Aibo and Webots: Simulation, Wireless Remote Control and Controller Transfer

Lukas Hohl, Ricardo Tellez, Olivier Michel and Auke Jan Ijspeert *

School of Computer and Communication Sciences, Swiss Federal Institute of Technology, Lausanne (EPFL)

Abstract

This article introduces a new software tool providing an accurate simulation of the Sony Aibo robots and the capability to transfer controller programs from the simulation to the real robot. Five components are described: 1) a simulated physics-based model of the Sony Aibo ERS-210(A) and ERS-7 quadruped robots, 2) a graphical user interface for controlling the simulated and the real robot, 3) a wireless communication protocol for controlling the robot from within Webots, 4) software components on the robot that enable remote control, and 5) a method for cross-compiling Webots robot controllers. The complete system has been calibrated and proof tested. It enables simultaneous control of both a simulated and a real Aibo robot and provides the user with a platform for convenient robot programming without any knowledge of the underlying robot firmware.

Key words: Aibo, Webots, robot simulation, remote control, cross-compilation

1 Introduction

Simulations play an important role in robotics research. In comparison with real robot investigations, simulations are easier to setup, less expensive, faster, more convenient to use and they allow the user to perform experiments without the risk of damaging the robot. Building up new robot models and setting up experiments only takes a few hours and control programs can be extensively

* Corresponding author
Email addresses: `lukas.hohl@bluewin.ch` (Lukas Hohl), `r_tellez@ouroboros.org` (Ricardo Tellez), `Olivier.Michel@cyberbotics.com` (Olivier Michel), `auke.ijspeert@epfl.ch` (Auke Jan Ijspeert).

tested on a host computer, offering convenient debugging facilities. Simulators are especially preferred when using time-consuming algorithms for learning or evolution of intelligent controllers. Simulation are really useful if their results can be easily transfered onto real robots. Simulators integrating remote control facilities provide additional advantages. They enable fast transitions from simulations to real robots without controller program transfers and thus they significantly speed up testing procedures, exploration of the robot's capabilities and simulation calibration. Nowadays, wireless communication protocols, offering easier handling and greater range, are an interesting alternative to cable connections between a host computer and a robot.

Simulators of course do have limitations and there are some problems to be solved. It is difficult to realistically simulate the physics of a robot (actuators, interaction with the environment, sensors) and the transfer from simulation to the real robot is not always simple.

One of our main motivations was the development of a robotics research tool with a variety of applications, e.g., fast design and testing of controllers, simulation calibration, use of the robot as a programming interface or interaction between real and simulated environments. We wanted to provide the user with a convenient environment where she/he is able to program the robot independently of the underlying firmware and as close to simulation control as possible.

The Webots^{TM} mobile robotics simulation software developed by Cyberbotics provides the user with a rapid prototyping environment for modelling, programming and simulating mobile robots. Webots relies on ODE (Open Dynamics Engine) to perform accurate dynamic physics simulation. With Webots it is possible to define and modify a complete mobile robotics setup, even several different robots sharing the same environment.

Aibo^{TM} is a four-legged robotic dog produced by Sony. Despite the fact that it is principally sold as an entertainment robot system (at an affordable price), it has powerful capabilities such as wireless network communication, a wide range of input and output devices such as speaker and microphone, color camera, distance sensor, acceleration sensors, various touch sensors, LEDs and of course controllable joints. What makes the robot even more interesting (besides its sophisticated hardware), is the possibility to write advanced programs for Aibo using the SDKs provided by Sony. The binary files are put onto a Memory Stick^{TM} which is plugged into Aibo.

In this article, we first make an overview of related work (section 2). We then describe the system as a whole (section 3). The reader can then learn how we developed an Aibo simulation model in Webots (section 4) and added a graphical user interface for its control (section 6) in addition to the control by a controller program which is a fundamental concept in Webots (section 7). We also implemented wireless remote control facilities for a real Aibo robot, making it possible to command the robot by the means of Webots controller programs or via the same graphical user interface that is used for simulation control. To achieve this, we designed a communication protocol and programmed a special software for Aibo (section 5). The protocol is not restricted to Aibo or Webots and might be reused for the control of other robots or simulators. The concepts adopted for programming the robot are not limited to remote movement control. Finally, we implemented the cross-compilation of Webots controller programs for direct execution on Aibo's hardware (section 8). Based on this work, other languages than the Webots robot controller programming interface could potentially be used for cross-compilation. Some examples of applications (section 9) and a detailled evaluation (section 10) are presented before future work (section 11) and the final conclusion (section 12).

2 Related Work

We explain in this section what were the features for robot remote control and cross-compilation in Webots before we included these functionalities for Aibo (subsection 2.2). The possibilities for Aibo programming that exist are also presented (subsection 2.1) as well as other Aibo and robot simulators (subsection 2.3).

2.1 Aibo Software Development Environment

Sony is promoting the Aibo Software Development Environment (SDE) for the creation of software that either executes on Aibo or on a PC and controls Aibo by using a wireless LAN. The SDE is provided free of charge and contains three Software Development Kits (SDKs) and the Aibo Motion Editor. The OPEN-R^{TM} SDK is typically used for research in robotics programming. The R-CODE SDK is used with a scripting language and the Aibo Remote Framework is a development environment for Windows PC applications that can control Aibo via wireless LAN. Different SDKs can not be combined. The Aibo Motion Editor creates MTN files containing joint positions over a certain number of so called key frames for the usage in the three SDKs.

In all development environments except the OPEN-R SDK only predefined movements can be played back. The OPEN-R SDK is a cross-development environment based on gcc (C++) offering the greatest flexibility. Programming with the OPEN-R SDK is the only possibility to create software that exploits Aibo's hardware limits. The remote control and cross-compilations software we implemented for Aibo puts additional layers of abstraction above OPEN-R and therefore programming the robot becomes much easier.

2.2 Transfer from Webots to real robots

The Webots simulation software already include transfer capability for a number of commercially available robots. This includes the Khepera robot (http://www.k-team.com), the Lego Mindstorms^{TM} (http://mindstorms.lego.com) and the Hemisson robot (http://www.hemisson.com). For these robots, different transfer system have been developed including remote control and cross-compilation.

The remote control mode consists in redirecting the inputs and outputs of a Webots controller program to a real robot using a wired or wireless connection between the robot and the computer. This means that a special program has to be running on the real robot to interface webots requests to the robot's hardware. In the second mode, the controlling program is cross-compiled on the host computer and downloaded to the robot which in turn executes the controller on board and no longer interacts with the host computer.

The cross-compilation consist in using a cross-compiler to compile the controller program on the computer for the target robot. The resulting binary program that can be uploaded onto the real robot and executed autonomously, i.e., without any interaction with the computer.

2.3 Aibo and Robot Simulators

There is currently several 3D simulators available that are either specifically designed for the Aibo robot, usually for the Sony Four-Legged Robot League (http://www.tzi.de/4legged/) of the Robocup competition, or general purpose simulators that include or may include Aibo simulation. For the RoboCup competition, multiple teams have developed their own Aibo simulators, but only a few of them are publicly available and documented.

The AISim simulator from the ASURA Robocup team [4] [22] is one of the open-source simulators available for the specific simulation of Aibo within the Robocup environment. This simulator allows for the simulation of a soccer match, including all type of interesting processes like vision processing, planning and behavior generation, but not including a detailed simulation of the Aibo robot, that is treated as a unique block. It can generate controllers through the simulation and transfer them to the real robot, but these are high level controllers, being not possible to issue low level commands to control every joint, and to perform experiments outside the Robocup framework is not possible.

The German team simulation, called SimRobot [11] [23], is a generic 3D robot simulator relying on an XML description for the modeling of the robots and their environment. The robot controllers are directly linked with the simulator library to produce an executable file. A preliminary simulation of rigid body dynamics by using the ODE library was included. The sensor library includes cameras (OpenGL rendering), distance sensors and bumpers. The actuator library appear to be limited to motors. This simulator was designed for the Aibo RoboCup competition, but is not any more restricted to the simulation of a RoboCup soccer match, other setups and other types of robots are now allowed. However, in difference to Webots, it has a smaller library of sensors and actuators and special domains like underwater environments or simulation of flying robots are not supported. Furthermore, transfer of controllers from simulation to real robot is not possible.

