Humanoid Behaviors: From Simulation to a Real Robot

Edgar Domingues[1], Nuno Lau[1], Bruno Pimentel[1,2], Nima Shafii[2], Luís Paulo Reis[2], and António J. R. Neves[1]

[1] DETI/UA - Dep. of Electronics, Telecommunications and Informatics,
IEETA - Inst. of Electronics and Telematics Engineering of Un. of Aveiro,
Campus Universitário de Santiago, 3810-193 Aveiro, Portugal,
{edgar.domingues,nunolau,brunopimentel,an}@ua.pt

[2] DEI/FEUP - Dep. Informatics Engineering, Faculfy of Engineering of the University of Porto,
LIACC - Artificial Intelligence And Computer Science Lab. of the University of Porto,
Rua Dr. Roberto Frias, s/n 4200-465 Porto, Portugal,
nima.shafii@gmail.com,lpreis@fe.up.pt

Abstract. This paper presents the modifications needed to adapt a humanoid agent architecture and behaviors from simulation to a real robot. The experiments were conducted using the Aldebaran Nao robot model. The agent architecture was adapted from the RoboCup 3D Simulation League to the Standard Platform League with as few changes as possible. The reasons for the modifications include small differences in the dimensions and dynamics of the simulated and the real robot and the fact that the simulator does not create an exact copy of a real environment. In addition, the real robot API is different from the simulated robot API and there are a few more restrictions on the allowed joint configurations. The general approach for using behaviors developed for simulation in the real robot was to: first, (if necessary) make the simulated behavior compliant with the real robot restrictions, second, apply the simulated behavior to the real robot reducing its velocity, and finally, increase the velocity, while adapting the behavior parameters, until the behavior gets unstable or inefficient. This paper also presents an algorithm to calculate the three angles of the hip that produce the desired vertical hip rotation, since the Nao robot does not have a vertical hip joint. All simulation behaviors described in this paper were successfully adapted to the real robot.

Keywords: humanoid robot, behaviors, simulation, real robot, Aldebaran Nao

1 Introduction

Since the year 2004 the FC Portugal team [8, 13] participates in the RoboCup 3D Simulation League. This year, for the first time, following this collaboration and also including members from CAMBADA [10] and 5DPO [3] teams, the Portuguese Team will participate in the Standard Platform League (SPL).

In the RoboCup 3D Soccer Simulation League robotic soccer games are run on a simulator. Each participating team programs an agent to control a humanoid robot that models the Nao robot from Aldebaran. In the SPL all robots are the same, being nowadays the Nao robot, the only difference being the software which controls them. Since 2008 both the Simulation League and the SPL use the same robot model: the Aldebaran Nao (Fig. 1 and 2). This would allow to develop agents that can compete on both leagues, test the agent on a simulated robot before running it on the real one, and use on the latter the results of machine learning techniques ran in the simulation.

Fig. 1. The Nao robot.

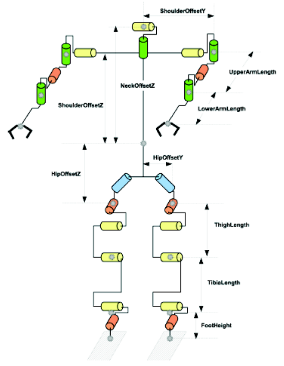

Fig. 2. Nao joints (adapted from [5]).

However, 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, since it has a few differences in the dimensions and one more degree of freedom. Hence, some modifications must be made to adapt the low level behaviors from simulation to a real robot. 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.

This paper presents the modifications needed on the FC Portugal simulation agent to run on the real Naos. All behaviors needed for the real robot are present in the simulation agent. However, the get up behaviors were developed from scratch, since those used in the simulation execute motions that are not possible on a real robot. A kick to the side that did not exist in the simulation was also created. The next two sections describe the simulation environment and the real robot. Section 4 describes the architecture of our agent. Section 5 and Section 6 describes the behaviors we use in simulation and their adaptation to the SPL. Section 7 shows the results achieved with real Naos. The last section presents the conclusions.

