE SMC '99 Conference Proceedings. 1999 IEEE International Conference on*, pages 739–743, 1999.

[6] Henrik Hautop Lund and Luigi Pagliarini. Robot soccer with LEGO mindstorms. In Minoru Asada and Hiroaki Kitano, editors, *RoboCup*, volume 1604 of *Lecture Notes in Computer Science*, pages 141–151. Springer, 1998.

[7] Tijn van der Zant and Thomas Wisspeintner. RoboCup X: A proposal for a new league where robocup goes real world. In Bredenfeld et al. [85], pages 166–172.

[8] Itsuki Noda, Hitoshi Matsubara, Kazuo Hiraki, and Ian Frank. Soccer server: A tool for research on multiagent systems. *Applied Artificial Intelligence*, 12(2-3):233–250, 1998.

[9] Frans Groen Remco De Boer, Jelle Kok. Uva trilearn 2001 team description. In Birk et al. [86], pages 551–554, 2001.

[10] Martin Riedmiller, Thomas Gabel, Johannes Knabe, and Hauke Strasdat. H.: Brainstormers 2d team description 2005. In Bredenfeld et al. [85].

[11] Noda et al. *RoboCup Soccer Server Users Manual, RoboCup Federation*, June 2001.

[12] Luís Paulo Reis and Nuno Lau. Coach unilang - a standard language for coaching a (robo)soccer team. In Birk et al. [86], pages 183–192, 2001.

[13] RoboCup Community. *RoboCup Soccer Server 3D Users Manual*, 2008.

[14] Rodrigo da Silva Guerra, Joschka Boedecker, Norbert Mayer, Shinzo Yanagimachi, Yasuji Hirosawa, Kazuhiko Yoshikawa, Masaaki Namekawa, and Minoru Asada. Introducing physical visualization sub-league. In Ubbo Visser, Fernando Ribeiro, Takeshi Ohashi, and Frank Dellaert, editors, *RoboCup*, volume 5001 of *Lecture Notes in Computer Science*, pages 496–503. Springer, 2007.

[15] Robin Murphy. *Introduction to AI Robotics.* MIT Press, 2000.

[16] Maria Prado, Antonio Simón, Ana Pérez, and Franscisco Ezquerro. Effects of terrain irregularities on wheeled mobile robot. *Robotica*, 21:143–152, 2003.

[17] Sten Grillner. *Control of motion in bipeds, tetrapods and fish.* American Physiology Society, Bethesda, 1981.

[18] Blair Calancie, Belinda Needham-Shropshire, Patrick Jacobs, Kate Willer, Gregory Zych, and Barth Green. Involuntary stepping after chronic spinal-cord injury — evidence for a central rhythm generator for locomotion in man. *Brain*, 117(5):1143–1159, 1994.

[19] Margareta Nordin and Victor Frankel. *Basic Biomechanics of the Musculoskeletal System*, chapter Biomechanics of the lumbar spine, pages 183–207. Lea & Febiger, 1989.

[20] Daniel Hein. *Evolution of Biped Walking Using Physical Simulation.* PhD thesis, Humboldt, University of Berlin, 2007.

[21] Julien Nicolas. Artificial evolution of controllers based on non-linear oscillators for bipedal locomotion, Diploma Thesis, EPFL / Computer Science, 2004.

[22] Dirk Wollherr. Design and control aspects of humanoid walking robots. Master's thesis, Technische Universität München, 2005.

[23] Shinichiro Nakaoka, Atsushi Nakazawa, Kazuhito Yokoi, Hirohisa Hirukawa, and Katsushi Ikeuchi. Generating whole body motions for a biped humanoid robot from captured human dances. In *Proceedings of 2003 IEEE International Conference on Robotics and Automation, ICRA' 03. IEEE International Conference*, volume 3, pages 3905–3910, 2003.

[24] Raymond Serway. *Physics for Scientists and Engineers with Modern Physics.* Brooks Cole Publishing Company, 6th edition, 2003.

[25] Ambarish Goswami. Postural stability of biped robots and the foot-rotation indicator (fri) point. *The International Journal of Robotics Research*, 18(6):523–533, 1999.

