Universidade de Aveiro
Departamento de
Electrónica, Telecomunicações e Informática,
2011

Edgar Filipe
da Silva Domingues

Desenvolvimento de Comportamentos para Robô Humanoide
Development of Behaviors for Humanoid Robot

Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2011

Edgar Filipe da Silva Domingues

Desenvolvimento de Comportamentos para Robô Humanoide
Development of Behaviors for Humanoid Robot

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Prof. Nuno Lau e do Prof. Bruno Pimentel

o júri / the jury

presidente / president

Doutor Tomás António Mendes Oliveira e Silva
Professor Associado da Universidade de Aveiro

vogais / examiners committee

Doutor Luís Paulo Gonçalves dos Reis
Professor Auxiliar da Universidade do Porto

Doutor José Nuno Panelas Nunes Lau
Professor Auxiliar da Universidade de Aveiro (orientador)

Doutor Bruno Figueiredo Pimentel
Professor Auxiliar Convidado da Universidade do Porto (co-orientador)

agradecimentos

Quero agradecer ao meus orientadores, Prof. Nuno Lau e Prof. Bruno Pimentel, pelo tempo despendido comigo e pelo apoio dado ao longo da realização desta dissertação.
Quero também agradecer à minha família, especialmente aos meus pais, Manuel Domingues e Graça Tereso, e à minha irmã, Inês Domingues, pelo apoio que me deram ao longo do meu percurso académico.
Por fim, agradeço aos meus amigos e colegas de laboratório—Mário Antunes, Gustavo Pires, Ricardo Machado, Aneesh Chauhan e Alina Trifan—pelo apoio e motivação que me deram. E por se terem esforçado por compreender o tópico desta disseração para mais facilmente me ajudarem e corrigirem.
Agradeço ainda a todas as pessoas que estiveram presentes ao longo da minha vida, sem elas não seria a mesma pessoa que sou hoje.

acknowledgements

I want to thank my supervisors, Prof. Nuno Lau and Prof. Bruno Pimentel, for their time and support during the realization of this thesis.
I want also to thank my family, specially my parents, Manuel Domingues and Graça Tereso, and my sister, Inês Domingues, for the support they gave me during my academic path.
At last, I thank my friends and colleagues of laboratory—Mário Antunes, Gustavo Pires, Ricardo Machado, Aneesh Chauhan, and Alina Trifan—for their support and motivation. And for their effort to understand the topic of this thesis to more easily help and correct me.
I also thank all people that have been present along my life, without them I would not be the same person I am today.

Resumo

A robótica humanoide é uma área em ativo desenvolvimento. Os robôs com forma humana estão melhor adaptados para executarem tarefas em ambientes desenhados para humanos. Além disso, as pessoas sentem-se mais confortáveis quando interagem com robôs que tenham aparência humana. O RoboCup incentiva a investigação na área da robótica através da realização de competições de robótica. Uma destas competições é a *Standard Platform League* (SPL) na qual robôs humanoides jogam futebol. O robô usado é o robô Nao, criado pela Aldebaran Robotics. A diferença entre as equipas que competem nesta liga está no software que controla os robôs. Outra liga presente no RoboCup é a *3D Soccer Simulation League* (3DSSL). Nesta liga o jogo de futebol é jogado numa simulação por computador. O modelo de robô usado é também o do robô Nao. Contudo, existem umas pequenas diferenças nas dimensões e este tem mais um grau de liberdade do que o robô real. O simulador também não consegue reproduzir a realidade com perfeição. Ambas estas ligas são importantes para esta dissertação, pois usam o mesmo modelo de robô. O objectivo desta dissertação é desenvolver comportamentos para estas ligas, aproveitando o trabalho prévio desenvolvido para a 3DSSL. Estes comportamentos incluem os movimentos básicos necessários para jogar futebol, nomeadamente: andar, chutar a bola e levantar-se depois de uma queda. Esta dissertação apresenta a arquitetura do agente desenvolvida para a SPL, que é similar à arquitetura do agente da equipa FC Portugal da 3DSSL, para permitir uma mais fácil partilha de código entre as ligas. Foi também desenvolvida uma interface que permite controlar uma perna de maneira mais intuitiva. Ela calcula os ângulos das juntas da perna, usando os seguintes parâmetros: três ângulos entre o torso e a linha que une anca ao tornozelo; dois ângulos entre o pé e a perpendicular do torso; e a distância entre a anca e o tornozelo. Nesta dissertação foi também desenvolvido um algoritmo para calcular os três ângulos das juntas da anca que produzam a desejada rotação vertical, visto o robô Nao não ter uma junta na anca que rode verticalmente. Esta dissertação também apresenta os comportamentos desenvolvidos para a SPL, alguns dos quais foram baseados nos comportamentos já existentes na 3DSSL. É apresentado um modelo de comportamento que permite criar movimentos para o robô definindo uma sequência de poses, um algoritmo para um andar *open-loop* e omnidirecional e um andar otimizado no simulador e adaptado para o robô real. A este último andar foi adicionado um sistema de *feedback* para o tornar mais robusto. Usando os comportamentos apresentados nesta dissertação, o robô atingiu uma velocidade de 16 cm/s para frente, 6 cm/s para o lado e rodou sobre si próprio a 40 graus/s. O trabalho desenvolvido nesta dissertação permite ter um agente que controle o robô Nao e execute os comportamentos básicos de baixo nível para competir na SPL. Além disso, as semelhanças entre a arquitetura do agente para a SPL com a arquitetura do agente da 3DSSL permite usar os mesmos comportamentos de alto nível em ambas as ligas.

Abstract

Humanoid robotics is an area of active research. Robots with human body are better suited to execute tasks in environments designed for humans. Moreover, people feel more comfortable interacting with robots that have a human appearance. RoboCup encourages robotic research by promoting robotic competitions. One of these competitions is the Standard Platform League (SPL) in which humanoid robots play soccer. The robot used is the Nao robot, created by Aldebaran Robotics. The difference between the teams that compete in this league is the software that controls the robots. Another league promoted by RoboCup is the 3D Soccer Simulation League (3DSSL). In this league the soccer game is played in a computer simulation. The robot model used is also the one of the Nao robot. However, there are a few differences in the dimensions and it has one more Degree of Freedom (DoF) than the real robot. Moreover, the simulator cannot reproduce reality with precision. Both these leagues are relevant for this thesis, since they use the same robot model. The objective of this thesis is to develop behaviors for these leagues, taking advantage of the previous work developed for the 3DSSL. These behaviors include the basic movements needed to play soccer, namely: walking, kicking the ball, and getting up after a fall. This thesis presents the architecture of the agent developed for the SPL, which is similar to the architecture of the FC Portugal team agent from the 3DSSL, hence allowing to port code between both leagues easily. It was also developed an interface that allows to control a leg in a more intuitive way. It calculates the joint angles of the leg, using the following parameters: three angles between the torso and the line connecting hip and ankle; two angles between the foot and the perpendicular of the torso; and the distance between the hip and the ankle. It was also developed an algorithm to calculate the three joint angles of the hip that produce the desired vertical rotation, since the Nao robot does not have a vertical joint in the hip. This thesis presents also the behaviors developed for the SPL, some of them based on the existing behaviors from the 3DSSL. It is presented a behavior that allows to create robot movements by defining a sequence of poses, an open-loop omnidirectional walking algorithm, and a walk optimized in the simulator adapted to the real robot. Feedback was added to this last walk to make it more robust against external disturbances. Using the behaviors presented in this thesis, the robot achieved a forward velocity of 16 cm/s, a lateral velocity of 6 cm/s, and rotated at 40 deg/s. The work developed in this thesis allows to have an agent to control the Nao robot and execute the basic low level behaviors for competing in the SPL. Moreover, the similarities between the architecture of the agent for the SPL with that of the agent from the 3DSSL allow to use the same high level behaviors in both leagues.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Although walking is easy for humans, it is not easy to develop biped walking robots. Beside the fact that no machine has the same anatomy of the human body, so far there is no algorithm that executes a motion so precise and robust to external disturbances as the human. This thesis talks about the behaviors developed for the humanoid robot Nao, created by Aldebaran, with the purpose of participating in the Standard Platform League (SPL) of the RoboCup competition.

This first chapter presents the motivation behind this thesis, its objectives, and contributions. An outline of the thesis is provided in the end.

1.1 Motivation

RoboCup is an international competition held regularly to stimulate robotics and artificial intelligence research [1, 2]. It is formed by various leagues including soccer, rescue, and @home. Soccer leagues include both simulated and real robot leagues. Moreover, there are different real robot leagues depending on the robot hardware. The only leagues relevant for this thesis are the SPL and the 3D Soccer Simulation League (3DSSL). Both use the same robot model: the Aldebaran Nao (Section 2.3.4). However, while the former's platform is a real robot, the latter's is a simulated robot. The use of the same robot model has the advantage of allowing the behaviors developed for both leagues to have small or no differences between them. The goal proposed by RoboCup is:

> "By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, comply with the official rule of the FIFA[1], against the winner of the most recent World Cup." [3]

This is the motivation behind the teams that compete in RoboCup: to use soccer—a game that is simple for humans but complex for robots—as a challenge to promote research, taking the resulting technologies outside the soccer field for improving people's lives.

Since the year 2000 the FC Portugal team [4, 5] participates in the RoboCup 2D Soccer Simulation League (2DSSL). It was world champion in that year and European champion in 2000 and 2001. FC Portugal participates also in the 3DSSL, since this league was created in 2004. In this league, it was world champion in 2006 and European champion in 2006 and

[1] Fédération Internationale de Football Association

2007. This year, for the first time, following this collaboration and also including members from CAMBADA [6] and 5DPO [7] teams, the Portuguese Team will participate in the SPL.

Although there are behaviors already developed for the 3DSSL, adapting these to the real robot is not an easy task, since the simulator cannot reproduce reality with precision. The simulator does not detect collisions between body parts of the same robot. The ground friction is low compared to the official SPL field. The simulated robot model is not an exact copy of the real one, having a few differences in the dimensions and one more Degree of Freedom (DoF). Therefore, new behaviors must be developed for the real robot, using those developed for simulation only as a base. Once we have stable low level behaviors on both the simulation and the real robot, the same high level behaviors can be used on both, with little or no modifications.

1.2 Objectives

The objective of this thesis is to develop behaviors for the humanoid robot Nao, making them work on both the real robot and the simulated robot, whenever possible. The behaviors to develop include the common human soccer game behaviors: walking, getting up after a fall, kicking the ball, dribbling, intercepting the ball, running, jink, heading, etc.

The behaviors developed shall be:

Stable The behaviors shall execute without making the robot fall;

Robust The behaviors shall be stable even when facing unexpected external disturbances (e.g. collisions, uneven ground, insufficient motor reaction, etc);

Efficient The behaviors shall achieve the desired objective with the best performance (e.g. in the least time possible);

Adaptable The behaviors should be parameterized to allow for easy adaptation to different floor conditions and robot models, namely those of the real and the simulated Nao;

Energy efficient The behaviors should consume as few power as possible, so the battery of the robot lasts longer.

The middle level behaviors will be developed on the basis of the previous enumerated ones. This allows the upper layers of strategy to be abstracted from the details of the lower level behaviors. The middle level behaviors are, for instance: go to, pass, shoot.

Another important part to consider is how to make the transition between different behaviors, since we are not executing always the same behavior. For example, the transition between walking and kicking the ball, or between different walking algorithms. To assure the robot remains stable when switching behaviors, the transition must produce smooth joint trajectories and result in a stable robot movement.

1.3 Contributions

This thesis contributes with an agent architecture for the SPL based on the existing agent architecture for the 3DSSL. The similarities between them allow to easily port code between the simulated and the real agents. This means the simulator can be used as a platform to test

and optimize algorithms before applying them to the real robot. Also, the same agent can be used to compete in both the 3DSSL and the SPL, therefore reducing code redundancy, apply improvements achieved in one league in the other one, and take advantage of the previous success of the FC Portugal team from the 3DSSL to compete in the SPL.

Another contribution of this thesis is the Leg Interface, which allows to control the movement of a leg using six parameters:

leg angle three angles between the torso and the line connecting hip and ankle;

foot angle two angles between the foot and the perpendicular of the torso;

leg extension distance between the hip and the ankle.

The Leg Interface was used in the walk algorithms to allow more control over the movements of the legs. Unlike most humanoid robots, the Nao does not have a vertical hip joint. It has only one joint inclined 45 degrees from the vertical axis which can be used to rotate the leg around this axis. This means that the desired vertical motions cannot be directly applied to this joint. To compensate this inclination an algorithm that calculates the angles of the three hip joints of the Nao robot needed to achieve the desired vertical rotation is presented in this thesis.

This thesis also contributes with behaviors developed for the real Nao robot with the objective of participating in the RoboCup SPL. These include different walking algorithms, namely an omnidirectional walk based on online trajectory generation and a walk which was optimized in the simulator and can walk forward (straight and turning), walk backward, and turn in place. Get up motions to put the robot in the stand position after a fall and movements to kick the ball in different directions were also successfully developed. These behaviors were created using the Slot Behavior model which allows to create behaviors specifying the desired joint angles and the time to reach them. Other generic model presented in this thesis is the CPG Behavior, which allows to develop periodic behaviors by generating the trajectory of the angle of each joint as a sum of sinusoids, specifying the parameters of each sinusoid. Some periodic behaviors developed with this model include a forward walk, a turn in place, and a rotate around the ball.

A middle level behavior was developed to make the real Nao robot follow the ball. It uses the TFSWalk behavior for locomotion, the vision to localize the ball, and the agent architecture developed.

1.4 Thesis Structure

The remainder of this thesis is organized as follows:

Chapter 2 Humanoid Robotics This chapter presents the motivation behind developing humanoid robots, the state of the art of the biped locomotion in detail, and some of the most recent and important humanoid robots developed.

Chapter 3 RoboCup Humanoid Competitions This chapter describes the current competitions of RoboCup using humanoid robots, giving special focus to the SPL and the 3DSSL.

Chapter 4 Portuguese Team Agent Software Architecture This chapter presents the Portuguese Team agent software architecture, with detail over the agent and the communication between the latter and the hardware of the real robot.

Chapter 5 Humanoid Behaviors This chapter presents the developed behaviors. It describes the Slot Behavior, CPG Behavior, OmnidirectionalWalk, and TFSWalk (all these developed for the real robot based on existing behaviors of the simulated robot). It then presents the COMWalk: a walk developed on the basis of that of the B-Human team in 2009. Then it is presented the walk developed by Aldebaran and how the agent can control it. The chapter finishes with a middle level behavior that makes the robot follow the ball. The results of the tests executed for each behavior are presented after the behavior description.

Chapter 6 Conclusion This last chapter presents the conclusions and discusses the future work.

Chapter 2

Humanoid Robotics

Humanity has long been fascinated by the idea of creating an autonomous machine that acts and looks like a human. This machine could help humans in labor, or even replace them in dangerous activities. A humanoid robot is an autonomous machine resembling human form. Usually it is composed by a trunk with two legs, two arms, and a head. However some research labs developed robots with just the part of the body they study. Some robots have only the two legs, used for studying biped locomotion. Others have arms only, to study object manipulation. There are also robots only with a head, to do research on vision or facial expressions. The development of humanoid robots also contributes to understand the human body better.

This chapter presents the motivations behind developing humanoid robots, the previous research and theories on biped locomotion, and some of the most recent and important humanoid robots developed.

2.1 Motivation

In 1973 the first full-scale anthropomorphic robot—the WABOT-1 [8] (Fig. 2.1)—was developed in the Waseda University in Tokyo, Japan. It was able to communicate with a person in Japanese and to measure distances and directions to the objects using external receptors, artificial ears and eyes, and an artificial mouth. The WABOT-1 walked with its lower limbs and was able to grip and transport objects with its hands that used tactile-sensors. Since then, many other humanoid robots have been developed, and in the last years the number of commercial humanoid robots has grown significantly. Humanoid robots are used in many areas, including: scientific research, entertainment, and factories [9, 10, 11]. The human appearance makes people comfortable to interact with them, and the human body makes them suitable for executing tasks in environments designed for humans.

Current robots are special-purpose machines, created to work in specific environments and solve specific tasks. But the robots of the future will be developed to work safely with humans in any environment solving a variety of tasks. This means they should have a humanoid body. Indeed, humanoid robots are better suited for operating in environments made for humans and have a higher acceptance by people [12]. They shall accomplish a wide variety of tasks in homes, battlefields, nuclear plants, factory floors, and space stations. And there will be no need to preprogram them to execute tasks, since they will learn through interaction with humans and the environment [13, 14].

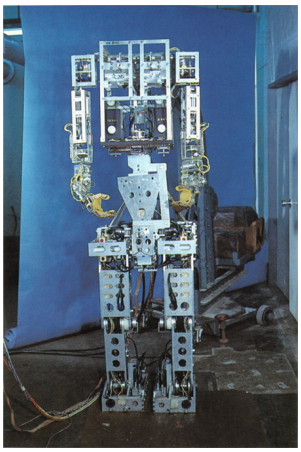


Figure 2.1: The WABOT-1 robot.

The motivations behind research on humanoid robots are many. They include: biped locomotion investigation; remote operated or autonomous robots to replace people in dangerous tasks or nocive environments (e.g. outer space, radioactive facilities); development of robots for domestic purposes, namely house cleaning and helping elder or ill people; entertainment; and to learn more about the human body and human interaction with the world.

2.2 Biped Locomotion

Biped locomotion is an active area of scientific research. Many approaches exist to try to move a robot in a human-like way. The locomotion process must be stable and robust against external disturbances to assure the robots do not fall. It should also move the robot in a fast and energy-efficient way. The development of human-like movements is important, since it is safer when walking surrounded by humans, as these can easily predict the reactions of the robot. Although more complex, legged robots have some advantages over wheeled: they can move on rough terrain, inaccessible to wheeled robots that need a continuous support surface; they can have active suspension, decoupling body from path of feet; and legs do less damage on terrain. However legged robots need complex coordination, control and balance. In locomotion we can define two goals: go to a specific location; or follow a particular trajectory. The latter is harder to achieve, since it requires an omnidirectional walk or a walk composed by simple behaviors (e.g. walk forward, stop, and turn). There are also situations where each foot must be positioned in specific locations, e.g. when using a ladder or walking over stepping-stones. In the context of this thesis, a robot that plays soccer has advantage if it can follow an arbitrary trajectory, using an omnidirectional walk to go faster to the ball and move avoiding obstacles. Only a few humanoid robots use biped locomotion successfully, since the simple act of walking on even ground without any external disturbances is not trivial to execute. During the walk, the robot's Center of Mass (CoM) is always moving and it cannot be directly controlled. It is only controlled by the momentum and contact forces from the periodic feet interaction with the ground. The inverse is also true, the foot contact with the ground can only be indirectly controlled by manipulating the robot dynamics above the foot.

Throughout this thesis we refer to three orthogonal planes that divide the human body (Fig. 2.2):

sagittal plane is the vertical plane which passes through the body from back to front dividing it in left and right sections;

coronal plane is the vertical plane which divides the body in front and back sections, also known as frontal plane;

transverse plane is the plane parallel to the ground.

Moreover, we use the following system of coordinates (Fig. 2.2): the x axis goes from back to front; the y axis from right to left; and the z axis from down to up. The rotations around these axis are called roll, pitch, and yaw, respectively.

Walking is a repetitive sequence of movements composed by alternation of single- and double-support phases (Fig. 2.3). In the single-support phase only one foot is in contact with the ground. In the double-support phase both feet are on the ground. When walking forward, the single-support phase has the support leg on the ground while the swinging leg is moved from rear to front. The double-support phase begins when the foot moving forward touches the ground and ends when the rear foot leaves the ground. After this, there is another single-support phase, this time with the other leg as the swinging leg. These two phases alternate continuously. The leg movement can also be partitioned in two others phases: the stance-phase; and the swing-phase. In the stance-phase the leg is on the ground, supporting the body. In the swing-phase the leg is in the air, moving from back to front.

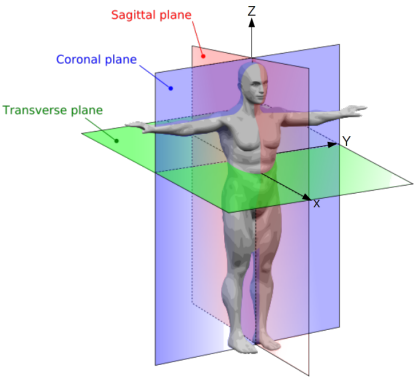


Figure 2.2: Orthogonal planes of the human body.

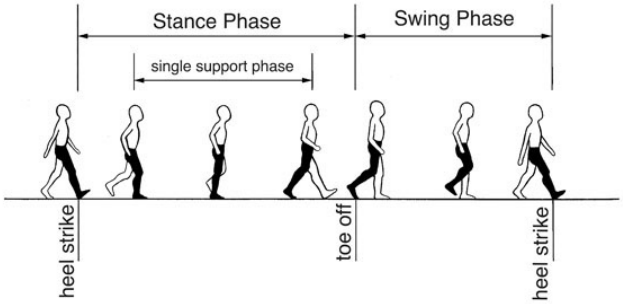


Figure 2.3: Complete walk cycle of a human walking forward (adapted from [15]).

Developing an algorithm for making a robot walk is defining how to control each joint of the robot through time. The number of joints and the number of Degree of Freedoms (DoFs) are key aspects in the algorithm creation as they bound the robot movements. More DoFs allow more flexibility of movements, but imply more parameters to control and a more complex algorithm.

One important property all behaviors must have is stability. There are two approaches to achieve stability: static stability; and dynamic stability. These two are presented in the next two sections.

2.2.1 Static Stability

Static stability, or static balance, is the simplest way to achieve stability. A robot is statically stable if the ground projection of its CoM is inside the support polygon [16, 17, 18].

However, it must move slowly, so that inertial effects do not disturb its balance. The support polygon is the convex hull of the foot support area. When the robot has only one foot on the ground (single-support phase) the support polygon is the contact area between the foot and the ground. When both feet are on the ground (double-support phase) the support polygon is the convex hull of both contact areas between the feet and the ground. Using static stability, any motion can be interrupted at anytime and the robot will remain stable. However, such restrictions make this kind of walk slow.

2.2.2 Dynamic Stability

Human walk is not statically stable, there are moments in which the center of mass leaves the support polygon, making the motion unstable for a short period before becoming stable again. However, it has dynamic stability because it results in a stable walk if the walking motion is continuous. In a dynamically stable walk, stability is assured when the Zero-Moment Point (ZMP) is kept inside the support polygon. The ZMP [19] is the point on the ground where the tipping moment acting on the robot, due to gravity and inertial forces, equals zero. This tipping moment is the component of the moment tangential to the ground. The perpendicular component can only rotate the robot changing its direction, without affecting its stability. When the ZMP leaves the support polygon the ground cannot exert the forces needed to keep the robot from tipping over, making it unstable and fall.

In a dynamically stable walk, the ZMP coincides with the Center of Pressure (CoP) [19, 20]. CoP is the point where the pressure force between the foot and the ground acts. When the gait is not dynamically stable, the ZMP leaves the support polygon. The ZMP outside the support polygon can be called Foot Rotation Indicator (FRI) [21] point or the Fictitious ZMP (FZMP) [19]. The distance from the latter to the foot edge is proportional to the intensity of the instability.

There are different approaches to develop a walking robot using dynamic stability. One approach is to replay trajectories generated offline for the joints or for the ZMP, which are modified during the walk according to sensor feedback [22, 23, 24, 25, 26, 27]. This requires precise knowledge of robot dynamics, mass distribution and inertia of each link to generate motion patterns. It is a computationally intensive process, meaning the trajectories can only be generated offline.

A different approach is to generate trajectories in real-time based on the state of the kinematic system and the wanted speed and direction of the walk [28, 18, 29, 30, 31, 32, 33]. Since trajectory generation is computationally intensive these approaches use limited knowledge about the robot dynamics, namely: total center of mass location, total angular momentum, etc. One simplified model used is the inverted pendulum [34]. This model assumes that when a biped robot is supported on a single foot, its dominant dynamics can be represented by a single inverted pendulum which connects the supporting foot to the CoM.

2.2.3 Central Pattern Generator

Central Pattern Generators (CPGs) are neural oscillators that can be used to generate rhythmic movements [35, 36]. Studies show that many animals use them in behaviors and that it is also present in the control of the human walking [37, 38]. It has the advantage of not needing neither sensor feedback nor model information of the robot. However, it is hard to determine the parameter settings that generate the desired walking pattern. Some

approaches use optimization algorithms to search for the best parameter settings that produce a stable and fast walking pattern, namely Genetic Algorithms (GA) [39] and Particle Swarm Optimization (PSO) [40].

2.2.4 Sven Behnke's Omnidirectional Walk

Sven Behnke presents in [41] an omnidirectional walk developed for the humanoid robot Jupp (Section 2.3.1). It is fully parameterizable and has low computational complexity. This work was the base for the OmnidirectionalWalk behavior developed for simulation, whose adaptation to the real robot is presented in this thesis (Section 5.3).

To control the movement of a leg, this algorithm uses the Leg Interface. The latter receives six parameters and calculates the joint angles to be applied to the leg. The six parameters are: leg angle, $\theta_{\text{Leg}} = \left(\theta^{\text{r}}_{\text{Leg}}, \theta^{\text{p}}_{\text{Leg}}, \theta^{\text{y}}_{\text{Leg}}\right)$, angle between the torso and the line connecting hip and ankle; foot angle, $\theta_{\text{Foot}} = \left(\theta^{\text{r}}_{\text{Foot}}, \theta^{\text{p}}_{\text{Foot}}\right)$, angle between the foot and the perpendicular of the torso; and leg extension, γ, distance between the hip and the ankle. A trajectory is generated for each parameter, for both legs, producing a stable walk with the desired direction and velocity.

All trajectories are synchronized with a central clock $-\pi \leq \phi_{\text{Trunk}} < \pi$ controlled by the step frequency. Each leg has its own leg phase $-\pi \leq \phi_{\text{Leg}} < \pi$ which is the trunk phase shifted by $\pm\pi/2$. The three parameters that control the omnidirectional walk are the swing amplitudes $a_{\text{Robot}} = (a^{\text{r}}_{\text{Robot}}, a^{\text{p}}_{\text{Robot}}, a^{\text{y}}_{\text{Robot}})$. Using the three swing amplitudes and the leg phase, the various walk components are generated, as explained next, for each leg, and then combined to produce the trajectories that are the input for the Leg Interface.

