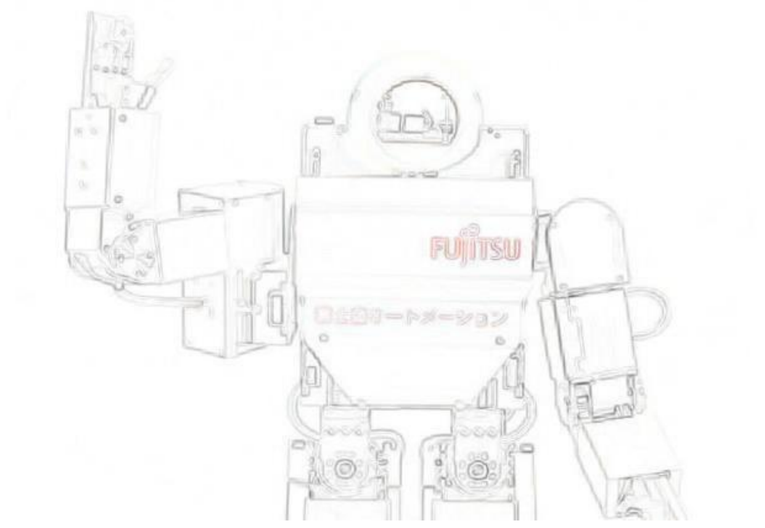
Development of a physical simulation of a real humanoid robot

Pascal Cominoli, Diploma Thesis

February 20, 2005

BIRG, Logic Systems Laboratory (LSL)
School of Computer and Communication Sciences
Swiss Federal Institute of Technology, Lausanne

Supervision:
Prof. Auke Jan Ijspeert & Dr. Olivier Michel



Acknowledgments

I would like first to thank Auke Jan Ijspeert, for giving me the opportunity to carry out two interesting projects in his lab.

I also would like to thank Olivier Michel, for our weekly meetings, for all the changes he introduced in his software to suit my needs, and also for letting me write some code of his software.

I thank all of the BIRG and LSL people, and specially Yvan Bourquin, Alessandro Crespi and Ludovic Righetti, who have always been there when I needed them.

I make a point of thanking the EPFL's ASL3 laboratory, for helping me to succeed in controlling the robot, and specially Adrien Brossard, for the assistance he brought me and the critical tests which he carried out in my place.

I would like also to say “ありがとう ございます” [1] to Fumio Nagashima, my contact (and future boss for the next twelve months...) at Fujitsu, Kawasaki, JP, for providing me within a few hours with the files I needed.

And last but not least, thanks a lot to my family and all of my friends.

[1] It means “thank you”, and is pronounced “ARIGATOO GOZAIMASU”.

Contents

List of Figures

List of Tables

1 Introduction

HOAP-2 is a compact, light weight, and easy-to-treat genuine humanoid robot with two arms and two legs. HOAP-2's gesture expression has been enhanced from its first version HOAP-1 with moving head, waist, and hands. Using motor current control, the movements are much smoother. Its connection to a PC is really easy, so one can use it as a human robotics research tool for studying areas, such as movement control and communications with humans. It is only 50[cm] tall, weights less than 7[kg], so it can be handled by a single person, what is a great advantage.

The Swiss Federal Institute of Technology purchased one HOAP-2 during the summer 2004. So it was really important to develop a simulator for this robot. Performing tests on the real robot could damage it, depending on the kinds of things one is doing. And knowing the price of such a robot, one could really easily understand that a simulator is necessary. The real robot needs to be handled carefully, calibrations need to be done. There is no such issues on a simulated robot. If one performs a mistake with a simulated robot, just reload the simulation, and everything is fine. There is no calibrations, no problems of simultaneous access to one robot shared by many people. It also allows someone not having access to a robot, like a stand alone student or small universities, to perform good simulations.

This simulator will be used for performing for example walking controllers, or various manipulations, that could be dangerous or hazardous if they were performed first on the real robot. After a validated test on the simulator – meaning that the real robot will not suffer any harm, the user could then upload its controller on the real robot. The simulator could also be used to do genetics evolution much faster than on the real robot. On a good computer, simulation can run up to ten times real time, what is unrealizable on a real robot.

It is then important to have a physical simulator, instead of just a kinematic simulator. The simulator must no only look like the real robot, but also react like it, with the same physical laws. A physical simulator takes care of the forces, the torques, what includes the gravity, the slipping it could have with the ground, etc., contrary to the kinematic simulator, that does not take care of the physics, but only off the velocities. The BIRG laboratory being interested in biped humanoid walking for example, it would have been impossible to realize it with a simple kinematic simulator. A good physical simulator makes it then possible to perform realistic dynamical moves. Capabilities of the real robot motors should also be respected, in order to have a "realistic" simulation.

The goal of this diploma project was the development of a physical simulation of that HOAP-2 robot. I already began developing it during the summer semester 2004, as my last semester project[3]. But the work was still not over at the end of the semester, and we decided I could continue improve it as a diploma work.

I used the mobile robotics simulation software Webots, developed by Cyberbotics Ltd, which is collaborating with my laboratory, the Biologically Inspired Robotics Group (BIRG). In this report, I will suppose you have basic knowledge about that software, such as world definition, format, nodes, and so on. Please do refer to its website for more informations. I think that you also should first quickly traverse the report I wrote for my semester project[3], this could help you understand what I already did during the semester before my diploma. You should also have a look at the web page I made for this thesis. It contains movies, pictures, etc[1].

My last semester project, and this diploma work, was the very first attempt to create an accurate design of an humanoid robot in Webots. There was already a humanoid robot (Sony QRIO) implemented in Webots, but it hadn't been made with respect to accurate values or data, and just been reversed engineered from pictures and feelings[7]. My work allowed us to realize there were limitations on this software, and also a few bugs. These last ones have all been corrected, and new features have been implemented.

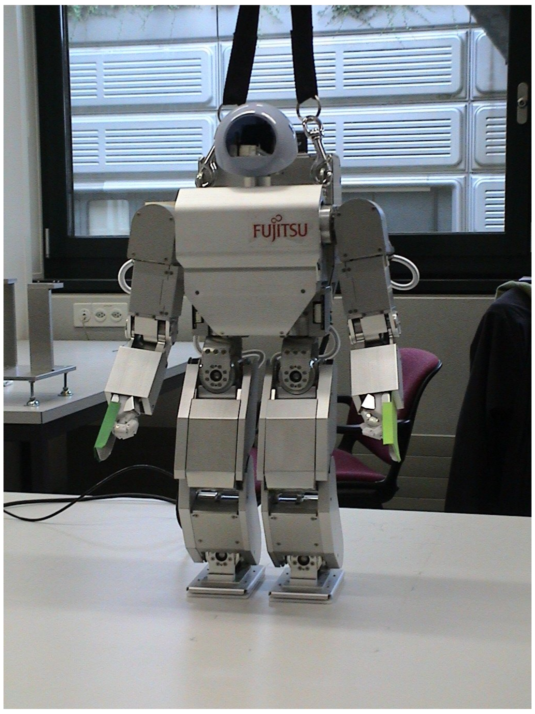


Figure 1: EPFL's HOAP-2

2 Humanoid robotics and available simulators

2.1 Humanoid robotics

”The 90s was the era of the PC and the Internet. The first decade of the 21st century will be dominated by robots.”

Toshitada Doi, Vice President of Sony Corporation

Humanoid robotics is a an emergent field. And like many new technologies, the early generations are costly curiosities, mainly useful for entertainment and for research. To date, there is not much humanoid robots in our surroundings, but, in time, they will accomplish a wide variety of tasks in homes but also in various places where human beings do not like to be, doing hazardous tasks like taking care of nuclear plants, rescuing people after a disaster, etc. So let’s have a look at the most important and famous humanoid robots. Most of them are just demonstration ones, and are not supposed to be sold, at least recently. And most of them come from Japan.

2.1.1 Honda series

ASIMO Honda describes its ASIMO (Advanced Step in Innovative MObility, or from the Japanese ashi[leg or foot] and mo[move or mobility]) as the most advanced humanoid robot in the world. Designed at its research and development Wako Fundamental Technical Research Center in Japan, ASIMO took more than 18 years of persistent study, research, trial and error before Honda engineers achieved their dream of creating an advanced humanoid robot (see figure 3).

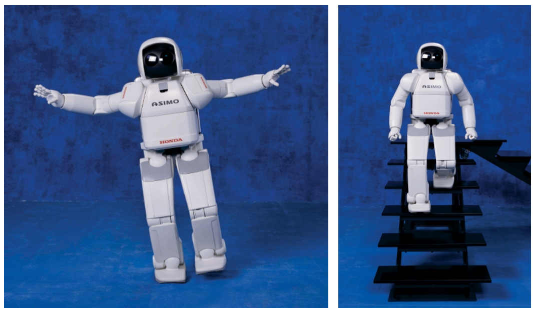


Figure 2: Honda ASIMO

ASIMO is then the culmination of nearly two decades of humanoid robotics research by Honda scientists and engineers, and it was the world’s first humanoid robot to walk dynamically, as humans walk, when it was introduced to the world, October 31st, 2000. It is 120[cm] tall, weights about 52[kg], and has 26 degrees of freedom (DOF). ASIMO can walk on uneven slopes and surfaces, turn smoothly, climb stairs, reach for and grasp objects, switch lights on and off, and open and close doors. Now, ASIMO can also comprehend and respond to simple voice commands. ASIMO has the ability to recognize the face of a selected group of individuals. Using its camera eyes, ASIMO can map its

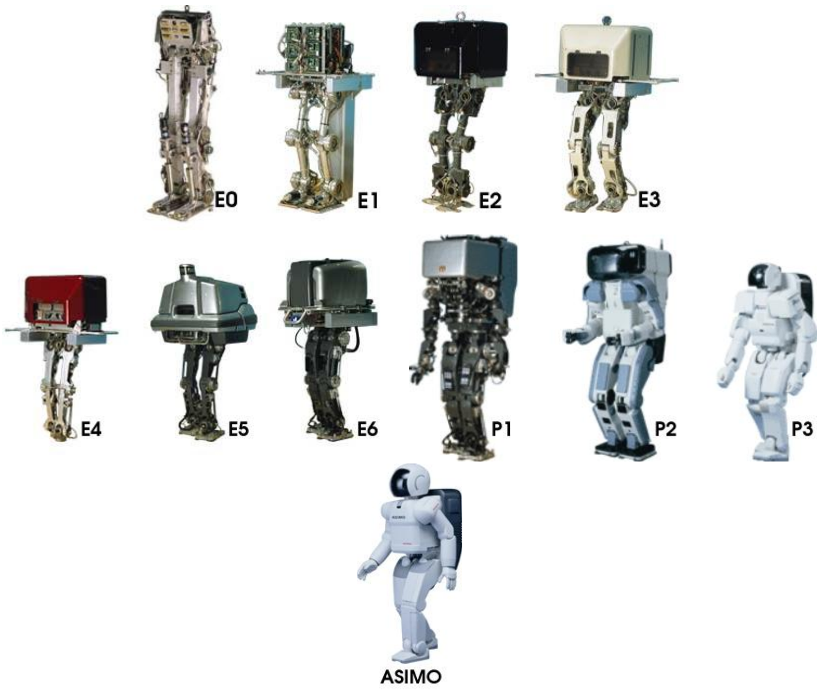


Figure 3: In 1986, Honda engineers set out to create a walking robot (E0). Early models (E1, E2, E3) focused on developing legs that could simulate the walk of a human. The next series of models (E4, E5, E6) were focused on walk stabilization and stair climbing. Next, a head, body and arms were added to the robot to improve balance and add functionality. Honda's first humanoid robot, P1, was rather rugged at 187[cm] tall, and 175[kg]. P2 improved with a more friendly design, improved walking, stair climbing/descending, and wireless automatic movements. The P3 model was even more compact, standing 157[cm] tall and weighing 130[kg]. ASIMO is then the culmination of nearly two decades of humanoid robotics research by Honda scientists and engineers

environment and register stationary objects. It can also yield to pedestrians in its path until they have cleared its path. It can walk at 1.6[km/h], even if some special experimental versions of the robot can reach the double.