[26] Miomir Vukobratovic and Yury Stepanenko. On the stability of anthropomorphic systems. *Mathematical Biosciences*, 15 i1:1–37, 1972.

[27] Miomir Vukobratovic, Branislav Borovac, Dusan Surla, and Dragan Stokic. Biped locomotion: dynamics, stability, control and applications (scientific fundamentals). *Springer*, 1990.

[28] Takemasa Arakawa and Toshio Fukuda. Natural motion generation of biped locomotion robot using hierarchical trajectory generation method consisting of GA, EP layers. In *Proceedings of 1997 IEEE International Conference on Robotics and Automation, ICRA'97*, volume 1, pages 211–216, 1997.

[29] Karl Muecke, Patrick Cox, and Dennis Hong. *DARwIn Part 1: Concept and General Overview*, pages 40–43. SERVO Magazine, 12 2006.

[30] Stefano Carpin and Enrico Pagello. The challenge of motion planning for humanoid robots playing soccer. *Humnaoids soccer workshop, held at IEEE Humanoid 2006*, 2006.

[31] Yoshiaki Sakagami, Ryujin Watanabe, Chiaki Aoyama, Shinichi Matsunaga, Nobuo Higaki, and Kikuo Fujimura. The intelligent asimo: system overview and integration. In *Intelligent Robots and System, 2002. IEEE/RSJ International Conference on*, volume 3, pages 2478–2483, 2002.

[32] Masahiro Fujita, Kohtaro Sabe, Yoshihiro Kuroki, Tatsuzo Ishida, and Toshi Doi. SDR-4X II: A small humanoid as an entertainer in home environment. In Paolo Dario and Raja Chatila, editors, *ISRR*, volume 15 of *Springer Tracts in Advanced Robotics*, pages 355–364. Springer, 2003.

[33] Jerry Pratt, Chee-Meng Chew, Ann Torres, Peter Dilworth, and Gill Pratt. Virtual model control: An intuitive approach for bipedal locomotion. *The International Journal of Robotics Research*, 20:129–143, 2001.

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

[35] Max Kurz and Nicholas Stergiou. Do horizontal propulsive forces influence the nonlinear structure of locomotion? *Journal of NeuroEngineering and Rehabilitation*, 4:30+, 2007.

[36] José Lima, José Gonçalves, Paulo Costa, and António Moreira. Inverted pendulum virtual control laboratory. In *Proceedings of the 7th Portuguese Conference on Automatic Control*, pages 11–13, 2006.

[37] Kawaguchi Tsutomu, Hase Kazunori, Obinata Goro, and Nakayama Atsushi. An inverted pendulum model for evaluating stability in standing posture control. In *Dynamics & Design Conference Proceedings CD*, 2006.

[38] Eric Kandel, James Schwartz, and Thomas Jessell. *Principles of Neural Science*. McGraw-Hill Medical, 4th edition, 2000.

[39] Felix Faber and Sven Behnke. Stochastic optimization of bipedal walking using gyro feedback and phase resetting. In *Proceedings of 7th IEEE-RAS International Conference on Humanoid Robots (Humanoids), Pittsburg, USA*, 2007.

[40] Carla Pinto. Central pattern generator for legged locomotion: a mathematical approach. In *Proceedings of the Workshop on Robotics and Mathematics, RoboMat07*, Coimbra, 2007.

[41] Ludovic Righetti and Auke Jan Ijspeert. Programmable central pattern generators: an application to biped locomotion control. In *Proceedings of International Conference on Robotics and Automation, ICRA'06*, pages 1585–1590, 2006.

[42] Yuichi Murase, Yusuke Yasukawa, Katsushi Sakai, and Miwa Ueki. Design of a compact humanoid robot as a platform. In *Proceedings of the 19th Conference of Robotics Society of Japan*, pages 789–790, 2001.

[43] Sven Behnke. Online trajectory generation for omnidirectional biped walking. *In Proceedings of 2006 IEEE International Conference on Robotics and Automation (ICRA' 06)*, pages 1597–1603, 2006.

[44] Jacques Denavit and Richard Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. *ASME Journal of Applied Mechanics*, 23:215–221, 1955.