2 RoboCup 3D Simulator

RoboCup 3D Soccer Simulation League uses SimSpark simulator [2] to simulate the games. This simulator uses the Open Dynamics Engine (ODE) [16] to simulate the physical environment. The robot model used is based on the Aldebaran Nao robot. However, it is not a precise replica of the real one, since it has a few differences in the dimensions and 22 degrees of freedom (one more than the real version). 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, but this is not important for this paper, since we talk only about low level behaviors.

3 Nao Hardware and Software

The real Nao robot has been developed by the French company Aldebaran Robotics [4]. It is a humanoid robot with 21 degrees of freedom, one less than the simulation because the two hip yaw pitch joints are controlled by the same motor. Joint control is based on providing angle targets. Each joint has an internal PID controller which then acts autonomously to achieve its target. This means that the joint control on the real robot is higher level than the used on simulation. Also unlike in simulation, the real robot provides stiffness control for each joint, which can be useful to save energy and improve the behavior robustness and speed [7].

Like the simulated robot, the real robot has a gyroscope and an accelerometer in the torso. Aldebaran provides a programming interface that delivers the filtered angles of the torso position relative to the vertical. Each foot has 4 force sensitive resistors and there are two cameras in the head.

The Nao robot runs NaoQi [4] 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, 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 modules provided.

4 Portuguese Team Agent Architecture

Our approach to allow the agent architecture access the robot hardware, like many others SPL teams [14, 12, 9], was to create a simple NaoQi module to communicate with the DCM. This module reads the actuators values from the shared memory, sends them to be executed by the DCM, and copies the sensors values from the DCM to the shared memory. By accessing the shared memory, our agent becomes independent from the Aldebaran software.

The architecture of the agent that uses the shared memory is identical to our agent in the simulation league. The only modifications made were in the low level communications. In simulation, the agent communicates with the server to send the actuator values and receive the sensor values. In the real robot, instead of communicating with a server, the agent sends the actuator values and reads the sensor values from the shared memory. As presented above, the real robot has a 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 copy this software PID controller from simulation to the real robot.

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

5 Behavior models

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. The sinusoidal trajectory is used since the initial and final speed are zero, and it assures the lowest second derivative maximum, hence the acceleration will be minimized [1]. 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 need to recompile the agent.

This was used to develop some behaviors, namely: kick the ball, get up, among others.

5.2 CPG Behaviors

Central Pattern Generator (CPG) Behaviors execute, on each targeted joint, a trajectory defined by a sum of sine waves [11]. 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.

This was used to create periodic behaviors: walk, turn in place, rotate around the ball, etc.

5.3 OmnidirectionalWalk Behavior

This behavior was based on Sven Behnke's work [1]. It is an open-loop omnidirectional walk developed for the humanoid robot Jupp. It had to be adapted to the simulated Nao, since the two robots are different.

This behavior uses a Leg Interface to control the leg movements. The Leg Interface allows to specify leg positions using three parameters: leg angle, roll, pitch, and yaw angles between the torso and the line connecting hip and ankle; leg extension, the distance between the hip and the ankle; and 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 leg angle, $\theta_{\mathrm{Leg}} = \left(\theta^{\mathrm{r}}_{\mathrm{Leg}}, \theta^{\mathrm{p}}_{\mathrm{Leg}}, \theta^{\mathrm{y}}_{\mathrm{Leg}}\right)$, is the angle between the torso and the line connecting hip and ankle. $\theta^{\mathrm{r}}_{\mathrm{Leg}} = 0$ when the leg is parallel to the trunk, on the coronal plane, and $\theta^{\mathrm{r}}_{\mathrm{Leg}} > 0$ when the leg is moved outwards to the side. $\theta^{\mathrm{p}}_{\mathrm{Leg}} = 0$ when the leg is parallel to the trunk, in the sagittal plane, and $\theta^{\mathrm{p}}_{\mathrm{Leg}} > 0$ when the leg is moved to the front. $\theta^{\mathrm{y}}_{\mathrm{Leg}} = 0$ when the foot points to the front and $\theta^{\mathrm{y}}_{\mathrm{Leg}} > 0$ when the leg is rotated vertically pointing the foot outward. The foot angle, $\theta_{\mathrm{Foot}} = (\theta^{\mathrm{r}}_{\mathrm{Foot}}, \theta^{\mathrm{p}}_{\mathrm{Foot}})$, controls the foot angle relative to the torso. If $\theta_{\mathrm{Foot}} = (0, 0)$ 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 shortened when $\gamma = -1$. When shortened, the leg is $\eta_{\min}$ of its fully extended length. The relative leg length can be calculated as $\eta = 1 + (1 - \eta_{\min})\gamma$.