Currently, there is no plan of selling ASIMO. It serves as a tour guide in museums and as a greeter at high-tech companies in Japan, and there is only about thirty ASIMO around the world. But in the future, Honda hopes that ASIMO may serve as another set of eyes, ears, hands and legs for all kinds of people in need. Someday ASIMO might help with important tasks like assisting the elderly or a person confined to a bed or a wheelchair. ASIMO might also perform certain tasks that are dangerous to humans, such as fighting fires or cleaning up toxic spills.[17][18]

2.1.2 Sony series

QRIO Sony's QRIO(Quest for CuRIOsity) was introduced September 2003. It is a lot smaller than the ASIMO, as it is only 60[cm] tall, and weights 7[kg] not including battery. QRIO is a remarkable assemblage of three powerful microprocessors, 38 DOF, three accelerometers, two charge coupled device (CCD) cameras, and seven microphones. It can hear, speak, sing, recognize objects and faces, walk, run, dance, and grasp objects. It can even pick itself up if it falls. At the moment, there are dozens of QRIOSs in existence, and it is not yet for sale, but it is the one that will probably be the first to be sold, as Sony already sell to dog AIBO[13].

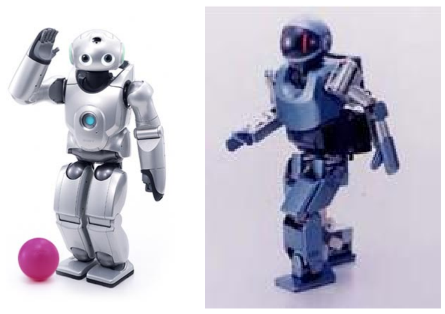


Figure 4: Sony Corporation originally developed a small biped walking robot "SDR" (3X and 4Xprototype). The right picture shows the SDR-3x. They are the previous series before the release of the QRIO (left)

2.1.3 Fujitsu series

Fujitsu designed two humanoid robots, the HOAP-1 (Humanoid for Open Architecture Platform), in 2001, and then the HOAP-2, in 2003 (see figure 5). To date, they are the only open source commercial robots, with a research purpose.

HOAP-2 HOAP-2 is designed as an aid to robotics research and therefore runs on open source, Linux-based software, and is sold since July 2003. It has 25 DOF, is 70[cm] tall, and weights around 7[kg] including battery. This is the robot I used in this project, so I will come back to this latter on.

HOAP-1 Fujitsu Automation Limited and Fujitsu Laboratories Ltd. released HOAP-1 on September the 10th, 2001. It has been designed for wide application in research and development of robotics technologies. This robot has been, from the beginning, designed for domestic sales. Weighting 6[kg] and standing 48[cm] tall, the light and compact HOAP-1 can be used for developing motion control algorithms in such areas as two-legged walking, as well as in research on human-to-robot communication interfaces. Fujitsu is disclosing the internal interface architecture of HOAP-1 to allow users to freely develop their own programs. The wide range of use and safe and efficient program development environment make of HOAP-1 an ideal tool for research and development work in robotics.[22]

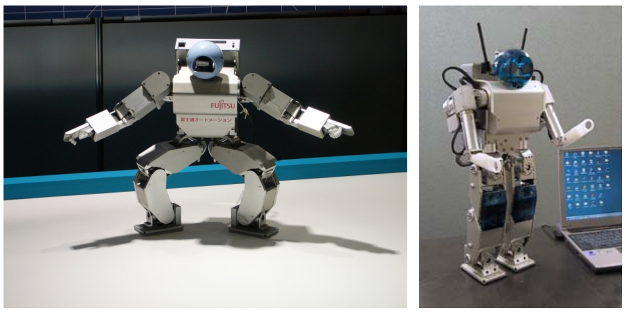


Figure 5: Fujitsu HOAP-2 (left) and HOAP-1 (right)

2.1.4 Humanoid Robotics Project series

“For research and development of humanoid robot performing applications tasks, the Ministry of Economy, Trade and Industry (METI) of Japan had run Humanoid Robotics Project (HRP for short) from 1998 to 2002. The final goal of HRP is to create 'useful' humanoids robots. Toward the goal, HRP have developed a humanoid robot called HRP-2 that can walk, lie down and get up.”[12]

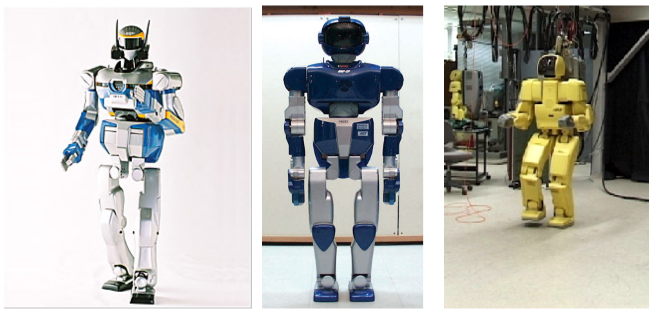


Figure 6: HRP-2 (left) and earlier version HRP-2P (middle). The height, mass, and DOF of the HRP-2 are the same as the HRP-2P that was released in March, 2002. The HRP-1 (right), is an even earlier model looking a lot like the Honda P3 robot with a large external backpack and rigid torso.

HRP-2 HRP-2's height is 154[cm] and mass is 58[kg] including batteries. With its size and weight, the HRP-2 is the first humanoid robot to be human-sized. It has 30 DOF including two DOF for its hip. Its highly compact electrical system packaging allows it to forgo the commonly used ”backpack” used on other humanoid robots. HRP-2 will be used for experiments to further develop robotics technologies in the areas of walking on uneven surfaces (HRP-2's feet are designed for that), tipping-over control, getting up from a fallen position, and ”human-interactive operations in open spaces. The beautiful external appearance of HRP-2 was designed by Mr. Yutaka Izubuchi, a mechanical animation designer famous for his robots that appear in Japanese anime.

Kawada Industries is renting HRP-2 as a humanoid robot R&D platform. Internal API for HRP-2 is expected to be open to the public and its users will be able to develop their own software. It is anticipated that HRP-2s will greatly enhance humanoid robot technology research activities. Users will be able to develop application software due to open architecture.[12]

2.1.5 KAIST Humanoid Robot Platform series

The Korea Advanced Institute of Science and Technology (KAIST) has produced a series of robots on the way to a practical humanoid. KAIST is developing a "remote-brained" robot –meaning it is connected to a central server whose computing capacity may be expanded.

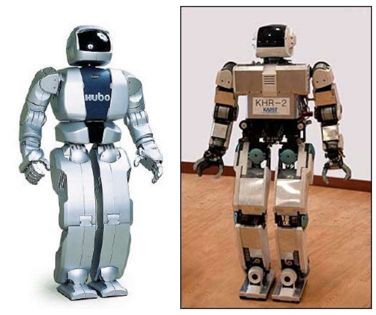


Figure 7: HUBO or KHR-3 (left) and its predecessor KHR-2 (right). Compared to the KHR-2, the HUBO is more streamlined and has smoother, more natural walking, has fingers instead of claws, voice recognition, and faster response times.

HUBO The HUBO (or KAIST Humanoid Robot Platform, KHR) is 150[cm] tall, weighs 67[kg], and has a range of 3[km]. The robot apparently does primary processing of sound and vision, and then sends its "perceptions" to a central computer, which does actual decision-making. This allows it to learn and interact using a computer that would be too heavy to carry. The HUBO debuted in December 2004.

2.1.6 Toyota series

Toyota has introduced lately a suite of robots –one destined for health care, one for factories, one for a human exoskeleton, and one for entertainment. The walking robot is the one designed for health care. Among other features, it can blow air through a trumpet and finger the valves to actually play a musical instrument, what shows a significant advance over the Honda robots. Its legs are less "bent knee" than the Honda ASIMO and its walk seems more natural. Toyota promises an entire robot orchestra by 2005. Toyota wants its partner robots to have human characteristics, such as being agile, warm and kind and also intelligent enough to skillfully operate a variety of devices in the areas of personal assistance, care for the elderly, manufacturing, and mobility. The walking robot is 120[cm] tall, and weights 35[kg]. These robots are to be unveiled at Expo 2005 (Nagoya, JP), starting March 2005. The company also announced the creation of a new division chartered with developing “partner robots”, with the aim of commercializing humanoid robots by 2010.

Figure 8: Toyota walking robot

2.2 Existing robotics simulators

2.2.1 Honda and Sony simulators

As the ASIMO and the QRIO are not (at least currently) commercial robots, there is no available simulator for them. But I am sure that Sony and Honda have one, because it is almost mandatory. If you want to improve the way the robot behaves toward its surroundings, one has to do some of the work in simulation, since it would be to difficult and expensive to do it on the real robot directly. But these companies do not let filter many information on this subject.

2.2.2 Fujitsu HOAP simulator

Fujitsu sells HOAP-1 with a basic simulation software (see figure 9). It enables virtual trial-runs of the control programs prior to actual implementation. The simulator and the user-developed programs are designed to run on RT-Linux on an operating command PC, which communicates with the robot through a USB interface. The included PC uses C/ C++ language with VRML model data to allow 3D simulation of actual robot actions before attempting action with the robot. So one has to have purchased a HOAP-1 to have the simulator. HOAP-2 is not provided with a simulator.

2.2.3 OpenHRP

OpenHRP (Open Architecture Humanoid Robotics Platform) is a software platform for humanoid robotics, and consists of a dynamics simulator, view (camera) simulator, motion controllers and motion planners of humanoid robots (see figure 10). OpenHRP has been developed by AIST, the University of Tokyo and MSTC. OpenHRP is integrated with CORBA, and each module, including the dynamics simulator, is implemented as a CORBA server. Users can develop their own software on OpenHRP as well as replace its building blocks by one of their own. The dynamics simulator and the view simulator from OpenHRP

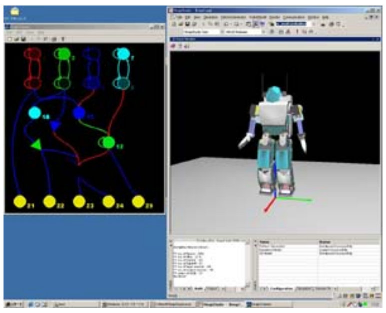


Figure 9: Comprised of a neural network display/edit unit, a robot simulation unit, and a mechanical interface, this system enables even people without any expertise in the field of dynamics to generate the desired movements in humanoid robots

are distributed for free for non-commercial use[11]. One can design whatever kind of robot on OpenHRP, a model of the HRP-2 robot is contained in this software, and some teams designed HOAP-1 models also.

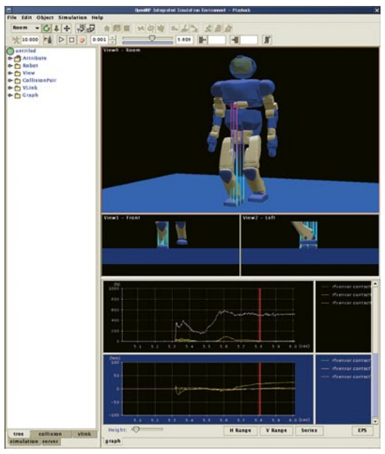


Figure 10: OpenHRP

2.2.4 RoboWorks

RoboWorks is a commercial software developed by Newtonium, founded in March 2000. "RoboWorks is an easy to use software tool for 3D modeling, simulation and animation of any physical system [...]. When using RoboWorks you will benefit from an extremely intuitive model development, and a high quality, fully interactive 3D graphics with full animation, even while building your model."[20]

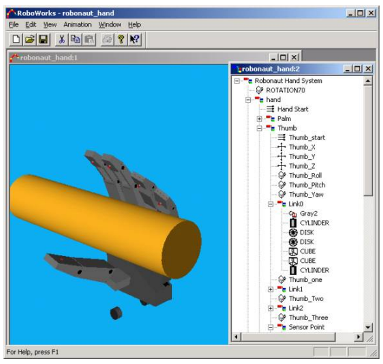


Figure 11: Roboworks screen shot

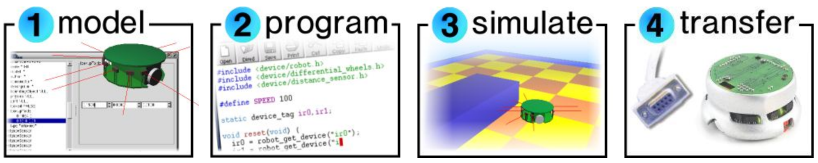
2.2.5 SD/FAST

"SD/FAST was developed by Michael Sherman and Dan Rosenthal of Symbolic Dynamics and has been distributed by PTC since January 2001. With over 31,000 customers worldwide, PTC is the leader in providing product development solutions for manufacturing. SD/FAST provides physically-based simulation of mechanical systems by taking a short description of an articulated system of rigid bodies (bodies connected by joints) and deriving the full nonlinear equations of motion for that system. The equations are then output as C or Fortran source code, which can be compiled and linked into any simulation or animation environment. The symbolic derivation of the equations provides the fastest possible simulations. Many substantial systems can be executed in real time on modest computers."[21]