[45] Reza Jazar. *Theory of Applied Robotics: Kinematics, Dynamics, and Control (Hardcover).* Springer, 1st edition, 2007.

[46] David Gouaillier, Vincent Hugel, Pierre Blazevic, Chris Kilner, Jerome Monceaux, Pascal Lafourcade, Brice Marnier, Julien Serre, and Bruno Maisonnier. The NAO humanoid: a combination of performance and affordability. *CoRR*, abs/0807.3223, 2008.

[47] Sander van Dijk, Martin Klomp, Herman Kloosterman, Bram Neijt, Matthijs Platje, Mart van de Sanden, and Erwin Scholtens. Little Green Bats humanoid 3D simulation team research proposal. In *Proceedings CD of RoboCup 2007*, 2007.

[48] Sander van Dijk, Martin Klomp, Herman Kloosterman, Bram Neijt, Matthijs Platje, Mart van de Sanden, and Erwin Scholtens. Little Green Bats humanoid 3D simulation team description paper. In *Proceedings CD of RoboCup 2008*, 2008.

[49] Xu Yuan and Tan Yingzi. SEU-3D 2007 soccer simulation team description. In *Proceedings CD of RoboCup 2007*, 2007.

[50] Xu Yuan, Shen Hui, Qian Cheng, Chen Si, and Tan Yingzi. SEU-RedSun 2008 soccer simulation team description. In *Proceedings CD of RoboCup 2008*, 2008.

[51] Xue Feng, Tai Yunfang, Xie Jiongkun, Zhou Weimin, Ji Dinghuang, and Xiaoping Chen Zhang Zhiqiang. Wright Eagle 2008 3D team description paper. *Proceedings CD of RoboCup 2008*, 2008.

[52] Johannes Jahn. *Introduction to the Theory of Nonlinear Optimization.* Springer, 3rd edition, July 2007.

[53] Thomas Weise. *Global Optimization Algorithms Theory and Application.* Thomas Weise, july 16, 2007 edition, July 2007. Online available at http://www.it-weise.de/projects/book.pdf.

[54] Shahid Bokhari. On the mapping problem. *IEEE Trans. Computers*, 30(3):207–214, 1981.

[55] David Johnson, Christos Papadimitriou, and Mihalis Yannakakis. How easy is local search? (extended abstract). In *FOCS*, pages 39–42. IEEE, 1985.

[56] Scott Kirkpatrick, Daniel Gelatt Jr., and Mario Vecchi. Optimization by simmulated annealing. *Science*, 220(4598):671–680, 1983.

[57] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986.

[58] Peter Laarhoven and Emile Aarts, editors. *Simulated annealing: theory and applications.* Kluwer Academic Publishers, Norwell, MA, USA, 1987.

[59] John Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, 1975.

[60] Melanie Mitchell. *An Introduction to Genetic Algorithms.* The MIT Press, 1998, 1998.

[61] Matthew Wall. *GAlib: A C++ Library of Genetic Algorithm Components, Documentation Revision B, Version 2.4*, August 1996.

[62] The MathWorks. *Genetic Algorithm and Direct Search Toolbox, User's Guide, Version 2*, September 2005.

[63] Ryszard Michalski, Jaime Carbonell, and Tom Mitchell, editors. *Machine Learning: An Artificial Intelligence Approach*, volume I. Morgan Kaufmann, Los Altos, California, 1983.

[64] Stephen Jose Hanson, Werner Remmele, and Ronald Rivest, editors. *Machine Learning: From Theory to Applications - Cooperative Research at Siemens and MIT*, volume 661 of *Lecture Notes in Computer Science.* Springer, 1993.

[65] Amit Konar. *Artificial intelligence and soft computing: behavioral and cognitive modeling of the human brain.* CRC Press, Inc., Boca Raton, FL, USA, 2000.

[66] Heiko Müller, Martin Lauer, Roland Hafner, Sascha Lange, Artur Merke, and Martin Riedmiller. Making a robot learn to play soccer using reward and punishment. In Joachim Hertzberg, Michael Beetz, and Roman Englert, editors, *KI*, volume 4667 of *Lecture Notes in Computer Science*, pages 220–234. Springer, 2007.