UCHILSIM [20] is a 3D robot simulator developed by the University of Chile. It is specially designed for the RoboCup Four-legged League. The simulator contains dynamics simulation using the Open Dynamics Engine (ODE) [12], a graphics engine, and has a window based graphical interface. The environment and the robots are described in a VRML structure extended by nodes for simulator elements and physical attributes. It has interfaces to their UChile1 software package and their learning component. At the current stage, this simulator is rather specific to the Aibo RoboCup competition.

In addition to the simulators mainly dedicated to the simulation of Aibo, there exist other general simulators that could in principle be used for the simulation of Aibo, everthough, is the user himself that has to create the whole model of the Aibo robot on them, since it is not provided by the simulator environment like in Webots.

In this category stands the Gazebo simulator [24] from the Player/Stage project [25]. This is an open source simulator which gives you the tools to simulate your own robot, includen several types of sensors and actuators. It provides off the self simulation models for Pioneer robots, but not for Aibo. Nevertheless, a simulation of the Aibo robot is possible in theory using this simulator.

Another simulator in the same line is the Ubersim simulator [26] by the Carnegie Mellon University. Like Webots, it relies on ODE for the simulation of the dynamics of the system, and allows the creation of the user own robot, eventhough the types of sensors currently available are a little limited. Ubersim has a focus on vision-centric robots in dynamic environments and includes Aibo models. It has a client/server based architecture, where clients communicate with the server over TCP/IP. The simulator also uses ODE for dynamics simulation. Own robots can be modeled by programming their structure in C classes. In the current release, only two sensors are predefined: a camera sensor and an inclinometer. At the moment there seems to be no graphical user interface for interacting directly with the robots or the environment during simulation time.

A lot of work on robot simulators in general has been done by Jerry Pratt at the MIT Leg Lab [10] and Oussama Khatib at Stanford University [5]. They developed various simulation software for mobile robot, including very advanced physics simulation, but none of them were specific to the Aibo robots, nor intended to be reused or adapted by Aibo robot users. Hence, covering the details of these interesting simulation tools is out of the scope of this paper.

3 System Overview

We conceived a system that allows a Webots user to control both a real Sony Aibo robot and its physical simulation. We currently support the Aibo ERS-210(A) model equipped with an ERA-201D1 wireless LAN card and the newer Aibo ERS-7 model which has a built-in wireless LAN card. The system consists of five developments: 1) a simulated physics-based model of Aibo (section 4), 2) a GUI for controlling the simulated and the real robot (section 6), 3) a wireless communication protocol for controlling the robot from within Webots (sections 5 and 7), 4) software components on the robot that enable remote control (section 5), and 5) a method for cross-compiling Webots robot controllers (section 8). Fig. 1 illustrates the main components.

There are three modes in which Webots and Aibo can be used: Simulation, wireless remote control, cross-compilation. All modes can be active in parallel and numerous combinations for controlling the robot and/or its simulation are possible. When working in simulation or remote control mode, command input can be given by a graphical user interface (two arrows starting at GUI window on Fig. 1) or by a Webots controller program compiled and running on the host computer (two arrows starting at source file icon on Fig. 1). In cross-compilation mode, Webots controllers written in C or C++ are cross-compiled on the host computer and wirelessly transferred to Aibo where the commands are executed without user interaction (arrow from source file icon to Aibo on Fig. 1). Giving commands (in the GUI or in controller programs) can either mean to individually control joints and other robot components or to start the playback of MTN files containing predefined movements.

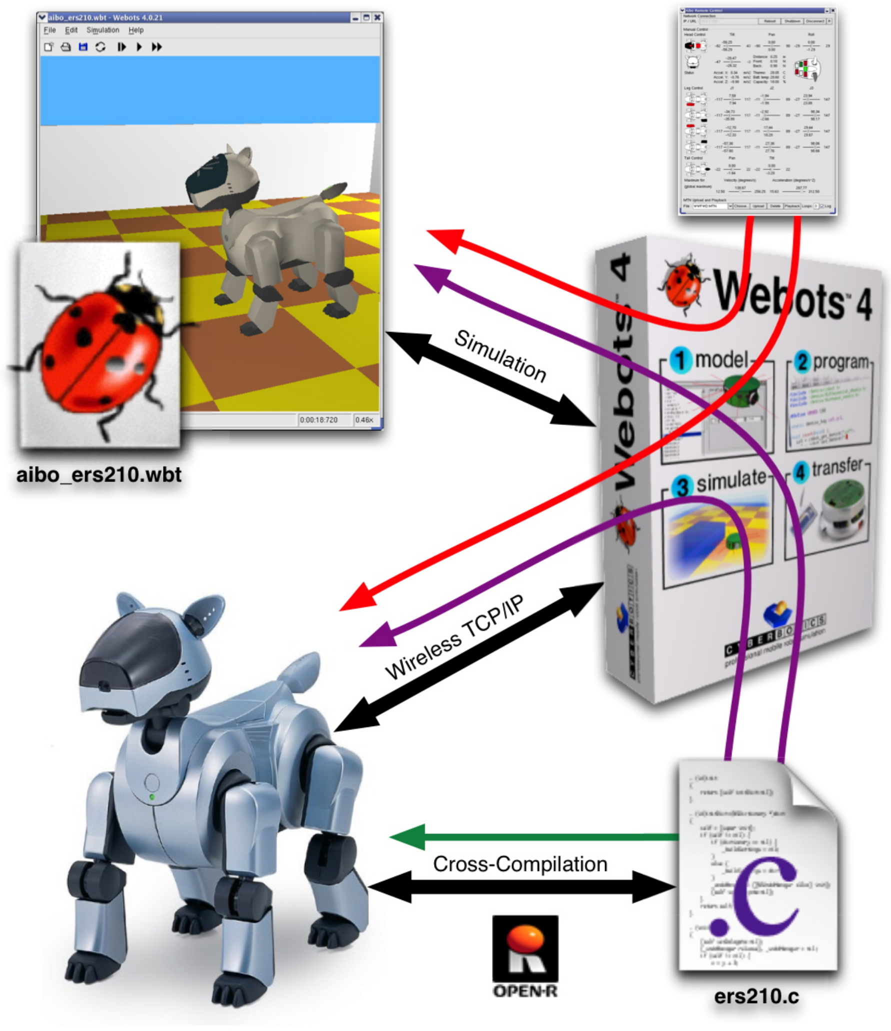

Fig. 1. Operating modes: simulation, remote control and cross-compilation

4 Simulation

This section describes how we developed a simulated model of Aibo in Webots. We first explain how models can be constructed in Webots in general (subsection 4.1) and then discuss how we represent Aibo's hardware with the capabilities of Webots (subsection 4.2). The Aibo types that are used in research because they have comparable capabilities and can be programmed with the OPEN-R SDK are ERS-210, ERS-220 and ERS-7. More information on the different Aibo models can be found in [1–3,14–16].

4.1 Webots

Webots is a physics based general purpose mobile robotics simulation software. A demo version of the most recent Webots version is available from *http://www.cyberbotics.com.* The main components of Webots are the *world* (one or more robots and their environment), the *supervisor* (a user written program to control an experiment, i.e., to change and observe the world) and the *controller* of each robot (a user written program defining its behavior).

The user can chose from a library of robot models (and modify them) or construct her/his own models. Each robot can be equipped with a large number of available sensors and actuators. For each object, a number of properties can be defined, such as shape, color, texture, mass, friction, etc. The user can then program the robots using an arbitrary development environment, simulate them and optionally transfer the resulting programs onto a real robot.

4.2 Webots Model of Aibo

We developed a model of the Aibo robot. This involved implementing its graphical aspect, replicating its kinematic structure, its dynamics properties (masses and moments of inertia) and its control.