5.4 TFSWalk Behavior

The TFSWalk gait combines two approaches: a gait where the joints trajectory is generated using Partial Fourier Series optimized with Genetic Algorithms [11]; and a gait using Truncated Fourier Series, which are imitated from human walk, to generate the joints angles optimized with Particle Swarm Optimization [15]. This gait allows the robot to walk forward and backward (either straight or turning) and to turn in place.

6 Adaptation to the Real Robot

When adapting the behaviors from simulation to a real robot we tried to keep the architecture of the agent similar on both. Since all behaviors on simulation produce joints angles which are then passed to a software PID controller, this was replaced with some software to send these angles to the shared memory (as explained on Section 4), and convert them to the angle range of the real robot joints if needed.

6.1 Leg Interface

The Leg Interface calculates the angles of the 6 leg joints using: leg angle, $\theta_{\mathrm{Leg}} = \left(\theta^{\mathrm{r}}_{\mathrm{Leg}}, \theta^{\mathrm{p}}_{\mathrm{Leg}}, \theta^{\mathrm{y}}_{\mathrm{Leg}}\right)$; foot angle, $\theta_{\mathrm{Foot}} = (\theta^{\mathrm{r}}_{\mathrm{Foot}}, \theta^{\mathrm{p}}_{\mathrm{Foot}})$; and leg extension, γ. In the Jupp robot the hip joints are in the following order, from up to down: roll, pitch and yaw. But in the Nao robot the hip joints are in a different order: yaw pitch, roll, pitch. This means that we cannot use all the formulas presented in Sven Behnke's paper [1]. So we have developed our own formulas. Given 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, $l = \eta(l_{\mathrm{upperLeg}} + l_{\mathrm{lowerLeg}})$. Where l_{upperLeg} and l_{lowerLeg} are the lengths of the thigh and shank, respectively. Using the law of cosines (1), the knee joint angle can be calculated by (2). 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. In the same way, the law of cosines is used to calculate the angles of the hip (3) and ankle (4) to compensate the knee angle. Therefore, when the leg extension changes, θ_{Leg} and θ_{Foot} are not changed.

$$c^2 = a^2 + b^2 - 2 \cdot a \cdot b \cdot \cos(\theta) \tag{1}$$

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

$$\Delta\theta^{\mathrm{p}}_{\mathrm{Hip}} = \arccos\left(\frac{l^2_{\mathrm{upperLeg}} + l^2 - l^2_{\mathrm{lowerLeg}}}{2 \cdot l_{\mathrm{upperLeg}} \cdot l}\right) \tag{3}$$

$$\Delta\theta^{\mathrm{p}}_{\mathrm{Ankle}} = \arccos\left(\frac{l^2_{\mathrm{lowerLeg}} + l^2 - l^2_{\mathrm{upperLeg}}}{2 \cdot l_{\mathrm{lowerLeg}} \cdot l}\right) \tag{4}$$

The remaining leg joint angles can be calculated by equations (5), 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 $\Delta\theta^{\mathrm{p}}_{\mathrm{Hip}}$. The ankle angles are the differences between θ_{Foot} angles and the θ_{Leg} angles with the compensation $\Delta\theta^{\mathrm{p}}_{\mathrm{Ankle}}$.