[67] Verena Heidrich-Meisner, Martin Lauer, Christian Igel, and Martin Riedmiller. Reinforcement learning in a nutshell. In *Proceedings of the 15th European Symposion on Artificial Neural Networks*, pages 277–288, 2007.

[68] Christopher Watkins. *Learning from Delayed Rewards.* PhD thesis, University of Cambridge, England, 1989.

[69] Oliver Obst and Markus Rollmann. Spark - a generic simulator for physical multi-agent simulations. In Gabriela Lindemann, Jörg Denzinger, Ingo J. Timm, and Rainer Unland, editors, *MATES*, volume 3187 of *Lecture Notes in Computer Science*, pages 243–257. Springer, 2004.

[70] Patrick Riley. SPADES: A system for parallel-agent, discrete-event simulation. *AI Magazine*, 24(2):41–42, 2003.

[71] Hitay Ozbay. *Introduction to Feedback Control Theory.* CRC Press (July 28, 1999), 1st edition, 1999.

[72] José Lima, José Gonçalves, Paulo Costa, and António Moreira. Humanoid robot simulation with a joint trajectory optimized controller. *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*, pages 986–993, Sept. 2008.

[73] João Silva. Sensor fusion and behaviours for the CAMBADA robotic soccer team. Master's thesis, Universidade de Aveiro, 2008.

[74] Alberto Isidori. *Nonlinear Control Systems (Communications and Control Engineering)*. Springer (August 11, 1995), 3rd edition, 1995.

[75] Kiam Heong Ang, Gregory Chong, and Yun Li. PID control system analysis, design and technology. *IEEE Transactions on Control Systems Technology*, 13:559–576, 2005.

[76] Vítor Santos and Filipe Silva. Design and low-level control of a humanoid robot using a distributed architecture approach. *Journal of Vibration and Control*, 12:1431–1456, 2006.

[77] John Ziegler and Nathaniel Nichols. Optimum settings for automatic controllers. *Transactions of ASME*, 64:759–768, 1942.

[78] Amin Zamiri, Amir Farzad, Ehsan Saboori, Mojtaba Rouhani, Mahmoud Naghibzadeh, and Amin Milani Fard. An evolutionary gait generator with online parameter adjustment for humanoid robots. *Computer Systems and Applications, 2008, AICCSA 2008*, 2008.

[79] Raffaello D'Andrea. The Cornell RoboCup robot soccer team: 1999-2003. In *Handbook of Networked and Embedded Control Systems*, pages 793–804. Birkhäuser Boston, 2005.

[80] Josep Porta and Enric Celaya. Body and leg coordination for omnidirectional walking in rough terrain. In *Proceedings of the 3rd International Conference on Climbing and Walking Robots*, 2000.

[81] Bernhard Hengst, Darren Ibbotson, Son Bao Pham, and Claude Sammut. Omnidirectional locomotion for quadruped robots. In Birk et al. [86], pages 368–373, 2001.

[82] J. Hooper, A. R. Barclay, and J. C. Miles. Increasing the reliability and convergence of a genetic algorithm in a varying scale multi objective engineering problem. *Artificial Intelligence in Civil Engineering, IEEE Colloquium on*, pages 2/1–2/5, Jan 1992.

[83] Milton Heinen and Fernando Osório. Applying genetic algorithms to control gait of physically based simulated robots. In *Proceedings of 2006 IEEE Congress on Evolutionary Computation*, pages 500–505, 2006.

[84] Luís Paulo Reis, Nuno Lau, and Eugénio Costa Oliveira. Situation based strategic positioning for coordinating a team of homogeneous agents. In *Balancing Reactivity and Social Deliberation in Multi-Agent Systems, LNAI 2103*, pages 175–197, 2001.

[85] Ansgar Bredenfeld, Adam Jacoff, Itsuki Noda, and Yasutake Takahashi, editors. *RoboCup 2005: Robot Soccer World Cup IX*, volume 4020 of *Lecture Notes in Computer Science*. Springer, 2006.

[86] Andreas Birk, Silvia Coradeschi, and Satoshi Tadokoro, editors. *RoboCup 2001: Robot Soccer World Cup V*, volume 2377 of *Lecture Notes in Computer Science*. Springer, 2002.