The graphical representation was imported from a graphical model provided by Sony. The dimensions were determined using the official model information [14–16] and the weights were estimated based on measured block weights. The model parts have a uniform mass distribution inside of their bounding box. Fig. 3 shows the bounding boxes of those objects having a mass.

All robot actuators and sensors are called primitives in the OPEN-R terminology. The robot body is the only simulation model part which does not correspond to a robot primitive. Fig. 2 shows the hierarchy relations between all nodes of the Aibo ERS-210 model. The Aibo ERS-7 model has a very similar structure.

The Webots Servo node models a servo motor, which is adequate for the simulation of Aibo's joints. The Servo node also simulates a position sensor. All the parts of the model having a mass, except the body, are Servo nodes. In Webots, a servo can be controlled in position, maximum velocity, maximum force and acceleration from the Servo programming interface.

Aibo's Position Sensing Device (PSD) is modeled by a DistanceSensor node of type "infra-red" (see Fig. 3). We modeled Aibo's paw touch sensors by TouchSensor nodes which return binary values. Because only the paw touch sensors

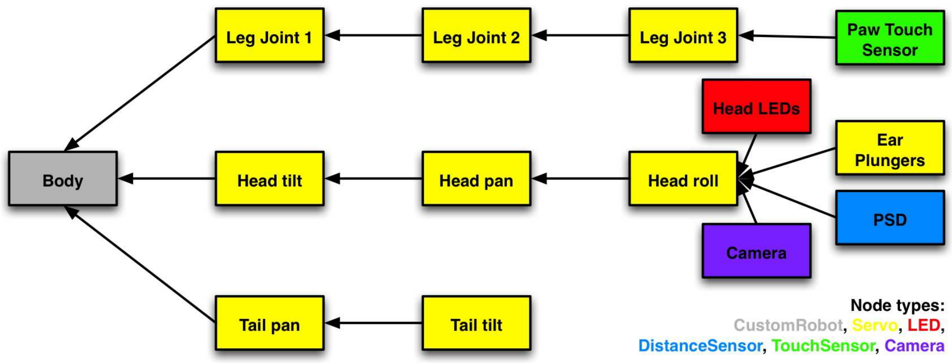

Fig. 2. Aibo ERS-210 Model Node Hierarchy

are of interest for the simulation of Aibo's movements, the back sensors, the chin switch and the head sensors are not yet included in the model. Acceleration and thermo sensors do not have corresponding Webots nodes and can therefore not be simulated. Similarly, speaker and microphone can't be simulated in Webots, but there is a Camera node type, which we used to include Aibo's color camera in the model.

In section 10, the accuracy of the physics-based mode is extensively analysed.

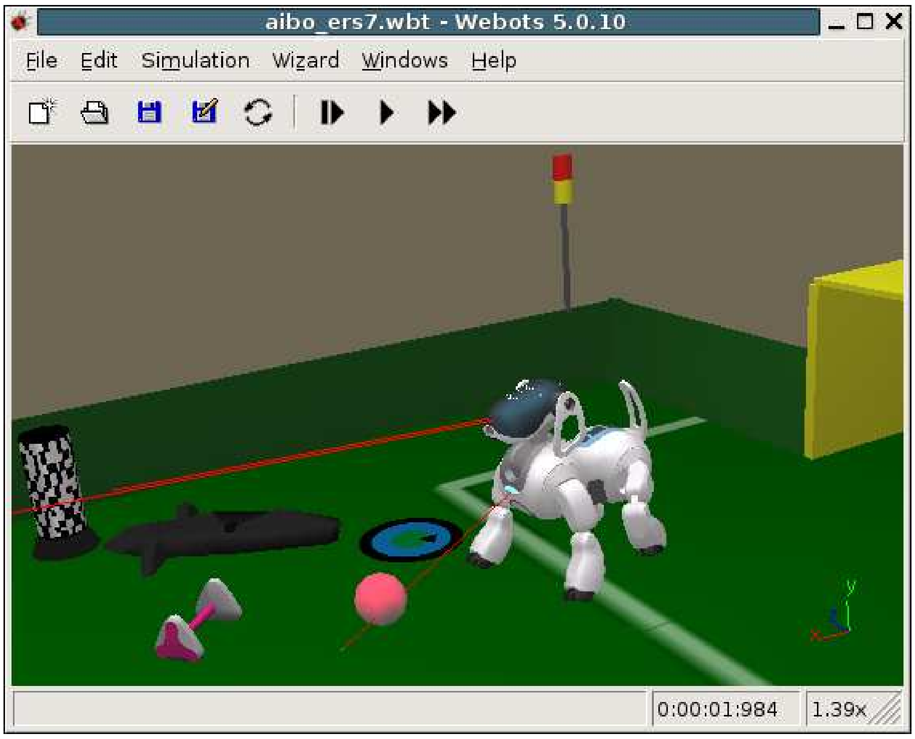

Fig. 3. Simulation of Aibo ERS-7 in Webots

5 Control of the Real Robot

In this section, we discuss the developments made to achieve the remote control of a real Aibo robot. The general concept of OPEN-R software is also treated (subsection 5.2). The remote control functionalities that we added to Webots require a computer equipped with an IEEE 802.11b compliant wireless LAN card. This is the client side. It establishes a wireless TCP/IP connection with its server counterpart on Aibo, a special OPEN-R software we developed (subsection 5.3). Then Aibo and Webots exchange messages defined in our communication protocol (subsection 5.1).

5.1 Communication Protocol

We developed a message protocol between Aibo and Webots. The messages are inspired by a command line interface. There are commands for the reading of sensor values, the control of LEDs, plungers and joints (position, speed, acceleration) and for (MTN) file handling. Aibo's answer is always the current value of the read sensor or the value that was actually taken into account for a piloting command. New commands are only read by Aibo when the previous response was successfully sent back. Aibo does not send any data that is not requested by a command. Thus the command protocol used by Aibo and the client is synchronous.

5.2 OPEN-R

Application software for Aibo consists of several OPEN-R software modules called "objects". An object corresponds to one executable file. Each object has its own thread of execution. Processing is performed by multiple objects with various functionality running concurrently and communicating via inter-object communication on their connection ports (entry points).

The OPEN-R system layer provides a set of services (e.g., output of control data to joints and input of data from various sensors) as the interface to the application layer. This interface is also implemented by inter-object communication. The system layer also provides the interface to the TCP/IP protocol stack, which enables the creation of wireless networking applications.

5.3 Remote Control Software

The remote control software on Aibo that we developed as the counterpart of Webots consists of four OPEN-R objects. Fig. 4 shows them as filled circles. PowerMonitor is isolated from the other objects. JointMover, RCServer and Controller, however, have multiple entry points (in addition to the default entry points, which are not shown). RCServer possesses additional entry points necessary for network communication (not drawn individually). The entry points are labeled with the service names. OVirtualRobotComm is part of Aibo's system software and provides services for sending commands to the robot, reading sensor values and retrieving images from the camera. Joint and LED commands need to be passed to the robot in the form of command vectors containing commands for multiple primitives and over a certain period of time. We will next describe the functionality of the four other objects on Fig. 4.