2.2.6 Webots

Webots is a commercial software developed by Cyberbotics Ltd. It is "a mobile robotics simulation software that provides you with a rapid prototyping environment for modeling, programming and simulating mobile robots. The provided robot libraries enable you to transfer your control programs to several commercially available real mobile robots. Webots$^{\mathrm{TM}}$lets you define and modify a complete mobile robotics setup, even several different robots sharing the same environment. For each object, you can define a number of properties, such as shape, color, texture, mass, friction, etc. You can equip each robot with a large number of available sensors and actuators. You can program these robots using your favorite development environment, simulate them and optionally transfer the resulting programs onto your real robots. Webots has been developed in collaboration with the Swiss Federal Institute of Technology in Lausanne, thoroughly tested, well documented and continuously maintained for over 7 years."[4]

This software already contains models of many existing robots, mostly wheeled ones. It also contains a representation of a QRIO-like[7]. This is the software I used for this project.



```
#include <device/robot.h>
#include <device/differential_wheels.h>
#include <device/distance_sensor.h>

#define SPEED 100

static device_tag ir0,ir1;

void reset(void) {
  ir0 = robot_get_device("ir0");
```

Figure 12: Webots stages of development of a robotics simulation

2.2.7 Yobotics

Yobotics Inc. was founded in January 2000 by four graduates of the M.I.T. Leg Laboratory, and as they say on their web page, they are "a cutting-edge robotic design, consulting, and research firm specializing in biomimetic robots, powered leg orthotics, and force-controllable actuators"[19]. Among other fields, they designed a simulation tool. "The Yobotics Simulation Construction Set is a full-featured software package for easily and quickly creating simulations of robots, biomechanical systems, and mechanical devices. With it you can accurately and quickly simulate rigid body physics, access all joint positions, velocities, and torques, plot real-time graphs of any variable, playback and rewind your simulations, easily generate 3D graphics with texture mapping and camera controls [...]. The Simulation Construction Set is easy to use, yet powerful for creating complex simulations of robotic devices. Simulations of multi-joint devices can be created in a matter of minutes. Arbitrary control can be added to these devices as each degree of freedom automatically has a simulated actuator associated with it. For power users, the simulations are easily extensible, as they are implemented in 100% Java, with a well-documented Application Programmers Interface (API)."[19]

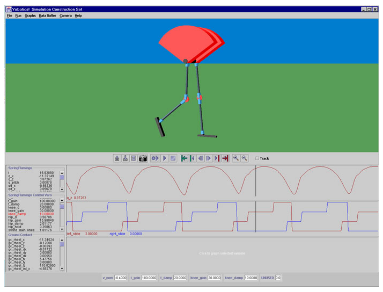


Figure 13: Yobotics simulation screen shot

3 HOAP-2 Design Specification

3.1 Foreword

This section contains the mechanical datas about the real HOAP-2 robot. I extracted these informations from the HOAP-2 instruction manual, and from the inspection report. I also got some results by measuring it on the robot. One can find here all the informations I needed to design my robot under Webots, such as link lengths, weights, positions, physical limitations, etc.

3.2 Robot link parameter

During [3], I thought that the specs were false concerning the length of arm_link3. After a few contacts with the real robot, I realized I was wrong. On the other hand, a few lengths were different from a few centimeters. So I modified my model in consequence. The table 1 contains the length of every link contained on the figure 14.

3.3 Joint freedom degree and joint composition

Here there was a few errors in the specs I had during [3]. The user manual is now more accurate, and there is no errors in it, as far as I know. The table 2 contains the motion and the motion range of every joint from the picture 15.

3.4 Mass property

I now have a very precise definition of the robot body parts weights, what was really not the case during my last semester project. In HOAP-2 inspection report, there is the weight of every link belonging to a joint, with the position of the center of mass of the corresponding link. I also have the inertia tensor at the center of mass position. These values were given with respect to the local axis orientation[2]. So it was really easy the update my model in order to correspond to the given weights, just by changing the *mass* field from the physics node. For the inertia matrices and the center of mass values, it has been a little bit more difficult, but I will come back to this latter on. The table 3 contains the weight of every joint, and one can find the center of mass position and inertia tensor values in the inspection report in appendix 12.1.

3.5 Local axes orientations

In HOAP-2 specs, the rotation of every joint is effective around the Z axis. So the orientation at every joint is different. The figure 16 contains these local orientations. I will explain in section 5 how I had to modify Webots, in order to respect this orientation that affect not only the inertia matrix on the concerned solid, but also its center of mass position.

[2]See section 3.5.

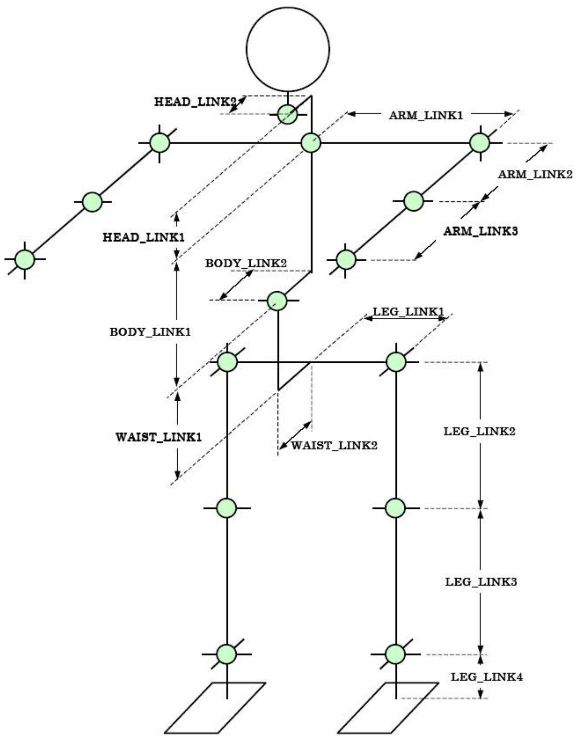


Figure 14: Parameter definition of HOAP-2 link length

Link	Length (m)
ARM_LINK1	0.0995
ARM_LINK2	0.101
ARM_LINK3	0.146
LEG_LINK1	0.039
LEG_LINK2	0.100
LEG_LINK3	0.100
LEG_LINK4	0.037
BODY_LINK1	0.090
BODY_LINK2	0.034
HEAD_LINK1	0.081
HEAD_LINK2	0.008
WAIST_LINK1	0.055
WAIST_LINK2	0.034

Table 1: HOAP-2 link length

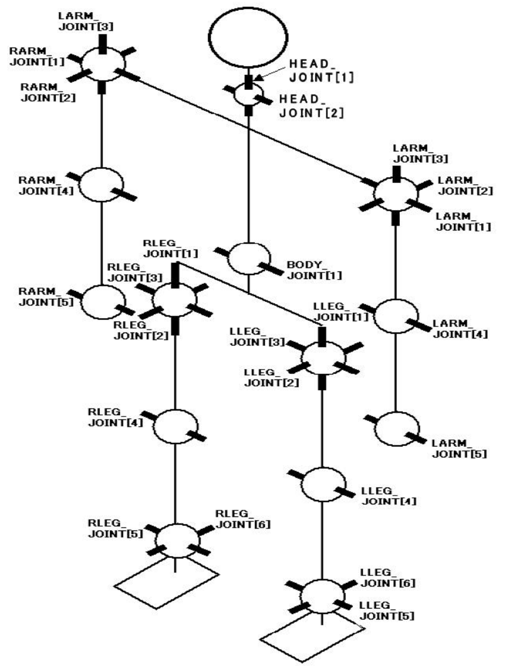


Figure 15: HOAP-2 joints positions

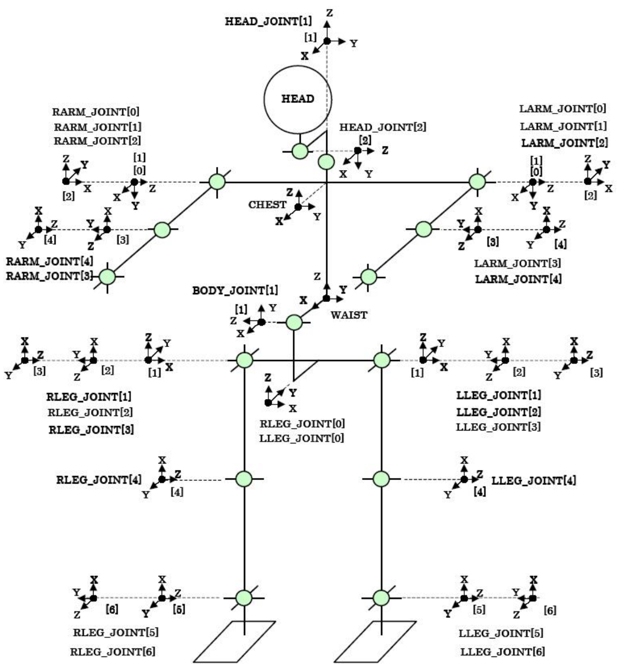


Figure 16: HOAP-2 local axes orientations

Part	Joint name	Motion	Motion range[deg] Min.	Max.
Right leg	RLEG_JOINT[1]	Thigh joint twist	-91	31
	RLEG_JOINT[2]	Thigh joint left & right	-31	21
	RLEG_JOINT[3]	Thigh front & back	-82	71
	RLEG_JOINT[4]	Knee	-1	130
	RLEG_JOINT[5]	Ankle front & back	-61	61
	RLEG_JOINT[6]	Ankle left & right	-25	25
Left leg	LLEG_JOINT[1]	Thigh joint twist	-31	91
	LLEG_JOINT[2]	Thigh joint left & right	-21	31
	LLEG_JOINT[3]	Thigh front & back	-82	71
	LLEG_JOINT[4]	Knee	-1	130
	LLEG_JOINT[5]	Ankle front & back	-61	61
	LLEG_JOINT[6]	Ankle left & right	-25	25
Right arm	RARM_JOINT[1]	Shoulder front & back	-91	151
	RARM_JOINT[2]	Shoulder left & right	-96	1
	RARM_JOINT[3]	Shoulder twist	-91	91
	RARM_JOINT[4]	Elbow	-115	1
	RARM_JOINT[5]	Finger open & close	-60	60
Left arm	LARM_JOINT[1]	Shoulder front & back	-91	151
	LARM_JOINT[2]	Shoulder left & right	-1	96
	LARM_JOINT[3]	Shoulder twist	-91	91
	LARM_JOINT[4]	Elbow	-115	1
	LARM_JOINT[5]	Finger open & close	-60	60
Waist	BODY_JOINT[1]	Waist front & back	-1	90
Head	HEAD_JOINT[1]	Neck twist	-60	60
	HEAD_JOINT[2]	Head front & back	-15	60

Table 2: HOAP-2 joint location and allowance motion

Side	Joint	Weight ([kg])
R	LEG_JOINT_1	3.93880^{-2}
L		3.93880^{-2}
R	LEG_JOINT_2	1.72696^{-1}
L		1.72696^{-1}
R	LEG_JOINT_3	4.38575^{-1}
L		4.38537^{-1}
R	LEG_JOINT_4	2.85982^{-1}
L		2.85827^{-1}
R	LEG_JOINT_5	1.71128^{-1}
L		1.71128^{-1}
R	LEG_JOINT_6	1.36753^{-1}
L		1.36753^{-1}
R	ARM_JOINT_1	1.99031^{-1}
L		1.99031^{-1}
R	ARM_JOINT_2	2.01439^{-1}
L		2.01439^{-1}
R	ARM_JOINT_3	2.20645^{-1}
L		2.18287^{-1}
R	ARM_JOINT_4	1.67552^{-1}
L		1.67592^{-1}
R	ARM_JOINT_5	0.47200^{-1}
L		0.47200^{-1}
	HEAD_JOINT_1	2.08892^{-2}
	HEAD_JOINT_2	6.69886^{-2}
	BODY_JOINT_1	2.37622^{+0}
	BODY_JOINT_2	4.97680^{-1}
	Total	7.0256448

Table 3: HOAP-2 weights

3.6 Joints angle values

On the real robot, joints angle values are expressed in *pulse*, one degree being equal to ± 209 [*pulses*]. In fact, for some joints, you divide the value in *pulses* by 209 to get the value in degrees, and for some other joints, you divide the value by -209[2].

3.7 Initialization and initial position

Before using the HOAP-2 it has to be initialized. You can see on figure 17 how it has to be done. This procedure will allow the robot to know its current position, and has to be done every time you start working with it.