The next sections present the equations which describe this walk, as presented in [41], after slight corrections.

Shifting

This component shifts the robot's CoM in a sinusoidal way:

$$\theta_{\text{Shift}} = a_{\text{Shift}} \cdot \sin(\phi_{\text{leg}} - \pi) \tag{2.1}$$

where $a_{\text{Shift}} = 0.12 + 0.08 \cdot \left\|\left(a^{\text{r}}_{\text{Robot}}, a^{\text{p}}_{\text{Robot}}\right)\right\| + 0.7 \left\|a^{\text{r}}_{\text{Robot}}\right\|$ is the shifting amplitude. The shifting results from the leg and foot roll angles:

$$\theta_{\text{LegShift}} = \theta_{\text{Shift}} \tag{2.2}$$

$$\theta_{\text{FootShift}} = -0.5 \cdot \theta_{\text{Shift}} \tag{2.3}$$

Shortening

After the robot shifts its CoM to one side, the leg on the opposite side can be shortened. The time course for the shortening is determined by the shortening phase:

$$\phi_{\text{Short}} = \upsilon_{\text{Short}} \cdot (\phi_{\text{Leg}} + \pi/2 + o_{\text{Short}}) \tag{2.4}$$

$\upsilon_{\text{Short}} = 3.0$ is the duration of the shortening and $o_{\text{Short}} = -0.05$ is the phase shift of the shortening relative to the lateral weight shifting. Using a cosine, a smooth transition between the fully extended leg and the shortened leg is produced:

$$\gamma_{\text{Short}} = \begin{cases} -a_{\text{Short}} \cdot 0.5 \cdot (\cos(\phi_{\text{Short}}) + 1) & \text{, if } -\pi \leq \phi_{\text{Short}} < \pi \\ 0 & \text{, otherwise} \end{cases} \tag{2.5}$$

$a_{\text{Short}} = 0.2 + 2 \cdot \left\| (a^{\text{r}}_{\text{Robot}}, a^{\text{p}}_{\text{Robot}}) \right\|$ is the shortening amplitude. The foot tip is lifted to avoid accidental contact with the ground during the swing:

$$\theta_{\text{FootShort}} = \begin{cases} -a^{\text{p}}_{\text{Robot}} \cdot 0.5 \cdot (\cos(\phi_{\text{Short}}) + 1) & \text{, if } -\pi \leq \phi_{\text{Short}} < \pi \\ 0 & \text{, otherwise} \end{cases} \tag{2.6}$$

Loading

Once the leg is fully extended and the heel has landed, the former is shortened a second time to facilitate its loading with the robot's weight:

$$\phi_{\text{Load}} = \upsilon_{\text{Load}} \cdot \text{piCut}(\phi_{\text{Leg}} + \pi/2 - \pi/\upsilon_{\text{Short}} + o_{\text{Short}}) - \pi \tag{2.7}$$

$\upsilon_{\text{Load}} = 3$ is the duration of this second shortening. The piCut function maps its arguments to the range $[-\pi, \pi)$ by adding multiples of 2π. The amplitude of this shortening is:

$$a_{\text{Load}} = 0.025 + 0.5 \cdot (1 - \cos(\left\| a^{\text{p}}_{\text{Robot}} \right\|)) \tag{2.8}$$

This shortening is computed as:

$$\gamma_{\text{Load}} = \begin{cases} -a_{\text{Load}} \cdot 0.5 \cdot (\cos(\phi_{\text{Load}}) + 1) & \text{, if } -\pi \leq \phi_{\text{Load}} < \pi \\ 0 & \text{, otherwise} \end{cases} \tag{2.9}$$

Swinging

Once the weight has moved to one leg and the other one has shortened, the latter is quickly moved to the walking direction. This swing is reversed during the rest of the gait cycle. The time course of the swing is determined by the swing phase:

$$\phi_{\text{Swing}} = \upsilon_{\text{Swing}} \cdot (\phi_{\text{Leg}} + \pi/2 + o_{\text{Swing}}) \tag{2.10}$$

$\upsilon_{\text{Swing}} = 2.0$ is the swing speed and $o_{\text{Swing}} = -0.15$ is the phase shift of the swing. While the forward swing motion is sinusoidal, the reverse is linear:

$$\theta_{\text{Swing}} = \begin{cases} \sin(\phi_{\text{Swing}}) & \text{, if } -\pi/2 \leq \phi_{\text{Swing}} < \pi/2 \\ b \cdot (\phi_{\text{Swing}} - \pi/2) + 1 & \text{, if } \pi/2 \leq \phi_{\text{Swing}} \\ b \cdot (\phi_{\text{Swing}} + \pi/2) - 1 & \text{, otherwise} \end{cases} \tag{2.11}$$

The speed of the reverse motion is:

$$b = -(2/(2 \cdot \pi \cdot \upsilon_{\text{Swing}} - \pi)) \tag{2.12}$$

The swing is applied using the leg angle and is partially balanced with the foot angle:

$$\theta^{\text{r}}_{\text{LegSwing}} = ls \cdot a^{\text{r}}_{\text{Robot}} \cdot \theta_{\text{Swing}} \tag{2.13}$$

$$\theta^{\text{p}}_{\text{LegSwing}} = a^{\text{p}}_{\text{Robot}} \cdot \theta_{\text{Swing}} \tag{2.14}$$

$$\theta^{\text{y}}_{\text{LegSwing}} = ls \cdot a^{\text{y}}_{\text{Robot}} \cdot \theta_{\text{Swing}} \tag{2.15}$$

$$\theta^{\text{r}}_{\text{FootSwing}} = ls \cdot 0.25 \cdot a^{\text{r}}_{\text{Robot}} \cdot \theta_{\text{Swing}} \tag{2.16}$$

$$\theta^{\text{p}}_{\text{FootSwing}} = 0.25 \cdot a^{\text{p}}_{\text{Robot}} \cdot \theta_{\text{Swing}} \tag{2.17}$$

ls (leg side) is -1 for the left leg and 1 for the right leg.

Balance

The robot is balanced by tilting it every step in the sagittal and coronal planes. To do so, offsets are added to the leg and foot angles:

$$\theta^{\mathrm{r}}_{\mathrm{FootBal}} = 0.5 \cdot ls \cdot a^{\mathrm{r}}_{\mathrm{Robot}} \cdot \cos(\phi_{\mathrm{Leg}} + 0.35) \tag{2.18}$$

$$\theta^{\mathrm{p}}_{\mathrm{FootBal}} = 0.02 + 0.08 \cdot a^{\mathrm{p}}_{\mathrm{Robot}} - 0.04 \cdot a^{\mathrm{p}}_{\mathrm{Robot}} \cdot \cos(2 \cdot \phi_{\mathrm{Leg}} + 0.7) \tag{2.19}$$

$$\theta^{\mathrm{r}}_{\mathrm{LegBal}} = 0.01 + ls \cdot a^{\mathrm{r}}_{\mathrm{Robot}} + |a^{\mathrm{r}}_{\mathrm{Robot}}| + 0.1 \cdot a^{\mathrm{y}}_{\mathrm{Robot}} \tag{2.20}$$

The leg roll angle assures the legs do not collide while walking to the side or while rotating.

Output

The previously presented components are combined to produce the input for the Leg Interface as follows:

$$\theta^{\mathrm{r}}_{\mathrm{Leg}} = \theta^{\mathrm{r}}_{\mathrm{LegSwing}} + \theta_{\mathrm{LegShift}} + \theta^{\mathrm{r}}_{\mathrm{LegBal}} \tag{2.21}$$

$$\theta^{\mathrm{p}}_{\mathrm{Leg}} = \theta^{\mathrm{p}}_{\mathrm{LegSwing}} \tag{2.22}$$

$$\theta^{\mathrm{y}}_{\mathrm{Leg}} = \theta^{\mathrm{y}}_{\mathrm{LegSwing}} \tag{2.23}$$

$$\theta^{\mathrm{r}}_{\mathrm{Foot}} = \theta^{\mathrm{r}}_{\mathrm{FootSwing}} + \theta_{\mathrm{FootShift}} + \theta^{\mathrm{r}}_{\mathrm{FootBal}} \tag{2.24}$$

$$\theta^{\mathrm{p}}_{\mathrm{Foot}} = \theta^{\mathrm{p}}_{\mathrm{FootSwing}} + \theta_{\mathrm{FootShort}} + \theta^{\mathrm{p}}_{\mathrm{FootBal}} \tag{2.25}$$

$$\gamma = \gamma_{\mathrm{Short}} + \gamma_{\mathrm{Load}} \tag{2.26}$$

The resulting Leg Interface parameters trajectories are shown in Figs. 2.4, 2.5, and 2.6, for forward walking ($a_{\mathrm{robot}} = (0, 0.25, 0)$), side walking ($a_{\mathrm{robot}} = (0.0625, 0, 0)$), and rotating ($a_{\mathrm{robot}} = (0, 0, 0.25)$), respectively.

2.2.5 Passive Dynamic Walking

Passive Dynamic Walking (PDW) are simple mechanical devices that can walk down a slope without actuators or control [42] (Fig. 2.7a). They use their dynamics powered only by gravity to achieve a stable human-like gait. Based on this, PDW robots were developed to walk on level ground using an active power source in a more energy efficient way [43] (Fig. 2.7b). Some PDW robots with knees have also been developed [44, 45].

2.3 Some Humanoid Robots

This section presents some of the most recent and important humanoid robots developed. The first is the Jupp robot created by the NimbRo team to play soccer in the RoboCup Humanoid League. The second is the HRP-2 robot—a robot developed to research cooperative work between humans and robots. The third is the ASIMO robot—the famous robot developed to operate in human environments. Last, the robot used in this thesis—the Aldebaran Nao—is presented in detail.

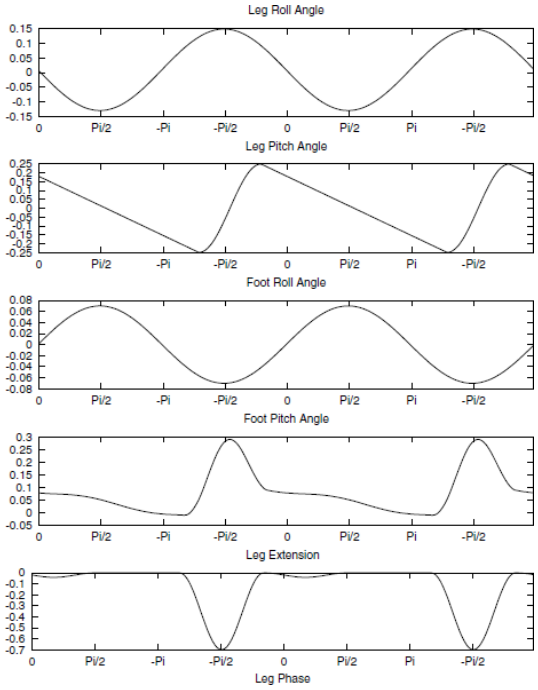


Figure 2.4: Trajectories generated while walking forward (adapted from [41]).

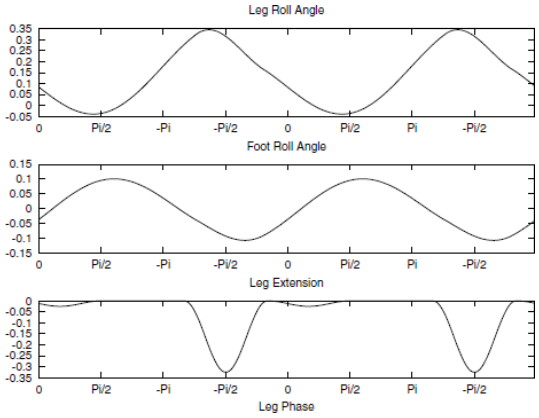


Figure 2.5: Trajectories generated while walking to the side (adapted from [41]).

2.3.1 Jupp

The Jupp robot [41, 46] (Fig. 2.8) was developed by the NimbRo team [47] to participate in the 2005 RoboCup Humanoid League competitions, KidSize class. It is a 60 cm tall human-like robot with a total weight of 2.3 kg and 19 DoFs. Each leg has three orthogonal joints in the hip, two orthogonal joints in the ankle, and one knee joint. Each arm has two orthogonal joints in the shoulder and one elbow joint. The 19th joint is in the trunk. Jupp is equipped with an attitude sensor located in the upper trunk, consisting of a dual-axis accelerometer and two gyroscopes. Jupp is fully autonomous, powered by high-current Lithium-polymer rechargeable batteries and controlled by a Pocket PC located in its chest.

It uses the walking algorithm presented in Section 2.2.4, an omnidirectional walking algorithm with low computational complexity that uses the Leg Interface to control the movements of the legs. Using this walk, the Jupp robot achieves a velocity of 14.4 cm/s while walking forward, 3.8 cm/s while walking sideways, and 23.6 deg/s while rotating. The Jupp robot can also get up after a fall.

2.3.2 HRP-2 "Promet"

The HRP-2 [11] (Fig. 2.9) is a human-size humanoid robot developed to conduct research on cooperative work among humans and robots. It is the result of the Humanoid Robotics Project headed by the Manufacturing Science and Technology Center (MSTC). The latter is sponsored by the Ministry of Economy, Trade, and Industry (METI) of Japan through New Energy and Industrial Technology Development Organization (NEDO). The development process was concluded in 2002. The HRP-2 robot is 154 cm tall, weights 58 kg, and has a

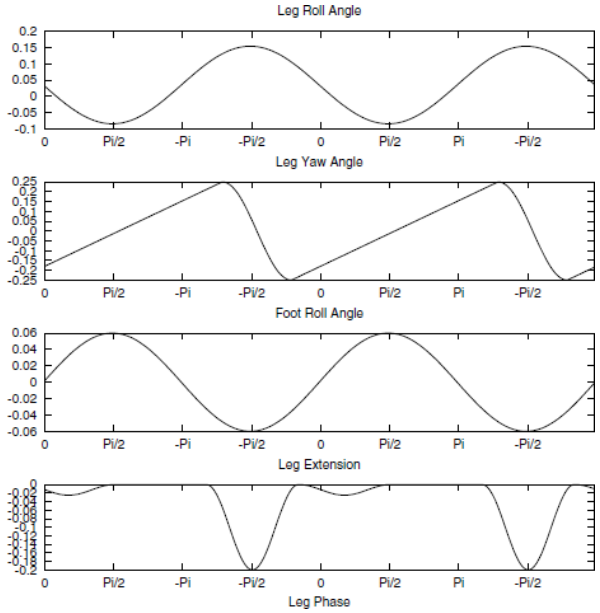
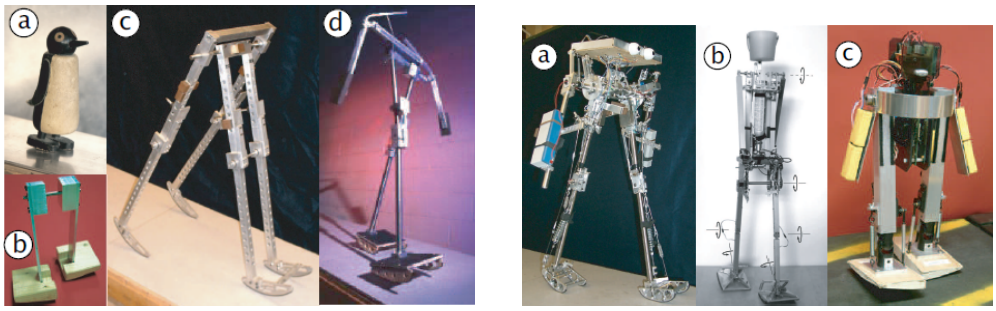


Figure 2.6: Trajectories generated while rotating (adapted from [41]).



(a) unpowered (b) powered

Figure 2.7: Example of machines using Passive Dynamic Walking (PDW) (adapted from [43]).

human-friendly appearance. Its mechanical configuration was inherited from its prototype HRP-2P [48]. It has 30 DoF: six in each leg, two in the head, six in each arm, one in each hand, and two in the waist. It has three Charge-Coupled Device (CCD) cameras inside the head and two CPU boards inside the body.

Different walking algorithms to control the HRP-2 robot were proposed [49, 50]. It can walk on narrow paths and uneven surfaces and it can achieve a velocity of 2.5 km/h. It can also lie down and get up on its own.

2.3.3 Honda ASIMO

ASIMO stands for Advanced Step in Innovative MObility [51, 52] (Fig. 2.10). It is a humanoid robot which was conceived to function in an actual human living environment in the near future. ASIMO is easy to operate, has a convenient size and weight, can move freely within a human environment, and has a people-friendly design. It was designed to reach with its hands things in the daily life environment, namely doorknobs, light switches, electric

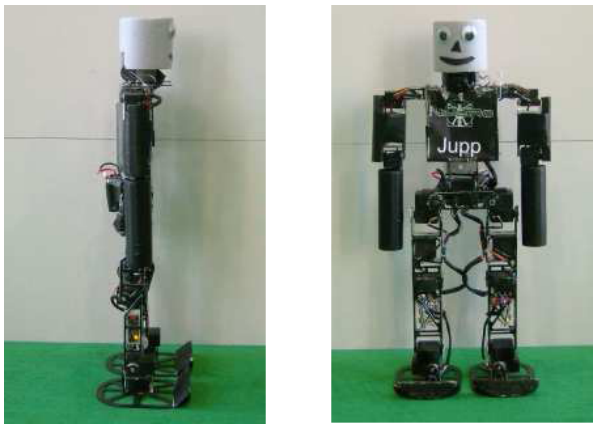


Figure 2.8: The Jupp robot of the NimbRo team (adapted from [46]).

outlets, etc. It has 34 DoF, is 130 cm tall, and weights 54 kg. It has a gyroscope and an accelerometer in the torso, and force sensors in the feet. ASIMO can change its walking pattern smoothly, freely, and flexibly at any moment without the need to stop. It generates a walking pattern in real time, calculating the optimum shift of the CoM at every instant. ASIMO's maximum running speed is 6 km/h. The first version was created in 2000 and the latest version in 2005.

Figure 2.9: The HRP-2 "Promet" robot.

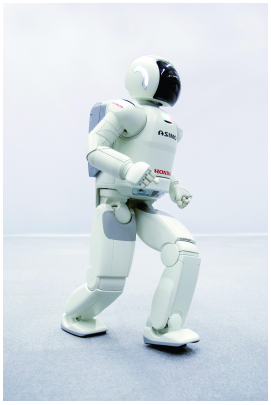


Figure 2.10: The Honda ASIMO robot.

2.3.4 Aldebaran Nao

Nao (Fig. 2.11) is a humanoid robot developed by the French company Aldebaran Robotics [53]. Aldebaran Robotics was founded in 2005 by chief executive Bruno Maisonnier. In 2008 the Nao robot replaced the Sony AIBO in RoboCup Standard Platform League (SPL). Like a human, it has two legs, two arms and a head connected to a torso. However, there are various versions with different DoFs: one with two DoFs, composed only by torso and head; another with 14 DoFs, formed by the upper body without the legs; and there are two full

body versions, with 21 and 25 DoFs, the latter being able to move the hands, unlike the former.

The version used in the SPL, and therefore important for this thesis, is the one with 21 DoFs (Fig. 2.12). It is 57 cm tall and weights 5 kg. The head has two joints. The arms have 4 joints. Although it has hands, they do not move in this version. Each leg has six joints: three on the hip, one on the knee, and two on the foot. Unlike the previous presented humanoid robots, the Nao does not have a vertical hip joint. Instead, it has a joint inclined 45 degrees towards the trunk. The objective is to replace the classical set of three joints composed by one horizontal joint and one vertical joint for each leg (Fig. 2.13). Those two hip yaw-pitch joints are controlled by the same servomotor, hence being just one DoF. Aldebaran decided to use this since it only requires one motor, instead of three [54]. This allows to reduce building costs and save space. The legs are the most important effector on this thesis, because all the developed behaviors use them. Joint control is based on providing angle targets. Each joint has an internal Proportional-Integral-Derivative (PID) controller which then acts autonomously to achieve its target. This means that the joint control on the real robot is at a higher level than that on the robot of the 3D Soccer Simulation League (3DSSL). Also, unlike the last one, the real robot provides stiffness control for each joint, which can be useful to save energy and improve the behavior robustness and speed [55]. The Nao robot has a gyroscope and an accelerometer on the torso and each foot has four Force Sensitive Resistors (FSRs) to measure the pressure force applied on them. Aldebaran provides a programming interface that delivers the filtered angles of the torso position relative to the vertical. It has two sonar sensors on the front of the torso. The head has two cameras, two loudspeakers, and two microphones. One of the cameras is in the center of the head pointing forward and the other one in the chin. The Nao has one button in the torso and one bumper sensor at the tip of each foot. The computer that controls the robot has a x86 AMD GEODE 500MHz CPU with 256 MB of SDRAM and a 2 GB flash memory.

The Nao robot runs NaoQi [53] which is a framework that manages the execution of modules. These modules are programming libraries that can be created and executed on the robot. Aldebaran made available some modules with the robot: modules to control motions, Light-Emitting Diodes (LEDs), text-to-speech, etc. One of these modules is Device Communication Manager (DCM), that allows low level access to the actuators and sensors of the robot. Modules can only communicate between each other, so it is necessary to create our own module in order to use any of the provided modules.

2.4 Conclusion

This chapter started with the motivation behind developing humanoid robots. The importance of developing robots with human appearance and human-like body was discussed. We concluded that people feel more comfortable to interact with a robot with human appearance and that a robot with a human-like body is better suited for operating in environments designed for humans.

Then, biped locomotion was presented, in which static and dynamic stability were explained. Dynamic stability, although more complex, allows to create faster walks than using static stability. Different approaches exist to create a dynamically stable walk. Some walking techniques were introduced, including the omnidirectional walk developed by Sven Behnke, which was previously used to create a behavior for the 3DSSL and is the base for the Om-

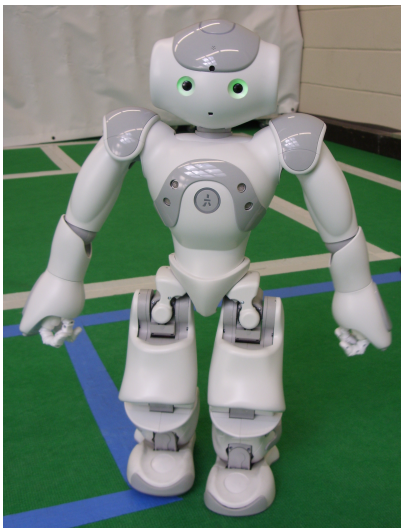


Figure 2.11: The Aldebaran Nao robot.

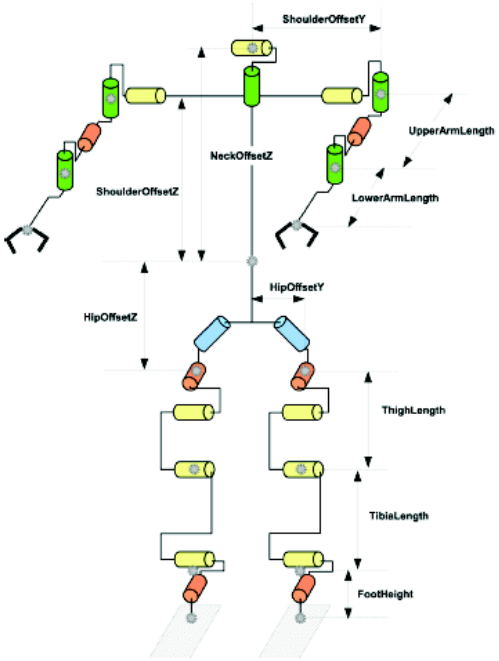


Figure 2.12: Joints of the Nao robot (adapted from [54]).

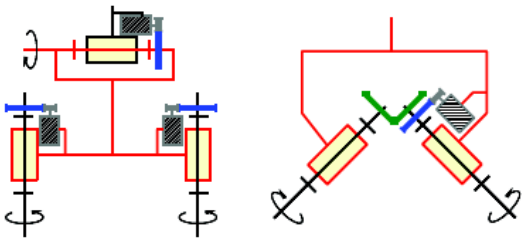


Figure 2.13: Comparison between the classical set and the Nao joints. Left: classical set composed by three joints. Right: inclined joints controlled by the same servomotor, as in the Nao robot. (Adapted from [54])

nidirectionalWalk Behavior, developed as part of the work of this thesis for the SPL, that will be presented in Section 5.3. Central Pattern Generators (CPGs) are a simple method to generate periodic movement, like the one present in a gait. However, the right parameters to generate the desired walking pattern are not simple to find. Some approaches exist to apply automatic optimization algorithms to find the best parameters that produce the desired walking pattern. The study of Passive Dynamic Walking (PDW) can help develop walking algorithms that are more energy efficient.

This chapter finished with the description of some humanoid robots, including the Jupp robot—developed to compete in the RoboCup Humanoid League—, the HRP-2 robot—created for research humanoid robotics and human robot cooperation at work—, the ASIMO robot—a robot with a human friendly appearance designed to function in a human environment—

, and the robot used in this thesis, the Aldebaran Nao. From all these robots the Nao is the only that does not have a vertical joint in the hip. Instead, it has a joint inclined 45 degrees towards the body. This means that the desired vertical rotation of the leg cannot be directly applied to this joint. This problem was solved mathematically and the solution will be presented in Section 5.3.2.

Chapter 3

RoboCup Humanoid Competitions

RoboCup has three leagues in which the objective is to play soccer using humanoid robots. These leagues are: the Humanoid League; the Standard Platform League (SPL); and the 3D Soccer Simulation League (3DSSL). The last two will be presented in the next two sections in more detail. In these three leagues, the robots must be autonomous and have a human-like body. Research areas related to these leagues include: walking; kicking the ball while maintaining balance; visual perception of the ball, other players, and field; self-localization; and team play.

In the Humanoid League, introduced in 2002, the game is played by autonomous robots with a human-like body developed by the teams. The sensors of those robots must have an equivalent in human senses. Hence, they must be placed at a position roughly equivalent to the location of the human biological sensors, as defined by the rules. In this league there are three size classes: KidSize (30-60 cm height); TeenSize (100-120 cm); and AdultSize (130 cm and taller). In the KidSize class, soccer competitions are played between teams of three robots. In TeenSize, each team is formed by two robots. In AdultSize, a striker robot plays against a goalkeeper robot, and the roles are exchanged between games.

All these three leagues need similar behaviors to play soccer. In this thesis, the only two relevant leagues are the SPL and the 3DSSL, because they use the same robot model—the Aldebaran Nao.