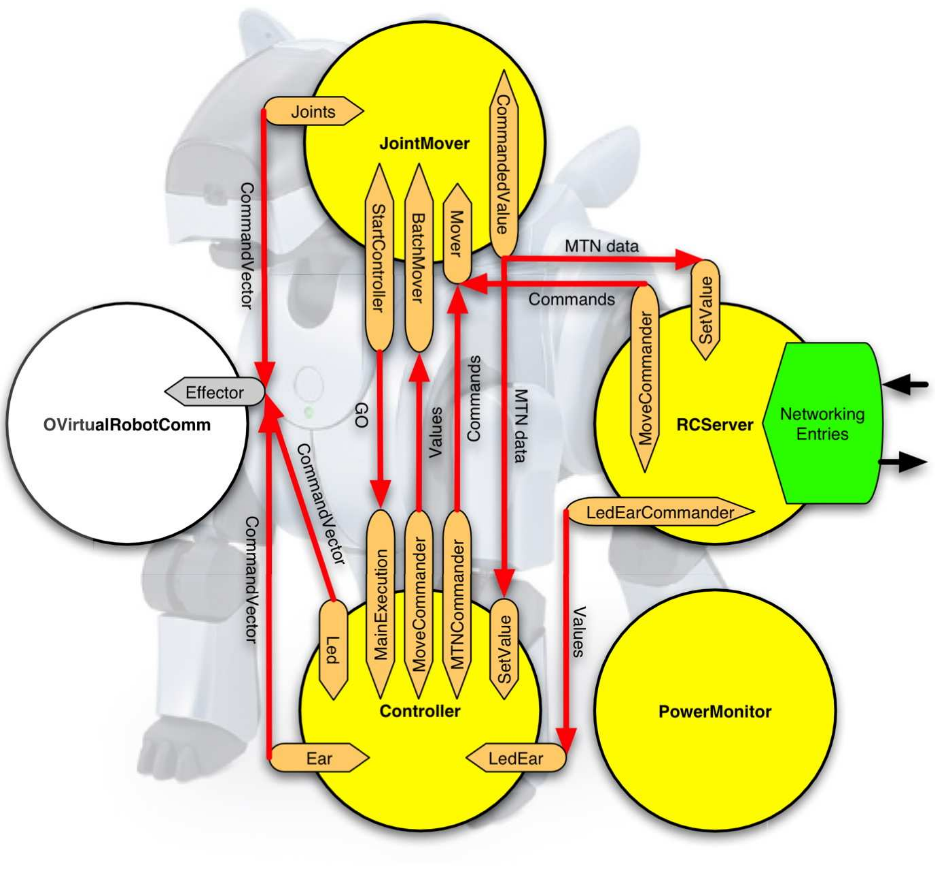

Fig. 4. OPEN-R Objects on Aibo

5.3.1 PowerMonitor

In PowerMonitor, every change in the robot and battery status is observed. Aibo is immediately shut down when the pause switch is on, the battery capacity is low or when Aibo is connected to an external connector.

5.3.2 RCServer

The RCServer object listens for TCP connection requests and then handles all the network traffic with the client. It interprets the commands received and performs the appropriate actions, either directly (sensor reading) or by delegating them to specialized objects (JointMover, Controller). The answer to a command is generated and returned over the same network connection.

5.3.3 Controller

For the cross-compilation of robot controller, we developed the Controller OPEN-R object which can integrate custom Webots controller programs. Controller implements the Webots controller programming interface and the controller main loop, including the exact timing of consecutive loop iterations. Controller also treats LED and ear commands. Section 8 gives more details on the role of the Controller object for the cross-compilation of Webots controller programs.

5.3.4 JointMover

The JointMover object executes all the commands concerning joints (setting positions, velocities, accelerations and playback of MTN files). It receives them from RCServer and Controller.

Speed and acceleration are concepts not foreseen in OPEN-R. We had to implemented ourselves the algorithm to smoothly reach individual goal positions respecting a user given speed and acceleration limit.

In the algorithm for individual commands, the command vector frames (position values, one frame lasts 8 ms) are filled by iteratively adding or subtracting the acceleration or maximum speed value (per frame) to the current velocity, depending on the current phase. In order to obtain smooth trajectories, we designed JointMover to decompose every movement in the three following phases. The duration of every phase (a certain number of frames) is reevaluated every time a joint receives a new command:

- acceleration

- constant speed
- deceleration

MTN files (see section 2.1) contain joint positions and other commands for a certain number of so called “key frames” and thus completely define a sequence of movement. Besides the control of individual joints using our algorithm, JointMover can also play back MTN files. As soon as an MTN playback command comes in, JointMover pilots Aibo to the position specified by the first key frame in the file (using the algorithm for individual smooth joint commands). Afterwards, the MTN file is executed as many times as loops are desired by simply mapping the file content to consecutive command vectors. No calculations as for individually commanded goal positions are done. When Aibo has successfully reached the final posture, JointMover again accepts commands.

Fig. 5 shows how both a simulated and a real Aibo robot execute one cycle of a forward walking movement defined by an MTN file. Eight positions are shown on the figure, whereas the MTN file contains approximately 80 key frames to specify the joint trajectories.

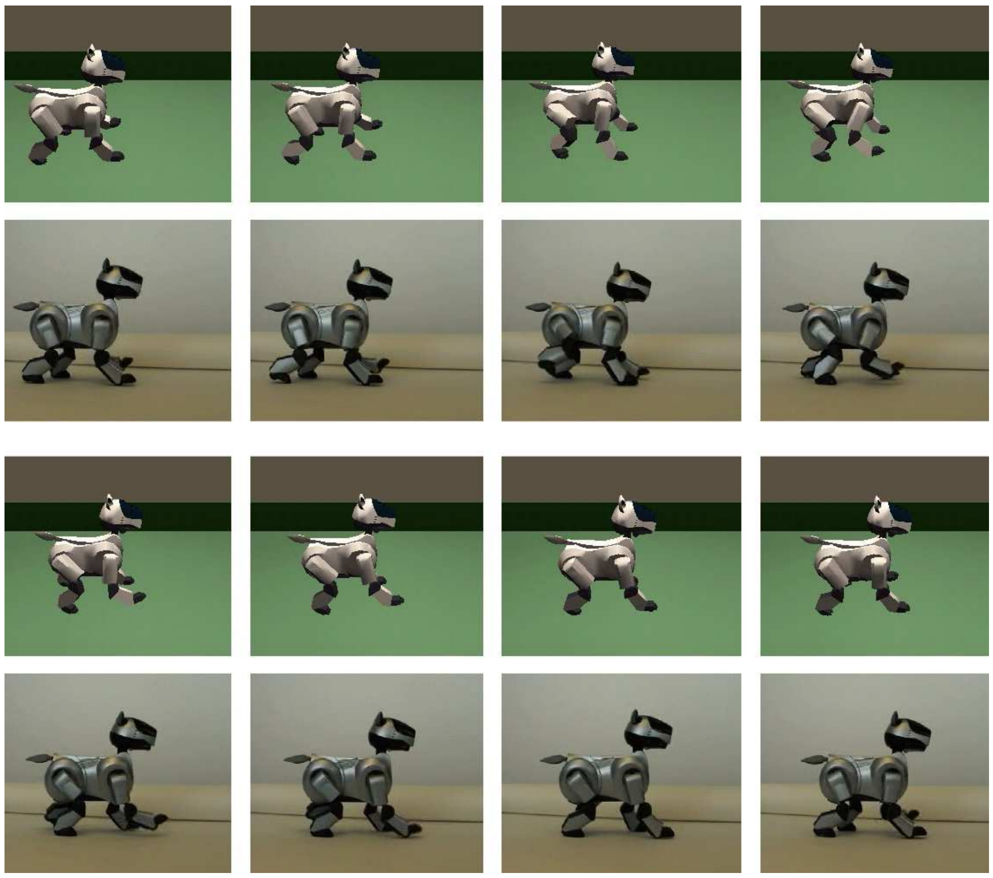

Fig. 5. MTN File Playback: Walking Cycle

6 Graphical User Interface

We discuss here the graphical user interface that we added for the manual control of both the simulation and the real robot. The 3D view is an integral part of Webots and we did not have to implement it. Every simulated Aibo has its own instance of the graphical user interface for controlling the simulation and/or a real robot. There is no theoretical limit on the number of Aibos that can be simulated in parallel. In practice, it is possible to simulate a soccer match (4 robots in each team) in real time on a high-end personal computer (see Fig. 7).

The interface is divided into two parts, actually tabs, with logically grouped controls. On tab includes configuration items (like the IP address the real robot) while the other tab includes individual controls for robot devices (like servos, leds, sensors, etc.).We specially designed the (interactive) robot pictures and the sliders, the other controls are generic. Fig. 6 shows how the interface and the controlled model look like when both the simulation and the real robot are commanded.