Once that the robot has been initialized, it can be used, and the motors can be set to angle values. If you set all motors to an angle value of 0[*pulse*], it is then in its initial position. This is really important to know this position. And one needs to respect it when one designs a simulator, otherwise the angle values

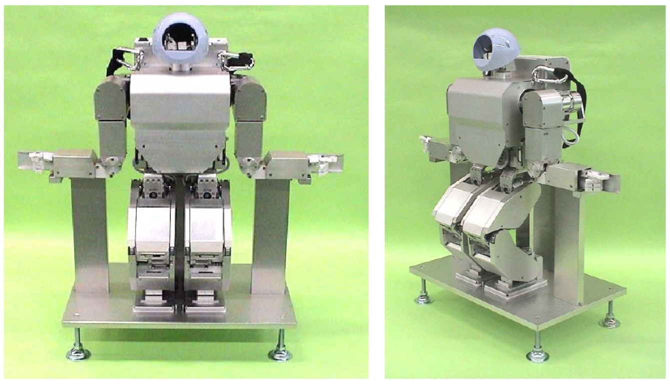


Figure 17: HOAP-2 initialization consists in putting the robot in a very precise position on a jig. The purpose of this manipulation is to put the robot on a known position (start position), and to pre-set the encoders counters of the joints in that posture. By knowing its start position, the robot can then calculate its current position.

will not be correct when you set them on the simulated robot. The figure 18 shows that initial position. You maybe realize that the real robot seems to be leaning ahead a little bit. I will come back to this in section 3.9.1.

3.8 Joints limitations

The joints on the real robot have of course limitations. Every joint has an allowable movement range, included between a maximum and a minimum position (see table 2). It also has a maximum speed and a maximum acceleration that it can not exceed. And last but not least, there is a maximum torque that can be applied to a joint, if you do not want to break it.

In the HOAP-2 user manual, one can find the min and max positions for every joint. But there is not the maximum speed and acceleration, so it had to be measured on the real robot. The EPFL ASL3 laboratory calculated that the maximum speed for every joint was $57[pulse/ms]$, and that the maximum acceleration was about $5[pulse/ms^2]$. But this first value depends on the kind of command that is sent to the robot, and the second of the value of the surroundings joints. This maximum speed value is the default value in the mode *direct command*, but it can be raised up to 70 by the user, by changing the value of a register. In the other mode (*fixed command*), the maximum speed that can be reached is $114[pulse/ms]$. For the maximum acceleration, these $5[pulse/ms^2]$ is to some extent the smallest maximum value that can be reached by every joint, in whatever position. Some joints, in some special positions, can reach higher accelerations. The maximum acceleration depends not only on the joint, but also on the angle value of the other joints, for example the shoulder acceleration depends on the value of the elbow joint, so it is impossible to know precisely this value in every situation[15].

The robot has hardware protections, preventing the user to set a too high joint value (an out of bounds one). If the value is out of bounds, the motor

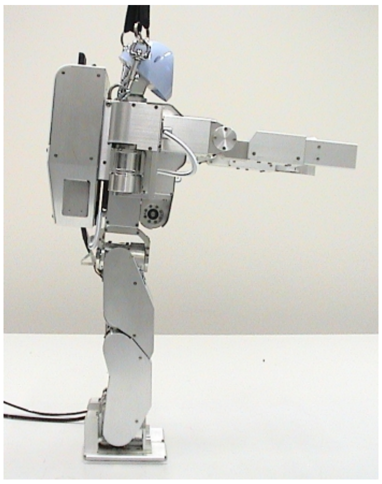


Figure 18: HOAP-2 initial position

stops a few pulse before the limit. So one can not exceed the fixed range. But as far as I know, there is no protection if the user sets a value too different from the current joint position (not reachable in the given time at *max velocity*). So one must be careful not to exceed the maximum speed.[15]

Concerning the maximum torque, there was no information about it in the instruction manual, and the ASL3 lab has unfortunately not calculated these values.

3.9 Backlash

3.9.1 Definition

The backlash is the purposeful clearance between mating components, or the play between mating teeth, and it is present in most mechanical systems (see figure 19). Without backlash, a system would be subject to overloading and overheating, and then as a consequence, the failure of the whole system. The backlash allows also a good lubrication of the teeth, what is important. One can then easily understand that the backlash is necessary, and can not be completely avoided. And it is also important to realize that the backlash is not constant for a system, as the teeth are not all perfectly identical, and that the wear increases backlash over time. The consequence of this backlash is of course the decrease of the control performance, and this problem is a well known one in robotics[8][9][10].

3.9.2 HOAP-2 backlash

On HOAP-2, there is backlash. Effectively, when the robot is in its initial position (all motors in position 0), the robot is not able to stand in a perfectly vertical position (see figure 20). I calculated the difference between the "perfect" initial position, and the "backlashed" initial position with the help of the two

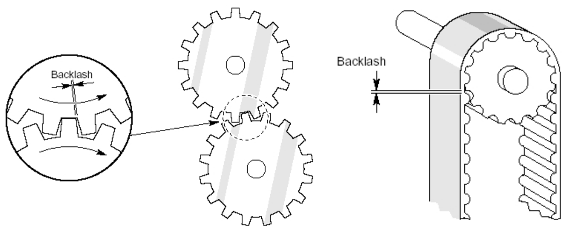


Figure 19: Backlash

pictures, and of the *measure* tool of the Gimp software[3], allowing to measure angles, because it was impossible to calculate this precisely on the real robot with the tools I had in my possession. And I realized that there was about 2[°] of difference with the [R and L]leg5 joints (ankle front and rear), and also about 2[°] difference with the [R and L]leg3 joints (hips front and rear). These angles are represented by the red lines in figure 20.

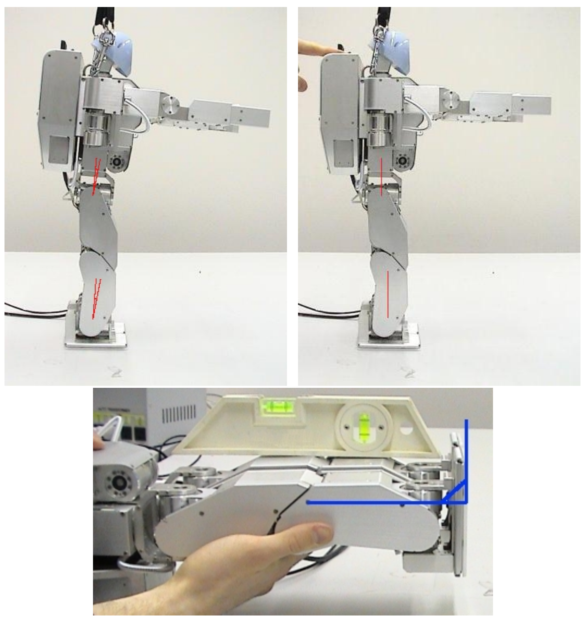


Figure 20: In the top left picture, the robot is leaning ahead a little bit, due to the backlash. But if the robot is hold, as in the top right picture, the robot then stands in a perfectly vertical position, and its arms are perfectly horizontal. If one keeps the robot in the same motors position, but just hang the robot in the air, or lean it on its back, as in the bottom picture, then one observes that the feet stand perfectly at the wanted position. This is then effectively backlash, and the robot is bending under its own weight.

[3]www.gimp.org

As a test to see the accuracy of my measurements, I did the three pictures of the figure 21. It is pretty hard to realize that there is two robots on the last picture of figure 21. But that is exactly what I wanted to demonstrate: after these two transformations of 2[°], the position is then virtually the same.

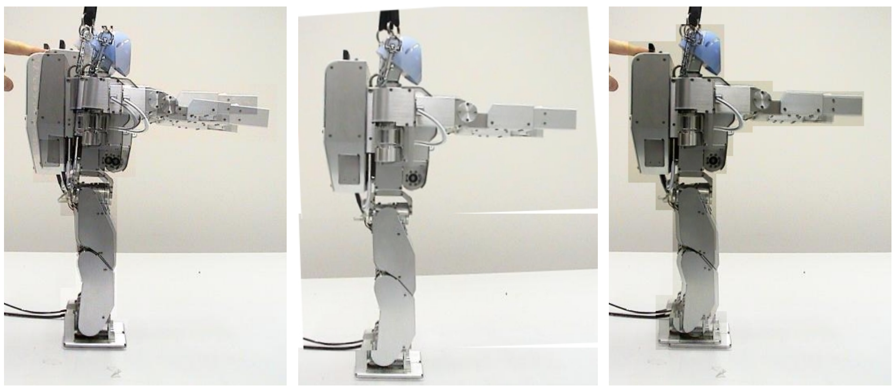


Figure 21: The left picture is just both initial positions (the one with backlash, and the one without) superposed with transparency on the same picture. The picture from the middle is the bended initial position, but I rotated parts of the picture with the help of Gimp, the whole robot by 2[°] around the leg5 joints, and the torso and arms by 2[°] around the leg3 joints. The right picture is the incrustation of the picture of the middle, in the picture of the robot without backlash (top right of figure 20)

4 Webots design of the robot

4.1 Foreword

My simulator is contained in the *hoap2.wbt* Webots' world. I will not explain clearly here how I designed it under Webots, since it is not really exciting or interesting. Basically, I just followed the instructions explained on the Webots user manual on how to design a new robot under Webots. I used both the *scene tree editor* and a simple text editor to write it. In the section 6, you can find a few hints and recommendations on how to design a new robot easily.

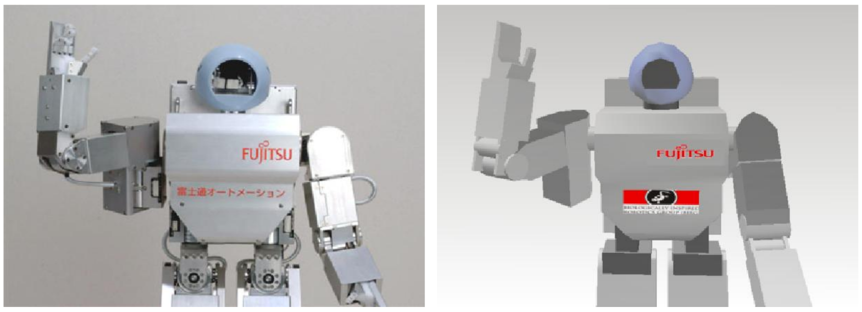


Figure 22: HOAP-2 robot and its Webots model

4.2 Modifications of the semester project model

A few weeks before the end of my semester project[3], I still had no precise specs of the robot. I found a very precise scale drawing (see figure 23) of the robot, on the Fujitsu Japan web site a few days before the dead line, so it was to late to take care of this during this semester project. At the beginning of my diploma work, I have also been given two very important docs: the HOAP-2 instruction manual[2] [4], and the inspection report [5] of the robot the EPFL bought. This last document has been made by Fujitsu's Engineers, and it is the very precise specifications of the robot. Basically, after they built the robot, they studied every part of the robot, every joint [6], every sensor, etc. And then they came with that document, concerning only our robot, containing, for every joint, the mass values, center of mass positions, and inertia matrices [7], precise at 10^{-5}[kg] and 10^{-5}[mm]. And all of this for every single joint. So it was very useful for the realization of my simulator.

So I had in my possession a very precise design of the shapes, and of the weight of the different parts of the robot. I also had almost permanent access to the real robot. I shall then be able to design an accurate model of it.

[4]This document was to big to find its place in this report, but it is easy to find it on Fujitsu website, so I did not put it as an appendix.

[5]See Appendix 12.1.

[6]For the real robot, I will talk of *joints* and of *motors*. In Webots the name *servo* is used to design whatever kind of servo motor.

[7]This is a matrix that describes how the body's mass is distributed around the center of mass.

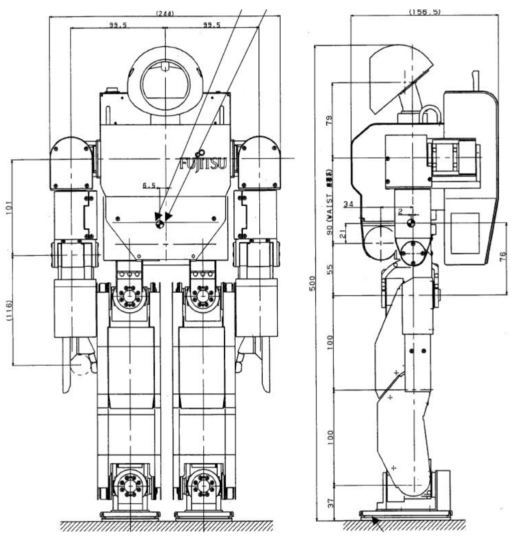


Figure 23: HOAP-2 scale drawing

4.3 Robot nodes

The table 4 is an overview of the hoap2 world definition, including the HOAP-2 robot. The Webots version is the 5.0.0.. The level of indentation of the text reflects the hierarchy in the model.