3.1 Standard Platform League

In the SPL, all teams play with the same robot platform, the only difference between the teams being the software that controls them. This way the teams are focused on developing software while facing the challenge of using a real robot.

Since 2008 the robot model used in both the 3DSSL and the SPL is the Aldebaran Nao (Section 2.3.4). Before, the robot used was the Sony AIBO and the league name was Four-Legged League (FLL). This constitutes a very significant change , as walking and kicking with two legs present much more complex stability issues than with the AIBO's four legs. The FFL started in 1999 organized by Sony with the name "Sony Four-Legged League". In 2004, the league name changed to FLL and was organized by the members of the teams. In 2008, with the change of the platform, from the AIBO to the Nao, the name changed to"Standard Platform League", the current name.

Figure 3.1 shows a game of the SPL. The rules of this game were made to be similar

to the ones used in a human soccer game. Each team plays with four robots. The game is played in a 6 m by 4 m soccer field built on a carpet (Fig. 3.2). The goals are 1.4 meters wide and 80 cm tall. The carpet is green and the field lines are white. The red team defends the yellow goal and the blue team defends the blue goal. The official ball is an orange street hockey Mylec ball, having a 65 mm diameter and weighting 55 grams.

Figure 3.1: Standard Platform League (SPL) game (adapted from [56]).

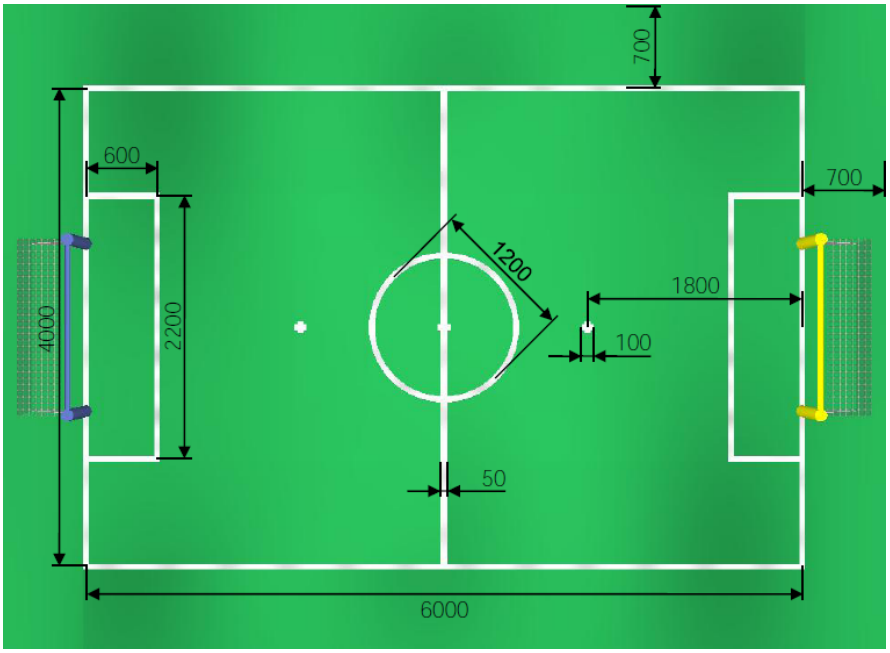


Figure 3.2: Standard Platform League (SPL) field dimensions (adapted from [57]).

A game is divided in three parts: the first half, the half-time brake, and the second half. Each part lasts 10 minutes. During the half-time brake, teams may switch robots, software, or anything that can be accomplished in that period. During this time the teams also switch their color and defended goals. All teams must use Nao robots and no modifications or

additions to the hardware are allowed. No external sensing or processing systems are allowed either. The robots can communicate between them through acoustic communication—using a microphone and a speaker—or through wireless data transfer.

During a game, a robot can be in one of six states: initial, ready, set, playing, penalized, and finished. These states are set by the GameController through the wireless network. The teams must implement code for the robot to receive the messages from the GameController. In case this is not implemented a robot is only allowed to play if it can be, at least, configured by the referee using its buttons (e.g. the torso button and the bumpers in the feet). The robots should show, using their Light-Emitting Diodes (LEDs), the team color and the game state.

For the kick-off, there are two options to place the robots in the field:

manual placement the referee manually places the robots in specific positions defined by the rules;

autonomous placement the robots have 45 seconds to autonomously move to the positions they want to occupy in the field.

The team leader can choose which placement option to use. The autonomous placement yields more freedom for choosing the positions for the robots in the field. However, the robots must be able to decide where to go without colliding with any obstacle.

The rules force the robots to move using a bipedal walk similar to the human walk and forbids them to stay in a stance wider that their shoulders for more than five seconds. The rules also state that, after a robot falls, it must start the getting up action within five seconds. If a robot cannot stand up autonomously within 20 seconds after a fall it will be removed and subject to a penalty. The only robot allowed to fall deliberately is the goalkeeper, when inside its team's penalty area. A robot that has ceased activity for 10 seconds or has turned off will also be removed. For a robot to be considered active, it must be walking, searching for the ball, or looking at the ball. The only robot allowed to touch the ball with its hands or arms is the goalkeeper when inside its team's penalty area. Also, the goalkeeper is the only robot allowed to be inside its team's penalty area. The rules do not allow the robots to push each other. A robot may not hold the ball for more than three seconds, meaning that at least half of the ball must not be covered by the convex hull of the projection of the robot's body onto the ground. The goalkeeper is allowed to hold the ball for up to five seconds, as long as it has a foot inside in its own penalty area.

3.2 3D Simulation League

In the RoboCup 3DSSL robotic soccer, games are ran on a simulator (Fig. 3.3). Each participating team programs autonomous software agents that play soccer. The agents of the same team should cooperate in order to achieve the best results. Each agent controls a humanoid robot, whose model is based on that of the Aldebaran Nao robot. The 3DSSL uses SimSpark [58] to simulate the games. This simulator uses the Open Dynamics Engine (ODE) [59] to simulate the physical environment.

The first 3D simulator for this league was introduced at RoboCup 2004, in Lisbon, Portugal. The field had the standard Fédération Internationale de Football Association (FIFA) size and the teams were formed by 11 agents. The model used for the agents was a 75 kg

Figure 3.3: A 3D Soccer Simulation League (3DSSL) game with 9 vs 9 Nao agents (adapted from [60]).

sphere. It moved omnidirectionally, had a kick effector for kicking the ball, and omnidirectional vision. The objective was to promote research on the high level behaviors, strategy, tactics, formations, player role, and multi-agent collaboration.

The field of view of the vision was reduced to 120 degrees in 2006.

In 2007, the 3DSSL started using the Fujitsu's HOAP-2 robot model. This was the first time a humanoid agent was used on this league, forcing the teams to develop low level behaviors (walking, kicking the ball, etc).

Since 2008, the robot model used is based on the Aldebaran Nao. However, it is not a precise replica of the real one, since it has a few differences in the dimensions and 22 Degree of Freedom (DoF) (one more than the real version). The ranges of the joint angles, for the SPL and the 3DSSL, are compared in Table 3.1. The joints are controlled specifying the desired angular speed for each joint. The simulated robot, like the real robot, has a gyroscope and an accelerometer on the torso and foot force sensors. Instead of cameras, the vision information is given as spherical coordinates of the objects. The robot can see the position of the ball, other players' positions, the four corners of the field, and the goal posts. Only recently, the simulation server started sending also the position of the field lines. But vision processing is not relevant for this thesis, since it is not used by the behaviors addressed.

The rules are identical to those of a human soccer game with some modifications. A match is played by two teams, with nine players each. A match consists of two halves of five minutes each and the teams switch sides between them. The game is played in a soccer field with a size of 21 by 14 meters. Each goal is 2.1 meters wide and 80 cm tall. The ball has a diameter of 8 cm and a mass of 26 grams. Each agent receives the relative position of the four corners of the field and the four goal posts if in line of sight. The players are prohibited to impede other players' movement or to obstruct the ball. The goalkeeper is the only player allowed to handle the ball with its arms or hands and only within its own penalty area. At any time during the game no more than three players of a team are allowed inside their own penalty area. Crowding the ball when an opponent is near is illegal. Therefore when two players of the same team are next to the ball the furthest is repositioned. A player must not be immobile for more than 15 seconds, except the goalkeeper whose limit is 30 seconds.

Table 3.1: Ranges, in degrees, for each joint in both the Standard Platform League (SPL) and the 3D Soccer Simulation League (3DSSL).

Joint Name	SPL Range	3DSSL Range
Head Yaw	-119.5 to 119.5	-120 to 120
Head Pitch	-38.5 to 29.5	-45 to 45
Right Shoulder Pitch	-119.5 to 119.5	-120 to 120
Right Shoulder Roll	-76 to 18	-95 to 1
Right Elbow Yaw	-119.5 to 119.5	-120 to 120
Right Elbow Roll	2 to 88.5	-1 to 90
Left Shoulder Pitch	-119.5 to 119.5	-120 to 120
Left Shoulder Roll	-18 to 76	-1 to 95
Left Elbow Yaw	-119.5 to 119.5	-120 to 120
Left Elbow Roll	-88.5 to -2	-90 to 1
Right Hip Yaw Pitch	-65.62 to 42.44	-90 to 1
Right Hip Roll	-42.30 to 23.76	-45 to 25
Right Hip Pitch	-101.54 to 27.82	-100 to 25
Right Knee Pitch	-5.90 to 121.47	-1 to 130
Right Ankle Pitch	-67.97 to 53.40	-75 to 45
Right Ankle Roll	-22.27 to 45.03	-25 to 45
Left Hip Yaw Pitch	-65.62 to 42.44	-90 to 1
Left Hip Roll	-21.74 to 45.29	-25 to 45
Left Hip Pitch	-101.63 to 27.73	-100 to 25
Left Knee Pitch	-5.29 to 121.04	-1 to 130
Left Ankle Pitch	-68.15 to 52.86	-75 to 45
Left Ankle Roll	-44.06 to 22.79	-45 to 25

After a fall a player must stand up within 30 seconds, or within 60 seconds in case it is the goalkeeper. Players must avoid touching other players. Therefore if three or more players are touching each other, a player from the team with most players in this situation or a random player will be repositioned outside the field.

3.3 Conclusion

This chapter presented the current humanoid leagues available at the RoboCup competition: the Humanoid League, the SPL, and the 3DSSL. The last two were presented in detail, since their are the most relevant for this thesis. Both of these use the same robot model: the Aldebaran Nao. However, the model used in the simulation is not an exact replica of the real robot. There are a few differences in the dimensions and it has one more DoF than the real robot. The joint control also works at different levels. While in the simulation the agent specifies the angular speed, in the real robot the agent specifies the desired angle. In the league using real robots, the SPL, the teams have less players then the 3DSSL since they are expensive to buy and costly to maintain. Also, the limited battery of the real robots restricts the time they can be used. One important difference between the two leagues, which constrains the software agent used, is the computational power. The real Nao robot has a x86 AMD GEODE CPU at 500MHz with 256MB of SDRAM. In the simulation league the

agents run in a Intel Core(TM) 2 Quad CPU Q8200 at 2.33 GHz with 4GB of DDR3 RAM. Therefore the software agent that controls the real robot cannot have as much computational and space complexity as the 3DSSL agent. Beside these differences, the use of the same robot model in both leagues has some advantages. The same developed behaviors can be used in both leagues with only a few modifications. The simulation can be used to test the behaviors before executing them in a real robot and is also a good tool for applying automatic optimization techniques to the behaviors. After the low level behaviors are developed for both leagues, the high level behaviors can be ported between them with little or no modifications.

Chapter 4

Portuguese Team Agent Software Architecture

The Portuguese Team agent software architecture is the result of contributions made by all teams collaborating in the Portuguese Team. It is composed by the DCMController, the agent, the vision, and the Real-Time Data Base (RTDB).

This chapter presents all these components, with more detail over the agent, its behaviors, and the communication with the robot hardware through the DCMController. It first describes the architecture of the software and then presents the architecture of the agent in detail.

4.1 Software Architecture

The software developed to control the robot has a distributed architecture composed by the agent, the DCMController, the vision, and the RTDB (Fig. 4.1). The vision is responsible for acquiring the visual data from the robot cameras. It processes the acquired image trying to detect and classify the objects in sight. The information extracted from the captured image is sent to the agent through the RTDB. Using the information from the vision and the other sensors, the agent creates an internal representation of the world, over which decisions are taken. It then interacts with the world through the actuators. The agent accesses the actuators and sensors of the robot through the DCMController. The RTDB is used for inter-process communication between the agent and the vision, and it also allows to share information among robots.

Figure 4.1: Software architecture.

4.1.1 DCMController

The low level access to the actuators and sensors of the Nao robot is made through the Device Communication Manager (DCM) module, presented in Section 2.3.4. Our approach to allow the agent architecture to control the robot hardware, like that of many other

Standard Platform League (SPL) teams [61, 62, 63], was to create a simple NaoQi module—the DCMController—through which the agent communicates with the DCM (Fig. 4.2). The agent uses shared memory to interact with the DCMController and the latter communicates with the DCM.

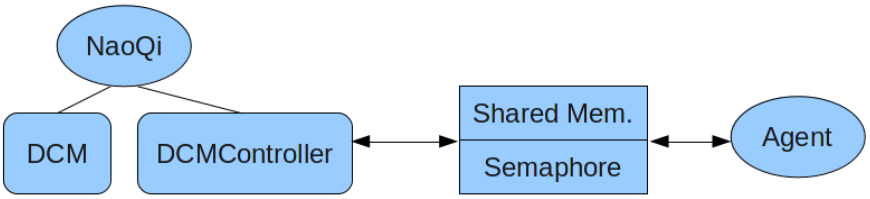


Figure 4.2: Architecture of the agent for communicating with the Device Communication Manager (DCM).

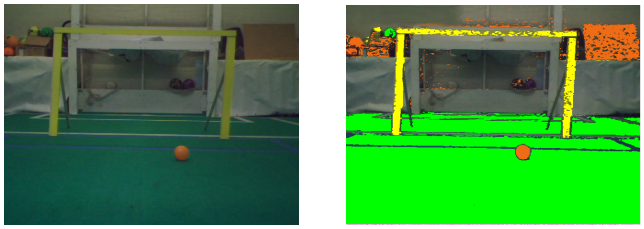
When the DCM reads new sensor values it notifies the DCMController calling the *atPostProcess* callback function. The latter writes the sensor values to the shared memory and raises a semaphore. The agent is blocked waiting for this semaphore, reads the values from the shared memory updating the world state, executes a behavior which produces actuator values to be written back to the shared memory. When the DCM is about to send values to the actuators it notifies the DCMController calling the *atPostProcess* callback function. The DCMController sends the actuator values in the shared memory to the DCM. This cycle repeats every 10 ms.

The alternative to this approach would be to create the agent as a NaoQi module instead of an independent process. The advantages of the approach we have chosen are the following:

- if the agent crashes the DCMController continues running and moves the robot to a safe position;
- it is faster to restart the agent, since the NaoQi takes a long time to restart;
- the agent only needs to know how to access the shared memory, becoming independent of the Aldebaran software.

4.1.2 Vision

The vision process is separated from the agent process. Its purpose is to process the images captured by one of the Nao cameras (the Nao robot has two cameras, however, only one can be used at a time) and give information to the agent process, namely: the position of the ball, the position of the goals, which field lines and marks are on sight, player positions and poses, among others. An example of the vision process is shown in Fig. 4.3. Fig. 4.3a shows an image captured by the robot camera. This is then processed by vision resulting in the segmented image shown in Fig. 4.3b. From this segmented image, the center of the orange circle surrounded by green is returned as the position of the ball, in number of pixels from the left upper corner of the image. This is just a simple description of the vision process. More complex steps are needed to extract information from the captured image. All information given by the vision process has coordinates in number of pixels relative to the image captured by the camera. This means they depend on the camera position and orientation. Therefore the agent must convert them to the global field coordinates. For instance, the ball position



(a) Real image (b) Segmented image

Figure 4.3: Example of the vision process. On the left is the real image captured by the robot camera. On the right is the image after color segmentation.

from the previous example must be converted to the absolute position inside the soccer field. Neither the vision process, nor the conversion of coordinates are discussed in detail in this thesis.

4.1.3 RTDB

The vision information is passed to the agent through the Real-Time Data Base (RTDB) [64, 6] (Fig. 4.4). The RTDB is a blackboard through which all robots can share information. Some of the data present in the RTDB is replicated in all robots in real-time (Fig. 4.5). This can be used not only to share information between local processes running in the same robot, but also to share information among robots. The data is sent using broadcasts, according to the producer-consumer cooperation model, and the blackboard is replicated according to the distributed shared memory model. All this is automatically done by an autonomous communication system. The RTDB is divided in two areas: a local one and a shared one. The local area has information only available to local processes and the shared area has the information broadcasted to all robots. At this point the RTDB is used only to pass information from the vision to the agent and to monitor the state of all robots. However, in the future, it can be used for sharing more information with varied purposes. The information shared among robots can be used for cooperative sensing, meaning that each robot shares its world state and complement it with others robots information. For example, if the robots share the ball position in field coordinates, a robot that does not see the ball can know where it is using the information of the other robots. The information shared by the robots through the RTDB can also be used to monitor and control them using an external application. The base station application [65] of the CAMBADA team was modified to be used with the Nao robots (Fig. 4.6). This application allows to monitor each robot, see their world state, change their configuration, send high level commands, and change their internal game state.

Figure 4.4: Architecture for communication between the agent and the vision using the Real-Time Data Base (RTDB).

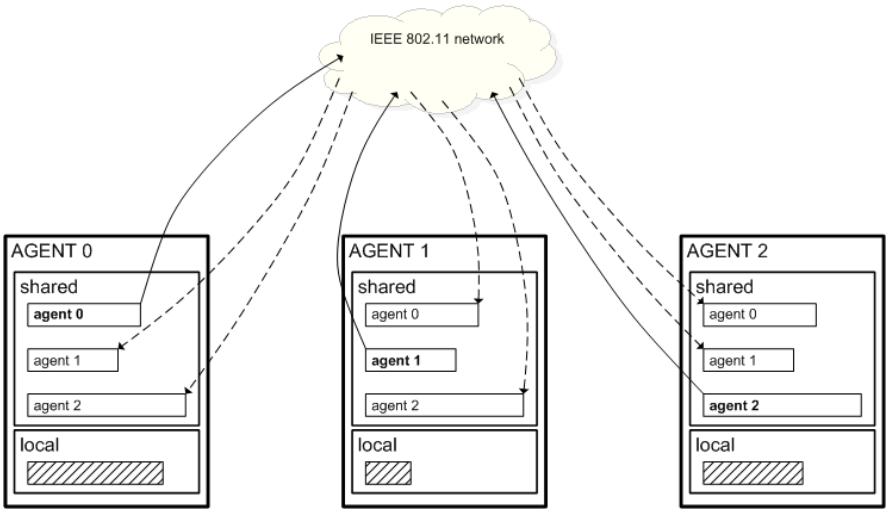


Figure 4.5: RTDB data replication among agents (adapted from [66]).

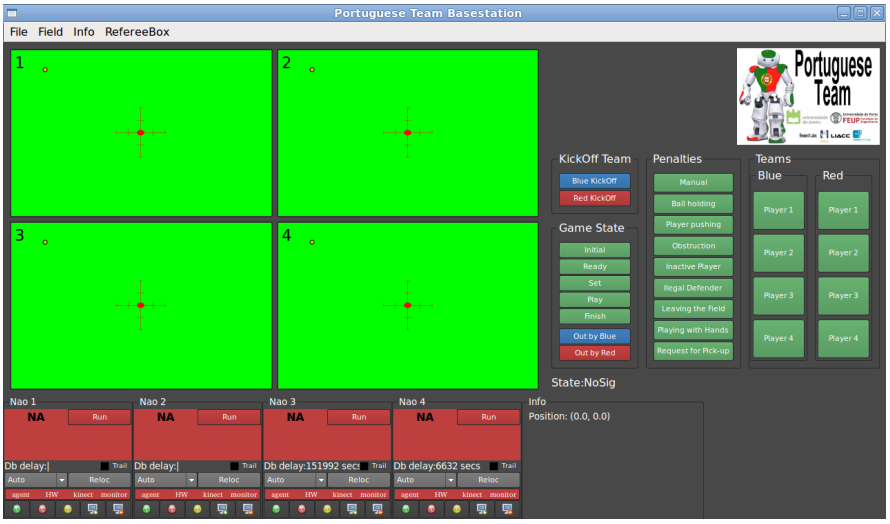


Figure 4.6: The base station application.

4.2 Agent Architecture

The architecture of the agent for the SPL is identical to that of the FC Portugal team agent in the 3D Soccer Simulation League (3DSSL). The only modifications made were in the low level communications. In the 3DSSL, the agent communicates with the server to send the actuator values and receive the sensor values, through a network connection using the Transmission Control Protocol (TCP). In the real robot, instead of communicating with a server, the agent sends the actuator values and reads the sensor values using the shared memory. The real robot has a hardware Proportional-Integral-Derivative (PID) controller for each joint as opposed to the simulated robot, whose PID controllers must be implemented by the user. Therefore, it is unnecessary to use these software PID controllers in the real robot.

Figure 4.7 shows the class diagram of the part of the agent responsible for interacting with the DCMController. All presented classes already existed in the agent of the 3DSSL except the

DCMReader. The *ServerComm*, *JointControl*, *AgentModel*, and *WorldState* classes suffered the major modifications. The agent is composed by a loop which repeats indefinitely and is synchronized with the DCM cycles. The sequence of function calls happening in the loop of the agent is shown in Fig. 4.8. The *HumanoidAgent* class starts by calling the functions *waitNextCycle* and *readSensors* of the *ServerComm* class. The former function blocks waiting for the semaphore. The latter instructs the *DCMReader* to read the sensor values from the shared memory and update the *AgentModel* and *WorldState* with those values. After this, when the agent wants to run a behavior, it calls the *execute* function of the desired behavior. The behaviors can use information from the *AgentModel* and *WorldState* and specify the desired joint targets using the *setJointTarget* function of the *JointControl* class. The *commit* function, when called, writes the joint target values to the shared memory. In the 3DSSL the *JointControl* class includes a software PID controller, however, since the real Nao has a hardware PID controller for each joint, this class was changed to simply copy the joint target values to the shared memory to be sent to the robot actuators. Since the joint angles in the real robot are in radians and in the simulation in degrees, the *DCMReader* and *JointControl* classes convert the joint's sensor and target values between these two metrics. In addition to converting the values, the following joints need to be multiplied by -1, so the behaviors developed for the simulation and for the real robot use the same convention:

- left and right shoulder pitch;
- left and right hip pitch;
- left and right knee;
- left and right ankle pitch.

This allows the code of behaviors to be compatible between both the 3DSSL and SPL.

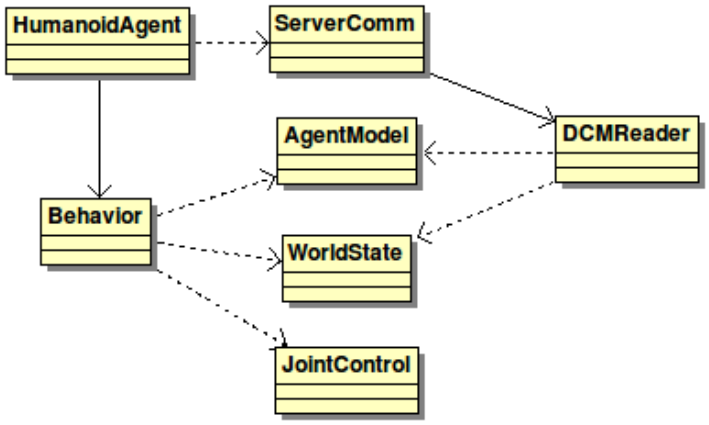


Figure 4.7: Class diagram of the part of the agent that communicates with the DCMController.

The high level behaviors can be ported from the simulated robot to the real robot with little or no modifications, as soon as the low level behaviors are developed for both the simulated and the real robot.

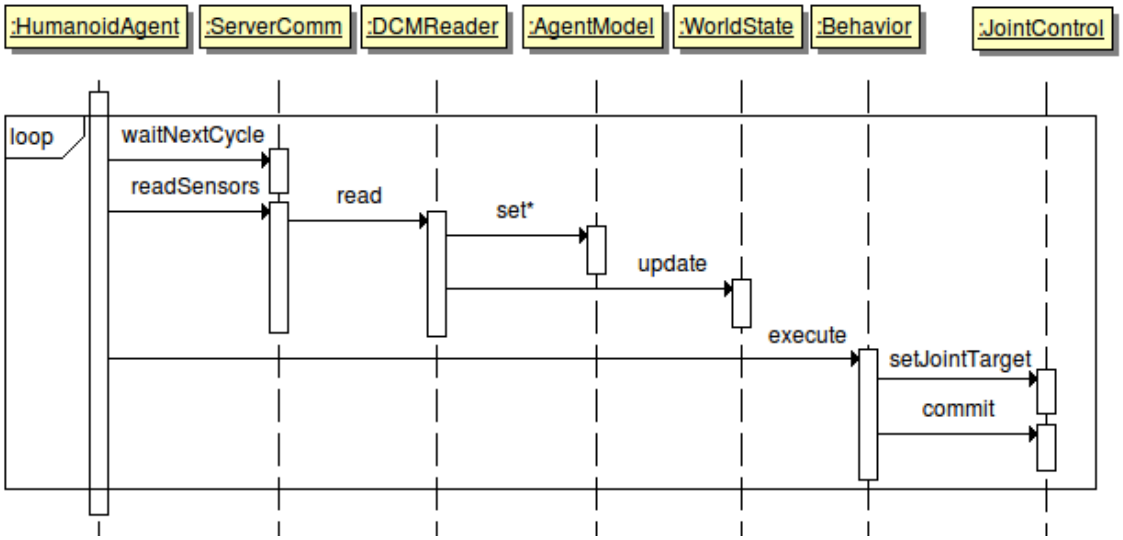


Figure 4.8: Sequence diagram of the agent loop.

4.2.1 Behaviors

All behaviors, either in the simulation agent or in the real robot agent, are a specialization of the parent class *Behavior* (Fig. 4.9). This class assures that all behaviors implement three functions:

init this function initializes the behavior's internal state before execution;

execute this function is to be called once per cycle to instruct the behavior to send new targets to the joints;

finished function that returns *true* when the behavior finished execution and the robot is in a stable position.