$$\begin{aligned}
\theta_{\mathrm{HipYawPitch}} &= \theta^{\mathrm{y}}_{\mathrm{Leg}} \\
\theta_{\mathrm{HipRoll}} &= -ls \cdot \theta^{\mathrm{r}}_{\mathrm{Leg}} \\
\theta_{\mathrm{HipPitch}} &= \theta^{\mathrm{p}}_{\mathrm{Leg}} + \Delta\theta^{\mathrm{p}}_{\mathrm{Hip}} \\
\theta_{\mathrm{AnklePitch}} &= \theta^{\mathrm{p}}_{\mathrm{Foot}} - \theta^{\mathrm{p}}_{\mathrm{Leg}} + \Delta\theta^{\mathrm{p}}_{\mathrm{Ankle}} \\
\theta_{\mathrm{AnkleRoll}} &= -ls \cdot \left(\theta^{\mathrm{r}}_{\mathrm{Foot}} - \theta^{\mathrm{r}}_{\mathrm{Leg}}\right)
\end{aligned} \tag{5}$$

Note that in (5) the leg vertical rotation is applied directly in the hip yaw pitch joint. 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 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 (6).

$$\begin{aligned} Rot_{\mathrm{Hip}} &= Rot_{\mathrm{x}}\left(ls \cdot \frac{\pi}{4}\right) \cdot Rot_{\mathrm{z}}(\theta^{\mathrm{y}}_{\mathrm{Leg}}) \\ &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{\sqrt{2}}{2} & -ls \cdot \frac{\sqrt{2}}{2} \\ 0 & ls \cdot \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{pmatrix} \cdot \begin{pmatrix} \cos(\theta^{\mathrm{y}}_{\mathrm{Leg}}) & -\sin(\theta^{\mathrm{y}}_{\mathrm{Leg}}) & 0 \\ \sin(\theta^{\mathrm{y}}_{\mathrm{Leg}}) & \cos(\theta^{\mathrm{y}}_{\mathrm{Leg}}) & 0 \\ 0 & 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} \cos(\theta^{\mathrm{y}}_{\mathrm{Leg}}) & -\sin(\theta^{\mathrm{y}}_{\mathrm{Leg}}) & 0 \\ \frac{\sqrt{2}}{2} \cdot \sin(\theta^{\mathrm{y}}_{\mathrm{Leg}}) & \frac{\sqrt{2}}{2} \cdot \cos(\theta^{\mathrm{y}}_{\mathrm{Leg}}) & -ls \cdot \frac{\sqrt{2}}{2} \\ ls \cdot \frac{\sqrt{2}}{2} \cdot \sin(\theta^{\mathrm{y}}_{\mathrm{Leg}}) & ls \cdot \frac{\sqrt{2}}{2} \cdot \cos(\theta^{\mathrm{y}}_{\mathrm{Leg}}) & \frac{\sqrt{2}}{2} \end{pmatrix} \end{aligned} \tag{6}$$

Then, part of the B-Human inverse kinematics [6] 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 $\cos(\delta_{\mathrm{x}})$) (7).

$$\begin{aligned} Rot_{\mathrm{Hip}} &= Rot_{\mathrm{z}}(\delta_{\mathrm{z}}) \cdot Rot_{\mathrm{x}}(\delta_{\mathrm{x}}) \cdot Rot_{\mathrm{y}}(\delta_{\mathrm{y}}) \\ &= \begin{pmatrix} c_{\mathrm{x}}c_{\mathrm{z}} - s_{\mathrm{x}}s_{\mathrm{y}}s_{\mathrm{z}} & -c_{\mathrm{x}}s_{\mathrm{z}} & c_{\mathrm{z}}s_{\mathrm{y}} + c_{\mathrm{y}}s_{\mathrm{x}}s_{\mathrm{z}} \\ c_{\mathrm{z}}s_{\mathrm{x}}s_{\mathrm{y}} + c_{\mathrm{y}}s_{\mathrm{z}} & c_{\mathrm{x}}c_{\mathrm{z}} & -c_{\mathrm{y}}c_{\mathrm{z}}s_{\mathrm{x}} + s_{\mathrm{y}}s_{\mathrm{z}} \\ -c_{\mathrm{x}}s_{\mathrm{y}} & s_{\mathrm{x}} & c_{\mathrm{x}}c_{\mathrm{y}} \end{pmatrix} \end{aligned} \tag{7}$$