4.4 Shapes

I had already designed the shapes of the robot during[3], using a software called Art Of Illusion (AOI)[8]. It is a free, open source 3D modeling and rendering studio, entirely written in Java. I only changed a few details from these initial shapes, like their sizes, when they were not correct (just remember that I had no precise data about the size of the shapes during[3]). I also changed the shape of the battery. But most of the time I just had to rescale the shapes.

[8]http://www.artofillusion.org

```
# Webots version 5.0.0
# Hoap-2 world ”hoap2.wbt” – structure of the world
# Lists the structure of the world, and for the robots the servo and sensor structure
WorldInfo ”Model of a hoap-2”
    ”Pascal Cominoli”
    ”Date: WINTER 2004-05”
Viewpoint
Background
PointLight
PointLight
GROUND Solid
TATAMI Solid ”ground”
    WHITE_CENTER Shape texture ”hoap2/fuji.png”
    RED_BORDER Shape
    STAGE Shape
CustomRobot ”Hoap2_0”
back_1_0 Servo ”body_joint_1”
    left_hip_1_0 Servo ”lleg_joint_1”
        left_hip_3_0 Servo ”lleg_joint_3”
            left_hip_2_0 Servo ”lleg_joint_2”
                left_knee_0 Servo ”lleg_joint_4”
                    left_ankle_1_0 Servo ”lleg_joint_5”
                        left_ankle_2_0 Servo ”lleg_joint_6”
                            TouchSensor ”left touch”
    right_hip_1_0 Servo ”rleg_joint_1”
        right_hip_3_0 Servo ”rleg_joint_3”
            right_hip_2_0 Servo ”rleg_joint_2”
                right_knee_0 Servo ”rleg_joint_4”
                    right_ankle_1_0 Servo ”rleg_joint_5”
                        right_ankle_2_0 Servo ”rleg_joint_6”
                            TouchSensor ”right touch”
left_shoulder_1_0 Servo ”larm_joint_1”
    left_shoulder_2_0 Servo ”larm_joint_2”
        left_shoulder_3_0 Servo ”larm_joint_3”
            left_elbow_0 Servo ”larm_joint_4”
                left_hand_1_0 Servo ”larm_joint_5”
right_shoulder_1_0 Servo ”rarm_joint_1”
    right_shoulder_2_0 Servo ”rarm_joint_2”
        right_shoulder_3_0 Servo ”rarm_joint_3”
            right_elbow_0 Servo ”rarm_joint_4”
                right_hand_1_0 Servo ”rarm_joint_5”
neck_0 Servo ”head_joint_1”
    neck_tilt_0 Servo ”head_joint_2”
Emitter ”emitter” range 100
GPS ”gps”
controller ”hoap2”
```

Table 4: hoap2 world definition

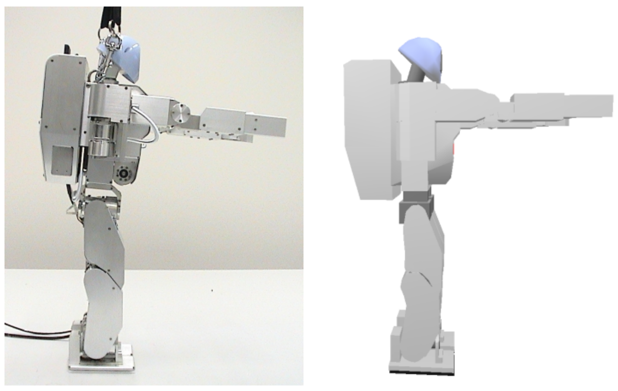


Figure 24: hoap2 initial position

4.5 Joints

4.5.1 Initial position

I did not know the initial position (see figure 24) of the robot during[3]. So as soon as I knew it, I modified my model accordingly. Basically I rotated the shapes that were not oriented correctly, and I changed the positions of some joints, to correspond with the robot initial position. So if you set a joint at a certain position on the real robot, the position is exactly the same on the simulated robot, what is of course mandatory for a simulator. As I already told you, the real robot suffers some backlash[9]. I will explain in section 4.5.3 how I tried to adapt my model to correct this problem.

4.5.2 Joints limitations

It was really important to respect the real robot joints restrictions on the simulated robot, because it is crucial that prohibited or impossible moves on a real robot are not realizable on the simulated one. Otherwise when one will export its simulation on a real robot, one will be really disappointed to see that the real robot can not perform this move, or one could be far more than just disappointed, if this impossible move broke an arm of the robot. So one can easily understand that only the moves allowed on the real robot should be allowed on the simulated one.

For the maximum velocity and maximum acceleration, as I said in section 3.8, there is no precise values valuable in every situation, and some values are reachable only in special cases. So it can not be precisely set in a simulator. So I decided to set the maximum speed of the joints in my simulator at the value of $57[pulse/ms]$, which is a value that can be reached by every joint in whatever situation and whatever kind of command sent to the robot. But one must just keep in mind that this value is only the default value of the real robot, and that if the user decides to change this value contained in a register to a smaller value, this could be problematic. But on the usual cases, users are not supposed to change this register. For the maximum acceleration, I used the $5[pulse/ms^2]$

[9]see section 3.9.1

value, which is a already good approximation, since I am sure that every joint is able to reach at least this value.

In Webots, one can set these min positions, max acceleration, etc. values, in the *servo* node. I let you have a look at the section 6 to see how one can set these fields.

So if one asks for an impossible move, for example if a joint value is 0.0[rd] at time T, and you set it at 1.0[rd] (with a controller or with Webots' sliders) at time T+1ms, then you will see the joint reach max acceleration, then max speed, then slow down, and finally stop smoothly at position 1.0 after a lot more than just 1ms.

But this will not prevent the user to upload this controller on the real robot, and then maybe damage it. The only thing it does is that the simulated robot will perform the asked move as fast as it can, respecting the real joints specs. So we decided to implement a procedure directly in Webots source code, showing a warning in the log window when one is asking too much to a joint [10]. So one is sure that if one's controller functions properly and without warnings on Webots, it can be safely uploaded on the HOAP-2.

4.5.3 Backlash consequence on the simulator

Now that we know that there is backlash on HOAP-2, we need to find a way to take care of this in Webots. The motors I use in simulation are effectively supposed to be perfect ones, and without backlash. But in real life, a motor is never perfect. Two motors engineered on the same construction line, by the same tools, will not be 100% identical. There is always a few differences. The backlash is another problem that is very difficult to realize perfectly in simulation, since it depends on the kind of motor, on the weight of the robot, on its construction, on the motor wear, and as I said is not perfectly constant on a single motor. Webots 5.0.0 does not handle the motor backlash (and I do not know if it will in the future), and I had not the time to implement it by myself. So I could not render this realistically, and the only thing I could do during this project, was to modify the angle values of the initial position of my simulated robot, to make it bend like the real one, as you can see in figure 25 [11].

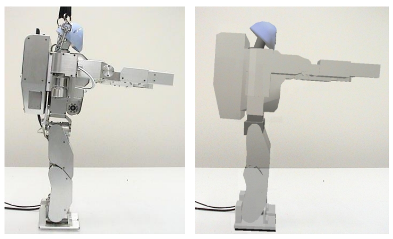


Figure 25: This modified initial position consists just in setting the servos from the ankle (front and rear) and from the hips(front and rear) to a value of 2[o]

[10] At the time I was handing this report, this was still a project, and its implementation had not been done by Olivier.

[11] See section 8.

5 Changes introduced in Webots

Working on the design of my HOAP-2 model, I sometimes realized that there were bugs –or things that we could simply not do– in Webots. Then it was always the occasion of big talks between me and Olivier Michel, searching for how to correct the bug, or how to implement a new functionality that would be really useful for me, and for the next users.

For example, on a previous Webots version (<4.0.27), a servo's center of mass and center of rotation were indissociable. And we can reasonably assume that it is rarely the case in robotics. So Olivier implemented a new field, allowing to move the center of rotation of a servo, the center of mass being the center of the servo. But I succeeded in convincing him that it was not appropriate for the design of a robot.

Effectively, when one designs a new robot, most of the time, what one is sure of, is the relative position of the servos, not the positions of the different centers of mass. For example in my situation, during [3], I knew perfectly the positions of the servos, but I had not a single clue about the centers of mass positions. Hence, if I had to design my robot starting from the positions of the centers of mass, I would have had to imagine where they were, and then to displace every center of rotation to its well known position. And if I suddenly realized that one center of mass was not where I thought it should be, I would have had to modify its position, then the position of the corresponding center of rotation, and then do exactly the same for every servo inheriting from this servo, moving center of mass, and then center of rotation. It would really not be practical.

And even now that I have precise position of both centers of mass and of centers of rotation, I really do not think it would be convenient to base everything on the centers of mass. I do know the positions of the centers of rotation. And theses values are fixed and will never change. But I will maybe have to modify by a few the center of mass' positions, to be more accurate with real robot [12]. If centers of rotation position was dependent of centers of mass position, it would be a big mess to rearrange it every time I think I should displace a center of mass by a few millimeters.

So we managed it the other way, having the center of the servo (its translation) as the center of rotation, and then allowing the user to displace the center of mass, if he wants to. With an additional field (*Joint*), it is also possible to displace the servos center of rotation (as an offset from the *translation*). I did not use this last field, but it had been added to Webots for a long time. Due to the ascendant compatibility that Olivier Michel wants to keep on his software, we kept this node.

But I then realized that there was a problem with the center of mass position. We had decided to enter its value in the *inertiaMatrix* field, the three firsts of its values being it (as it is implemented in ODE [13]). But it didn't seem to have a single effect on the simulation. Having, for a servo, its center of mass in (0,0,0) or in (10,0,0) –so ten meters away, there was no reaction. After searching

[12] See section 8.

[13] As you probably know, Webots relies on ODE for every physics simulation.

a little bit, on ODE forums and archives, I became conscious that on ODE, the center of mass position was always equal to the body center position (=(0,0,0)), and that there is no way of changing it. Hence, moving it to a different position was not even taken in account.

We then again had to modify Webots software. Instead of letting the body in this position, and moving its center of mass, we decided to translate the whole body, at the position where we wanted to put the center of mass. Hence the center of mass was still in (0,0,0), and everything was fine. In fact, we created the ODE bodies at the position of their center of mass. Then, after ODE has calculated all the physics at time t, and its effects on the bodies, we translated every body back to the wanted position (basically, we applied the inverse transformation than the one that was applied on the body). We used for this two new fields: *translation* and *rotation*, both located in the *physics* node. The field *translation* contains the position of the center of mass, relative to the local axes. These local axes are taken from the *rotation* node. We effectively added the possibility to give a new orientation to a servo, allowing for example the user to have rotations always effective around the same axis. In the inspection report, I had inertia matrices and center of mass positions that were relative to every servo given orientation (corresponding to a rotation around the Z axis). This new *rotation* field allows the user to modify the local orientation of the axes. If we had not implemented that new functionality, I would have had to rotate every inertia matrix and center of mass position, to be corresponding with Webots axes orientation.

Olivier then gave me the sources of Webots, and let me add this new functionality by myself. As I already said, we wanted to implement to following:

- The center of mass of a solid is located at a position equal to the body *translation* field, plus the offset represented by the field *translation* from the *Physics* node.
- The position of this center of mass can be rotated. This rotation also rotate the inertia matrix of the solid. This rotation is contained in the *rotation* field from the *Physics* node.

So I had to face two main problems: displace and rotate the center of mass, but also rotate the mass. But first of all, let me just explain quickly what are a Body and a Geom in ODE.

Body An ODE Body has various properties from the point of view of the simulation. Some properties change over time, like the position vector and the orientation. Some other properties are usually constant over time: the mass, the position of the center of mass, and the inertia matrix (this is a 3x3 matrix that describes how the body's mass is distributed around the center of mass). It is equivalent to the *Solid* node in Webots.

Geom Geometry objects (or "geoms" for short) are the fundamental objects in the collision system. A geom is usually represented as a single rigid shape (such as a sphere or box). The geom is used by ODE to calculate the collisions between objects. The easiest way to represent it, is to see it as a box surrounding the object, and that is used to calculate collision detection with the other objects geom, without caring of the object real shape. It is equal to the *BoundingObject* in Webots.

Conceptually each body has an x-y-z coordinate frame embedded in it, that moves and rotates with the body, as shown in figure 26.