Apart from these, each behavior has its own functions to allow control over it and change parameters.

4.3 Conclusion

This chapter presented the architecture of the software and the architecture of the agent for the SPL. The former is composed by the DCMController, the agent, the vision, and the RTDB. All these components were presented, with more detail over the agent, its behaviors, and the communication with the robot hardware through the DCMController.

The vision is responsible for processing images acquired from the robot cameras and give information to the agent, namely: the position of the ball, the position of the goals, which field lines and marks are on sight, and player positions and poses. This information is passed to the agent through the RTDB. The RTDB is a blackboard through which all robots can share information. It can be used not only to share information between local processes running in the same robot, but also to share information among robots. The information shared by the robots through the RTDB can also be used to monitor and control them using

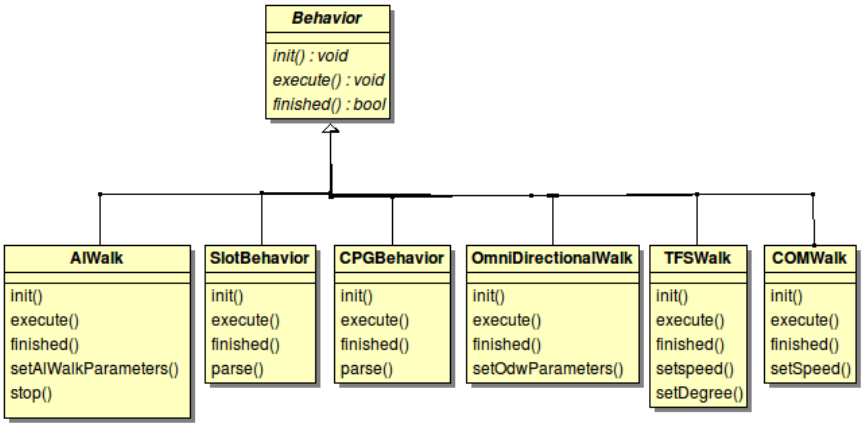


Figure 4.9: Class diagram of the behaviors.

an external application. The DCMController is a NaoQi module that allows low level access to the actuators and sensors of the Nao robot. The agent communicates with this module through a shared memory.

The architecture of the SPL agent is identical to that of the 3DSSL agent. The only modifications made were in the low level communications. The high level behaviors can be ported from the simulated robot to the real robot with little or no modifications, as soon as the low level behaviors are developed for both the simulated and the real robot.

The agent process only depends on the DCMController and the RTDB. This means that it could be executed outside the robot by creating a test environment. One needs only to develop a process that creates a shared memory equal to the one created by the DCMController and simulates the robot hardware by manipulating the sensor and actuator values. The RTDB can already be executed outside the robot, but a debug application could be developed to read and write the information contained in it.

Chapter 5

Humanoid Behaviors

Humanoid behaviors are basic human-like movements executed by the robot, namely: walk, kick the ball, get up after a fall, etc.

When developing behaviors for the real Nao robot, we tried to keep the architecture of the agent similar to that of the simulated robot. This way, the behaviors could be easily ported between the simulated and the real robot. Hence, the simulator can be used as a tool to test the behaviors before executing them in a real robot, which is damageable and expensive. Moreover, it is also faster to execute repetitive tasks in the simulator, making it more suitable for optimizing the behaviors using techniques such as machine learning. Since all behaviors on simulation produce joint angles which are then passed to a software Proportional-Integral-Derivative (PID) controller, this was replaced with software which sends these angles to the shared memory (as explained in Chapter 4), adapting them to the convention used by the real robot.

This chapter presents all behaviors developed for the real Nao robot, starting with those which were developed for the Standard Platform League (SPL) on the basis of existing behaviors from the 3D Soccer Simulation League (3DSSL). These include the Slot Behavior, which generates robot movements by specifying sequences of joint positions and the time interval between them; the CPG Behavior, which moves joints with trajectories defined by sums of sine waves; an omnidirectional walking algorithm, which has been developed on the basis of Sven Behnke's work (introduced in Section 2.2.4); and the TFSWalk algorithm, which has been optimized in the simulator and adapted to the real robot.

In the context of the omnidirectional walk, we also present the Leg Interface and its implementation for the real Nao robot, including the compensation for the inclination of Nao's hip joints.

We will then describe the COMWalk—a walking behavior based on the walk of the B-Human team in 2009—and also the walk made available by Aldebaran and how the agent can control it.

Last, we describe a middle level behavior developed to make the robot follow the ball.

5.1 Slot Behaviors

A Slot Behavior is defined by a sequence of slots. Each slot has a time duration and a target angle for each joint to be controlled. When a slot is executed the joints are moved with a sinusoidal trajectory, from the angle they have at the beginning of the slot, to the

target angle that is achieved at the end of the slot. A sinusoidal trajectory is applied because its initial and final speeds are zero and it assures the lowest second derivative maximum, hence the acceleration will be minimized [41]. This produces a smooth motion for the joints. Slot behaviors are defined in eXtensible Markup Language (XML) files, so they can be easily manipulated without the need to recompile the agent.

Listing 5.1 shows an example of the content of an XML file of a Slot Behavior. The joint target trajectories generated by this code are shown in Fig. 5.1. Each behavior has a sequence of slot elements. Each slot has a name and a duration (the attribute named delta) in seconds. Inside each slot there is a list of joint names and respective target angles, in degrees, to be achieved at the end of this slot. The joints that are not mentioned keep their values during the slot execution. In the example, the two joints were at -90 deg before the behavior was executed. When it is executed, the first slot moves the two joints to 45 and 90 deg, in 1 second. The next slot moves only one of the joints to 30 deg, in 0.5 seconds, keeping the other joint at 90 deg.

Listing 5.1: Example of the content of a XML file of a Slot Behavior.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <!DOCTYPE joints [
3  <!ENTITY larm1  "14"    >
4  <!ENTITY rarm1  "15"    >
5  ]>
6  <behavior name="Example" type="ExpediteSlotBehavior2" >
7      <slot name="slot1" delta="1" >
8          <move id="&larm1;" angle="45"  />
9          <move id="&rarm1;" angle="90"  />
10     </slot>
11     <slot name="slot2" delta="0.5" >
12         <move id="&larm1;" angle="30"  />
13     </slot>
14 </behavior>
```

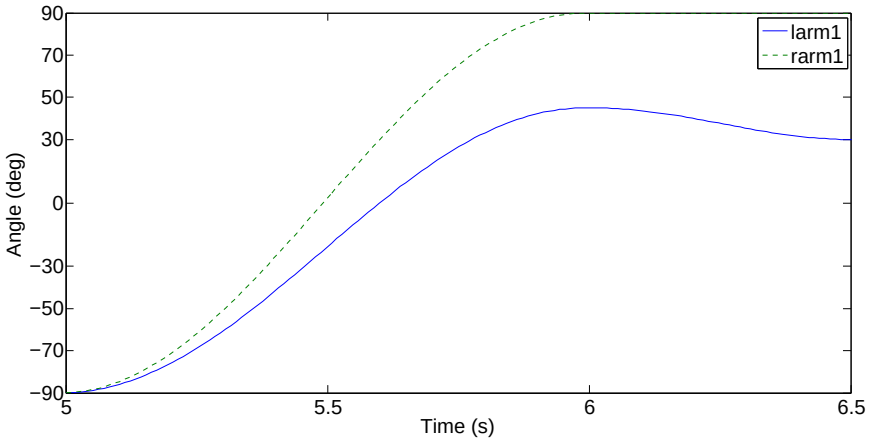


Figure 5.1: Example of the joint trajectories generated by a Slot Behavior.

The trajectory generated to move a joint from the angle at the beginning of the slot, θ_b, to the angle at the end, θ_e, in d seconds, given the time elapsed, t, since the beginning of the

slot is:

$$\theta_{\text{joint}} = -A \cdot \sin\left(t \cdot \frac{2\pi}{T} + \frac{\pi}{2}\right) + \alpha$$
$$T = 2 \cdot d$$
$$\alpha = A + \theta_{\text{b}}$$
$$A = \frac{\theta_{\text{e}} - \theta_{\text{b}}}{2}$$

Slot Behaviors were used to develop some behaviors, such as: kick the ball and get up after a fall. When switching the behavior being executed, Slot Behaviors can be used, since they move the joints from the actual position to the target one by generating a continuous and smooth trajectory.

The development of the Slot Behavior model for the real robot, based on the existing algorithm for simulation, was easy, since it only generates joint trajectories based on XML files. The behaviors themselves were more complex to adapt from the simulated to the real robot. The general approach used was to start with making the behavior from simulation compliant with the real robot restrictions and then apply the simulator behavior to the real robot with reduced velocity. Finally, we gradually increase the behavior's velocity while adjusting the parameters, so as to raise its efficiency and keep its stability.

The get up behaviors (after falling forward and backward) were developed from scratch because the ones used on the simulation execute motions that are not possible on a real robot. The sequence of poses of our get up behaviors were based on Aaron Tay's thesis [67] and on the B-Human 2010 Code Release [61]. The robot detects that it is falling when the filtered angle of the torso position relative to the vertical is bigger the defined threshold, either in the x or the y axis. When falling, the robot sets the stiffnesses of all joints to a low value. This has been considered a good approach to protect the robot since, if the joints were left with a high stiffness value they could get damaged by exerting too much force against the ground and deactivated joints tend to gain too much momentum before hitting the ground [61]. When lying on the ground, the robot detects its position, using also the angle of the torso with respect to the vertical, and executes the right get up behavior to move it to the stand up position.

The kick forward behavior was adapted from simulation and had to be changed to be stable on the real robot. Some slot durations were increased to make the behavior slower. The angles of the joints that move on the coronal plane (hip roll and ankle roll) were also increased, so the center of mass is better shifted to the support foot. A kick to the side that did not exist on the simulation was also created. When optimizing a kick, the priority is to assure stability. This means that the robot should not fall when kicking a ball. It is also important to kick fast, so the robot kicks the ball before an opponent, and return quickly to the stand position, allowing other behaviors to be executed. The kick should also be robust according to the position of the ball, hence the kick does not depend on the position of the ball. It is also desired to kick the ball with precision according to the direction and distance traveled.

5.1.1 Results

The get up behaviors, developed as Slot Behaviors, take the robot to the stand up position, in 16.5 seconds, from a face-down lying position, and in 8.5 seconds, from a face-up lying

position.

The kick forward behavior (Fig. 5.2) takes 3.2 seconds to hit the ball and then 2.2 seconds to return to the stand up position. The kick to the side behavior (Fig. 5.3) hits the ball in 3 seconds and returns to the stand position in 3.2 seconds. The distance traveled by the ball was measured in both kicks. Each kick was executed five times, always starting with the robot and the ball from the same position. The ball was placed, for both kicks, in front of the right foot with a 5 cm distance from its center to the start of the right foot, according to the robot's x axis, and a 2 cm distance from its center to the center of this foot, according to the y axis. Both kicks were executed with the right foot. For each kick, the distance from the ball final position (when the ball stopped moving) to its initial position before the kick was measured, for both the x and y axis. The kick forward behavior had an average distance of 287 cm with a standard deviation of 46 cm, in the x axis, and an average distance of 32 cm with a standard deviation of 65 cm, in the y axis. The kick to the side behavior had an average distance of 7 cm with a standard deviation of 18 cm, in the x axis, and an average distance of 105 cm with a standard deviation of 17 cm, in the y axis.

Watching videos from some of the best teams, the get up behavior from a face-up lying position takes approximately 13 seconds to execute for the TT-UT Austin Villa team[1] and 7 seconds for the Nao Team Humboldt[2] and the B-Human team[3]. The get up behavior from a face-down lying position takes about 11 seconds to execute for the B-Human team and 5 seconds for the Nao Team Humboldt. Regarding the kick forward, the Nao Team Humboldt takes approximately 3 seconds to kick the ball and more 2 seconds to return to the stand up position, the TT-UT Austin Villa can kick the ball up to 3.5 meters, and the B-Human team takes 2 seconds to kick the ball, 2 seconds to return to the stand up position, and the ball can reach up to 5.4 meters.

All these behaviors were successfully tested on the floor of the laboratory and in the SPL field and are stable.

5.2 CPG Behaviors

Central Pattern Generator (CPG) Behaviors execute, on each targeted joint, an angle trajectory defined by a sum of sine waves [39]. Each sine has four parameters: amplitude, period, phase, and offset. Like Slot Behaviors, CPG Behaviors are defined using XML files, to be easily manipulated without the need to recompile the agent. Listing 5.2 shows an example of an XML file with a forward walk developed as a CPG behavior. Lines from 26 to 41 define the pattern of each joint. Each of these lines starts with the joint name followed by any number of 4-tuples. Each 4-tuple represents a sine wave: the first element is the amplitude in degrees, the second is the period in seconds, the third is the phase in radians, and the fourth is the offset in degrees. In this example the period is defined using the *ENTITY* feature of the XML, which allows to define its value only once for all joints. In the same way, three amplitudes are defined. The trajectory generated for a joint is:

$$\theta_{\text{joint}} = \sum_i \left(a_i \cdot \sin\left(\frac{2\pi}{T_i} \cdot t + \phi_i \right) + o_i \right) \tag{5.1}$$

[1] `http://www.cs.utexas.edu/~AustinVilla/?p=nao`
[2] `http://www.naoteamhumboldt.de/category/media/videos/`
[3] `http://www.b-human.de/en/media/`

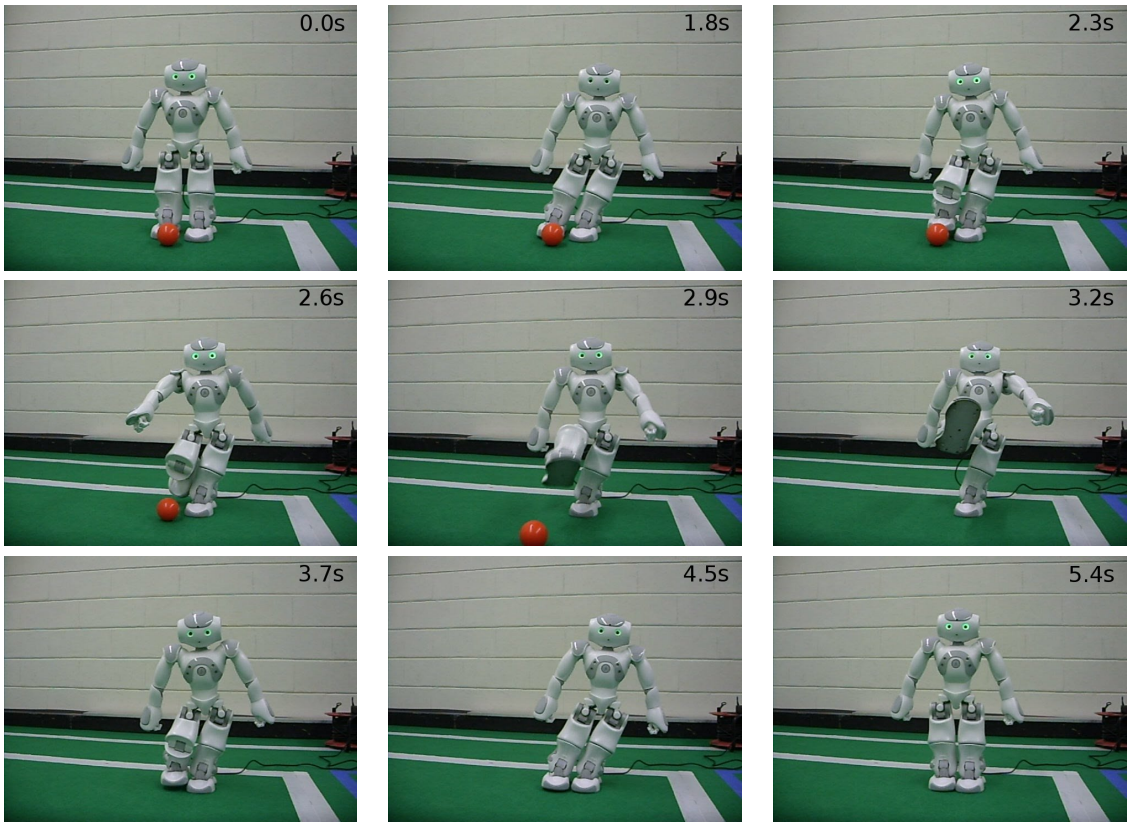


Figure 5.2: Execution of the Forward Kick Slot Behavior.

where a_i is the amplitude, T_i the period, ϕ_i the phase, and o_i the offset.

Listing 5.2: Forward walk developed using a CPG behavior.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE joints [
 <!ENTITY lleg1   "2"      >
<!ENTITY rleg1   "3"      >
<!ENTITY lleg2   "4"      >
<!ENTITY rleg2   "5"      >
<!ENTITY lleg3   "6"      >
<!ENTITY rleg3   "7"      >
<!ENTITY lleg4   "8"      >
<!ENTITY rleg4   "9"      >
<!ENTITY lleg5   "10"     >
<!ENTITY rleg5   "11"     >
<!ENTITY lleg6   "12"     >
<!ENTITY rleg6   "13"     >
<!ENTITY larm1   "14"     >
<!ENTITY rarm1   "15"     >
<!ENTITY larm2   "16"     >
<!ENTITY rarm2   "17"     >
<!ENTITY amp1 "3">
<!ENTITY amp2 "6">
<!ENTITY amp3 "0">
<!ENTITY period "0.6">
```

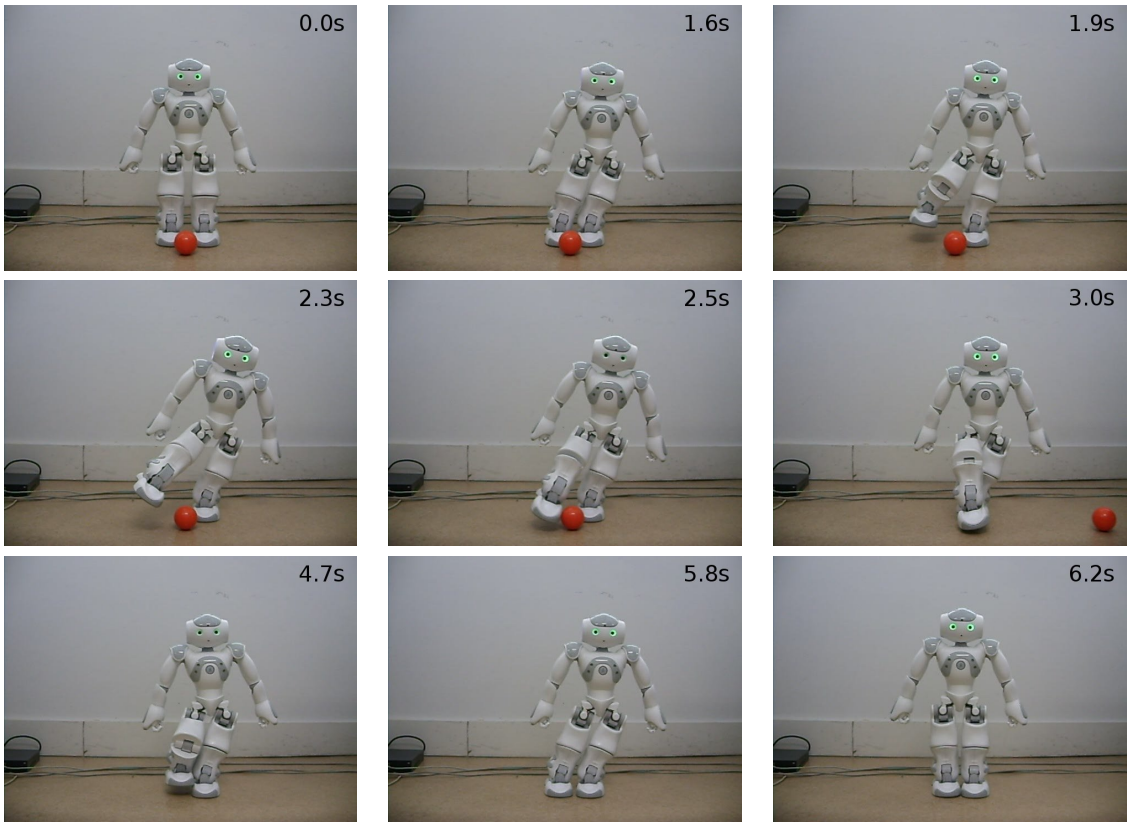


Figure 5.3: Execution of the Kick to the side Slot Behavior.

```
23 ]>
24 <behavior name="FrontWalkS2" type="CPGBehavior">
25     <patterns>
26         &larm1;: -5 &period; 1.570796327 -90;
27         &rarm1;: 5 &period; 1.570796327 -90;
28         &larm2;: 3 &period; 1.570796327 20;
29         &rarm2;: 3 &period; 1.570796327 -20;
30         &lleg1;: 0 &period; 1.570796327 0;
31         &rleg1;: 0 &period; 1.570796327 0;
32         &lleg2;: -&amp3; &period; 1.570796327 3;
33         &rleg2;: &amp3; &period; 1.570796327 -3;
34         &lleg3;: -&amp1; &period; 0 37.44, &amp1; &period; 1.570796327 0;
35         &rleg3;: &amp1; &period; 0 37.44, -&amp1; &period; 1.570796327 0;
36         &lleg4;: &amp2; &period; 0 -62.33;
37         &rleg4;: -&amp2; &period; 0 -62.33;
38         &lleg5;: -&amp1; &period; 0 31.0, -&amp1; &period; 1.570796327 0;
39         &rleg5;: &amp1; &period; 0 31.0, &amp1; &period; 1.570796327 0;
40         &lleg6;: &amp3; &period; 1.570796327 0;
41         &rleg6;: -&amp3; &period; 1.570796327 0;
42     </patterns>
43     <delta>&period;</delta>
44 </behavior>
```

The CPG Behavior model was used to create periodic behaviors, namely: walk, turn in place, rotate around the ball, etc.

In the CPG Behaviors, like the Slot Behaviors, the algorithm was easily developed for the real robot based on the existing algorithm of the simulation. However, the behaviors needed to be adapted from the simulation to the real robot. The common modification was to slowdown the behaviors and reduce the joint amplitudes. However, each behavior needed to be independently adapted. Like the Slot Behaviors, the first step was to make the simulation behavior compliant with the real robot restrictions, followed by applying it to the real robot with reduced velocity. Then, we increased the velocity, while adapting the behavior parameters, assuring the behavior does not become unstable or inefficient.

5.2.1 Results

The forward walk created as a CPG Behavior (shown in List. 5.2) was ran five times in the SPL field and the time it took to walk a meter and a half was measured. The walk is stable, since the robot never fell, and achieved an average velocity of 6 cm/s with a standard deviation of 0.4 cm/s. The trajectories of the hip pitch and knee joints are shown in Fig. 5.4 and Fig. 5.5, respectively. Another behavior developed as a CPG Behavior was the rotate around the ball behavior. Executing this behavior on the floor of the laboratory several times and measuring the time and degrees rotated, the average rotation velocity was 18 deg/s.

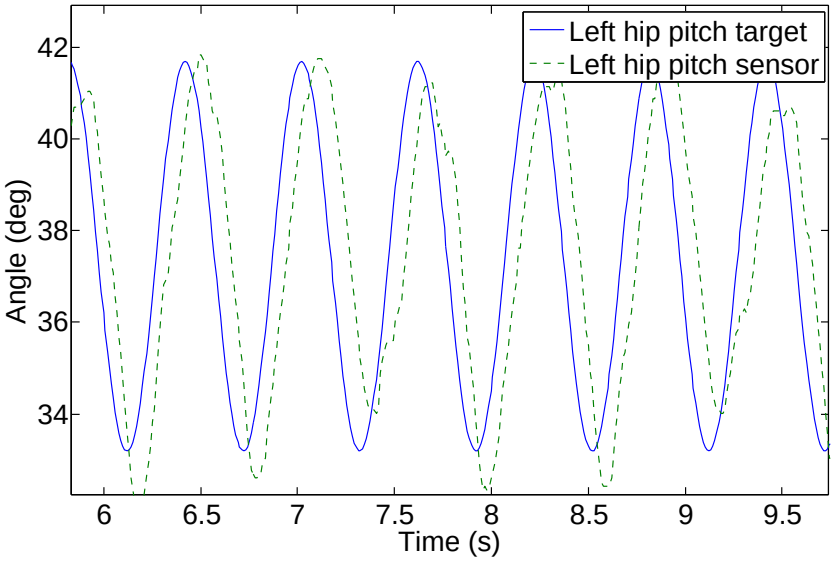


Figure 5.4: Left leg hip pitch joint target and sensor regarding the forward walk CPG Behavior.

5.3 OmnidirectionalWalk Behavior

The OmnidirectionalWalk behavior was based on Sven Behnke's work [41]. It is an open-loop omnidirectional walk developed for the humanoid robot Jupp (Section 2.3.1). This walk allows to move forwards, sideways, rotate, or a combination of these. However, it does not use sensor information to react against external disturbances. It had to be adapted to the simulated Nao, since the latter and the Jupp robot are different.

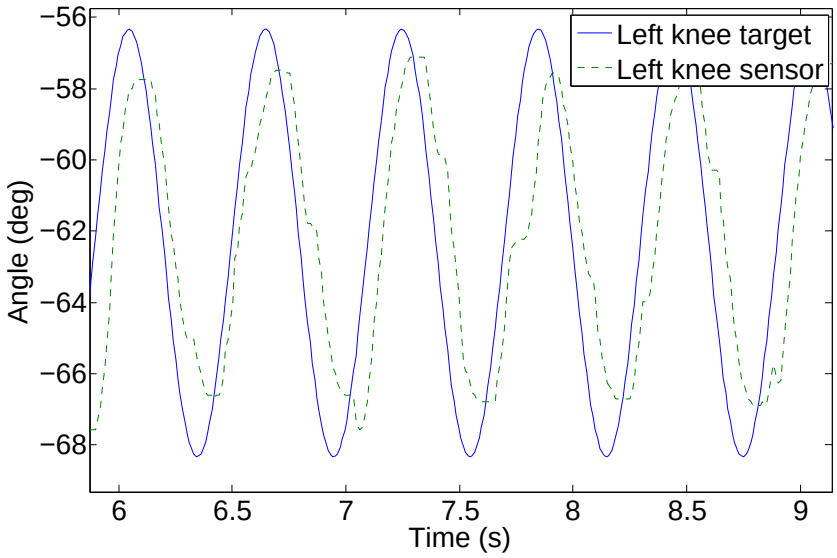


Figure 5.5: Left leg knee joint target and sensor regarding the forward walk CPG Behavior.