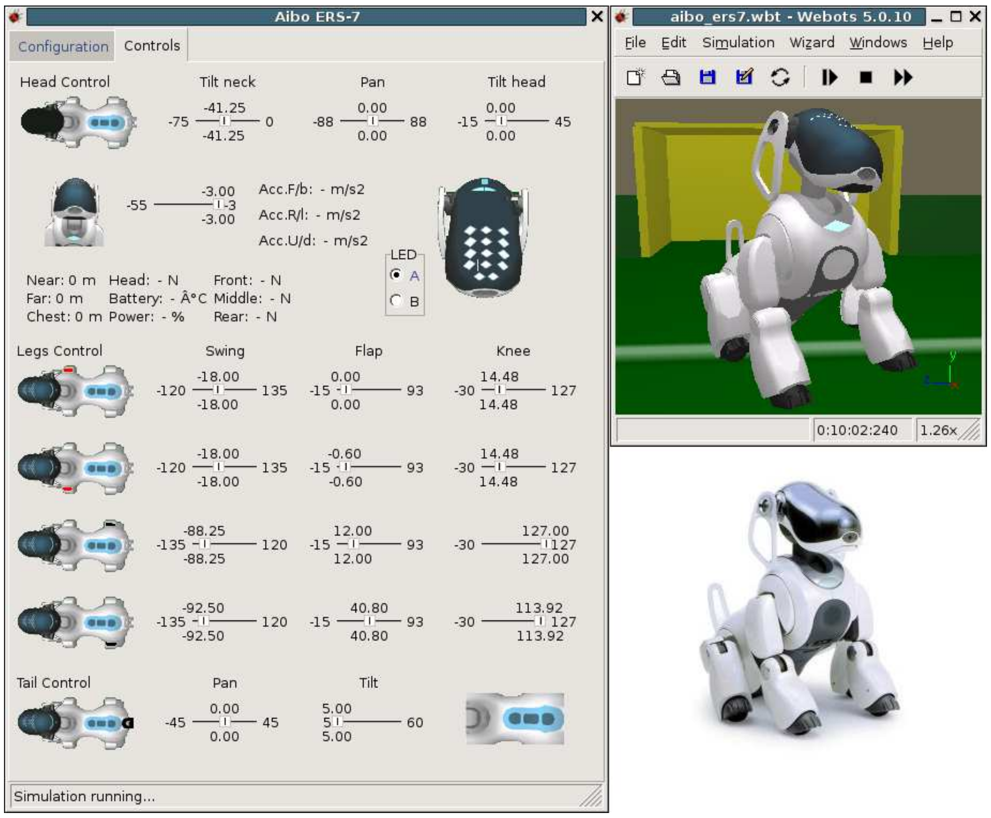

Fig. 6. Simulation and Robot controlled by Graphical User Interface and Controller

The simulation is always controlled when no connection to a real Aibo is open. Once a connection is established, simultaneous control of the simulation and the real robot can be turned on and off. All this is done in the top part of the interface. The bottom part of the interface handles the upload and playback of MTN files, possibly with multiple loops.

In the middle part, head lights and ears are represented as a sliced bitmap. The slices change their appearance when they are clicked and send a command to the real robot and/or call the appropriate method of the simulated device. Every joint has an associated slider. These sliders contain a position indicator which is set according to the read sensor value. The moveable part of the slider indicates the desired goal position Aibo will go to with respect to velocity and acceleration limits. There is a slider for the global velocity limit and one for the acceleration limit. If both the real and the simulated robot are controlled at the same time, only the sensor values of the real robot are displayed.

7 Controller Program

This section explains how we implemented simulation and remote control by Webots controller programs. We explain what controller programs are (subsection 7.1) and how they can interact with the real robot (subsection 7.2).

7.1 Webots Robot Controller Programs

Webots robot controller programs are typically written in C and they are used to control a simulated robot in Webots. The Webots controller programming interface is usually the only way for the user to give commands to her/his simulation models. No modification had to be made in Webots in order to program the control algorithms for the Aibo robots, as Webots controller programming relies on standard Webots devices included in the models of the Aibo robots.

7.2 Remote Control by Controller Programs

Remote control functionalities for controller programs are foreseen in the concept of Webots. A special function can be called when the value of a simulated device is read or when it is given a command by a controller program. In our implementation of this function, the simulation system is overridden when connected to Aibo by setting (via Webot's internal programming interface) the simulated sensor value to the measurement received from Aibo. Thus the object representing the simulated sensor is actually a representation of the real sensor on Aibo. As a consequence, the call to the sensor reading function in the controller program returns the real value. For actuators, it is checked whether the commanded value differs from the last value set by the controller program and a corresponding command is sent to Aibo if necessary.

8 Cross-Compilation

In this section, we describe how we achieved the controller cross-compilation for direct execution on Aibo (subsection 8.1) while taking into account Aibo's programming scheme (subsection 8.2).

Cross-compilation has the advantage that the robot can run independently of the host computer and no network connection is needed. This potentially enables applications where a large amount of data must be handled, e.g., image and sound processing. We provide a graphical interface for the uploading of binary files in Webots, but the files can also be transfered by other means, e.g., a Memory Stick Reader. When Aibo boots the next time, the new binary file is loaded and executed.

8.1 Controller Cross-Compilation

What was developed for controller cross-compilation is not really a cross-compiler but an OPEN-R object which can integrate custom Webots controller programs without modifications on the controller code itself. This OPEN-R object is called Controller and it runs concurrently with other objects (see Fig. 4) and a robot running a cross-compiled compiler can still receive and execute commands coming in over the wireless network connection. The actual cross-compiler is Sony's OPEN-R cross-compiler included in the OPEN-R SDK, which has to be previously installed in order to take advantage of the cross-compilation functionality in Webots. Because OPEN-R programs are written in C++, it was rather straightforward to combine Webots controller program files and the source code files of the Controller object. The Controller object source code contains Aibo specific implementations of the most important and most interesting functions of the controller programming interface. Every cross-compilation process creates a new binary file that replaces the complete Controller object on Aibo (see Fig. 4).

The major problem for cross-compilation was the infinite loop in the Webots controller main function. Aibo's programming scheme is contrary to writing infinite loops in order to have a program running without interruption. The whole life cycle of an object is an infinite loop, but it is an implicit infinite loop, consisting of the continuous reception and sending of messages from and to other objects.

With respect to this particularity, it must be guaranteed that all OPEN-R methods terminate. In order to make Controller compatible with this programming scheme, we had to extend the Webots controller programming interface in order to replace the explicit infinite loop by a run function which is implicitly repeated infinitely.

There are no timing or wait methods in OPEN-R. The most convenient and precise way to respect the desired time step between consecutive calls to the run function is to take advantage of the period for LED commands. For every LED value in a command vector, an illumination duration (period) can be specified in Aibo's basic time unit (frame of 8 ms). We thus set the period of all LED commands to the time step selected in the controller program (should be a multiple of 8 ms).

9 Examples of Applications

Fig. 6 shows a situation where both the simulation and the real robot are commanded. Parallel control of both the simulation and the real robot can be used for direct comparison and has proven to be a convenient testing and debugging tool during the construction and calibration of the simulated model.

Fig. 7 shows two simulation-only scenarios, i.e., with multiple robots simultaneously simulated and on a ground with special properties.

In one of the controllers that were tested for cross-compilation, the motors of the front left leg are turned off and the values of this leg's three joints are read and passed to the other three legs as their goal positions. This allows a user to move the front left robot leg while the others are quickly following on a trajectory generated by JointMover. This suggests than an Aibo robot running a cross-compiled controller could be used as a programming interface for learning trajectories, i.e., the trajectory is recorded on the robot. Furthermore, simulation could directly copy or record the positions of the real robot by running a simple controller that just reads sensor values in Webots

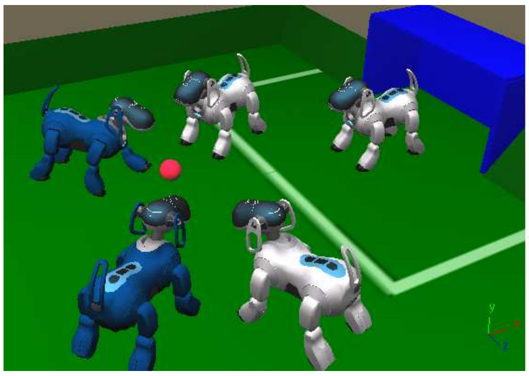

Fig. 7. Simulation of multiple ERS-7 robots (left) and an ERS-210 robot on rough terrain (right)

in remote control mode.