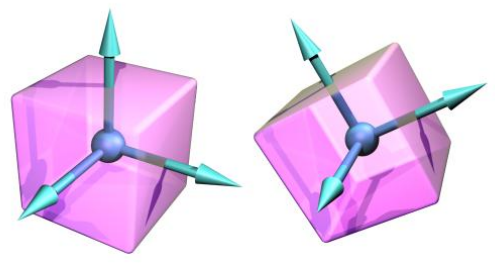
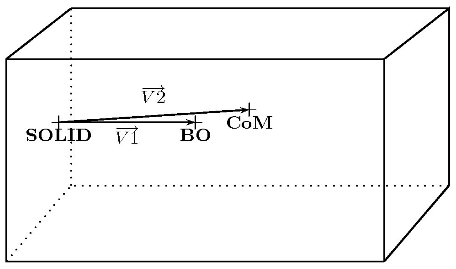


Figure 26: ODE body coordinate frame

To use the collision engine in a rigid body simulation, geoms are associated with body objects. This allows the collision engine to get the position and orientation of the geoms from the bodies. Note that geoms are distinct from rigid bodies in that a geom has geometrical properties (size, shape, position and orientation) but no dynamical properties (such as velocity or mass). A body and a geom together represent all the properties of the simulated object. [23]

So I had to take care of this in four Webots functions: createGeomFromTransform (sets the geom for a body), setBody (sets the corresponding body), runAfter (recover the body position after ODE applied the involved forces on the bodies, in order to display them correctly under Webots), and addMass (assigns a mass to a body).



$\overrightarrow{V1} =$ ***BoundingObject->translation***
$\overrightarrow{V2} =$ ***Physics->translation***

createGeomFromTransform() The center of mass (**CoM**) is the center of the ODE body. But I want to have the center of my bounding object in the position **BO**. So I have to subtract the *Physics->translation* to the *BoundingObject->translation* in order to get back to the bounding object[14]. But we must not forget that the translation contained in the *Physics*

[14]Bounding object position inherits from the corresponding body's position. So this translation is only locally effective.

node might be rotated by *Physics->rotation*. So I have to rotate the vector *Physics->translation* by the rotation *Physics->rotation* before withdrawing it from the *BoundingObject->translation*. And it all goes with matrices calculations. I calculated the Rotation Matrix with the ODE function dRFromAxisAndAngle, that takes a vector and an angle (here equal to the *Physics->rotation* field), and returns a rotation matrix. I then had to multiply my vector *Physics->translation* by this rotation matrix, to get my vector correctly rotated. Mathematically:

$$\underbrace{\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}}_{\overrightarrow{Rotation\ Matrix}} \times \underbrace{\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}}_{\overrightarrow{physics->translation}} = \underbrace{\begin{pmatrix} a*X+b*Y+c*Z \\ d*X+e*Y+f*Z \\ g*X+h*Y+i*Z \end{pmatrix}}_{\overrightarrow{result}}$$

I can then set my geom to the position $\overrightarrow{BoundingObject\text{->}translation}$ - $\overrightarrow{result}$. Concerning its rotation, I just have to rotate the BO with the possible rotation *BoundingObject->rotation*. I do not need to care with the body rotation, since as I said, the BO inherits from its body translation and rotation.

setBody() The **CoM** is the position at which I want to set the body, but this body is by default at the position **SOLID** (equal to *Solid->translation*). So I have to take the field *Physics->translation* from the *Solid* node, and rotate it by the field *Physics->rotation*, and then again by the field *Solid->rotation*, which is the orientation of the solid in the world, and changes throughout the simulation. The vector obtained is then added to the field *Solid->translation*, to get the position of the center of mass of this solid. Finally, I set the rotation of the solid, that is equal to the rotation contained in the matrix solid.matrix[], that is the OpenGL 4x4 matrix, maintained by Webots to know the absolute transformation that gives the position and orientation of a Solid. Then we check if the body has children, and if it is the case, we apply recursively the setBody() function on them.

runAfter() This is the procedure that took me the most time to correct. I had to recover the body positions after ODE made its job and applied the physics on the different bodies that are concerned by the physic. Basically, I had to compute exactly the inverse transformation that I applied on the body in the setBody function. So I had to go from the **CoM** position that ODE knows, to the body position. For this, I took the relative position of the current body center of mass ($\overrightarrow{V2}$), that I rotated by the *Physics->translation*, to get its rotated position. I then had to rotate this point by the body orientation by using the body rotation contained in solid.matrix[]. If this body has no parents, then the position of the body is the body position returned by ODE, at which I subtracted the vector I got from the previous matrix calculations. On the other hand, if the body has a parent, we need to get the position relative to its parent, because it has been constructed this way. So we need to get the body position relatively to its parent, in order to set its position and rotation relatively to its parent. I used for this the ODE::dBodyGetPosRelPoint() function, that takes a point in global coordinates and returns the point's position in body-relative coordinates. The problem is that the point it gives us back is the **CoM** position of the ODE body, and I need the Webots body position. So I need to rotate the $\overrightarrow{V2}$ vector of its parent by the *Physics->rotation* of its parent, and then add the result to the previously got vector, to finally get what I wanted.

addMass The best way I found to rotate the mass (and trust me, I tried many) was to do it before it was assigned to a body. So in the procedure addMass(), I checked if the field *Physics->rotation* was set, and in this case I applied the corresponding rotation to the mass. This rotation induces a rotation of the inertia matrix from the corresponding body, but does not modify the position and the orientation of the body. It is only the effect of the mass on the body that is changed.

5.1 Evaluation of my new Webots version

Before I could go any further, I had to verify that this new Webots version was correctly calculating the physics. So I designed some little worlds with just a few simple boxes to test the improvements, as one can see in the two examples from figures 27 and 28. And I got really good results. So now that Webots had been improved, and that the results I got with simple physic tests seemed correct, I was able to continue the development of my simulator [15].

[15]For more informations about these modifications, see section 6, and I will also let you have a look at Webots change log, and at its user guide and reference manual, available on its website.

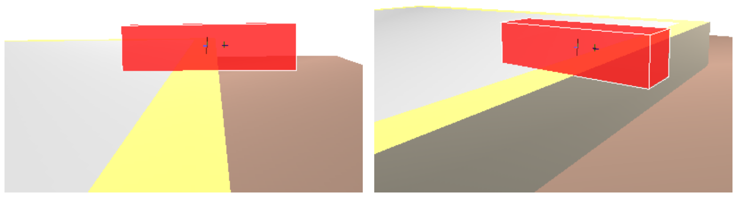


Figure 27: This world only contains a white floor (in fact an elevated tatami) and a red box. I designed it to test if the center of mass rotation was correctly effective, by putting the box on the edge of the tatami, with the box center that was on the tatami, and by moving and rotating the center of mass to a position that was outside the tatami. By doing it while Webots was in *run* mode, one could see exactly when the balance limit was reach. And it was when the center of mass was outside the tatami. So one can say that this test is a valid one.

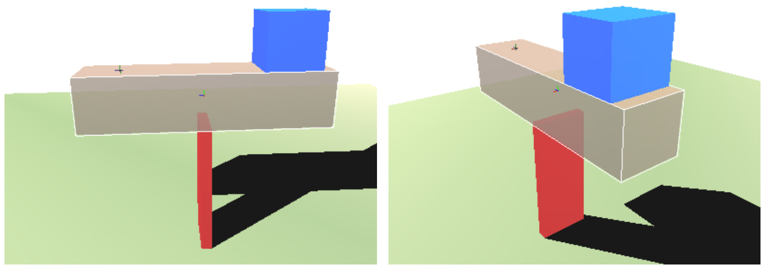


Figure 28: This world contains three boxes and a floor (light green). The brown box is standing on the red one, and an extra blue box is standing on the brown on. The little cross in the middle of the brown box is its center (**SOLID**). One can see that it is centered perfectly on the red box. The other cross on the left side of the top face is the position of the box center of mass. The blue box is then situated at exactly the same distance from the middle of the brown box, than the center of mass. Both brown and blue boxes weight exactly the same. So physically, the system is supposed to be stable, as both masses are at equal distance at the point at which the forces are effective (the contact point between brown and red boxes). And if one runs the simulation, the system happens to be effectively stable, as it stands in this position and does not fall. But as soon as the blue box or the center of mass of the brown box are displaced by a little offset, system becomes unstable, and the three boxes fall on the ground. So again, the test is valid.

6 How to design a new (humanoid) robot on Webots

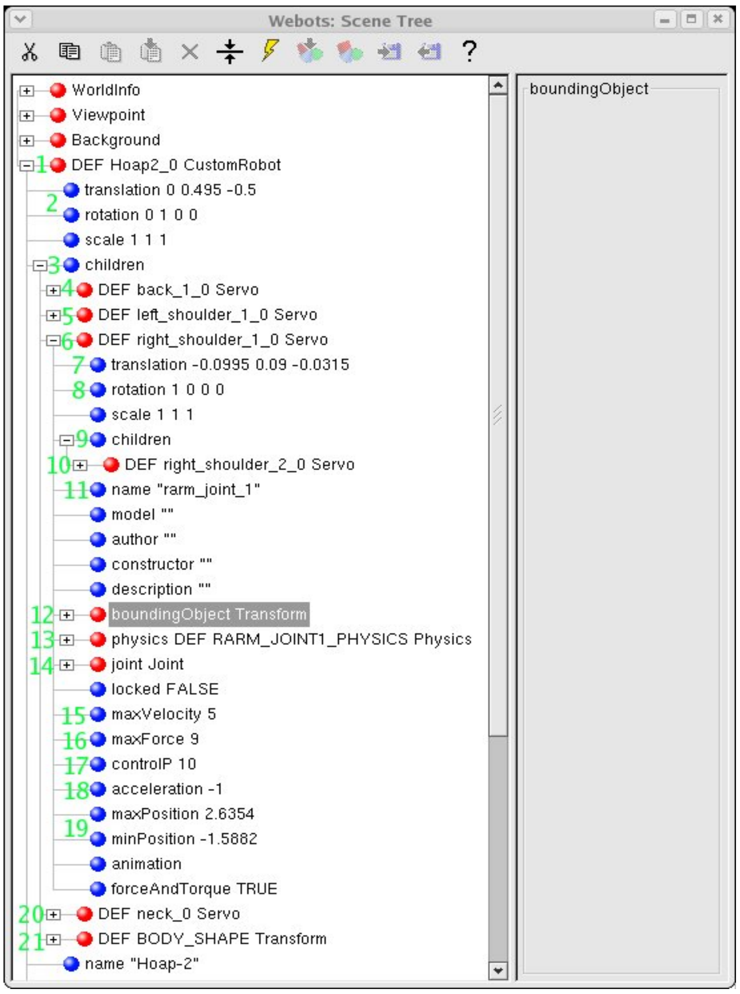
As I already said, this project was the first one, which aim was to develop a very precise simulation of a real humanoid robot on Webots. The QRIO model was the first humanoid developed on Webots, but it had been done without a single precise information about the real robot. They did not get the specs from Sony, and they had to recover the shapes from pictures (as I did during [3], before having precise specs). They had no weights, velocity, etc information. And they also had no real QRIO to make experiments, to check if their model was accurate. For me it was totally different, while I had access to the real robot, and had precise specs. I think thanks to my work, one is now able to design quite easily whatever kind of humanoid robot in Webots. There has been a lot of Webots updates since this project about QRIO[7]. And a lot of things have changed. So here is a little procedure, about how to design a new humanoid robot on Webots, and a little explanation of every important node and fields. I won't talk here about the very first nodes, like the *WorldInfo*, or the definition of the ground. I won't neither talk about the way one creates and inserts nodes using the scene tree window. Please refer to [4] for further informations, if needed. The easiest way to perform such a design is probably to do it with the scene tree editor of Webots. During[3], I worked only with a simple text editor, and did not use the scene tree editor. But now, it has been a lot improved, and is a lot more stable.

But before starting with this little tutorial, I just would like to warn you. In this section, I will introduce two new fields contained in the *physics* node, called *physics->translation* and *physics->rotation*. I implemented this on Webots, and it is working just fine (see section 5). But the names of these two fields might change very soon, as we were unsure with Olivier of the correct names for it. So maybe these names will be changed to something like *centerOfMass* for the first one, and *orientation* for the second one. But the location, the use and the functionality of it will not change. During my diploma (winter 2004-2005), the current Webots version was Webots 5.0.0.. I just hope that this protocol will still be valid for the next versions[16].

The first node one will need is a *CustomRobot* node (1[17]). Its position in space (its *translation* and *rotation* fields (2)) is somehow the initial position of your robot. As its children (3), one has a transform node containing the corresponding shape, the torso for example (21). The other children will be the servos, respecting the hierarchy. In my case for example, I had the body servo (4), the left shoulder servo (front and rear) (5), the right shoulder servo (front and rear) (6), the neck servo (20), etc. One can give them a *DEF* name for every of them, but this is not mandatory. On the other hand, you should give them a *name*. It will allow you to get them back in the controllers you will write to control your robot.