The OmnidirectionalWalk behavior is controlled by specifying the amplitude of the swing in three directions: forward, lateral, and rotating. To avoid abrupt movements when these parameters are changed, a linear interpolation is applied between the actual and the desired parameters.

This walking algorithm is always moving the feet up and down, even when the amplitude of the swing is zero for the three direction. This makes it harder to transition between it and a still stand up position in a smooth and stable way, either when starting or stopping walking. To solve this a Slot Behavior was created to move the robot to a stand up position with the joint angles close to the ones used by this walk. Therefore, to start walking the Slot Behavior is executed and then the OmnidirectionalWalk behavior is executed with an initial trunk phase of $\pi/2$. To stop walking, the Slot Behavior is executed after the OmnidirectionalWalk behavior to move the robot to the still stand up position.

This behavior uses a Leg Interface to control the leg movements. The Leg Interface allows to specify leg positions using six parameters:

leg angle roll, pitch, and yaw angles between the torso and the line connecting hip and ankle;

leg extension distance between the hip and the ankle;

foot angle roll and pitch angles between the foot and the perpendicular of the torso.

This behavior uses this interface to generate a stable omnidirectional walk.

The development of the OmnidirectionalWalk behavior for the real robot is explained in the next sections. First, the Leg Interface is explained and its formulae are presented. Then, the adjustments for the 45 degrees rotation of the Nao hip yaw-pitch joints are described. Finally, the fact that the two hip yaw-pitch joints are controlled by a common servomotor, as opposed to simulation, is addressed.

5.3.1 Leg Interface

The Leg Interface (Fig. 5.6) calculates the angles of the six leg joints using: leg angle, $\theta_{\text{Leg}} = \left(\theta^{\text{r}}_{\text{Leg}}, \theta^{\text{p}}_{\text{Leg}}, \theta^{\text{y}}_{\text{Leg}}\right)$; foot angle, $\theta_{\text{Foot}} = \left(\theta^{\text{r}}_{\text{Foot}}, \theta^{\text{p}}_{\text{Foot}}\right)$; and leg extension, γ. The leg angle, $\theta_{\text{Leg}} = \left(\theta^{\text{r}}_{\text{Leg}}, \theta^{\text{p}}_{\text{Leg}}, \theta^{\text{y}}_{\text{Leg}}\right)$, is the angle between the torso and the line connecting hip and ankle. $\theta^{\text{r}}_{\text{Leg}} = 0$ when the leg is parallel to the trunk, on the coronal plane, and $\theta^{\text{r}}_{\text{Leg}} > 0$ when the leg is moved outwards to the side. $\theta^{\text{p}}_{\text{Leg}} = 0$ when the leg is parallel to the trunk, in the sagittal plane, and $\theta^{\text{p}}_{\text{Leg}} > 0$ when the leg is moved to the front. $\theta^{\text{y}}_{\text{Leg}} = 0$ when the foot points to the front and $\theta^{\text{y}}_{\text{Leg}} > 0$ when the leg is rotated vertically pointing the foot outward. The foot angle, $\theta_{\text{Foot}} = \left(\theta^{\text{r}}_{\text{Foot}}, \theta^{\text{p}}_{\text{Foot}}\right)$, controls the foot angle relative to the torso. If $\theta_{\text{Foot}} = (0, 0)$ then the foot base is perpendicular to the torso. Hence, if the torso is perpendicular to the ground, the foot base is parallel to the ground. The leg extension, $-1 \leq \gamma \leq 0$, denotes that the leg is fully extended when $\gamma = 0$ and is fully shortened when $\gamma = -1$. When shortened, the leg is $\eta_{\min}$ of its fully extended length. The relative leg length can be calculated as $\eta = 1 + (1 - \eta_{\min})\gamma$.

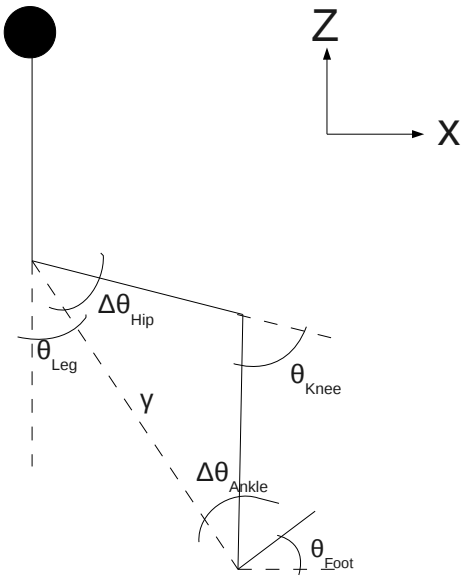


Figure 5.6: Leg Interface parameters and calculated angles, in the sagittal plane.

The Jupp robot's hip joints have the following order, from up to down: roll, pitch and yaw. In the Nao robot the hip joints have a different order: yaw-pitch, roll, pitch. This means that we cannot use all the formulae presented in Sven Behnke's paper [41]. Thus we have developed our own formulae. There is also the problem of not having a pure vertical yaw joint. Instead, the Nao robot has a inclined yaw-pitch joint. The solution for this problem is presented in Section 5.3.2.

From the leg angle, foot angle, and leg extension, the Leg Interface (Fig. 5.6) calculates the angles of the six leg joints, as described next. Starting with the leg extension γ, we calculate

the relative leg length as $\eta = 1 + (1 - \eta_{\min})\gamma$ which is then multiplied by the length of the fully extended leg to get the desired leg length, $l = \eta(l_{\text{upperLeg}} + l_{\text{lowerLeg}})$, where l_{upperLeg} and l_{lowerLeg} are the lengths of the thigh and shank, respectively. The law of cosines (5.2) relates the lengths, a, b, and c, of the sides of a triangle with the angle, θ, opposite to the side of length c. Using this law, the knee joint angle can be calculated by (5.3). It is subtracted by π because it is the outside angle, contrary to the one used in the law of cosines which is the inside angle. θ_{Knee} is 0 when the leg is stretched and negative when the leg is retracted. The law of cosines is also used to calculate the angle variation of the hip (5.4) and ankle (5.5) to compensate the knee angle. Therefore, when the leg extension changes, θ_{Leg} and θ_{Foot} are not changed.

$$c^2 = a^2 + b^2 - 2 \cdot a \cdot b \cdot \cos(\theta) \tag{5.2}$$

$$\theta_{\text{Knee}} = \arccos\left(\frac{l_{\text{upperLeg}}^2 + l_{\text{lowerLeg}}^2 - l^2}{2 \cdot l_{\text{upperLeg}} \cdot l_{\text{lowerLeg}}}\right) - \pi \tag{5.3}$$

$$\Delta\theta_{\text{Hip}}^{\text{p}} = \arccos\left(\frac{l_{\text{upperLeg}}^2 + l^2 - l_{\text{lowerLeg}}^2}{2 \cdot l_{\text{upperLeg}} \cdot l}\right) \tag{5.4}$$

$$\Delta\theta_{\text{Ankle}}^{\text{p}} = \arccos\left(\frac{l_{\text{lowerLeg}}^2 + l^2 - l_{\text{upperLeg}}^2}{2 \cdot l_{\text{lowerLeg}} \cdot l}\right) \tag{5.5}$$

The remaining leg joint angles can be calculated using equations (5.6), where ls (leg side) is -1 for the left leg and 1 for the right leg. The hip angles are the θ_{Leg} angles with the compensation $\Delta\theta_{\text{Hip}}^{\text{p}}$. The ankle angles are the differences between θ_{Foot} angles and the θ_{Leg} angles with the compensation $\Delta\theta_{\text{Ankle}}^{\text{p}}$.

$$\begin{aligned}
\theta_{\text{HipYawPitch}} &= \theta_{\text{Leg}}^{\text{y}} \\
\theta_{\text{HipRoll}} &= -ls \cdot \theta_{\text{Leg}}^{\text{r}} \\
\theta_{\text{HipPitch}} &= \theta_{\text{Leg}}^{\text{p}} + \Delta\theta_{\text{Hip}}^{\text{p}} \\
\theta_{\text{AnklePitch}} &= \theta_{\text{Foot}}^{\text{p}} - \theta_{\text{Leg}}^{\text{p}} + \Delta\theta_{\text{Ankle}}^{\text{p}} \\
\theta_{\text{AnkleRoll}} &= -ls \cdot \left(\theta_{\text{Foot}}^{\text{r}} - \theta_{\text{Leg}}^{\text{r}}\right)
\end{aligned} \tag{5.6}$$

Note that in (5.6) the leg vertical rotation is applied directly in the hip yaw-pitch joint. This is just a rough approximation, since it only works for small values. The Nao robot, unlike Jupp, has no vertical hip joint. It has only one joint rotated 45 degrees that can be used to obtain vertical rotation of the leg. When we applied the leg yaw rotation directly to the hip yaw-pitch joint, the robot was stable but the torso oscillated forwards and backwards. To correct this oscillation the previous formulae were corrected considering the 45 degrees rotation of the hip yaw-pitch joint, as explained in the next section.

5.3.2 Inclination of the Hip Yaw-Pitch Joint

The Nao robot does not have a vertical hip joint, as opposed to most humanoid robots. Instead, it has the hip yaw-pitch joint which is rotated 45 degrees from the vertical axis. To

correct the formulae of the previous section, and eliminate the referred oscillation, we used a formula that, given the desired yaw hip rotation, calculates the correct angles to apply on the three hip joints of the Nao robot.

We start by calculating the rotation matrix of the thigh in relation to the hip, aligned with the hip yaw-pitch joint axis, using (5.7).

$$\begin{aligned} Rot_{\text{Hip}} &= Rot_{\text{x}}\left(ls \cdot \frac{\pi}{4}\right) \cdot Rot_{\text{z}}(\theta^{\text{y}}_{\text{Leg}}) \\ &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{\sqrt{2}}{2} & -ls \cdot \frac{\sqrt{2}}{2} \\ 0 & ls \cdot \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{pmatrix} \cdot \begin{pmatrix} \cos(\theta^{\text{y}}_{\text{Leg}}) & -\sin(\theta^{\text{y}}_{\text{Leg}}) & 0 \\ \sin(\theta^{\text{y}}_{\text{Leg}}) & \cos(\theta^{\text{y}}_{\text{Leg}}) & 0 \\ 0 & 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} \cos(\theta^{\text{y}}_{\text{Leg}}) & -\sin(\theta^{\text{y}}_{\text{Leg}}) & 0 \\ \frac{\sqrt{2}}{2} \cdot \sin(\theta^{\text{y}}_{\text{Leg}}) & \frac{\sqrt{2}}{2} \cdot \cos(\theta^{\text{y}}_{\text{Leg}}) & -ls \cdot \frac{\sqrt{2}}{2} \\ ls \cdot \frac{\sqrt{2}}{2} \cdot \sin(\theta^{\text{y}}_{\text{Leg}}) & ls \cdot \frac{\sqrt{2}}{2} \cdot \cos(\theta^{\text{y}}_{\text{Leg}}) & \frac{\sqrt{2}}{2} \end{pmatrix} \end{aligned} \tag{5.7}$$

Then, part of the B-Human inverse kinematics [68] can be used to calculate the angles of the three hip joints that produce the desired rotation. First, the rotation matrix produced by the three hip joints is constructed (the matrix is abbreviated, e.g. c_{x} means $\cos(\delta_{\text{x}})$) (5.8).

$$\begin{aligned} Rot_{\text{Hip}} &= Rot_{\text{z}}(\delta_{\text{z}}) \cdot Rot_{\text{x}}(\delta_{\text{x}}) \cdot Rot_{y}(\delta_{\text{y}}) \\ &= \begin{pmatrix} c_{\text{x}}c_{\text{z}} - s_{\text{x}}s_{\text{y}}s_{\text{z}} & -c_{\text{x}}s_{\text{z}} & c_{\text{z}}s_{\text{y}} + c_{\text{y}}s_{\text{x}}s_{\text{z}} \\ c_{\text{z}}s_{\text{x}}s_{\text{y}} + c_{\text{y}}s_{\text{z}} & c_{\text{x}}c_{\text{z}} & -c_{\text{y}}c_{\text{z}}s_{\text{x}} + s_{\text{y}}s_{\text{z}} \\ -c_{\text{x}}s_{\text{y}} & s_{\text{x}} & c_{\text{x}}c_{\text{y}} \end{pmatrix} \end{aligned} \tag{5.8}$$

It is now clear that the angle of the hip roll (x axis) can be calculated using s_{x}, as shown in (5.9). This angle has to be rotated 45 degrees, because the hip roll joint space is rotated according to the hip yaw-pitch axis.

$$\delta_{\text{x}} = \arcsin(s_{\text{x}}) - ls \cdot \frac{\pi}{4} = \arcsin(ls \cdot \frac{\sqrt{2}}{2} \cdot \cos(\theta^{\text{y}}_{\text{Leg}})) - ls \cdot \frac{\pi}{4} \tag{5.9}$$

Calculating each of the other two hip joints requires 2 matrix entries. Equation (5.10) shows how we can obtain the rotation along the z axis by combining 2 entries of the rotation matrix. With this, the remaining two hip joints can be calculated using (5.11) and (5.12).

$$\frac{c_{\text{x}} \cdot s_{\text{z}}}{c_{\text{x}} \cdot c_{\text{z}}} = \frac{\cos(\delta_{\text{x}}) \cdot \sin(\delta_{\text{z}})}{\cos(\delta_{\text{x}}) \cdot \cos(\delta_{\text{z}})} = \frac{\sin(\delta_{\text{z}})}{\cos(\delta_{\text{z}})} = \tan(\delta_{\text{z}}) \tag{5.10}$$

$$\delta_{\text{z}} = \text{atan2}(c_{\text{x}} \cdot s_{\text{z}}, c_{\text{x}} \cdot c_{\text{z}}) = \text{atan2}(\sin(\theta^{\text{y}}_{\text{Leg}}), \frac{\sqrt{2}}{2} \cdot \cos(\theta^{\text{y}}_{\text{Leg}})) \tag{5.11}$$

$$\delta_{\text{y}} = \text{atan2}(c_{\text{x}} \cdot s_{\text{y}}, c_{\text{x}} \cdot c_{\text{y}}) = \text{atan2}(-ls \cdot \frac{\sqrt{2}}{2} \cdot \sin(\theta^{\text{y}}_{\text{Leg}}), \frac{\sqrt{2}}{2}) \tag{5.12}$$

These three angles (δ_{x}, δ_{y} and δ_{z}) are added to the calculated joint angles of (5.6) producing the final angles for the three hip joints:

$$\begin{aligned} \theta_{\mathrm{HipYawPitch}} &= \delta_{\mathrm{z}} \\ \theta_{\mathrm{HipRoll}} &= -ls \cdot \theta^{\mathrm{r}}_{\mathrm{Leg}} + \delta_{\mathrm{x}} \\ \theta_{\mathrm{HipPitch}} &= \theta^{\mathrm{p}}_{\mathrm{Leg}} + \Delta\theta^{\mathrm{p}}_{\mathrm{Hip}} - \delta_{\mathrm{y}} \end{aligned} \tag{5.13}$$

This method can be used whenever a Nao robot's leg must rotate around its vertical axis. After this procedure the differences between the Nao robot and other humanoid robots are less significant. Most humanoid robots have a joint to rotate each leg vertically, but the Nao robot does not. This means that code developed for a generic humanoid robot could not be applied to the Nao robot. With this algorithm, we obtain a virtual joint on the hip that rotates around the vertical axis, permitting to easily adapt code from other humanoid robots to this one.

5.3.3 Common Hip Yaw-Pitch Joints

Another difference from the Nao robot to most humanoid robots (like the Jupp robot for which Sven Behnke's walk [41] was developed) is that the hip yaw-pitch joints in both legs are controlled by a single motor. This means that any rotation is applied to both joints at the same time. In the 3DSSL the two joints can be controlled independently. So, when adapting this walk from simulation to the real robot, we need to combine both motions into one that produces the desired stable gait. When, in the simulation, the two motions are equal, we can apply them directly to the real robot. When they are different, the task is more difficult and each specific case must be analyzed.

For this walk the desired yaw motion of the legs is shown in Fig. 5.7. Each leg motion is composed by two kinds of movements: a sinusoidal movement, when the leg is in the air, and a linear movement, when it is on the ground. As presented in Fig. 5.7, the two motions are significantly different and cannot be directly executed by the common joint of the real Nao robot. We needed to combine the two motions creating one that is a continuous function (so the joint can follow it smoothly) and that produces a stable walk. Our choice was to follow a linear-like motion which results from selecting the motion of the leg that is on the ground (Fig. 5.8). To do this we switch between left and right leg yaw trajectory followed in the intersection of the two motions when both legs follow linear movements. The reason for this choice is that the motion of the leg on the ground (the support leg) is slower than that of the leg in the air and because the leg on the ground has all body weight over it, therefore the motion of this leg has more impact over the stability of the walk than the leg in the air.

5.3.4 Results

The OmnidirectionalWalk was tested on the floor of the laboratory and on a carpet similar to the one used in the SPL field. The parameters of this walk were optimized to make it faster without loosing stability. Tables 5.1 and 5.2 show the parameters that produce a stable forward walk and the achieved velocity. The a_{Shift} is the shifting amplitude of the robot's Center of Mass (CoM), used in the Sven Behnke's formulae (Section. 2.2.4). To make this walk more stable, the equation used to calculate this amplitude was replaced by

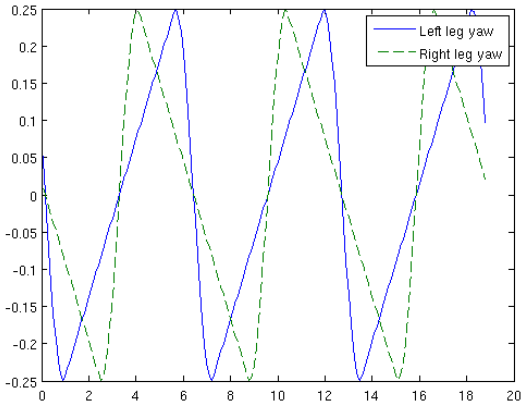


Figure 5.7: Leg yaw motions in the OmnidirectionalWalk.

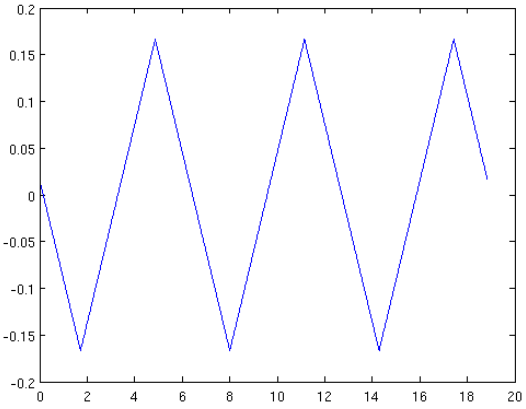


Figure 5.8: Common joint motion resulting from the combination of the two motions in Fig. 5.7.

a constant value. Walking to the side achieved a velocity of 1 cm/s, on both floors, using a step frequency of 0.32 steps/s, a swing amplitude of 0.05 rad and a a_{Shift} amplitude of 0.21 rad. The parameters that produce a stable rotation movement and the corresponding velocities are shown in Table 5.3. It was only possible to achieve velocities higher than 4 deg/s after the Leg Interface was changed to compensate the inclination of the hip yaw-pitch joints, as explained in Section 5.3.2. Before this inclination was compensated, the torso oscillated forwards and backwards becoming unstable at higher velocities. This compensation allowed to achieve a rotation velocity of 35 deg/s.

Table 5.1: Results regarding forward walking on the laboratory floor.

Step Frequency (steps/s)	Swing Amplitude (rad)	a_{Shift} (rad)	Velocity (cm/s)
0.32	0.2	0.20	3
0.48	0.2	0.18	5

Table 5.2: Results regarding forward walking on the carpet.

Step Frequency (steps/s)	Swing Amplitude (rad)	a_{Shift} (rad)	Velocity (cm/s)
0.32	0.2	0.22	3
0.40	0.2	0.20	4
0.40	0.25	0.20	5

This implementation of the OmnidirectionalWalk was also tested in the simulator and compared to the existing version, since the Leg Interface had not been correctly adapted to the simulated Nao. In the existing version of the OmnidirectionalWalk, used in the 3DSSL, there was no compensation for the inclination of the hip yaw-pitch joint, hence the desired vertical rotation was directly applied to this joint. The behavior was executed for 30 seconds, five times with the existing version and five times with the new version. The average velocities

Table 5.3: Results regarding the OmnidirectionalWalk rotation.

Step Frequency (steps/s)	Swing Amplitude (rad)	Velocity (deg/s)
0.32	0.2	4
0.40	0.2	10
0.80	0.4	35

are shown in Table 5.4. Similarly to the real robot, the compensation of the inclination of the hip yaw-pitch joint reduced the torso oscillation therefore allowing higher rotation velocities.

Table 5.4: Results regarding the OmnidirectionalWalk rotation in simulation.

Step Frequency (steps/s)	Swing Amplitude (rad)	Velocity (deg/s)	
		Existing Version	New Version
2.39	0.3	12	15
2.39	0.5	unstable	23

5.4 TFSWalk Behavior

The TFSWalk gait combines two approaches: a gait where the joints trajectory is generated using Partial Fourier Series (PFS) optimized with Genetic Algorithms (GA) [39]; and a gait using Truncated Fourier Series (TFS), which imitates human walk, to generate the joint angles optimized with Particle Swarm Optimization (PSO) [40]. The TFSWalk allows the robot to walk forward and backward (either straight or turning) and to turn in place.

When walking forward, this gait uses the first approach—PFS optimized with GA. In this approach (as presented in [39]), a sinusoidal trajectory is applied to the following joints from both sides of the robot: shoulder pitch, hip roll, hip pitch, knee, ankle pitch, and ankle roll. Each of these sinusoids has four parameters: amplitude, period, phase, and offset. This results in a total of 48 parameters. However, assuming sagittal symmetry, the same trajectories are used for both sides with a phase shift of π, reducing the number of parameters by half. Moreover, using the same period for all sinusoids, to keep them synchronized, the number of parameters is reduced to 19. Reducing the number of parameters has the advantage of reducing the search space of the optimization problem and allowing for a faster convergence

of the algorithm. The equations that produce the joint trajectories, for the left side, are:

$$
\begin{aligned}
f_{\text{lShoulder}}(t) &= A_1 \cdot \sin\left(\frac{2\pi}{T} \cdot t + \phi_1\right) + O_1 \\
f_{\text{lHipRoll}}(t) &= A_2 \cdot \sin\left(\frac{2\pi}{T} \cdot t + \phi_2\right) + O_2 \\
f_{\text{lHipPitch}}(t) &= A_3 \cdot \sin\left(\frac{2\pi}{T} \cdot t + \phi_3\right) + O_3 \\
f_{\text{lKnee}}(t) &= A_3 \cdot \sin\left(\frac{2\pi}{T} \cdot t + \phi_4\right) + O_4 \\
f_{\text{lAnklePitch}}(t) &= A_4 \cdot \sin\left(\frac{2\pi}{T} \cdot t + \phi_5\right) + O_5 \\
f_{\text{lAnkleRoll}}(t) &= A_5 \cdot \sin\left(\frac{2\pi}{T} \cdot t + \phi_6\right) + O_6
\end{aligned}
$$

where A_i are amplitudes, ϕ_i are phases, O_i are offsets, and T is the period. For the roll joints of the right side, the trajectories are equal to the ones of the left side. The pitch joints of the right side can be obtained by adding a phase shift of π to the trajectories of the left side.

The unknown 19 parameters form the genome used by the genetic algorithm. In each test, the robot is placed in a fixed position away from the ball and it must minimize the distance to the latter in a limited time. At the end of the test, the fitness function is calculated as the sum of the final distance to the ball with the average oscillation of the torso. When executed, the genetic algorithm tries to find the set of 19 parameters that produce the minimum value of the fitness function.

To apply the resulting gait to the real robot, some modifications were needed. The arms collided with the legs, therefore they were opened to the side by changing the values of the shoulder roll joints. Moreover, the torso was too much inclined forward, hence the offset of the sinusoidal applied to both hip pitch joints was reduced, to make the torso more vertical. These modifications were enough to have a stable gait on the floor of the laboratory. However, the floor of the SPL field has more friction and therefore the walk needed further modifications to be stable. These modifications are explained through this section.

For walking backward and turning in place, the TFS approach is used (as explained in [40]). The trajectories of the hip and knee joints are generated on the basis of an approximate imitation of the human walk. The trajectories of the ankle joints are calculated on the basis of the hip and knee trajectories to keep the foot parallel to the ground and thus avoid stumbling. The trajectory applied to the hip is a combination of the following two functions:

$$
\begin{aligned}
\theta_{\text{h}}^{+} &= A \cdot \sin\left(\frac{2\pi}{T} \cdot t\right) + C_{\text{h}} \\
\theta_{\text{h}}^{-} &= B \cdot \sin\left(\frac{2\pi}{T} \cdot t\right) + C_{\text{h}}
\end{aligned}
$$

where T is the period, A and B are amplitudes, C_{h} is the offset, θ_{h}^{+} is the upper portion of the trajectory, and θ_{h}^{-} is the lower portion. This means that the trajectory followed by the hip joint is equal to θ_{h}^{+} when this value is greater than C_{h} and it is equal to θ_{h}^{-} otherwise.

The knee joint follows a trajectory formed by two functions also:

$$\theta_{\mathrm{k}}^{+} = C \cdot \sin\left(\frac{2\pi}{T} \cdot t\right) + C_{\mathrm{k}}$$
$$\theta_{\mathrm{k}}^{-} = C_{\mathrm{k}}$$

where C is the amplitude and C_{k} is the offset. When a leg is in the stance-phase, supporting the body weight, it follows the trajectory defined by θ_{k}^{-}, keeping a constant angle. When it is in the swing-phase, moving in the walking direction, it follows the trajectory defined by θ_{k}^{+}. According to [69], by analyzing the relation between the hip and the knee trajectories in the human walk, it is conclusive that the stance-phase starts when the hip angle reaches its maximum value and it ends when the minimum value is reached. This means that when the leg is moved forward the knee must follow θ_{k}^{+} and when it is moved backward the knee must be equal to θ_{k}^{-}. The trajectories followed by both legs are identical but have a phase shift of half-period. Hence, the number of parameters to define the leg movement is six: T, A, B, C, C_{h}, and C_{k}. Each arm is moved by applying a sinusoidal trajectory to the shoulder pitch joint, to move the arm in the same direction as the opposite leg. This sine has the same period as the one of the legs. The amplitude is the only unknown parameter. Besides these parameters, there are two more: one to control the acceleration when starting to walk; and other to control the coronal movement. The PSO algorithm was used to optimized these nine parameters. In each test the robot walks for 15 seconds. At the end, the fitness function is calculated as the distance traveled along the x axis, thus inciting the robot to walk fast and straight.