It is now clear that the angle of the hip roll (x axis) can be calculated using s_{x}, as shown in (8). This angle has to be rotated 45 degrees, because the hip roll joint space is rotated according to the hip yaw pitch axis.

$$\delta_{\mathrm{x}} = \arcsin(s_{\mathrm{x}}) - ls \cdot \frac{\pi}{4} = \arcsin(ls \cdot \frac{\sqrt{2}}{2} \cdot \cos(\theta^{\mathrm{y}}_{\mathrm{Leg}})) - ls \cdot \frac{\pi}{4} \tag{8}$$

Calculating each of the other two hip joints requires 2 matrix entries. Equation (9) 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 (10) and (11).

$$\frac{c_{\mathrm{x}} \cdot s_{\mathrm{z}}}{c_{\mathrm{x}} \cdot c_{\mathrm{z}}} = \frac{\cos(\delta_{\mathrm{x}}) \cdot \sin(\delta_{\mathrm{z}})}{\cos(\delta_{\mathrm{x}}) \cdot \cos(\delta_{\mathrm{z}})} = \frac{\sin(\delta_{\mathrm{z}})}{\cos(\delta_{\mathrm{z}})} = \tan(\delta_{\mathrm{z}}) \tag{9}$$

$$\delta_{\mathrm{z}} = \operatorname{atan2}(c_{\mathrm{x}} \cdot s_{\mathrm{z}}, c_{\mathrm{x}} \cdot c_{\mathrm{z}}) = \operatorname{atan2}(\sin(\theta^{\mathrm{y}}_{\mathrm{Leg}}), \frac{\sqrt{2}}{2} \cdot \cos(\theta^{\mathrm{y}}_{\mathrm{Leg}})) \tag{10}$$

$$\delta_{\mathrm{y}} = \operatorname{atan2}(c_{\mathrm{x}} \cdot s_{\mathrm{y}}, c_{\mathrm{x}} \cdot c_{\mathrm{y}}) = \operatorname{atan2}(-ls \cdot \frac{\sqrt{2}}{2} \cdot \sin(\theta^{\mathrm{y}}_{\mathrm{Leg}}), \frac{\sqrt{2}}{2}) \tag{11}$$

These three angles (δ_{x}, δ_{y} and δ_{z}) are added to the calculated joints angles of (5) producing the final angles of the three hip joints:

$$\begin{aligned}\theta_{\text{HipYawPitch}} &= \delta_{\text{z}} \\ \theta_{\text{HipRoll}} &= -ls \cdot \theta^{\text{r}}_{\text{Leg}} + \delta_{\text{x}} \\ \theta_{\text{HipPitch}} &= \theta^{\text{p}}_{\text{Leg}} + \Delta\theta^{\text{p}}_{\text{Hip}} - \delta_{\text{y}}\end{aligned} \tag{12}$$

This method can be used whenever a Nao robot's leg must rotate around its vertical axis. After this 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.

6.2 Slot Behaviors

Adapting the Slot Behavior algorithm was easy, since it only generates joint trajectories based on XML files. Only the behaviors themselves needed to be adapted.

The get up behaviors (after falling forward and backward) were developed from scratch because the get up behaviors 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 [17] and on the B-Human 2010 Code Release [14].

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. Compared to the simulation, the slot durations were increased an average of 23%. The angles of the joints that move on the coronal plane (hip roll and ankle roll) were also increased from 11 to 20 degrees, so the center of mass is better shifted to the support foot.

Was also created a kick to the side that did not exist on the simulation.