[16]Depending on the current Webots version at the time when you begin your design, you would better read the reference manual, or even ask Olivier Michel, before starting, to be sure that there has not been too many modifications since I wrote this report.

[17]See figure 29.



```
WorldInfo
Viewpoint
Background
1 DEF Hoap2_0 CustomRobot
    translation 0 0.495 -0.5
  2 rotation 0 1 0 0
    scale 1 1 1
  3 children
    4 DEF back_1_0 Servo
    5 DEF left_shoulder_1_0 Servo
    6 DEF right_shoulder_1_0 Servo
      7 translation -0.0995 0.09 -0.0315
      8 rotation 1 0 0 0
        scale 1 1 1
      9 children
     10   DEF right_shoulder_2_0 Servo
     11 name "rarm_joint_1"
        model ""
        author ""
        constructor ""
        description ""
     12 boundingObject Transform
     13 physics DEF RARM_JOINT1_PHYSICS Physics
     14 joint Joint
        locked FALSE
     15 maxVelocity 5
     16 maxForce 9
     17 controlP 10
     18 acceleration -1
        maxPosition 2.6354
     19 minPosition -1.5882
        animation
        forceAndTorque TRUE
   20 DEF neck_0 Servo
   21 DEF BODY_SHAPE Transform
  name "Hoap-2"
```

Figure 29: Webots scene tree

And then one continue the hierarchy, with the following servos. For example, the children (9) of the right shoulder servo (front and rear) will be the right shoulder servo (right and left) (10), and so on, until the hand servos. And for every servo node, as its second child, one has the shape of the corresponding body part, if needed. Every servo will contain in its translation field (7) the translation needed to go from previous servo, to the current one. Here for example, the right shoulder servo is located (-0.0995,0.09,-0.0315) meters from the initial position 2 –notice also that translations are expressed in meters. This position is the center of rotation of the servo. If for whatever reason you would like to displace it, you can put an offset in the *Joint* field (14).

In its *rotation* field (8), one will insert the axis around which the servo is rotating. For example if the rotation is effective around the X axis, one would put *rotation 1 0 0 0.*

For every servo, one needs also to set its *boundingObject* (12). This bounding object is needed to perform the collisions detections. One can design it as accurate as one needs it to be. There is many predefined shapes, like boxes, cylinders, but one can also use complex shapes, like the precise shape of the corresponding body part. But the most complex your bounding object will be, the slower the simulation will run. Predefined shapes are already good enough for usual cases. This bounding object can be a *Transform* node or directly a shape. Use a *Transform* node if the *boundingObject* is not centered around the current body origin, or if you need to rotate the bounding object.

If many servos or solids are defined at the same position[18], one does not need to design a shape and a bounding box for each of them. One just has to design it for one of them (usually the last servo/solid in the hierarchy), and then either one does not define the unwanted bounding objects, or defines them with very small sizes (~ 0.001).

The *maxVelocity* (15), *maxForce* (16), *controlIP* (17) and *acceleration* (18) are all used to control the rotation of the servo. Refer to the user manual for more informations. Both *maxPosition* and *minPosition* (19) define the servo allowable movement range for the corresponding servo. The values are radians angles. They can be positive or negative.

If one wants to design a precise simulator, one needs precise physics definitions. This is easily feasible, thanks to the *physics* (13) node. In this node, there is many fields:

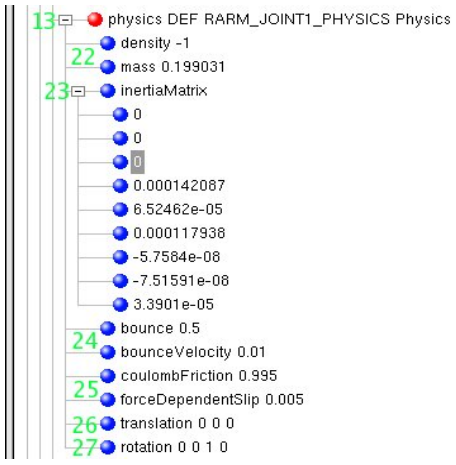

```
13 physics DEF RARM_JOINT1_PHYSICS Physics
      density -1
   22
      mass 0.199031
   23 inertiaMatrix
        0
        0
        0
        0.000142087
        6.52462e-05
        0.000117938
        -5.7584e-08
        -7.51591e-08
        3.3901e-05
      bounce 0.5
   24
      bounceVelocity 0.01
      coulombFriction 0.995
   25
      forceDependentSlip 0.005
   26 translation 0 0 0
   27 rotation 0 0 1 0
```

Figure 30: Webots physics tree

- *translation* (26): this allows to translate the center of mass of the corresponding solid (or servo). By default, the center of mass position is equal to the rotation center (the *translation* field (7) of the *servo* node). But it is seldom the case in robotics. So the value of this *translation* field is the offset added to the position of the servo, to reach the center of mass.

[18]For example in the shoulder, there is often much more than just a single servo, since one performs the right-left rotation (6), an other one the front-rear (10), etc.

- *density* and *mass* (22): one can set one of these fields to the value of density or mass of the current servo. If one defines a density, the mass must be set to -1. If one sets the mass to a value, then it is the density that must be set to -1.

- *inertiaMatrix* (23): as its name shows it, it contains the inertia matrix of the servo. It contains 9 values, even if only the last 6 are used: (COM1, COM2, COM3, I11, I22, I33, I12, I13, I23), represented as follow:

$$inertiaMatrix = \begin{pmatrix} I11 & I12 & I13 \\ I12 & I22 & I23 \\ I13 & I23 & I33 \end{pmatrix}$$

  COM1-3 should always be set to zero. We only realized recently that these three first values are not used by ODE[19]. But we decided to let it this way, to keep ascendant compatibility. Effectively, the field *inertiaMatrix* is present in Webots for a long time. We did not want to change it. If one does not set these three first values to zero, one will get a warning. As you already know, the center of mass position is set thanks to the *translation* (26) field from the *Physics* node. This matrix is ignored if this field contains not exactly 9 values.

- *bounce* and *bounceVelocity* (24): the bounce parameter defines the bounciness of a solid. This restitution parameter is a floating point value ranging from 0 to 1. 0 means that the surfaces are not bouncy at all, 1 is maximum bounciness. When two solids hit each other, the resulting bounciness is the average of the bounce parameter of each solid. If a solid has no Physics node, and hence no bounce parameter defined, the bounce parameter of the other solid is used. The bounceVelocity parameter defines the minimum incoming velocity necessary for bounce. Incoming velocities below this will effectively have a bounce parameter of 0.

- *coulombFriction* and *forceDependentSlip* (25): The coulombFriction parameter defines the friction parameter which applies to the solid regardless of its velocity. Friction approximation in ODE relies on the Coulomb friction model and is documented in the ODE documentation. The forceDependentSlip parameter defines the force-dependent-slip (FDS) for friction, as explained in the ODE documentation. FDS is an effect that causes the contacting surfaces to side past each other with a velocity that is proportional to the force that is being applied tangentially to that surface. If you have no idea of values for these two fields, just put values like 0.995 for the *coulombFriction*, and around 0.005 for the *forceDependentSlip*. This is just to be sure that you will not have the kind of strange behaviors, that one can have if these fields are at their default values.

- *rotation* (27): this field is useful for modifying the orientation of the local axes. Sometimes in robotics, a convention specifies that a rotation is always effective around the Z axis. But it is not the case in Webots. This field allows then one to modify the local orientation of the axes. This will of course affect the position of the center of mass, and the values of the inertia matrix.

[19] See section 5.

7 Controllers developed

7.1 csv2Webots

The purpose of this controller is to allow the user to run a csv[20] file on the simulated robot. This controller does the following:
When one runs the world, it will ask for a (csv) file. The program will idle until one will provide it with a file. When it finds it, it first interpolates the moves from its initial position, to the start position of the file. One will then see the robot move smoothly until the start position of your file (basically the first line) is reached. Then it will execute the position file. At the end of the file, it will ask for the next file to be executed. If one is not running a file, one can move the robot with the sliders, what allows to start the simulation in whatever wanted position.

7.2 hoap2moves

The purpose of this controller is to allow the user to give the wanted position for a servo as a text input, and to watch the robot smoothly reach this position. An interpolation is effectively set between the current position and the wanted position, in order to have smooth and safe moves. The working of this controller is easy: on the standard input, one can see the value of every servo, in radians and in pulse[21]. Then, to increase the servo number 04 by 0.3 radians, one just writes *04 0.3* and press *return*. One can also reset a single servo position to its initial value (*0*), or reset every servo at the same time. To reset a single servo, one puts the servo number, followed by an angle of *9*. To reset all servos, one puts whatever number of servo, followed by *666* as the angle. It is also possible to write the current position to a csv file, by putting the angle value as *999* (and whatever servo number). This is very useful, for example when I was doing balance tests, it allowed me to bend smoothly the robot, and as soon as the robot reached the ultimate balance posture, I asked for this csv file, and then I could very easily test it on the real robot, with exactly the same file, to verify if my model was accurate.

7.3 postures_protocol

This one was very useful to me, and will also be to the next people in charge of the updating of the simulator. This controller (and its dedicated supervisor) performs 16 balance tests, measuring the maximum angle that can be applied to some servos, before the fall of the robot. This fall is recognized thanks to foot sensors I put on the feet of the robot. When the value returned by the foot sensor is different from 1, it means that the robot is falling, because one foot is loosing contact with the ground.

For example for the first test (leg3_Back)[22], both lleg3 and rleg3 are incremented at every step by a small amount of angle, until the robot fall. And at

[20]A csv file is a position file, containing, every X ms (usually 2ms), the position for every motor, a line in this text file being one time step. These files can be easily generated on the HOAP-2 computer. Some demo csv files were also provided with the robot, like the m01 walking sequence.

[21]The pulse is the unity used by the robot, and 1 degree is equal to 209[pulse].

[22]See section 8.

every step, the robot controller sends to its supervisor the current angle value (the angle is the same for both servos, so one only needs to know the value for one of them), followed by the touch sensor value. At the same time, the supervisor sends to the robot the number of the test to perform (1-16). As soon as one foot sensor is different from 1 (value that the sensor has when the foot is on the ground), then the supervisor realizes that the robot is falling, records the current angle value the robot sent him, and finally reload the simulation, asking the robot to perform the next test. At the end of the tests, the supervisor writes the results for the 16 tests in a text file. This file contains the measured value for each of the sixteen tests, followed by the real value[23], and then by the difference between the two, what is a good way of evaluating the current version of the simulator. Thanks to this controller, it is then really easy to perform postures and balance tests on a new model of the hoap2. One just has to run this new world with these two controllers, and collect the results to see the accuracy of the new world. And by running it in fast mode, only about ten minutes are necessary to have the complete evaluation.

[23]See section 8.

8 Postures protocol

The goal of this diploma work was the realization of an accurate simulator of a real HOAP-2. So my simulator should react exactly like the real one. Hence it does not only concern the fact that when one applies some angle to a servo, it shall apply exactly the same angle on my simulator. Physics must also be reacting correctly. I then took a lot of time to proceed to precise measurements of balance tests. I first performed the tests on the real robot, by choosing some "key" moves. I chose 16 moves, reflecting well the effect off the gravity on the robot[24]. For every test, I started from the initial position of the robot, and then slowly incremented (or decremented) one or two servos, to have a symmetrical move. For example for test one, I incremented both LLEG_JOINT3 and RLEG_JOINT3 (front and rear thigh joint), until the fall. It is not really easy and safe to do it on a real robot, but I managed it by hanging the robot, and then uploading a csv file corresponding to the wanted final position. I then put the robot on a flat ground (I checked if the ground I used was flat with the help of a level), and looked if the robot could stay up with such angle values. Depending on the answer, I changed the csv file, by increasing or decreasing the concerned servos, until I got the balance limit.

On the simulated robot, I used the controller *postures_protocol* I developed. Before I had designed this controller, I did my balance tests by the hand, increasing slowly the wanted servos, and stopping right when the robot fell, by using the *hoap2moves* controller. But it took a lot of time, and was not very precise. So this first controller was of a great help. And running it in the *fast* mode, it ran about seven times faster than real time, so in about ten minutes I have precise results in a text file.

That kind of tests allowed me to realize that there was problems with mass repartition on Webots 4.0.27, because my results were far from identical between simulation and real. Effectively, for example with the body servo, the difference between the limit angle in simulation and in real was bigger than 30 degrees. So we modified Webots accordingly[25]. I did not get excellent results for all situations, but I did my best to get something accurate.