As explained, this gait was optimized in the simulator, resulting in the fastest gait we have for the 3DSSL. However, the ground on the simulator has low friction compared to the official SPL field, and thus the gait learned to use this as an advantage, not lifting the feet too much and almost not using coronal movement. When adapting this behavior to the real robot, it only worked on slippery ground. On the official SPL field, the carpet has too much friction, making the robot stumble and fall.

To solve this problem we tried two different approaches. The first approach was to add coronal movement, allowing the robot to better shift its CoM and resulting in a more stable gait. The coronal movement used is shown in Fig. 5.9 and was based on [40]. It consists of rotating the hip roll joint of the support leg (the one on the ground). This lifts the other leg (the swinging leg) from the ground. The ankle roll joint of the swinging leg is rotated with the same angle as the support leg hip, so the foot is always parallel to the ground.

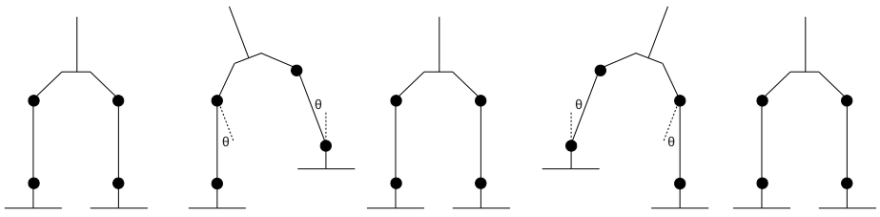


Figure 5.9: Coronal movement.

The hip motion is defined using (5.14) and produces the trajectories shown in Fig. 5.10. The left and right hips have a phase shift of π. θ is the amplitude and T the period. The latter is the same for the sagittal and the coronal movements. On the other hand, the

coronal movement has a phase shift of $\pi/2$ relative to the sagittal movement. The amplitude is empirically adjusted and depends on the walking speed. When the latter increases, the coronal movement amplitude decreases.

$$f(t) = \begin{cases} \theta \sin\left(\frac{2\pi}{T}t\right) & \text{, if } t < \frac{T}{2} \\ 0 & \text{, otherwise} \end{cases} \tag{5.14}$$

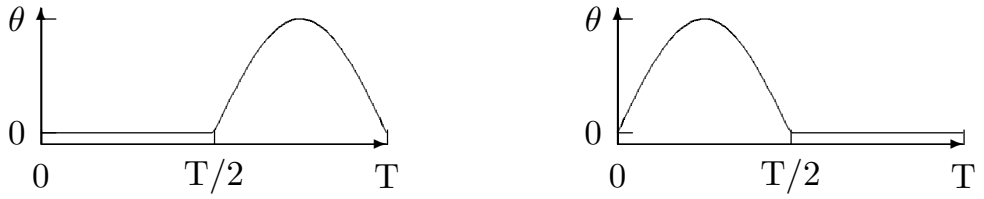


Figure 5.10: Left and right hip roll joints trajectories.

Instead of using coronal movement to shift the CoM, the second approach we took was based on lifting the feet higher and more rapidly, making use of the robot dynamics to keep it balanced. When the robot has only one leg on the ground, it will fall towards the other leg, having the dynamics of a single inverted pendulum. To assure dynamic stability, the leg on the ground must change at the right moment. In order to increase the height of the foot trajectories, the knee, hip and ankle joint trajectories should be changed in a coordinated way. However, the specification of TFSWalk, by defining each joint trajectory independently of the others, does not provide a good model for controlling the foot trajectory. To achieve a more controllable model, the TFSWalk specification was converted to use the Leg Interface (Section 5.3.1). To accomplish this, the equations of the Leg Interface, presented in Section 5.3.1, were solved in the inverse order. Taking the angles of the six joints of the leg as input the, values of the six parameters of the Leg Interface were obtained. This resulted in one sinusoidal trajectory for each parameter of the Leg Interface (Fig. 5.11, 5.13, and 5.14). The trajectories of both legs are identical. The only difference is that the trajectories of the left leg have a phase shift of π relative to the right leg. The four roll parameters ($\theta^{\mathrm{r}}_{\mathrm{Leg}}$ and $\theta^{\mathrm{r}}_{\mathrm{Foot}}$ for each leg) have a constant value of zero. The leg angle yaw, $\theta^{\mathrm{y}}_{\mathrm{Leg}}$, for both legs, is zero when walking forward and follows the trajectory shown in Fig. 5.15 when turning. The difference between turning left or right is a phase shift of π. With this new model, it is easy to control the foot height trajectory just changing the leg extension parameter. In addition, the velocity of the robot, controlled by the leg angle amplitude, becomes independent from the foot trajectory height. As an example, the feet may be kept moving up and down in the same place by setting the leg pitch angle amplitude to zero while keeping the normal value of the leg extension. This up and down movement is very useful to initiate and finish the walking behavior. The increase in the foot height trajectory diminished foot collisions with the ground and resulted in a stable walk without needing coronal movement.

The Leg Interface also allows control over the height of the robot CoM by changing the offset of the leg extension parameter. Reducing the offset of the leg extension makes the robot to bend the knees more, lowering the CoM. The inverse is also true, increasing the offset makes the CoM higher. A lower CoM leads to a more stable walk, but also slower and the bent knees consume more energy. It is a trade-off involving stability, speed, and energy efficiency. The trajectory of the leg extension parameter after applying the previous

improvements is shown in Fig. 5.12. The resulting trajectory of the leg extension parameter has a higher amplitude, a lower offset and no acceleration/deceleration phase.

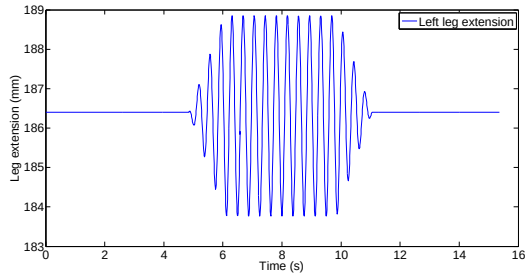


Figure 5.11: Left leg extension parameter.

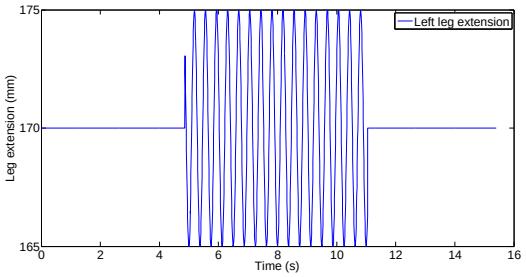


Figure 5.12: Left leg extension parameter after increasing the amplitude, reducing the offset, and removing the acceleration and deceleration.

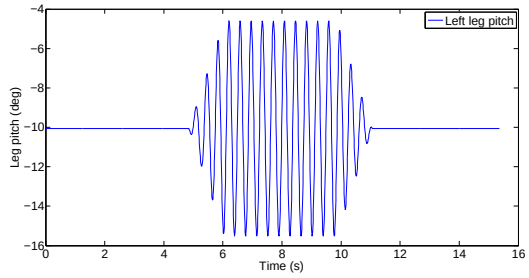


Figure 5.13: Left leg angle pitch parameter.

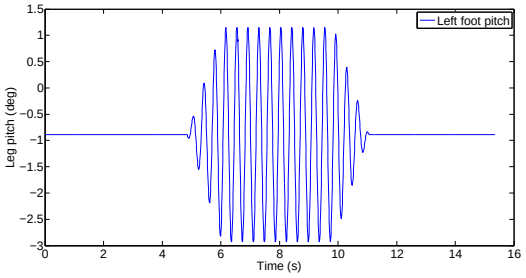


Figure 5.14: Left foot angle pitch parameter.

The use of the Leg Interface in the TFSWalk also compensated the fact that the hip yaw-pitch joint is rotated 45 degrees relative to the vertical. Before using the Leg Interface, the TFSWalk, when turning, applied a sinusoidal trajectory directly to the hip yaw-pitch joint (Fig. 5.15). This is the desired movement around the vertical axis. However, since this joint is rotated 45 degrees from the vertical axis, it results in a slight oscillation of the torso forwards and backwards. Using the Leg Interface, the sinusoidal movement around the vertical axis, needed when turning, is converted into the joint angles that produce the desired movement. This way, the hip yaw-pitch joint inclination is compensated, resulting in the elimination of the previous oscillation.

5.4.1 Feedback

Feedback was added to make the behavior more robust against external disturbances, such as uneven ground and collisions with obstacles. Without feedback the walk is open-loop. It generates the trajectories for the joints according only to its internal state and does not use sensors to verify if its objectives are achieved. This kind of system cannot compensate for disturbances. The walk with feedback is a closed-loop system. In this system, sensors are used to determine the error between the desired and current output, which is then used to

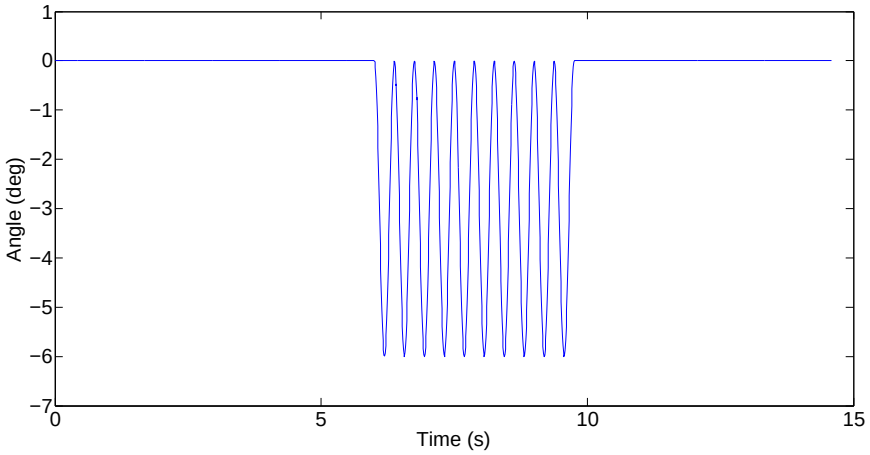


Figure 5.15: Trajectory of the leg vertical rotation used for turning.

compensate against disturbances. Different feedback methods were tested. It is possible to use different sensors for the feedback, including the accelerometer, the gyroscope, and the angle of the torso. The output of the feedback system can be used to balance the walk by changing the parameters of the walk or by changing the angle of some joints, namely those in the hip, the ankle, and the shoulder (using the arms to balance).

One successfully applied feedback method was based on the rUNSWift Team Report 2010 [62] and uses the accelerometer value to balance the walk using the hip pitch joints. This feedback is calculated using a filtered value of the accelerometer in the x direction, as shown in (5.15).

$$filAccelX = a \cdot filAccelX + (1 - a) \cdot accelX \cdot b \tag{5.15}$$

$accelX$ is the value of the accelerometer in the x direction. $filAccelX$ is the filtered value of the accelerometer that is added to the hip pitch joints. a is 0.9 and b is 0.2. These coefficients were obtained empirically making the robot walk with different parameters and keeping the parameters of the best run. The filtered accelerometer value is added to the two hip pitch joints, to balance the torso. This way, when the robot is falling to the front, the accelerometer will have a positive value that is added to the hip pitch joints, making the legs to move forward, to compensate. The inverse is also true, when the robot is falling backwards.

Another feedback method tried was to replace, in the previous one, the value of the accelerometer with the angle of the torso, relative to the vertical, around the y axis. We have replaced the value of the accelerometer with the angle of the torso because the latter value does not oscillate so much, has less noise, and because the accelerometer is affected by the gravitational acceleration. This resulted in smaller oscillations of the torso during the walk. This feedback is calculated using (5.16).

$$\begin{aligned} filAngleY &= a \cdot filAngleY + (1 - a) \cdot angleY \cdot b \\ balance &= filAngleY \cdot c + d \end{aligned} \tag{5.16}$$

$angleY$ is the angle of the torso around the y axis and $filAngleY$ is its filtered value. $balance$ is the feedback value added to the hip pitch joints. a is 0.9, b is 0.2, c is 50, and d is -1. These values were also obtained empirically while trying to make the average result of this

feedback system close to the previous one. So, although they respond in a different way to the disturbances, they both give similar angles to balance the torso. To achieve this values we did an experience with each feedback systems. The robot was kept still in the stand up position for a certain time, then it was picked up and inclined forward, and then it was inclined backward. The filtered values from both feedback systems are shown in Fig. 5.16 and Fig. 5.17. To make the filtered value of the angle of the torso close to that of the accelerometer, it was multiplied by 50 and then subtracted by 1, these are the coefficients c and d, respectively. Fig. 5.18 shows an example of the output of both feedback systems while the robot is walking. As one can see, the feedback using the angle of the torso has less noise and smaller oscillations than the one using the accelerometer.

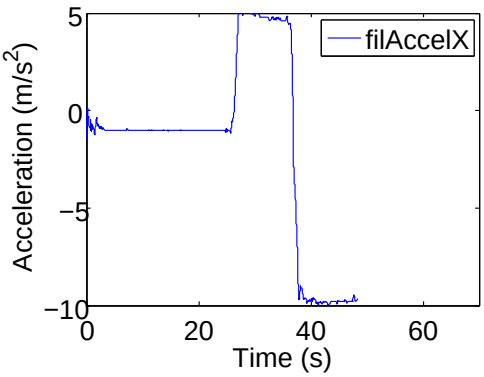


Figure 5.16: Filtered value of the accelerometer.

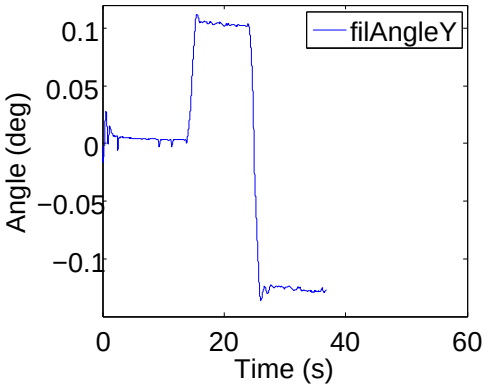


Figure 5.17: Filtered value of the angle of the torso.

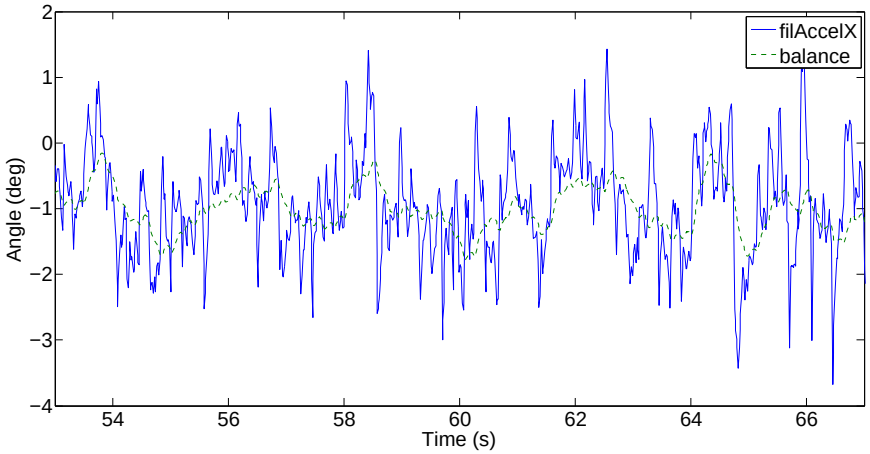


Figure 5.18: Example of the output of both feedback systems.

5.4.2 Results

To test the effectiveness of the feedback, the walk was executed with and without feedback. The feedback system used was the one based on the angle of the torso. Each configuration

(with and without feedback) was executed five times. We made the robot walk forward always from the same place of the SPL field, to assure equal conditions for the tests. This place is a flat and levelled area of the field. The robot was manually picked up to a vertical position whenever it fell. Without feedback, the robot fell a total of 12 times during the five runs. With feedback the robot never fell.

The CoM height was lowered by reducing the leg extension offset from 186.4 mm to 170 mm. Since a lower value means a slower walk, the value chosen must be the highest value that assures a stable walk. This value was obtained empirically by making the robot walk using different values. These experiments were carried out in two areas with different ground conditions. Both places were covered with a carpet similar to that in the SPL competitions but one was flat and levelled, whereas the other one had slight slopes and bumps. The values of the leg extension offset tested include 186.4 mm, 170 mm and 160 mm. For each value the walk was executed five times and the number of times the robot fell was counted. In the flat area, the robot never fell, as expected. However, in the irregular area, the robot only fell when using the original offset value of 186.4 mm. This proves that a lower CoM makes the walk more robust against external disturbances.

To measure the velocity, the robot was put to walk for five seconds, in the flat part of the SPL field, and the distance traveled was measured. This was executed for the same three values of the leg extension offset as the previous experience (186.4 mm, 170 mm, and 160 mm) and was repeated five times for each one of these values. The results are shown in Table 5.5. From this table is possible to see that a higher offset—which corresponds to a higher CoM—yields a higher velocity. However, the previous experience showed that a lower offset is more stable. Taking these two experiences in consideration, the offset value of 170 mm was chosen since it produces the fastest stable walk with an average velocity of 16 cm/s. The trajectories generated and followed by the joints, when walking forward, are shown in Figures 5.19, 5.20, 5.21, and 5.22.

Table 5.5: TFSWalk velocities for different leg extension offsets.

Offset (mm)	Average Velocity (cm/s)	Standard Deviation (cm/s)
160	15	0.2
170	16	0.2
186.4	17	0.5

The turn in place of the TFSWalk was executed in a SPL-like field and the time it took to do a number of turns was measured. This process was repeated several times and the average velocity calculated was 25 deg/s.

5.5 COMWalk Behavior

This behavior was developed based on the walk of the B-Human team in 2009 [68, 70]. It generates the trajectories for the position of the feet relative to the CoM assuming the torso is always vertical. This walk is a repetition of phases. In each phase the variable p_{phase} goes from 0 to 1 (excluding). Each phase is composed by two half-phases, each having a different support leg.

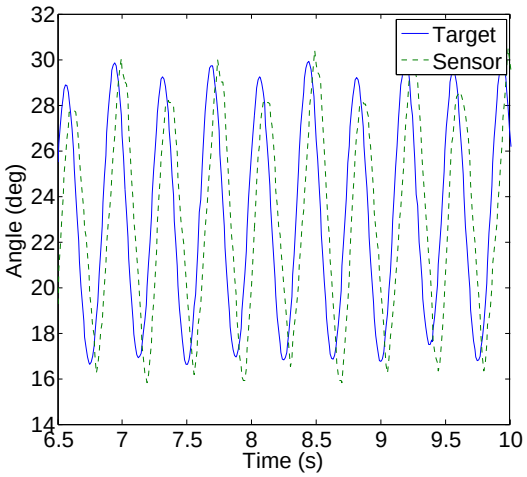


Figure 5.19: Hip pitch joint of the left leg regarding the TFSWalk behavior.

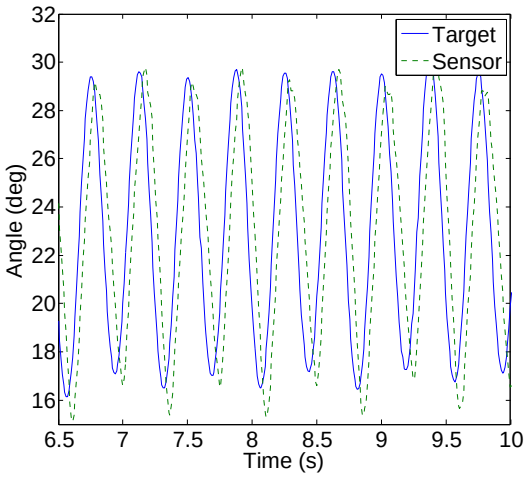


Figure 5.20: Hip pitch joint of the right leg regarding the TFSWalk behavior.

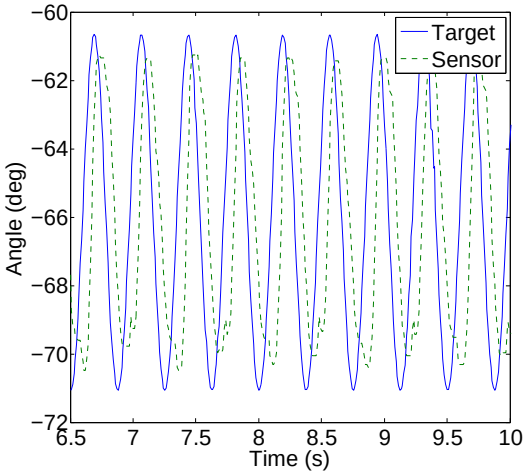


Figure 5.21: Knee joint of the left leg regarding the TFSWalk behavior.

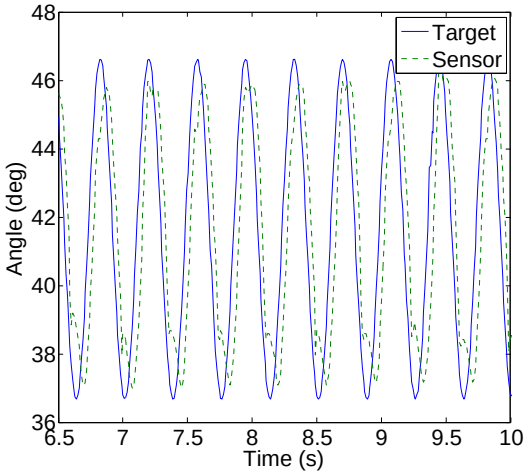


Figure 5.22: Ankle pitch joint of the left leg regarding the TFSWalk behavior.

The position of the feet in the walking direction are calculated as follows:

$$p_{\mathrm{lRel}} = \begin{cases} o_{\mathrm{l}} + s_{\mathrm{lLift}} \cdot t_{\mathrm{lLift}} + s_{\mathrm{l}} \cdot t_{\mathrm{lMove}} - s_{\mathrm{r}} \cdot (1 - t_{\mathrm{lMove}}) \cdot t_{\mathrm{l}} & \text{if } p_{\mathrm{phase}} < 0.5 \\ o_{\mathrm{l}} + s_{\mathrm{lLift}} \cdot t_{\mathrm{lLift}} + s_{\mathrm{l}} \cdot (1 - t_{\mathrm{r}}) & \text{if } p_{\mathrm{phase}} \geq 0.5 \end{cases} \tag{5.17}$$

$$p_{\mathrm{rRel}} = \begin{cases} o_{\mathrm{r}} + s_{\mathrm{rLift}} \cdot t_{\mathrm{rLift}} + s_{\mathrm{r}} \cdot t_{\mathrm{rMove}} - s_{\mathrm{l}} \cdot (1 - t_{\mathrm{rMove}}) \cdot t_{\mathrm{r}} & \text{if } p_{\mathrm{phase}} \geq 0.5 \\ o_{\mathrm{r}} + s_{\mathrm{rLift}} \cdot t_{\mathrm{rLift}} + s_{\mathrm{r}} \cdot (1 - t_{\mathrm{l}}) & \text{if } p_{\mathrm{phase}} < 0.5 \end{cases} \tag{5.18}$$

p_{lRel} and p_{rRel} are 3 dimensional vectors with the position of the feet relative to the torso in millimeters. o_{l} and o_{r} are the feet origin positions. s_{lLift} and s_{rLift} are the total offsets used for lifting the legs. s_{l} and s_{r} are the step sizes. t_{lLift} and t_{rLift} are the trajectories for

lifting the feet. These trajectories have two parameters: the moment, according to the phase, of the beginning of the lift (x_l) and its duration (y_l) (Fig. 5.23). They are calculated using equation (5.19). t_lMove and t_rMove are used to move the foot in the walking direction, as calculated by (5.20). They also have two parameters: the moment, according to the phase, of the beginning of the move (x_m) and its duration (y_m) (Fig. 5.24). t_l and t_r are used to subtract the step sizes so the body moves in the walking direction. They are calculated with equation (5.21) producing the result shown in Fig. 5.25.

$$t_\text{lLift} = \begin{cases} \frac{1-\cos((2\cdot p_\text{phase}-x_\text{l})/y_\text{l}\cdot 2\pi)}{2} & \text{if } 2\cdot p_\text{phase} > x_l \wedge 2\cdot p_\text{phase} < x_\text{l}+y_\text{l} \\ 0 & \text{otherwise} \end{cases} \tag{5.19}$$

$$t_\text{rLift} = \begin{cases} \frac{1-\cos((2\cdot p_\text{phase}-1-x_\text{l})/y_\text{l}\cdot 2\pi)}{2} & \text{if } 2\cdot p_\text{phase}-1 > x_\text{l} \wedge 2\cdot p_\text{phase}-1 < x_\text{l}+y_\text{l} \\ 0 & \text{otherwise} \end{cases}$$

$$t_\text{lMove} = \begin{cases} \frac{1-\cos((2\cdot p_\text{phase}-x_\text{m})/y_\text{m}\cdot \pi)}{2} & \text{if } 2\cdot p_\text{phase} > x_\text{m} \wedge 2\cdot p_\text{phase} < x_\text{m}+y_\text{m} \\ 1 & \text{if } 2\cdot p_\text{phase} \geq x_\text{m}+y_\text{m} \wedge 2\cdot p_\text{phase} < 1 \\ 0 & \text{otherwise} \end{cases} \tag{5.20}$$

$$t_\text{rMove} = \begin{cases} \frac{1-\cos((2\cdot p_\text{phase}-1-x_\text{m})/y_\text{m}\cdot \pi)}{2} & \text{if } 2\cdot p_\text{phase}-1 > x_\text{m} \wedge 2\cdot p_\text{phase}-1 < x_\text{m}+y_\text{m} \\ 1 & \text{if } 2\cdot p_\text{phase}-1 \geq x_\text{m}+y_\text{m} \wedge 2\cdot p_\text{phase}-1 < 1 \\ 0 & \text{otherwise} \end{cases}$$

$$t_\text{l} = \begin{cases} \frac{1-\cos(2\cdot p_\text{phase}\cdot \pi)}{2} & \text{if } p_\text{phase} < 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{5.21}$$

$$t_\text{r} = \begin{cases} \frac{1-\cos((2\cdot p_\text{phase}-1)\cdot \pi)}{2} & \text{if } p_\text{phase} < 0.5 \\ 0 & \text{otherwise} \end{cases}$$

The feet movement presented so far moves the CoM in the walking direction only. However,

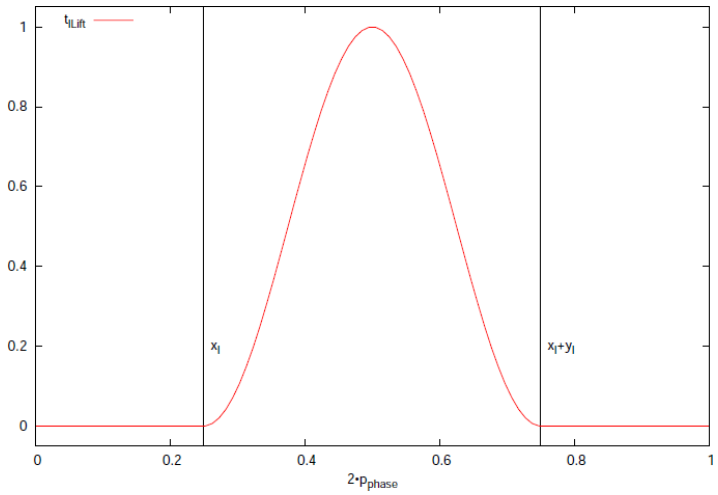


Figure 5.23: The t_lLift trajectory (adapted from [70]).

for the walk to be feasible, the CoM must be moved in the coronal plane also. To achieve

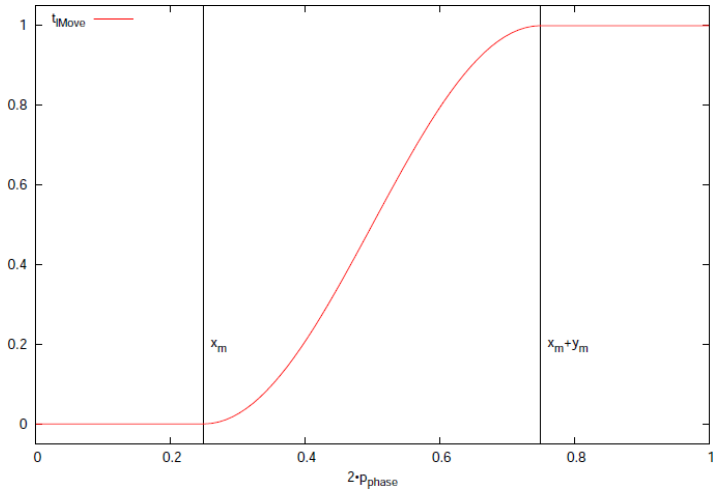


Figure 5.24: The t_{lMove} trajectory (adapted from [70]).