6.3 CPG Behaviors

In the CPG Behaviors, like the Slot Behaviors, the algorithm itself was easily adapted but the behaviors needed to be changed. The common modification was to slowdown the behaviors and reduce the joint amplitudes. However, each behavior needed to be independently adapted to the real robot. The forward walk created as a CPG Behavior was adapted from the simulation to the real robot by reducing the joint angle amplitudes 50% and slowing it down 67%. The rotate around the ball was slowdown in 58% and the joint angle amplitudes had an average reduction of 74%.

6.4 OmnidirectionalWalk Behavior

As said above, this behavior produces the desired motion calculating the trajectories of three parameters for each leg, and then uses the Leg Interface to convert these parameters into joint angles. To adapt this behavior from the simulation to the real robot we modified the Leg Interface according to the dimensions of the real robot.

A difference from the Nao robot to most humanoid robots (like the Jupp robot for which Sven Behnke's walk [1] 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 simulation the two joints can be controlled independently. So, when adapting this walk from simulation to the real robot, we need to combine the two 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 these two motions 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. 3. Each leg 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. 3, 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. 4). 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.

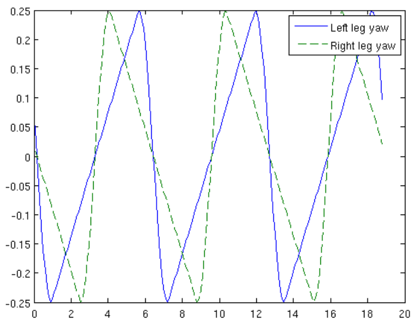

Fig. 3. Yaw motions of the legs in the OmnidirectionalWalk.

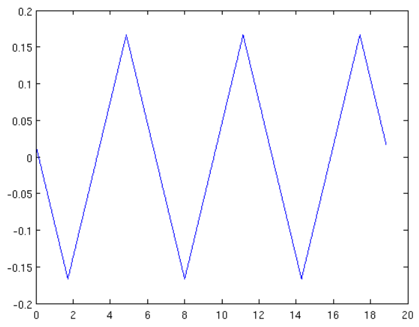

Fig. 4. Common joint motion, result of the combination of the two motions shown in Fig. 3.

6.5 TFSWalk Behavior

This gait was optimized on the simulator, resulting in the fastest gait we have. However, the ground on the simulator has low friction and so 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 more friction, making the robot stumble and fall.

The first approach we took to solve this problem was to add coronal movement, allowing the robot to better shift its Center of Mass (CoM), resulting in a more stable gait. The coronal movement used is shown in Fig. 5 and was based on [15]. 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.

The hip motion is defined using (13) and produces the trajectories shown in Fig. 6. 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 defined and depends on the walking speed. When the latter increases, the coronal movement amplitude decreases.

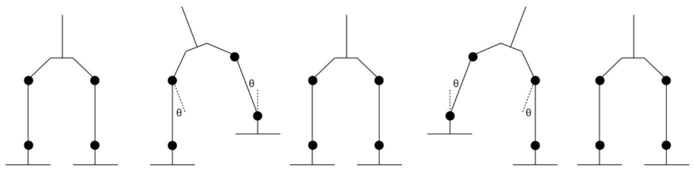

Fig. 5. Coronal movement.

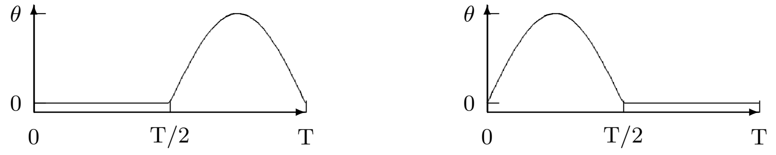

Fig. 6. Left and right hip roll joints trajectories.

$$f(t) = \begin{cases} \theta \sin\left(\frac{2\pi}{T}t\right) & \text{, if } t < \frac{T}{2} \\ 0 & \text{, otherwise} \end{cases} \tag{13}$$

The other approach we took, instead of using coronal movement to shift the CoM, lifts the feet higher and rapidly making use of the robot dynamics to keep the robot balanced. In order to increase the height of the feet trajectories, the knee, hip and ankle joints 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 6.1). With this new model it is easy to control foot height trajectory just changing the leg extension parameter. In addition, the velocity of the robot, controlled by the leg angle amplitude, becomes independent on the foot trajectory height. As an example, the feet may be kept moving up and down in the same place by setting the leg 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.