Another controller tested for cross-compilation was a program that consecutively plays back an MTN file for forward walking, but plays back a backward walking movement when an obstacle is detected by the distance sensor at a distance of 20 cm or less. Only a few lines of code were needed for both cross-compilation examples. They show how simple it is thanks to the Webots controller programming interface to write relatively complex behavior.

A more complex and extensive use of the cross-compilation was done in [27] were a group of 24 recurrent neural networks were evolved using the simulation in order to allow the Aibo robot to walk. The evolved controller was then cross-compiled and transferred the robot with no significant difference between simulation and real robot.

10 Evaluation of the simulation acuracy

In this section we qualitatively and quantitatively evaluate the Aibo simulation in Webots by comparing the simulation with the real Aibo while performing various tasks. We implement measurements on the robot in static and dynamic situations, and we compare performance between them. The comparison procedure is always the same: a Webots controller is created for the simulation. This controller makes the robot perform an action that we want to measure. The controller is then executed in the simulator and the measure taken. Next, the controller is cross-compiled using Webots cross-compilation system, and executed on the real robot. Finally, the measure of the real robot is taken and compared with the simulation result. The cross-compilation of the code ensures that the same control program will be executed in both simulator and real robot, and allows us to perform a fair comparison of performance between them.

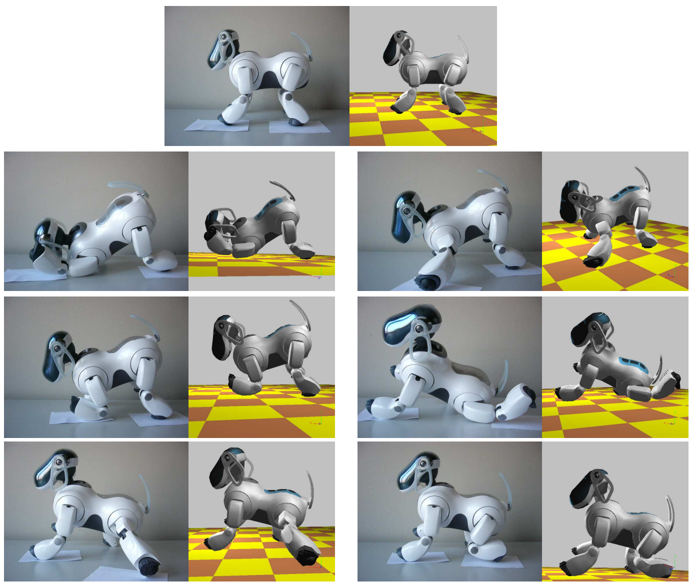

Fig. 8. List of pairs of static figures obtained in the simulator and the real robot when moving a joint at a time. The first two pictures show the initial position, and the resting pairs are the resultings of moving joints LeftFront J1, J2, J3, and Left Hind J1, J2 and J3 respectively.

10.1 Static measurement

The first comparison consists of measuring how the simulation differs from the real robot when confronted to extreme static positions of the joints. In this case, a Webots controller is created which slowly moves one of the leg joints to the maximum or minimum of the joint range, starting from an initial natural position. The controller is executed in the simulator and in the real robot, but in both cases, only one joint is moved at the same time. Final positions reached by simulator and real robot are recorded and compared visually and numerically. The velocity at which joints are moved is very slow, so static states can be assumed at any position of the joint.

Due to the fact that a single joint was moved at the same time, the robot was required to slide over the ground. For this reason, the friction against the ground was reduced to the minimum. In simulation this was achieved by reducing the friction coefficient of the simulated ground. In real robot this was achieved by placing some small sheets of paper under the paws of the robot.

The results of those tests can be seen in Fig. 8. We observe that both simulator and real robot achieve the same final position and, morever, the trajectory described by both from the initial position to the final one, is practically identical.

10.2 Dynamic measurement

We carried out dynamic measurements, to analyze the difference between the simulation and the real robot when they perform a movement. We will show the results obtained from three different experiments, with an increasing degree of complexity in the robot behavior. Experiments conducted consisted of the characterization of the movement of one leg alone, the characterization of the whole robot when performing a walking gait, and the characterization of the whole robot when performing a walking gait and stopping at a specified distance to an obstacle detected by the long distance sensor.

10.2.1 Movement of one leg

In this experiment, the three joints of a leg are moved following a sinusoid trajectory. This movement is the result of a direct command to the servos, not the execution of a MTN motion file. Trajectories obtained from the real and the simulated robot were obtained and a measure of error calculated, comparing differences between the desired trajectories and the ones obtained in both cases. For every joint three trajectories where performed at three different frequencies of oscillation. The error is calculated as the mean square error per period, normalised by the amplitude.

Fig. 9 shows the trajectories obtained for joint J1 at every testing frequency, including the desired trajectory, the trajectory obtained by the simulated leg, and the trajectory of the real robot leg. Fig. 10 summarizes the errors obtained between desired trajectory and actual trajectory, for the simulator and the real robot for all types of joints.

From both figures we observe that the error obtained at these frequencies is small. Nevertheless, error increases exponentially with the frequency of the oscillation, and higher frequencies started to be difficult to follow for the joints, especially for the real robot. At the same time, errors between simulation and real robot also increased exponentially showing that at high frequencies the simulator starts to differ more from the real robot. The two types of errors present at higher frequencies are phase differences and attenuation of the shape trajectory, i.e. typical tracking errors of PD controller. However, these are only observed at frequencies far from the typical use for this robot and should not affect much the simulation.

10.2.2 Performing a walking gait

This section shows how similar the simulator and the real robot behave when implementing a complex movement. Both the simulator and the real robot run

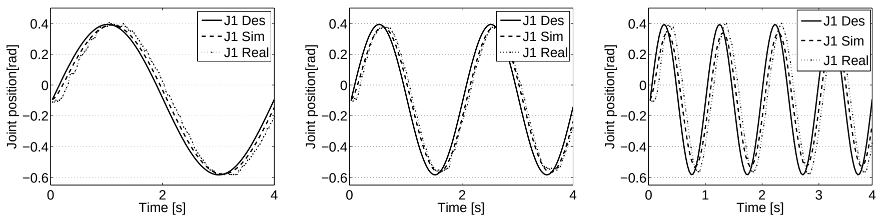

Fig. 9. Trajectories obtained for the J1 joint at three different frequencies. From left to right, oscillations at 0.25 Hz, 0.5 Hz and 1 Hz

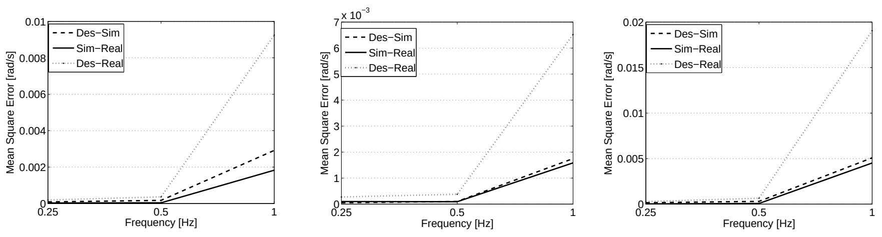

Fig. 10. Error measures obtained for the three types of joints. Each figure shows the error between the desired trajectory (Des), the simulator (Sim) and the real robot (Real). From left to right, errors for joints J1, J2 and J3

a Webots controler that executes an MTN file specifying a walking gait. Any MTN file specifies the exact sequence of movements for each robot joint at any time step in order to generate sequence of movements (in this case a walking pattern). The walking gait obtained for the simulator and the real robot are visually compared, joint trajectories are recorded and compared, and speed is measured and compared.