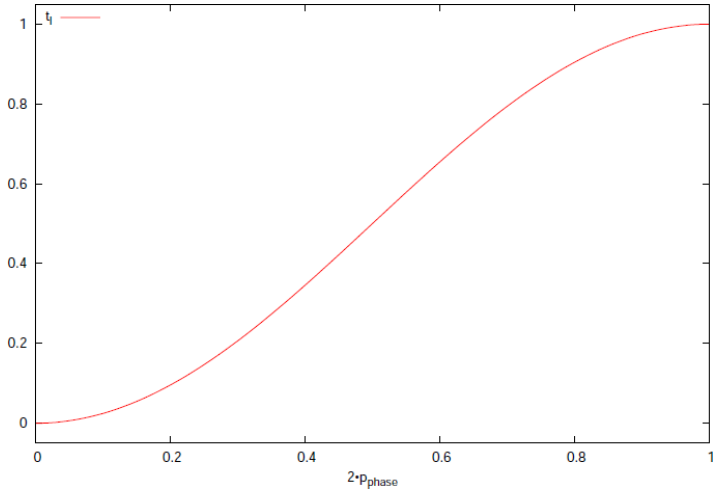


Figure 5.25: The t_l trajectory (adapted from [70]).

this, the position of the feet relative to the CoM are calculated as follows:

$$p_{\text{lCom}} = \begin{cases} -c_{\text{S}} + s_{\text{s}} \cdot t_{\text{com}} - \frac{s_{\text{l}} \cdot t_{\text{lin}} - s_{\text{r}} \cdot (1 - t_{\text{lin}})}{2} + o_{\text{l}} & \text{if } p_{\text{phase}} \geq 0.5 \\ p_{\text{rCom}} - p_{\text{rRel}} + p_{\text{lRel}} & \text{otherwise} \end{cases} \tag{5.22}$$

$$p_{\text{rCom}} = \begin{cases} -c_{\text{S}} + s_{\text{s}} \cdot t_{\text{com}} - \frac{s_{\text{r}} \cdot t_{\text{lin}} - s_{\text{l}} \cdot (1 - t_{\text{lin}})}{2} + o_{\text{r}} & \text{if } p_{\text{phase}} < 0.5 \\ p_{\text{lCom}} - p_{\text{lRel}} + p_{\text{rRel}} & \text{otherwise} \end{cases} \tag{5.23}$$

p_{lCom} and p_{rCom} are 3 dimensional vectors with the position of the feet relative to the CoM, in millimeters. c_{s} is the position of the CoM relative to the torso when the robot is in the standing position. s_{s} is the amplitude of the CoM movement along the coronal plane. t_{com} is the trajectory of the CoM movement. This trajectory is calculated by combining a sine, a square root of sine, and a linear function (Fig. 5.26):

$$t_{\text{com}} = \frac{x_{\text{c}} \cdot s(p_{\text{phase}}) + y_{\text{c}} \cdot r(p_{\text{phase}}) + z_{\text{c}} \cdot l(p_{\text{phase}})}{x_{\text{c}} + y_{\text{c}} + z_{\text{c}}} \tag{5.24}$$

$$s(p) = sin(2\pi \cdot p) \tag{5.25}$$

$$r(p) = \sqrt{|\sin(2\pi \cdot p)|} \cdot \text{sgn}(\sin(2\pi \cdot p)) \tag{5.26}$$

$$l(p) = \begin{cases} 4 \cdot p & \text{if } p < 0.25 \\ 2 - 4 \cdot p & \text{if } p \geq 0.25 \wedge p < 0.75 \\ 4 \cdot p - 4 \text{if } p \geq 0.75 \end{cases} \tag{5.27}$$

t_{lin} is a linear trajectory to move the CoM in the walking direction:

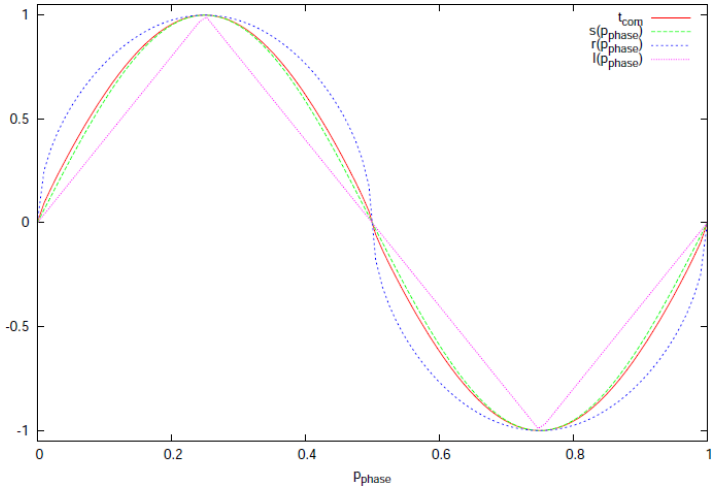


Figure 5.26: The t_{com} trajectory (adapted from [70]).

$$t_{\text{lin}} = \begin{cases} 2 \cdot p_{\text{phase}} & \text{if } p_{\text{phase}} < 0.5 \\ 2 \cdot p_{\text{phase}} - 1 & \text{if } p_{\text{phase}} \geq 0.5 \end{cases} \tag{5.28}$$

The joint angles for the legs are calculated from the position of the feet with coordinates relative to the torso using inverse kinematics. These positions are calculated adding the offset o_{new} to p_{lRel} and p_{rRel}. This offset is calculated by using the offset calculated in the previous iteration (o_{old}). This is only possible assuming that the CoM does not change much between successive iterations. Having the position of the feet added to the offset of the previous iteration ($p_{\text{lRel}} + o_{\text{old}}$ and $p_{\text{rRel}} + o_{\text{old}}$) and the position of all body parts, the position of the feet relative to the CoM is calculated. The difference between these positions and the desired positions (p_{lCom} and p_{rCom}) is added to the previous offset to get the new one. Finally, this new offset is added again to p_{lRel} and p_{rRel} and the result is converted to joint angles using inverse kinematics. The inverse kinematic model used is the one presented also by the B-Human team [68].

This walk has different balancing methods to resist against external disturbances. The filtered angle of the torso relative to the vertical is subtracted from the expected angle to obtain the rotation error. The latter is used to balance the robot by rotating the body, the feet, and the arms. The rotation error is also used to measure the error between the desired feet positions (p_{lCom} and p_{rCom}) and the same positions rotated according to the rotation error. This error allows to balance the robot by changing the step size and by adding an

offset to the desired positions of the feet. The last balance technique is to change the phase of the walk. To do this, the difference between the current angle of the torso, in the x axis, and the expected angle is used to calculate the measured phase position. The difference between the measured and the current phase position is used for calculating an offset that is added to the current phase position.

For this walk to be fully operational, the agent must be able to calculate the CoM position of the real Nao robot. However, at this point, the agent has only the model of the simulated robot, thus it can only calculate the CoM position of the latter.

5.5.1 Results

The COMWalk behavior when walking forward with a step duration of 0.9 seconds and a step amplitude of 15 mm produced a stable but slow walk, on the floor of the laboratory, with the trajectories shown in Figures from 5.27 to 5.31. Unfortunately, we could not measure its velocity, because at the time we were to measure it we only had one robot and it had a broken joint, thus it could not execute this behavior correctly.

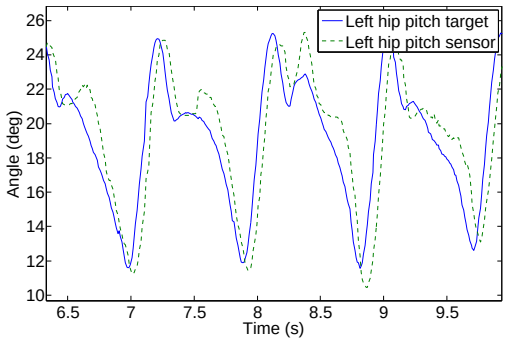


Figure 5.27: Left hip pitch joint target and sensor regarding the COMWalk behavior while walking forward.

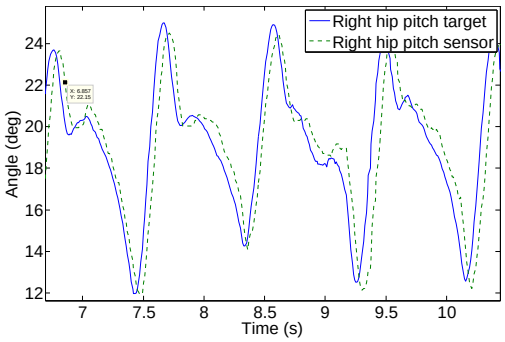


Figure 5.28: Right hip pitch joint target and sensor regarding the COMWalk behavior while walking forward.

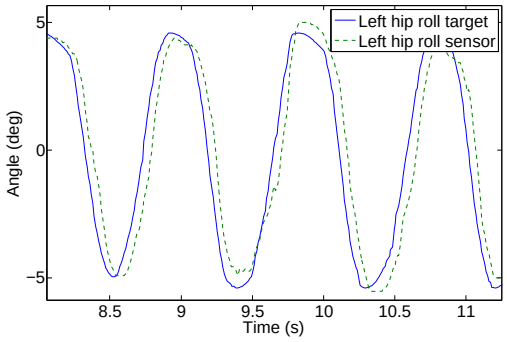


Figure 5.29: Left hip roll joint target and sensor regarding the COMWalk behavior while walking forward.

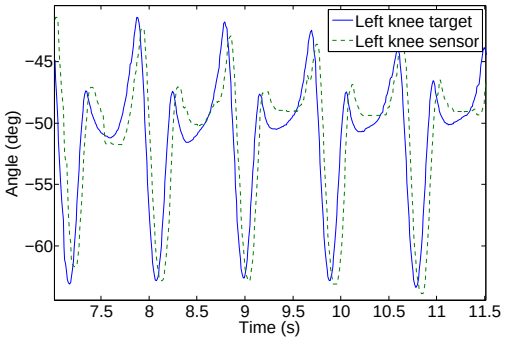


Figure 5.30: Left knee joint target and sensor regarding the COMWalk behavior while walking forward.

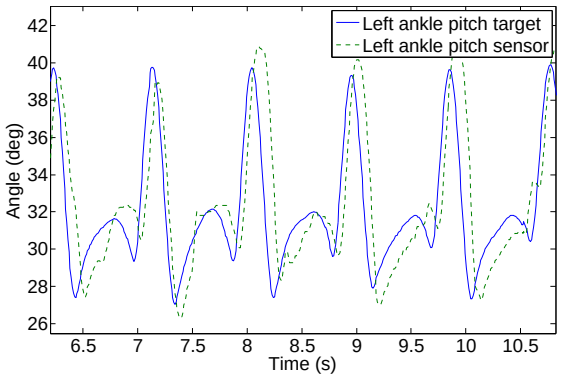


Figure 5.31: Left ankle pitch joint target and sensor regarding the COMWalk behavior while walking forward.

5.6 Aldebaran Walk

The Nao robot is already shipped with a closed-loop omnidirectional walk algorithm [71], which is executed by the ALMotion module provided by Aldebaran. Although not the fastest developed walk for the Nao robot, it is stable and freely available with the robot. Many teams use it on competitions because either they do not have their own walk algorithm, or they cannot adapt their algorithms to be stable in the field condition (which in some competitions is not the desired flat levelled ground). We decided to allow the agent to control the walk of Aldebaran so if the other walking algorithms cannot be optimized to be stable in the field, in the short time available before the competition, this walk can be used. Unfortunately this algorithm will only be open-source in the end of 2011 and, for now, it has only a few parameters to control. This means there are not many options to optimize it.

The walk of Aldebaran is based on the work of S. Kajita *et al.* [72] using Quadratic Programming [73]. It uses the theory of the preview controller and the 3D linear inverted pendulum mode to generate the trajectory of the walk. The outline of the algorithm is shown in Fig. 5.32. The preview controller used to guarantee stability uses a preview time of 0.8 seconds. This means the walk will take this time to react to new commands.

Aldebaran provides two Application Programming Interfaces (APIs) to control the walk. One is what they call the *Velocity Control*, which allows control over the velocity of the walk through four parameters:

x [−1.0 to 1.0] specifies the length of a step along the x axis as a fraction of maxStepX;

y [−1.0 to 1.0] specifies the length of a step along the y axis as a fraction of maxStepY;

theta [−1.0 to 1.0] specifies the angle between the feet as a fraction of maxStepTheta. A positive value results in a left turn and a negative value results in a right turn;

frequency [0.0 to 1.0] specifies the step frequency as a fraction of linear interpolation between minimumStepFrequency and maximumStepFrequency.

These four parameters are used by the foot step planner, as shown in Fig. 5.33, to generate the positions for walking.

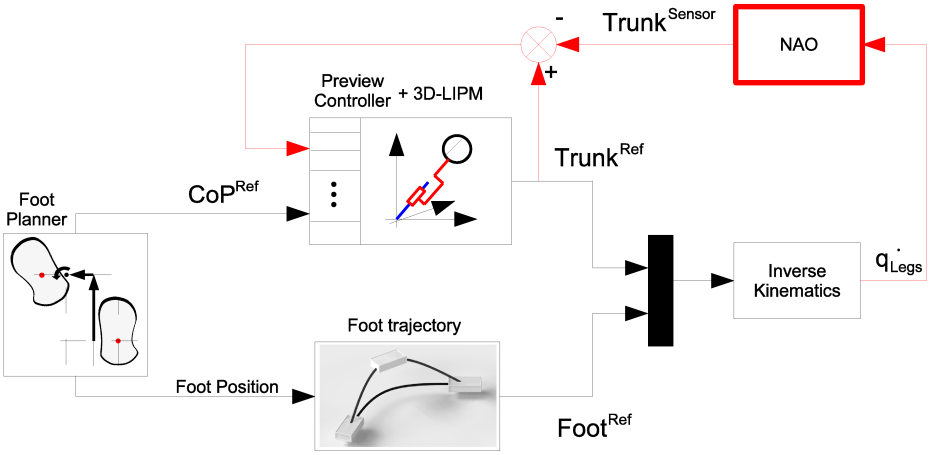


Figure 5.32: Outline of the walk of Aldebaran (adapted from [71]).

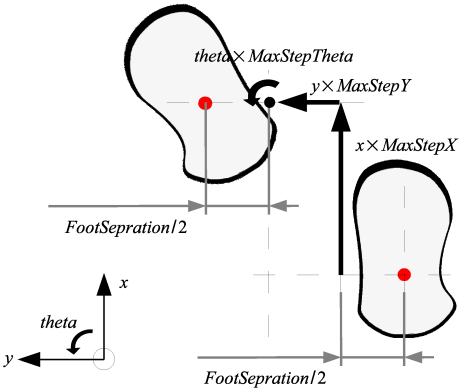


Figure 5.33: Foot step planner (adapted from [71]).

The other API is called *Destination Control* and it allows to specify the target position of the robot using three parameters:

x the distance to walk, in meters, along the x axis;

y the distance to walk, in meters, along the y axis;

theta the final robot orientation relative to the initial orientation, in radians.

When the distance to the desired position is less than 0.4 meters, the path followed by the walk is calculated using a SE3 Interpolation [74]. For bigger distances the path is generated using a Dubins curve (Fig. 5.34) composed by two circles with a 0.1 meters radius with a straight line between them.

Having these two APIs, we chose to use the *Velocity Control*, since it allows for a more flexible control over the walk and a more reactive agent, and also because the *Destination Control* API does not result in a precise destination as desired. When the robot is commanded to go to some position, using this API, it misses this position, because it uses no feedback from the world to localize itself and correct its trajectory. Giving this task to our agent, instead, and using the *Velocity Control* API, we can assure it arrives precisely at the desired destination, provided it can localize itself.

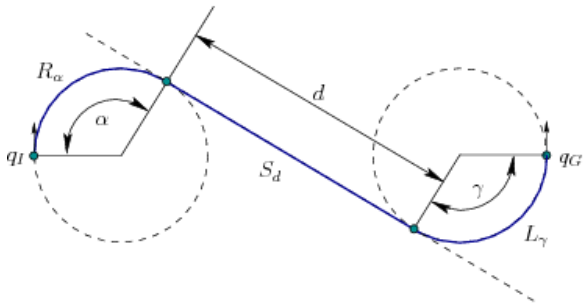


Figure 5.34: Path followed using the Destination Control.

To allow our agent architecture to use this API, the DCMController (Section 4.1.1) is used to send the desired walk parameters to the ALMotion module. The shared memory between the agent and the DCMController has all parameters of the walk and a flag that is set to *True* by the agent when it wants to execute the walk of Aldebaran, and *False* otherwise. When this flag is *True* the DCMController sends the parameters to the ALMotion module and does not send joint targets to the DCM, since these will be calculated and sent by the ALMotion module (with exception of the two head joints). When the flag is *False*, the DCMController behaves normally has explained in Sect. 4.1.1. In theory this would be enough to get the desired control over the walk of Aldebaran. However the robot behaved strangely in the transitions between the Aldebaran's walk and other behaviors. To correct this problem, we also needed to call the *setStiffnesses* function, of the ALMotion module, with the value zero, for all joints, when the flag is *False* and with the value one when the flag is *True*. Although this has no effect over the stiffness control, since the DCMController sends the values of stiffness directly to the DCM overriding the values defined by the ALMotion module, it does a reset of the walk of Aldebaran, solving the problem.

Beside the parameters presented above for each API, the walk of Aldebaran has some more parameters that can be used to optimize it. These parameters can be configured using the *setMotionConfig* function of the ALMotion module. The agent can modify them through the DCMController, in the same way the API parameters are defined. These parameters, and their default values, are:

- An angle in degrees added to the hip roll joint at each single-support phase. This value is linearly interpolated depending on the frequency parameter. When the frequency is lower the angle is bigger. When the frequency is higher the angle is smaller.
  - maxTrapezoid: 4.5 degrees;
  - minTrapezoid: 3.5 degrees;
- Time between foot steps, in number of motion cycles (each motion cycle is 20 ms).
  - stepMaxPeriod: 30
  - stepMinPeriod: 30
- Maximum step size and step height.

  - maxStepX: 0.04 meters
  - maxStepY: 0.04 meters
  - maxStepTheta: 20 degrees
  - stepHeight: 0.015 meters
- Foot default configuration.
  - footSeparation: 0.095 meters
  - footOrientation: 0 degrees
- Torso height and orientation.
  - torsoHeight: 0.31 meters
  - torsoOrientationX: 0 degrees
  - torsoOrientationY: 0 degrees

5.6.1 Results

The Aldebaran walk was tested on the floor of the laboratory. Measuring the time the robot takes to walk forward one meter, and repeating this five times, the average velocity achieved was 9 cm/s, with a standard deviation of 0.2 cm/s. A similar experience for side walking lead to an average velocity of 6 cm/s, with the same standard deviation of 0.2 cm/s. Making the robot turn in place for five turns, both clockwise and anticlockwise, for a total of 15 turns, the measured average velocity was 40 deg/s, with a standard deviation of 1.6 deg/s. The Aldebaran walk achieved similar results on the floor of the SPL field.

5.7 Follow The Ball Behavior

A middle level behavior was developed to make the real Nao robot follow the ball. The TFSWalk behavior is used to walk forward while turning in the direction of the ball.

The position of the ball is received from the vision process, in number of pixels from the upper left corner of the camera image. Only the horizontal component is used and it is a value between 0 and 319. This value is filtered using (5.29), since the ball position in the image oscillates during the walk.

$$preBallX = a \cdot preBallX + (1 - a) \cdot ballX \tag{5.29}$$

$ballX$ is the position of the ball received from the vision, $preBallX$ is the filtered position, and a is 0.9. When the ball is not in sight its last filtered position is used. The ball position is considered unknown when the ball is not seen for one second.

The TFSWalk behavior was commanded to walk forward when the filtered ball position has a value between 80 and 239, walk forward turning to the left when the value is lower than 80, and walk forward turning to the right when the value is greater than 239. The robot stops walking when the ball position is unknown. All these values were obtained empirically.

This behavior was executed successfully in the SPL field and it was stable. This means that all agent software components are working well when integrated.

5.8 Conclusion

This chapter presented the behaviors that the agent of the SPL can use. It started with the behaviors developed, namely: the Slot Behavior, the CPG Behavior, the OmnidirectionalWalk, the TFSWalk, and the COMWalk. The Slot Behavior allows to create robot movements by defining a sequence of joint angles that are followed by the joints using a sine interpolation. This was used to create the behaviors of getting up after a fall and the behaviors to kick the ball. It can be used to create movements that are sequences of robot poses. The CPG Behavior allows to develop periodic movements by moving the joints with a trajectory defined by a sum of sinusoids. This allowed to create a walk forward behavior, a turn in place behavior, and a rotate around the ball behavior.

The OmnidirectionalWalk is an open-loop omnidirectional walk. This means it can walk forward, backward, on sideways, rotate, or even a combination of these. In the context of this algorithm the Leg Interface is described. It allows to control a leg by defining: the distance between the hip and the ankle; the three angles between the torso and the line connecting hip and ankle; and the two angles between the foot and the perpendicular of the torso. The algorithm to compensate the inclination of the hip yaw-pitch joint is also presented. It calculates the angles of the three hip joints that produce the desired vertical rotation around the hip.

The TFSWalk is a walking algorithm optimized in the simulator and adapted to the real robot. It was modified to use the Leg Interface, which allows more control over it, and feedback was added to make it robust against external disturbances. The COMWalk is a walk based on the one from the B-Human team in 2009. It works by calculating the positions of the feet relative to the CoM of the robot and it uses inverse kinematics to convert these positions to joint angles. Then, it is explained how the walk created by Aldebaran works and how it is controlled by the agent using the DCMController. Last, it is explained the middle level behavior developed to make the real Nao robot follow the ball, using the TFSWalk behavior to move and the vision to know the location of the ball.

The comparison of the velocities achieved by the presented behaviors is shown in Table 5.6. The forward velocity achieved by the TFSWalk behavior is almost double of that achieved by the Aldebaran Walk. Although the TFSWalk behavior cannot walk to the side, it can turn while walking forward. Therefore, the TFSWalk is the best behavior to be used in the competitions. The Aldebaran Walk is a good choice when an omnidirectional walk is needed, since it allows movements in all directions.

Table 5.6: Velocity comparison of the presented behaviors.

Behavior	Forward Velocity (cm/s)	Lateral Velocity (cm/s)	Rotation Velocity (deg/s)
CPG Behaviors	6		18
OmnidirectionalWalk	5	1	35
TFSWalk	16		25
Aldebaran Walk	9	6	40

The rUNSWift team, one of the best teams competing in the SPL, has a gait that achieves

a velocity of 22 cm/s [62]. The Nao Team Humboldt[4] has a walk that can achieve 24 cm/s, according to their qualification video for the RoboCup 2011. And the B-Human team has a walking algorithm [75] that achieves 28 cm/s forward, 7 cm/s sideways, and 90 deg/s rotating.

This chapter proves it is possible to use behaviors from the simulation in a real robot. However, some modifications are required since the model of the robot used in the 3DSSL has a few differences in the dimensions and one more Degree of Freedom (DoF) than the real Nao robot. Also, the simulator cannot create an exact copy of the real world. The ground friction is low compared to the SPL field and it does not detect collisions between body parts of the same robot. In a real environment, there are factors that cannot be controlled and affect the performance of the behaviors, namely: external disturbances, uneven ground, and motor strength.

[4] `http://www.naoteamhumboldt.de/category/media/videos/`

Chapter 6

Conclusion

This last chapter starts with a review of the work presented in this thesis, then presents some conclusions taken, and finishes with the discussion over the future work.