Feedback was also added to make the behavior more robust against external disturbances, such as uneven ground and collisions with obstacles. The feedback is calculated using a filtered value of the accelerometer in the x direction (front), as shown in (14), and it is based on the rUNSWift Team Report 2010 [12]. This value is summed 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.

$$filAccelX = 0.5 * filAccelX + 0.5 * accelX * 0.2; \tag{14}$$

7 Results

All the described behaviors were tested and are stable.

The get up behaviors developed as Slot Behaviors put the robot in the stand position when the robot is lying down on the ground in 16.5 seconds, when lying on the front, and 8.1 seconds, when lying on the back. The kick forward behavior takes 3.2 seconds to hit the ball and then 2.2 seconds to return to the stand up position. The distance traveled by the ball was measured for five kicks and the average distance was 2.87 ± 0.46 meters. The kick to the side behavior hits the ball in 3 seconds and returns to the stand position in 3.2 seconds. This kick was executed five times and the ball traveled an average distance of 1.05 ± 0.17 meters.

The forward walk created using a CPG Behavior was executed five times in the real robot and had an average velocity 6 ± 0.4 cm/s. The turn in place and the rotate around the ball had an average velocity of 27 °/s. In the simulation, the forward walk had a velocity of 7.33 cm/s, the turn in place had an average velocity of 49.47 ± 0.42 °/s, and the rotate around the ball had an average velocity of 49.98 ± 1.24 °/s

The OmnidirectionalWalk achieved in the real robot a forward velocity of 5 cm/s, a side velocity of 1 cm/s, and rotates at 35 °/s. In the simulation this behavior achieved a forward velocity of 10 cm/s, a side velocity of 3 cm/s, and rotates at 12 °/s.

The TFSWalk was executed 10 times in the real robot and has an average forward velocity of 22 ± 1 cm/s and rotates at 25 °/s. In the simulation the TFSWalk behavior achieves a forward velocity of 51 cm/s and rotates at 42 °/s.

8 Conclusion

This paper described the adaption of behaviors from the simulation to a real robot. All presented behaviors were successfully adapted and tested on a real robot. It is harder to develop behaviors for a robot in a real environment instead of simulated environment because of factors that make behaviors unstable, namely: external disturbances; uneven ground; or motor strength that affects the precision the joints follow the 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, making the behavior 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.

Was also presented in this paper 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 a 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 robot. This allows to use the same agent on both the RoboCup 3D Simulation League and the Standard Platform League.

Future work include the optimization of the adapted behaviors in the real robot, making them more stable and faster.

References

1. Behnke, S.: Online trajectory generation for omnidirectional biped walking. In: Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on. pp. 1597 –1603. Orlando, Florida, USA (May 2006)
2. Boedecker, J., Asada, M.: Simspark concepts and application in the Robocup 3D Soccer Simulation League. In: Proceedings of SIMPAR-2008 Workshop on The Universe of RoboCup simulators. pp. 174–181. Venice, Italy (Nov 2008)
3. Conceição, A., Moreira, A., Costa, P.: Design of a mobile robot for Robocup Middle Size League. In: 6th Latin American Robotics Symposium (LARS-2009). pp. 1–6. Chile (Oct 2009)