The robot was placed on a parquet floor and its friction parameter introduced on the simulator. The paws of the robot are made of rubber. Typical values for a parquet floor friction against rubber are between 0.55 and 1.36 [21]. A mean value of 0.7 was selected. The robot executed then a sequence of five walking steps, and the distance and time to accomplish those was measured.

First, a visual comparison of the walking was performed. Fig. 11 shows a sequence of movements obtained in both systems. Walking gait was very similar with no appreciable visual difference.

Second, the motor positions on each legs were also recorded for comparison. Differences between the positions of the left fore leg in simulation and on the real robot can be seen in Fig. 12. Trajectories are also very similar as the visual inspection of the gait indicated, the only significant difference being the value of the paw sensor. This is due to the fact that real paw is more noisy than the simulated one and some contacts are not detected by it. Fig. 12 only shows joints of one leg, but other legs behave in the same manner (including paw sensor errors).

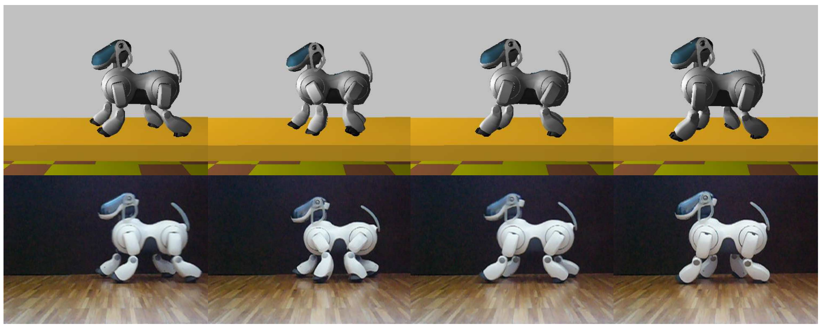
Fig. 11. Walking sequence performed by simulator and real robot

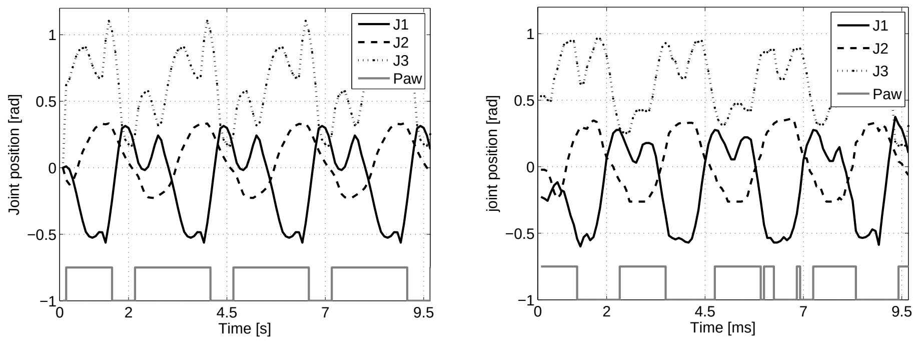

Fig. 12. Joint trajectories during walking for the left fore leg, in the simulator and with the real robot

Finally, velocity in both systems was measured. Due to the chaotic nature of the real robot, a statistical measurement was required to obtain its velocity. A mean value was calculated out of ten runs. For the simulator no statistical measurement was required since no noise was introduced in the simulation and the results could then be repeated all the times with the same final value. Nevertheless, due to small variations in some variables, it was noted that the final distance was not exactly the same in different rounds, but the difference was so small that it was discarded. This difference may indicate the presence of chaotic behavior in the simulator that may affect more complex setups, and may come from the accumulation of numerical errors due to loss of precision.

The measurements of the velocities showed a velocity of 3.50 cm/s for the simulator, and a mean velocity value of 3.25 cm/s for the real robot with a variance of 0.0261.

The results in this section show again that small differences between the simulator and the real robot can be observed, but none of them really significant.

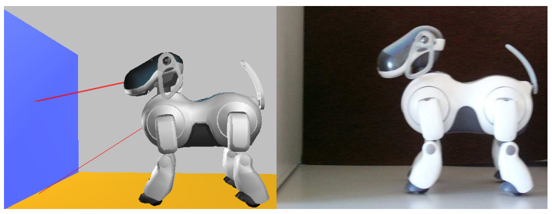

Fig. 13. Simulator and real robot setup for distance measurement

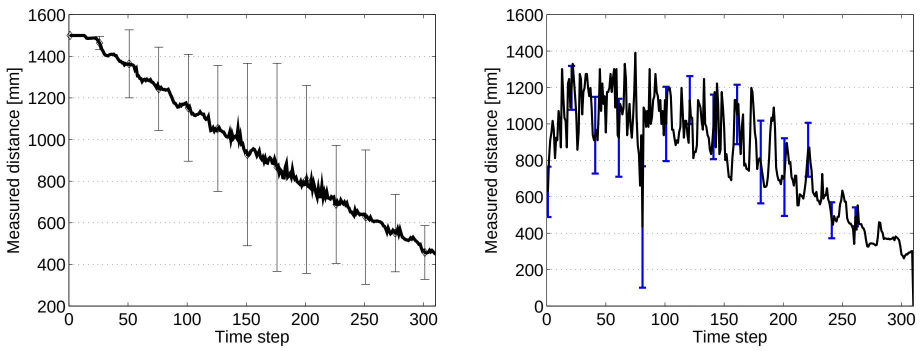

Fig. 14. Mean distance value to an obstacle for the simulator (left) and the real robot (right), obtained during the walking of the robot towards the obstacle. Mean value is represented by thick black line. Vertical lines represent the standard deviation of the measure at those points

10.2.3 Walking up to a distance

This experiment measures the difference between the simulator and the real robot when the whole robot is put into test, ina behavioral task involving movements and sensor readings.

In this case, Aibo is requested to start walking from an initial position in the same way as in the previous experiment, but to stop after it detects any obstacle at less than 40 cm. Obstacle detection is performed by using the *Far* distance sensor situated in the head of the robot. The mechanism is the same as in previous sections: a webots controller is created and executed in both the simulator and the real robot (cross-compiled), and the differences between both measured. In this case, we compare the distance at which the robot stopped from the obstacle, but also all the measures taken by the distance sensor during the walking period. The measures of distance taken during walking are obtained from the far distance sensor of the robot, which directly provides a value of the distance in milimeters.

Since there exist sensor errors in both simulated and real robot (simulated sensors are modeled with noise), a unique and absolute measure of the distance cannot be used for stopping the robot. Because of that, a hysteresis mechanism was implemented, consisting of stopping the robot only after a distance below 40 cm has been detected for ten consecutive time steps of 96 ms. The final distance and the trajectories were measured ten times in each type of robot.

The experiments showed that the simulated robot stopped at a mean distance of 33.6 cm of the obstacle with a variance of 0.22 cm, meanwhile the real robot stopped at a mean distance of 32.72 cm of the obstacle with a variance of 15.67 cm. Eventhough the mean distances at which both robots stopped are very similar, the variance values among them are very different. The simulated robot has a very small variance, only due to the noise of the distance sensor. On the other side, the real robot has a very large variance due to several factors not included in the simulation: first the distance measure method is far from exact in the real robot (it is a manually measurement performed with a ruler). Second, all the joints have their own noise that affect the final position of the robot head, amplifying the difference in the distance measured. Third, the movements on the head of the robot can produce reflections of the infrared ray, producing even more measure differences.

All these differences can be seen in Fig. 14. In that figure, the mean distance measured value obtained at each time step is presented, together with the standard deviation at some randomly selected points. It can be seen that while the simulator presents a *clean* line, the real robot has a more noisy line showing the high variance detected in the measurements.