This thesis described the adaptation of the agent architecture and behaviors from the 3D Soccer Simulation League (3DSSL) to the Standard Platform League (SPL). It is harder to develop behaviors for a robot in a real environment than for a simulated environment because of factors that make behaviors unstable, namely: external disturbances; uneven ground; or motor strength that affects the precision of the joints to follow trajectories. We proved that it is possible to use the behaviors of simulation in a real robot. The simulation environment can be used to test the behaviors before executing them on a real robot, or to use machine learning techniques to optimize them. Some modification were needed since the simulated and the real robot are different. In the TFSWalk behavior, feedback was added to increase stability and robustness, allowing the behavior to adapt to changes in the environment. The process of adapting behaviors from simulation to a real robot allowed us to find and correct some bugs on the existing behaviors. All presented behaviors were successfully adapted and tested on the real robot. They were stable on the floor of the laboratory. However, only some of them were stable in the SPL field, since the field used for tests is not flat levelled ground. The carpet of the field has more friction, irregularities, and slopes.

In this thesis was also developed an algorithm to calculate the angles of the three hip joints that produce the desired vertical rotation of the hip, since the Nao robot does not have a vertical hip joint. This algorithm was used with success on the presented behaviors and can be used in the future for any behavior, either in simulation or in the real robot, that needs to rotate the hip around the vertical axis.

The agent architecture and the high level behaviors are the same on both simulation and real robots. This allows to use the same agent in both the RoboCup 3DSSL and the SPL. Therefore, the high level behaviors can be ported from the simulated robot to the real one with little or no modifications, as soon as the low level behaviors are developed for both the simulated and the real robot.

To test the integration among all agent software components presented in Chapter 4, a simple agent was developed to make the robot follow the ball. This agent received the position of the ball from the vision, in number of pixels from the left upper corner of the image captured by the camera, and executed the TFSWalk behavior. When the horizontal position of the ball was in a region in the center of the image the robot walked forward, when it was in the left side of the image the robot walked forward turning to the left, and when

it was in the right side of the image the robot walked forward turning to the right. When the ball was not in sight the robot stopped. This test was a success, since the robot followed the ball in the SPL and it was stable. Hence, all software components are working well when integrated.

A paper about the work developed during this thesis was submitted to the EPIA 2011 conference [76] with the title "Humanoid Behaviors: From Simulation to a Real Robot". It was accepted for presentation at the conference and it is among the 25% papers selected for publication in a volume published by Springer in the LNCS/LNAI series. This paper will also be indexed at the ISI Web of Knowledge [77].

6.1 Future Work

Future work includes applying optimization techniques to optimize the behavior in the real robot, making them more stable, faster and robust. The Leg Interface can also be used in the CPG Behaviors and Slot Behaviors to develop new behaviors, making them more easily controllable and more stable, since using the Leg Interface in these two kinds of behavior will compensate the 45 degrees inclination of the hip yaw-pitch joint. Some interesting work to do in the future could be to make the model of the robot in the simulator equal to the real robot [78]. This could be used to get better results from the optimization techniques ran in the simulator, since the current differences between the models of the simulated and the real robot can affect the performance of the optimized behaviors. Another future work is to keep researching the most recent walking algorithms and try to develop a dynamic stable walk using the Zero-Moment Point (ZMP) and the inverted pendulum model, which is used by some teams of the SPL.

Appendix A

Compiling and Running the Agent

This appendix explains how to compile the DCMController module and the agent, and how to run them in the Nao robot. The version of the SDK of Aldebaran used is 1.10.25. In the following instructions the robot has the hostname `nao3`, this can be replaced by another name or by an IP address. The user account used is `nao` and the default password for this account is also `nao`.

A.1 Compiling the DCMController

The following commands compile the DCMController using the Cross Compilation Toolchain v1.10.25 and upload it to the Nao robot with hostname `nao3`:

```
$ cd /path/to/dcmcontroller
$ mkdir build
$ cd build
$ cmake -DCMAKE_TOOLCHAIN_FILE=
          /path/to/nao-cross-toolchain-1.10.25/toolchain-geode.cmake ..
$ make
$ scp sdk/lib/naoqi/libdcmcontroller.so nao@nao3:naoqi/lib/naoqi/
```

A.2 Running the DCMController

To run the DCMController module, the file `/home/nao/naoqi/preferences/autoload.ini` in the robot must be edited to have a line with the text `dcmcontroller` written in it. This file can be edited by logging in into the robot with:

```
$ ssh nao@nao3
```

Then, the NaoQi must be restarted in one of two ways. Either by executing the command `nao restart` in the robot command line. Or through the web interface (Fig. A.1) by opening a browser and connecting to the robot IP address or hostname. After logging in, go to the *advanced* menu and click *Naoqi*. Then click in the *restart* button. The NaoQi will take some time to restart. After it restarts there will an entry in the logs with the text "`NaoQi: INF: Registering module : dcmcontroller`" and the eyes of the robot will have the color red.

The DCMController module has a safety mechanism that allows to remotely remove the stiffness of the joints. This is controlled using python as follows:

```
$ export PYTHONPATH=/path/to/naoqi-sdk-1.10.25-linux/lib
$ python
>>> from naoqi import ALProxy
>>> proxy=ALProxy("dcmcontroller","nao3",9559)
>>> proxy.setRunning(True)      # apply stiffnesses
>>> proxy.setRunning(False)     # remove stiffnesses
```

When the stiffnesses is removed the eyes of the robot have the color red, otherwise, they have the color green.

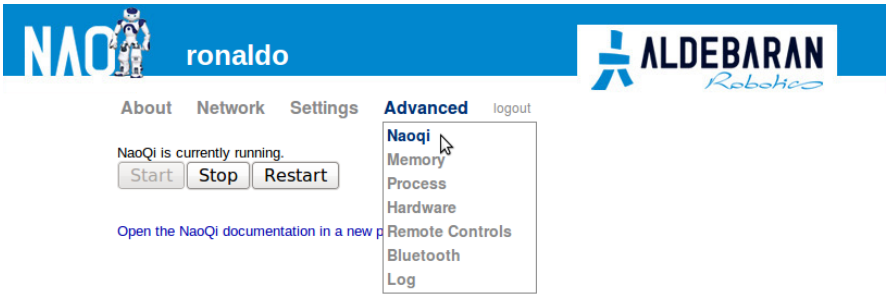


Figure A.1: Web interface of the Nao robot.

To stop the execution of the DCMController module, remove, or comment using a '#', the line with the text `dcmcontroller` in the `autoload.ini` file and then restart the NaoQi, as previously explained.

A.3 Compiling the Agent

The following commands compile the agent using the Cross Compilation Toolchain v1.10.25 and upload it to the robot:

```
$ cd /path/to/fcpagent
$ mkdir build
$ cd build
$ cmake -DCMAKE_TOOLCHAIN_FILE=
            /path/to/nao-cross-toolchain-1.10.25/toolchain-geode.cmake
        -DFCPAGENT_IS_REMOTE=on ..
$ make
$ scp sdk/lib/fcpagent nao@nao3:agent/
```

Note that before executing the last command, a directory named `agent` must exist in the robot.

A.4 Running the Agent

Before running the agent in the robot, a directory named `agent` must exist in the robot and some files must be copied from the `src` directory to this directory. These files can be copied using the following commands:

```
$ cd /path/to/fcpagent
$ scp src/strategy.conf.3D nao@nao3:agent/
$ scp src/nao.xml nao@nao3:agent/
$ scp -r src/movs/ nao@nao3:agent/
$ scp -r src/formations/ nao@nao3:agent/
$ scp ../rtdb/src/rtdb.ini nao@nao3:agent/
```

The directory where the log files are saved must also be created by executing the following command in the robot:

```
$ mkdir -p /home/nao/agent/logs/analyze/
```

To run the agent the DCMController module has to be running, as explained in the Section A.2. To run the agent execute:

```
$ export AGENT=3
$ ./fcpagent
```

or

```
$ export AGENT=3
$ ./fcpagent >/dev/null 2>&1
```

The latter form suppresses the program output. The `AGENT` environment variable defines the number of the agent, in this example is 3. The agent process is terminated using the *Ctrl-C* keystroke.

Bibliography

[1] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. RoboCup: The robot world cup initiative. In *Proceedings of the first international conference on Autonomous agents*, AGENTS '97, pages 340–347, New York, NY, USA, Feb. 1997. ACM.

[2] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, and H. Matsubara. RoboCup: A challenge problem for AI. *AI magazine*, 18(1):73, Mar. 1997.

[3] Objective - RoboCup. `http://www.robocup.org/about-robocup/objective/`, Jun. 2011.

[4] Nuno Lau and Luis Paulo Reis. FC Portugal - high-level coordination methodologies in soccer robotics. In Pedro Lima, editor, *Robotic Soccer*, pages 167–192. InTech, Dec. 2007.

[5] Luís Paulo Reis and Nuno Lau. FC Portugal Team Description: RoboCup 2000 Simulation League Champion. In Peter Stone, Tucker Balch, and Gerhard Kraetzschmar, editors, *RoboCup 2000: Robot Soccer World Cup IV*, volume 2019 of *Lecture Notes in Computer Science*, pages 29–40, London, UK, Jun. 2001. Springer-Verlag.

[6] Antonio J. R. Neves, Jose Luis Azevedo, Bernardo Cunha, Nuno Lau, Joao Silva, Frederico Santos, Gustavo Corrente, Daniel A. Martins, Nuno Figueiredo, Artur Pereira, Luis Almeida, Luis Seabra Lopes, Armando J. Pinho, Joao Rodrigues, and Paulo Pedreiras. CAMBADA soccer team: from robot architecture to multiagent coordination. In Vladan Papić, editor, *Robot Soccer*, pages 19–45. InTech, Jan. 2010.

[7] A.S. Conceição, A.P. Moreira, and P.J. Costa. Design of a mobile robot for Robocup Middle Size League. In *6th Latin American Robotics Symposium (LARS-2009)*, pages 1–6, Chile, Oct. 2009.

[8] Humanoid History -WABOT-. `http://www.humanoid.waseda.ac.jp/booklet/kato_2.html`, Jun. 2011.

[9] Rodney A. Brooks, Cynthia Breazeal, Matthew Marjanović, Brian Scassellati, and Matthew M. Williamson. The cog project: Building a humanoid robot. In Chrystopher L. Nehaniv, editor, *Computation for Metaphors, Analogy, and Agents*, volume 1562 of *Lecture Notes in Computer Science*, pages 52–87. Springer-Verlag, Berlin, Heidelberg, Jun. 1999.

[10] Y. Kuroki. A small biped entertainment robot. In *Micromechatronics and Human Science, 2001. MHS 2001. Proceedings of 2001 International Symposium on*, pages 3–4, Nagoya, Japan, Sep. 2001.

[11] K. Kaneko, F. Kanehiro, S. Kajita, H. Hirukawa, T. Kawasaki, M. Hirata, K. Akachi, and T. Isozumi. Humanoid robot HRP-2. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 2, pages 1083 – 1090, New Orleans, LA, USA, May 2004.

[12] T. Kanda, H. Ishiguro, M. Imai, and T. Ono. Development and evaluation of interactive humanoid robots. *Proceedings of the IEEE*, 92(11):1839–1850, Nov. 2004.

[13] Minoru Asada, Karl F. MacDorman, Hiroshi Ishiguro, and Yasuo Kuniyoshi. Cognitive developmental robotics as a new paradigm for the design of humanoid robots. *Robotics and Autonomous Systems*, 37(2–3):185–193, Nov. 2001.

[14] M.N. Nicolescu and M.J. Mataric. Task learning through imitation and human-robot interaction. *Models and Mechanisms of Imitation and Social Learning in Robots, Humans and Animals: Behavioural, Social and Communicative Dimensions*, pages 407–424, Apr. 2006.

[15] Ian J. Harrington. Symptoms in the opposite or uninjured leg. Workplace Safety and Insurance Appeals Tribuna, Aug. 2005.

[16] Ching-Long Shih. Ascending and descending stairs for a biped robot. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 29(3):255–268, May 1999.

[17] Ching-Long Shih and Chien-Jung Chiou. The motion control of a statically stable biped robot on an uneven floor. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 28(2):244–249, Apr. 1998.

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

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

[20] P. Sardain and G. Bessonnet. Forces acting on a biped robot. center of pressure-zero moment point. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 34(5):630–637, Sep. 2004.

[21] A. Goswami. Foot rotation indicator (FRI) point: a new gait planning tool to evaluate postural stability of biped robots. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 1, pages 47–52, Detroit, Michigan, USA, May 1999.

[22] K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka. The development of Honda humanoid robot. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 2, pages 1321–1326, Leuven, Belgium, May 1998.

[23] Qiang Huang, S. Kajita, N. Koyachi, K. Kaneko, K. Yokoi, H. Arai, K. Komoriya, and K. Tanie. A high stability, smooth walking pattern for a biped robot. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 1, pages 65–71, Detroit, Michigan, USA, May 1999.

[24] J. Yamaguchi, E. Soga, S. Inoue, and A. Takanishi. Development of a bipedal humanoid robot-control method of whole body cooperative dynamic biped walking. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 1, pages 368–374, Detroit, Michigan, USA, May 1999.

[25] K. Nishtwaki, K. Nagasaka, M. Inaba, and H. Inoue. Generation of reactive stepping motion for a humanoid by dynamically stable mixture of pre-designed motions. In *Systems, Man, and Cybernetics, 1999. IEEE SMC '99 Conference Proceedings. 1999 IEEE International Conference on*, volume 6, pages 902–907, Tokyo, Japan, Oct. 1999.

[26] Qiang Huang, K. Yokoi, S. Kajita, K. Kaneko, H. Arai, N. Koyachi, and K. Tanie. Planning walking patterns for a biped robot. *Robotics and Automation, IEEE Transactions on*, 17(3):280–289, Jun. 2001.

[27] J.Y. Kim, I.W. Park, and J.H. Oh. Experimental realization of dynamic walking of the biped humanoid robot KHR-2 using zero moment point feedback and inertial measurement. *Advanced Robotics*, 20(6):707–736, Jun. 2006.

[28] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Yokoi, and H. Hirukawa. A realtime pattern generator for biped walking. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, volume 1, pages 31–37, Washington, DC, USA, May 2002.

[29] A. Sano and J. Furusho. Realization of natural dynamic walking using the angular momentum information. In *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pages 1476–1481, Cincinnati, OH, USA, May 1990.

[30] S. Kajita and K. Tani. Study of dynamic biped locomotion on rugged terrain-theory and basic experiment. In *Advanced Robotics, 1991. 'Robots in Unstructured Environments', 91 ICAR., Fifth International Conference on*, pages 741–746, Pisa, Italy, Jun. 1991.

[31] Y. Fujimoto and A. Kawamura. Three dimensional digital simulation and autonomous walking control for eight-axis biped robot. In *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*, volume 3, pages 2877–2884, Nagoya, Aichi, Japan, May 1995.

[32] J. Pratt, P. Dilworth, and G. Pratt. Virtual model control of a bipedal walking robot. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 1, pages 193–198, Albuquerque, NM, USA, Apr. 1997.

[33] T. Sugihara, Y. Nakamura, and H. Inoue. Real-time humanoid motion generation through ZMP manipulation based on inverted pendulum control. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, volume 2, pages 1404–1409, Washington, DC, USA, May 2002.

[34] S. Kajita, F. Kanehiro, K. Kaneko, K. Yokoi, and H. Hirukawa. The 3D linear inverted pendulum mode: a simple modeling for a biped walking pattern generation. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 1, pages 239–246, Maui, USA, Nov. 2001.

[35] Auke Jan Ijspeert. Central pattern generators for locomotion control in animals and robots: A review. *Neural Networks*, 21(4):642 – 653, May 2008.

[36] L. Righetti and A.J. Ijspeert. Programmable central pattern generators: an application to biped locomotion control. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1585–1590, Orlando, Florida, USA, May 2006.

[37] B. Bussel, A. Roby-Brami, O.R. Néris, and A. Yakovleff. Evidence for a spinal stepping generator in man. *Spinal Cord*, 34(2):91–92, Feb. 1996.

[38] J. Duysens and H.W.A.A. Van de Crommert. Neural control of locomotion; Part 1: The central pattern generator from cats to humans. *Gait & Posture*, 7(2):131–141, Mar. 1998.

[39] Hugo Picado, Marcos Gestal, Nuno Lau, Luis Reis, and Ana Tomé. Automatic generation of biped walk behavior using genetic algorithms. In Joan Cabestany, Francisco Sandoval, Alberto Prieto, and Juan Corchado, editors, *Bio-Inspired Systems: Computational and Ambient Intelligence*, volume 5517 of *Lecture Notes in Computer Science*, pages 805–812. Springer Berlin / Heidelberg, Jun. 2009.

[40] Nima Shafii, Luís Reis, and Nuno Lau. Biped walking using coronal and sagittal movements based on truncated fourier series. In Javier Ruiz-del Solar, Eric Chown, and Paul Plöger, editors, *RoboCup 2010: Robot Soccer World Cup XIV*, volume 6556 of *Lecture Notes in Computer Science*, pages 324–335. Springer Berlin / Heidelberg, Jun. 2011.

[41] S. Behnke. Online trajectory generation for omnidirectional biped walking. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1597 –1603, Orlando, Florida, USA, May 2006.

[42] T. McGeer. Passive dynamic walking. *The International Journal of Robotics Research*, 9(2):62, Apr. 1990.

[43] S. Collins, A. Ruina, R. Tedrake, and M. Wisse. Efficient bipedal robots based on passive-dynamic walkers. *Science*, 307(5712):1082, 2005.

[44] T. McGeer. Passive walking with knees. In *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pages 1640–1645, may 1990.

[45] S.H. Collins, M. Wisse, and A. Ruina. A three-dimensional passive-dynamic walking robot with two legs and knees. *The International Journal of Robotics Research*, 20(7):607, Jul. 2001.

[46] J. Stückler, J. Schwenk, and S. Behnke. Getting back on two feet: Reliable standing-up routines for a humanoid robot. In *Proc. of The 9th International Conference on Intelligent Autonomous Systems (IAS-9)*, pages 676–685, Tokyo, Japan, Mar 2006.

[47] S. Behnke, M. Schreiber, J. Stückler, H. Strasdat, and M. Bennewitz. NimbRo KidSize 2006 team description. In *In RoboCup 2006 Humanoid League Team Descriptions*, Bremen, Jun. 2006.

[48] K. Kaneko, F. Kanehiro, S. Kajita, K. Yokoyama, K. Akachi, T. Kawasaki, S. Ota, and T. Isozumi. Design of prototype humanoid robotics platform for HRP. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 3, pages 2431–2436, Switzerland, Oct. 2002.

[49] K. Harada, S. Kajita, K. Kaneko, and H. Hirukawa. An analytical method on real-time gait planning for a humanoid robot. In *Humanoid Robots, 2004 4th IEEE/RAS International Conference on*, volume 2, pages 640–655, Santa Monica, California, Nov. 2004.

[50] H. Diedam, D. Dimitrov, P.-B. Wieber, K. Mombaur, and M. Diehl. Online walking gait generation with adaptive foot positioning through linear model predictive control. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 1121–1126, Nice, France, Sep. 2008.

[51] M. Hirose and K. Ogawa. Honda humanoid robots development. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 365(1850):11, Jan. 2007.

[52] ASIMO by Honda — the world's most advanced humanoid robot. `http://asimo.honda.com/`, Jun. 2011.

[53] D. Gouaillier, V. Hugel, P. Blazevic, C. Kilner, J. Monceaux, P. Lafourcade, B. Marnier, J. Serre, and B. Maisonnier. The NAO humanoid: a combination of performance and affordability. *ArXiv e-prints*, Jul. 2008.

[54] David Gouaillier, Vincent Hugel, Pierre Blazevic, Chris Kilner, Jerome Monceaux, Pascal Lafourcade, Brice Marnier, Julien Serre, and Bruno Maisonnier. Mechatronic design of NAO humanoid. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 769 –774, Kobe, Japan, May 2009.

[55] Jason A. Kulk and James S. Welsh. A low power walk for the NAO robot. *Australasian Conference on Robotics and Automation (ACRA)*, Dec 2008.

[56] RoboCup 2010 Singapore. `http://www.robocup2010.org/`, Jun. 2011.

[57] RoboCup Technical Committee. RoboCup Standard Platform League (Nao) Rule Book, May 2011.

[58] Joschka Boedecker and Minoru Asada. Simspark concepts and application in the Robocup 3D Soccer Simulation League. In *Proceedings of SIMPAR-2008 Workshop on The Universe of RoboCup simulators*, pages 174–181, Venice, Italy, Nov. 2008.

[59] R. Smith. Open dynamics engine. `www.ode.org`, Jul. 2008.

[60] Soccer Simulation - Simspark. `http://simspark.sourceforge.net/wiki/index.php/Soccer_Simulation`, Jun. 2011.

[61] Thomas Röfer, Tim Laue, Judith Müller, Armin Burchardt, Erik Damrose, Alexander Fabisch, Fynn Feldpausch, Katharina Gillmann, Colin Graf, Thijs Jeffry de Haas, Alexander Härtl, Daniel Honsel, Philipp Kastner, Tobias Kastner, Benjamin Markowsky, Michael Mester, Jonas Peter, Ole Jan Lars Riemann, Martin Ring, Wiebke Sauerland, André Schreck, Ingo Sieverdingbeck, Felix Wenk, and Jan-Hendrik Worch. B-Human team report and code release 2010, Oct. 2010. Only available online: `http://www.b-human.de/file_download/33/bhuman10_coderelease.pdf`.

[62] Adrian Ratter, Bernhard Hengst, Brad Hall, Brock White, Benjamin Vance, Claude Sammut, David Claridge, Hung Nguyen, Jayen Ashar, Maurice Pagnucco, Stuart Robinson, and Yanjin Zhu. rUNSWift team report 2010 Robocup Standard Platform League, Oct. 2010.

[63] Somchaya Liemhetcharat, Brian Coltin, Çetin Meriçli, Junyun Tay, and Manuela Veloso. CMurfs: Carnegie mellon united robots for soccer, 2010.

[64] Luis Almeida, Frederico Santos, Tullio Facchinetti, Paulo Pedreiras, Valter Silva, and L. Lopes. Coordinating distributed autonomous agents with a real-time database: The CAMBADA project. In Cevdet Aykanat, Tugrul Dayar, and Ibrahim Krpeoglu, editors, *Computer and Information Sciences - ISCIS 2004*, volume 3280 of *Lecture Notes in Computer Science*, pages 876–886. Springer Berlin / Heidelberg, Oct. 2004.

[65] Nuno Figueiredo, António Neves, Nuno Lau, Artur Pereira, and Gustavo Corrente. Control and monitoring of a robotic soccer team: The base station application. In Lus Lopes, Nuno Lau, Pedro Mariano, and Lus Rocha, editors, *Progress in Artificial Intelligence*, volume 5816 of *Lecture Notes in Computer Science*, pages 299–309. Springer Berlin / Heidelberg, Aveiro, Portugal, Oct. 2009.

[66] Frederico Santos, Luís Almeida, Luís Lopes, José Azevedo, and M. Cunha. Communicating among robots in the RoboCup Middle-Size League. In Jacky Baltes, Michail Lagoudakis, Tadashi Naruse, and Saeed Ghidary, editors, *RoboCup 2009: Robot Soccer World Cup XIII*, volume 5949 of *Lecture Notes in Computer Science*, pages 320–331. Springer Berlin / Heidelberg, Apr., 2010.

[67] Aaron James Soon Beng Tay. Walking Nao Omnidirectional Bipedal Locomotion, Aug. 2009.

[68] Colin Graf, Alexander Härtl, Thomas Röfer, and Tim Laue. A robust closed-loop gait for the Standard Platform League humanoid. In Changjiu Zhou, Enrico Pagello, Emanuele Menegatti, Sven Behnke, and Thomas Röfer, editors, *Proceedings of the Fourth Workshop on Humanoid Soccer Robots in conjunction with the 2009 IEEE-RAS International Conference on Humanoid Robots*, pages 30–37, Paris, France, Dec. 2009.

[69] N. Shafii, M.H.S. Javadi, and B. Kimiaghalam. A Truncated Fourier Series with genetic algorithm for the control of biped locomotion. In *Advanced Intelligent Mechatronics, 2009. AIM 2009. IEEE/ASME International Conference on*, pages 1781–1785, Singapore, Jul. 2009.

[70] Thomas Röfer, Tim Laue, Judith Müller, Oliver Bösche, Armin Burchardt, Erik Damrose, Katharina Gillmann, Colin Graf, Thijs Jeffry de Haas, Alexander Härtl, An-

drik Rieskamp, André Schreck, Ingo Sieverdingbeck, and Jan-Hendrik Worch. B-Human team report and code release 2009, Nov. 2009. Only available online: `http://www.b-human.de/file_download/26/bhuman09_coderelease.pdf`.

[71] D. Gouaillier, C. Collette, C. Kilner, and A. Robotics. Omni-directional closedloop walk for NAO. In *IEEE-RAS International Conference on Humanoid Robots*, pages 448–454, Nashville, TN, USA, Dec. 2010.

[72] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, volume 2, pages 1620–1626, Taipei, Taiwan, Sep. 2003.

[73] P.-B. Wieber. Trajectory free linear model predictive control for stable walking in the presence of strong perturbations. In *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*, pages 137–142, Genova, Italy, Dec. 2006.

[74] M. Zefran and V. Kumar. Two methods for interpolating rigid body motions. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 4, pages 2922–2927, Leuven, Belgium, May 1998.

[75] C. Graf and T. Röfer. A closed-loop 3D-LIPM gait for the RoboCup Standard Platform League humanoid. In C. Zhou, E. Pagello, S. Behnke, E. Menegatti, T. Röfer, and P. Stone, editors, *Proceedings of the Fourth Workshop on Humanoid Soccer Robots in conjunction with the 2010 IEEE-RAS International Conference on Humanoid Robots*, Nashville, TN, USA, Dec. 2010.

[76] EPIA.2011 - Home. `http://epia2011.appia.pt/`, Jun. 2011.

[77] ISI Web of Knowledge [v.4.10] - All Databases Home. `http://www.isiknowledge.com/`, Jun. 2011.

[78] Yuan Xu and Hans-Dieter Burkhard. Narrowing reality gap and validation: Improving the simulator for humanoid soccer robot. In *Concurrency, Specification and Programming CS&P'2010*, Helenenau, Germany, Sep. 2010.