4. Gouaillier, D., Hugel, V., Blazevic, P., Kilner, C., Monceaux, J., Lafourcade, P., Marnier, B., Serre, J., Maisonnier, B.: The NAO humanoid: a combination of performance and affordability. ArXiv e-prints (Jul 2008)
5. Gouaillier, D., Hugel, V., Blazevic, P., Kilner, C., Monceaux, J., Lafourcade, P., Marnier, B., Serre, J., Maisonnier, B.: Mechatronic design of NAO humanoid. In: Robotics and Automation, 2009. ICRA '09. IEEE International Conference on. pp. 769 –774. Kobe, Japan (May 2009)
6. Graf, C., Härtl, A., Röfer, T., Laue, T.: A robust closed-loop gait for the Standard Platform League humanoid. In: Zhou, C., Pagello, E., Menegatti, E., Behnke, S., Röfer, T. (eds.) Proceedings of the Fourth Workshop on Humanoid Soccer Robots in conjunction with the 2009 IEEE-RAS International Conference on Humanoid Robots. pp. 30–37. Paris, France (Dec 2009)
7. Kulk, J.A., Welsh, J.S.: A low power walk for the NAO robot. Australasian Conference on Robotics and Automation (ACRA) (Dec 2008)
8. Lau, N., Reis, L.P.: FC Portugal - high-level coordination methodologies in soccer robotics. In: Lima, P. (ed.) Robotic Soccer, pp. 167–192. InTech (Dec 2007)
9. Liemhetcharat, S., Coltin, B., Çetin Meriçli, Tay, J., Veloso, M.: Cmurfs: Carnegie mellon united robots for soccer (2010)
10. Neves, A.J.R., Azevedo, J.L., Cunha, B., Lau, N., Silva, J., Santos, F., Corrente, G., Martins, D.A., Figueiredo, N., Pereira, A., Almeida, L., Lopes, L.S., Pinho, A.J., Rodrigues, J., Pedreiras, P.: CAMBADA soccer team: from robot architecture to multiagent coordination. In: Papić, V. (ed.) Robot Soccer, pp. 19–45. InTech (Jan 2010)
11. Picado, H., Gestal, M., Lau, N., Reis, L., Tom, A.: Automatic generation of biped walk behavior using genetic algorithms. In: Cabestany, J., Sandoval, F., Prieto, A., Corchado, J. (eds.) Bio-Inspired Systems: Computational and Ambient Intelligence, Lecture Notes in Computer Science, vol. 5517, pp. 805–812. Springer Berlin / Heidelberg (Jun 2009)
12. Ratter, A., Hengst, B., Hall, B., White, B., Vance, B., Sammut, C., Claridge, D., Nguyen, H., Ashar, J., Pagnucco, M., Robinson, S., Zhu, Y.: rUNSWift team report 2010 Robocup Standard Platform League (Oct 2010)
13. Reis, L.P., Lau, N.: FC Portugal Team Description: RoboCup 2000 Simulation League Champion. In: Stone, P., Balch, T., Kraetzschmar, G. (eds.) RoboCup 2000: Robot Soccer World Cup IV. Lecture Notes in Computer Science, vol. 2019, pp. 29–40. Springer-Verlag, London, UK (Jun 2001)
14. Röfer, T., Laue, T., Müller, J., Burchardt, A., Damrose, E., Fabisch, A., Feldpausch, F., Gillmann, K., Graf, C., de Haas, T.J., Härtl, A., Honsel, D., Kastner, P., Kastner, T., Markowsky, B., Mester, M., Peter, J., Riemann, O.J.L., Ring, M., Sauerland, W., Schreck, A., Sieverdingbeck, I., Wenk, F., Worch, J.H.: B-Human team report and code release 2010 (Oct 2010), only available online: `http://www.b-human.de/file_download/33/bhuman10_coderelease.pdf`
15. Shafii, N., Reis, L., Lau, N.: Biped walking using coronal and sagittal movements based on truncated fourier series. In: Ruiz-del Solar, J., Chown, E., Plöger, P. (eds.) RoboCup 2010: Robot Soccer World Cup XIV, Lecture Notes in Computer Science, vol. 6556, pp. 324–335. Springer Berlin / Heidelberg (Jun 2011)
16. Smith, R.: Open dynamics engine. `www.ode.org` (Jul 2008)
17. Tay, A.J.S.B.: Walking Nao Omnidirectional Bipedal Locomotion (Aug 2009)