11 Future Work

Even the current version of the graphical interface has some capabilities which are unused in simulation. There are features of Aibo which are not included in the simulation model, either because there's no interest in modeling them or because it is not possible to simulate them, e.g., acceleration and thermo sensors, speaker and microphone. Those features could be integrated in the model by implementing new simulated device types with appropriate controller programming interfaces. On the other side, there are devices for which Webots nodes exist but which are not controllable via the remote control software and therefore not integrated in the user interface, e.g., the camera. The controller remote control mode and the cross-compilation could be extended to support the complete controller programming interface of the existing node types. Going even further, there are devices on Aibo, for which it would be nice to have a convenient programming interface in cross-compilation, even if there is no Webots equivalent, e.g., thermo and acceleration sensors.

12 Conclusion

This article has described how we implemented a Webots simulation model, a remote control system and the cross-compilation of Webots controllers for Aibo ERS-210 and ERS-7. The complete system enables simultaneous control of both a simulated and a real Aibo robot and provides the user with a platform for convenient robot programming without any knowledge of the underlying robot firmware. The simulation and remote control capabilities are almost identical, but do not cover all of Aibo's features. Most available functions are related to motion control and trajectory generation, but the system architecture was conceived to be extensible to other device types. The complexities of most system components (simulation model, Webots nodes, remote control, graphical user interface, controller programming interface) are closely related. For the purpose of our new developments, modifications to Webots and especially the controller program interface could not be avoided. Except for the challenging task of implementing all simulated devices, it is probably easiest to obtain equivalence between a (complete) Aibo model controlled by a simulated controller and a real Aibo running the same cross-compiled controller. The implementation of full wireless remote control by a controller and GUI running on the client side would imply an important extension of the communication protocol.

With the current set of features, the extended version of Webots can be used for a variety of applications, e.g., fast design and testing of controllers, simulation calibration, use of the robot as a programming interface or interaction between real and simulated environments. The combination of simulation, remote control and cross-compilation allows the users to test Aibo in various environments and under numerous conditions. Simulation results are easily transferrable to the real robot using controller cross-compilation and the convenient graphical user interface for the transfer. Despite the overhead compared to a direct solution programmed in OPEN-R, the results of the cross-compilation enable efficient and precise control of an Aibo robot in an easy way. The remote control mode enables even faster switching between the simulation and the real robot. Parallel control of both the simulation and the real robot can be used for direct comparison. Thanks to the graphical user interface for manual control of the simulation and the real robot, no controller program has to be written to explore the robot's capabilities. We already took advantage of this during the construction and calibration of the basic simulated model.

A Trademarks

- "Aibo", "Memory Stick", "OPEN-R" and "OPEN-R" logo are trademarks or registered trademarks of Sony Corporation.
- "Webots" is a registered trademark of Cyberbotics Ltd.
- "Mindstorms" and "RCX" are registered trademarks of the Lego Group.
- Other system names, product names, service names and firm names contained in this document are generally trademarks or registered trademarks of respective makers.

Acknowledgements

This work was funded by the Swiss Federal Office for Professional Education and Technology http://www.bbt.admin.ch/f/index.htm (Technology Transfer CTI http://www.bbt.admin.ch/kti/aufgaben/f/, project 6395.1 ENS-ET) and by the Swiss National Science Foundation (young professorship award to Auke Ijspeert).

References

[1] M. Fujita, Digital Creatures for Future Entertainment Robotics, *Proceedings IEEE International Conference on Robotics and Automation* (2000) 801–806

[2] M. Fujita, K. Kageyama, An Open Architecture for Robot Entertainment, *Proceedings First International Conference on Autonomous Agents* (1997) 435–442

[3] M. Fujita, H. Kitano, Development of an Autonomous Quadruped Robot for Robot Entertainment, *Autonomous Robots* **Volume 5 Number 1** (1998) 7-18

[4] T. Ishimura, T. Kato, K. Oda, T. Ohashi, An Open Robot Simulation Environment, in: *Proceedings Robot Soccer World Cup VII, Springer, 2003*

[5] O. Khatib, O. Brock, K. Chang, F. Conti, D. Ruspini, L.Sentis, Robotics and Interactive Simulation, *Communicaton of the ACM* **Volume 45 Number 3** (2002), 46–51

[6] F. Klassner, S. D. Anderson, Lego Mindstorms: Not Just for K-12 Anymore, *IEEE Robotics & Automation Magazine* **Volume 10 Issue 2** (2003) 12–18

[7] T. Makimoto, T. T. Doi, Chip Technologies for Entertainment Robots – Present and Future, *Digest of the International Electron Devices Meeting* (2002) 9–16

[8] O. Michel, Cyberbotics Ltd. Webots: Professional Mobile Robot Simulation, *International Journal of Advanced Robotic Systems* **Volume 1 Number 1** (2004) 39–42

[9] O. Michel, Webots: Symbiosis between Virtual and Real Mobile Robots, in: *Proceedings First International Conference on Virtual Worlds (VW'98), J.-C. Heuding (Ed.), LNCS/AI 1434, Springer, 1998*

[10] J. Pratt, G. Pratt, Exploiting Natural Dynamics in the Control of a 3D Bipedal Walking Simulation, in: *Proceedings International Conference on Climbing and Walking Robots (CLAWAR99), Portsmouth, UK, 1999*

[11] T. Roefer, German Team RoboCup 2003 Technical Report, http://www.germanteam.org/

[12] R. Smith, Open Dynamics Engine User Guide (2004) http://ode.org/

[13] Sony Corporation, Level2 Reference Guide, OPEN-R SDK Documents English (2004) http://openr.aibo.com/

[14] Sony Corporation, Model Information for ERS-210, OPEN-R SDK Documents English (2004) http://openr.aibo.com/

[15] Sony Corporation, Model Information for ERS-220, OPEN-R SDK Documents English (2004) http://openr.aibo.com/

[16] Sony Corporation, Model Information for ERS-7, OPEN-R SDK Documents English (2004) http://openr.aibo.com/

[17] Sony Corporation, OPEN-R Internet Protocol Version4, OPEN-R SDK Documents English (2004) http://openr.aibo.com/

[18] Sony Corporation, Programmer's Guide, OPEN-R SDK Documents English (2004) http://openr.aibo.com/

[19] L. F. Wang, K. C. Tan, V. Prahlad, Developing Khepera Robot Applications in a Webots Environment, *Proceedings International Symposium on Micromechatronics and Human Science* (2000) 71-76

[20] J. C. Zagal, J. Ruiz-del-Solar, A Dynamically and Visually Realistic Simulator for the RoboCup Four Legged League, in: *International Workshop on RoboCup 2004, Lecture Notes in Artificial Intelligence, Springer, Lisbon, Portugal, 2004*

[21] R. Brough, F. Malkin and R. Harrison, Measurement of the coeficient of friction of floors, *J. Phys. D: Appl. Phys.*, Vol. 12, 1979

[22] K. Asunuma, K. Umeda, R. Ueda, T. Arai, Development of a Simulator of Environment and Measurement for Autonomous Mobile Robots Considering Camera Characteristics, *Proceedings Robot Soccer World Cup VII*, Springer, 2003

[23] T. Laue, K. Spiess, T. Rfer, SimRobot - A General Physical Robot Simulator and its Application in Robocup, in RoboCup 2005: Robot Soccer World Cup IX, *Lecture Notes in Artificial Intelligence*, Springer, 2005

[24] N. Koening, A. Howard, Gazebo - 3D Multiple Robot Simulator With Dynamics, http://playerstage.sourceforge.net/index.php?src=gazebo

[25] B. Gerkey, R.T. Vaughan, A. Howard, The Player/Stage Project: Tools for Multi Robot and Distributed Sensor Systems, *Proceedings of the 11th International Conference on Advanced Robotics*, 2003, 317-323

[26] J. Go, B. Browing, M. Veloso, Accurate and Flexible Simulation for Dynamic, Vision-centric Robots, *Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'04)*, 2004

[27] R. Tellez, C. Angulo, D. Pardo, Evolving the walking behaviour of a 12 DOF quadruped using a distributed neural architecture, *Proceedings of the 2nd International Workshop on Biologically Inspired Approaches to Advanced Information Technology (Bio-ADIT'2006)*, Springer, 2006