The tests I did are really important, because the results I got will never change, since the real robot (and probably also earth gravity...) will never change, and will always behave almost the same way. On the other hand, the simulator and Webots will assuredly change. Hence, every time modifications are brought to my model or to the software, it will be necessary to perform new tests on the simulator, to watch if it still behaves like the real one. And by the fact that I have this automatic world evaluation, one will just have to run the world with the *postures_protocol* controller, and verify that the behavior and the accuracy have been improved.

I made many tests, with different worlds, and the results are contained in the table 6. The first column (*Move*) is the name of the kind of the move I did. It corresponds to a servo, and an orientation. For example, *leg3 Back* is the simultaneous modification of the servos RLEG_JOINT3 et LLEG_JOINT3, in order to make to robot fall backward. The L_ARM stands for arm lowered.

[24]See table 6 for the different tests I performed.
[25]See section 5.

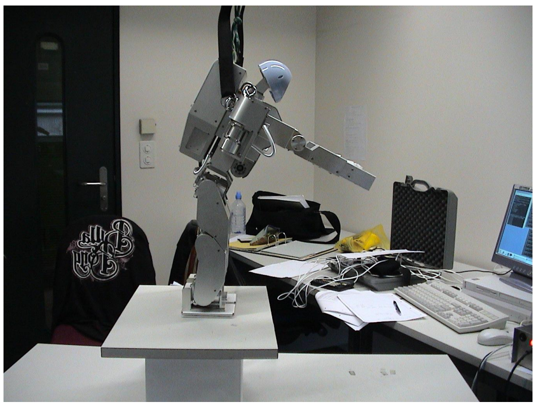


Figure 31: Performing postures protocol on HOAP-2

There is 16 different tests, and the different worlds I tested are represented in figure 32.

The column Value contains first the name of the world used, and then the angle value: The different worlds I tested are represented in figure 32. The last column is the difference between the real value, that I measured on the real robot, and the simulated one. For every test, I made a movie of the real robot (that is available on my project web page[1]).

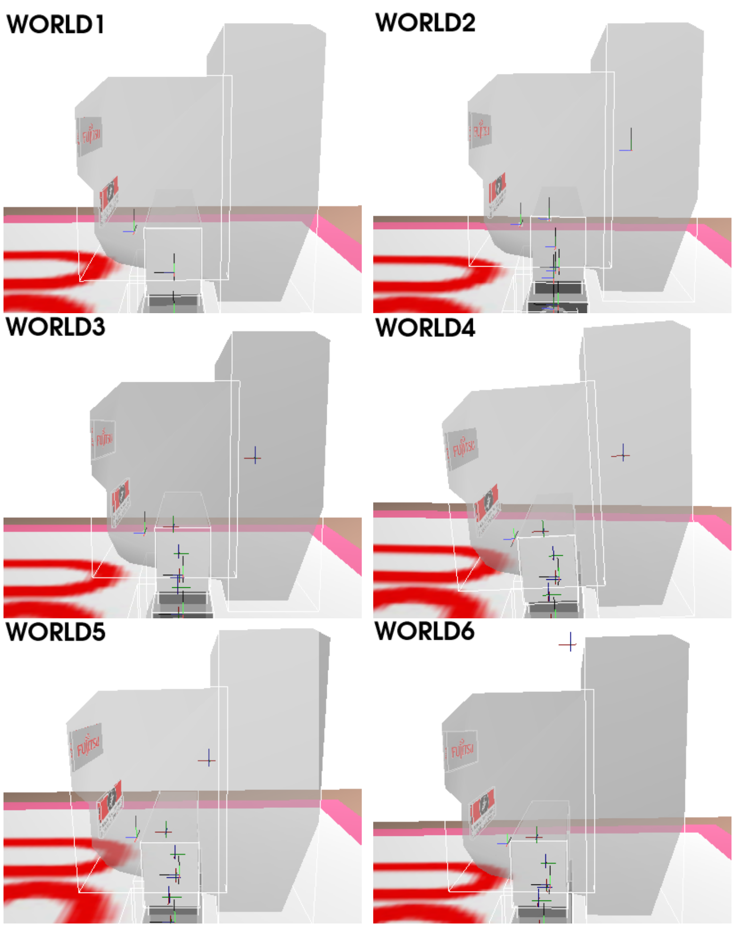
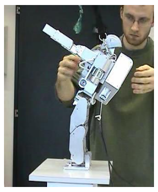
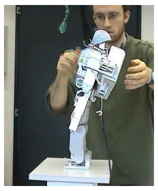
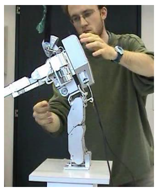
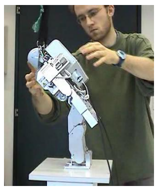
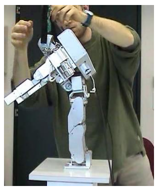
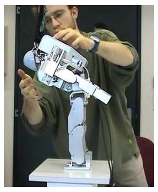
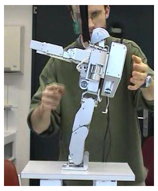
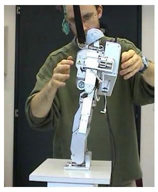
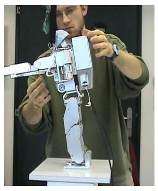


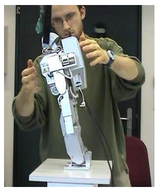
Figure 32: These pictures represent the robot torso from the different worlds that have been tested. WORLD1 is hoap2 in the 4.0.27 Webots version (where the CoM == center of rotation). WORLD2 is the same world than WORLD1, but with extra solids put at the centers of mass positions of the links, in order to simulate the correct weights at the correct positions. WORLD3 is hoap2 in Webots 5.0.0, after I had improved it, so with centers of mass correctly positioned, and it corresponds exactly to the real robot specifications. WORLD4 is a world where I tried to represent backlash by setting both (R & L)LEG_JOINT3 and both (R & L)LEG_JOINT5 from the WORLD3 world at an initial value of 2[°] (see section 3.9.1). WORLD5 is hoap2 in Webots 5.0.0, with a first try of displaced center of mass of the torso. And finally WORLD6 is the same world than WORLD5, but with an extreme and impossible torso CoM position. I made these worlds to try to find the best position for the center of mass position of the torso, which is the heaviest part of the robot, so changing it imply significant changes in the results. To improve pictures legibility, as I only changed the torso center of mass position, robot arms and head have been removed. The moving cross between the pictures is the torso center of mass position.

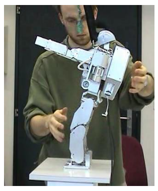








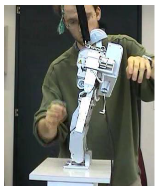
Move	Value [rd]		Difference[rd]
leg3 Back	real	0.4676	
	WORLD1	0.9218	-0.4542
	WORLD2	0.4098	0.0578
	WORLD3	0.4052	0.0624
	WORLD4	0.5227	-0.0551
	WORLD5	0.5477	-0.0801
	WORLD6	0.3932	0.0744
leg3 Back L Arm	real	0.2672	
	WORLD1	0.9457	-0.6785
	WORLD2	0.2116	0.0556
	WORLD3	0.2206	0.0466
	WORLD4	0.3522	-0.0850
	WORLD5	0.4033	-0.1361
	WORLD6	0.2766	-0.0094
leg3 Front	real	-0.3424	
	WORLD1	-0.8154	0.4730
	WORLD2	-0.7536	0.4112
	WORLD3	-0.7857	0.4433
	WORLD4	-0.6597	0.3173
	WORLD5	-0.6438	0.3014
	WORLD6	-0.4084	0.0660
leg3 Front L Arm	real	-0.6597	
	WORLD1	-1.4309	0.7712
	WORLD2	-1.1279	0.4682
	WORLD3	-1.1457	0.4860
	WORLD4	-0.9993	0.3396
	WORLD5	-1.0336	0.3739
	WORLD6	-0.6286	-0.0311
Body Front	real	0.5189	
	WORLD1	1.5696	-1.0507
	WORLD2	1.1665	-0.6476
	WORLD3	1.2206	-0.7017
	WORLD4	0.9567	-0.4378
	WORLD5	1.0971	-0.5782
	WORLD6	0.5953	-0.0764
Body Front L Arm	real	1.0021	
	WORLD1	1.5696	-0.5675
	WORLD2	1.5696	-0.5675
	WORLD3	1.5697	-0.5676
	WORLD4	1.3279	-0.3258
	WORLD5	1.5697	-0.5676
	WORLD6	0.9017	0.1004
leg5 Back	real	0.1545	
	WORLD1	0.2298	-0.0753
	WORLD2	0.1268	0.0277
	WORLD3	0.1283	0.0262
	WORLD4	0.1398	0.0147
	WORLD5	0.1752	-0.0207
	WORLD6	0.1559	-0.0014

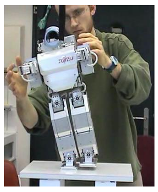


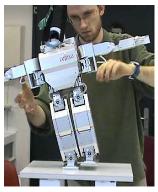




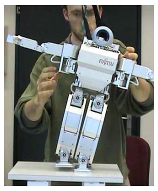








Move	Value [rd]		Difference[rd]
leg5 Back L Arm	real	0.0838	
	WORLD1	0.2140	-0.1302
	WORLD2	0.0739	0.0099
	WORLD3	0.0623	0.0215
	WORLD4	0.0254	0.0584
	WORLD5	0.1120	-0.0282
	WORLD6	0.1003	-0.0165
leg5 Front	real	-0.1050	
	WORLD1	-0.1562	0.0512
	WORLD2	-0.2308	0.1258
	WORLD3	-0.2417	0.1367
	WORLD4	-0.2301	0.1251
	WORLD5	-0.1948	0.0898
	WORLD6	-0.1659	0.0609
leg5 Front L Arm	real	-0.2004	
	WORLD1	-0.1815	-0.0189
	WORLD2	-0.3025	0.1021
	WORLD3	-0.3269	0.1265
	WORLD4	-0.3158	0.1154
	WORLD5	-0.2786	0.0782
	WORLD6	-0.2375	0.0371
leg4 Back	real	0.2422	
	WORLD1	0.3783	-0.1361
	WORLD2	0.1837	0.0585
	WORLD3	0.2028	0.0394
	WORLD4	0.2763	-0.0341
	WORLD5	0.2763	-0.0341
	WORLD6	0.2294	0.0128
leg4 Back L Arm	real	0.1418	
	WORLD1	0.3582	-0.2164
	WORLD2	0.1023	0.0395
	WORLD3	0.1013	0.0405
	WORLD4	0.1748	-0.0330
	WORLD5	0.1826	-0.0408
	WORLD6	0.1512	-0.0094
leg6 Right	real	-0.2505	
	WORLD1	-0.1149	-0.1356
	WORLD2	-0.1148	-0.1357
	WORLD3	-0.2394	-0.0111
	WORLD4	-0.2523	0.0018
	WORLD5	-0.2524	0.0019
	WORLD6	-0.2192	-0.0313
leg6 Right L Arm	real	-0.2589	
	WORLD1	-0.1144	-0.1445
	WORLD2	-0.1144	-0.1445
	WORLD3	-0.2232	-0.0357
	WORLD4	-0.2350	-0.0239
	WORLD5	-0.2363	-0.0226
	WORLD6	-0.2026	-0.0563



Move	Value [rd]		Difference[rd]
leg6 Left	real	0.2672	
	WORLD1	0.2477	0.0195
	WORLD2	0.2391	0.0281
	WORLD3	0.2388	0.0284
	WORLD4	0.2517	0.0155
	WORLD5	0.2518	0.0154
	WORLD6	0.2187	0.0485
leg6 Left L Arm	real	0.2756	
	WORLD1	0.2595	0.0161
	WORLD2	0.2201	0.0555
	WORLD3	0.2229	0.0527
	WORLD4	0.2347	0.0409
	WORLD5	0.2359	0.0397
	WORLD6	0.2023	0.0733

Table 6: Postures protocol results

8.1 Worlds evaluation

The table 7 contains an evaluation of the results obtained with the posture protocol for the six different worlds.

World	Total [rd]	Average	
		[rd]	[°]
WORLD1	4.9388	0.3087	17.69
WORLD2	2.9352	0.1835	10.51
WORLD3	2.8263	0.1766	10.12
WORLD4	2.0234	0.1265	7.25
WORLD5	2.1073	0.1317	7.55
WORLD6	1.4104	0.0880	5.05

Table 7: Evaluation of the different worlds of the postures protocol

First of all, one can see that between Webots 4.0.27 (WORLD1) and Webots 5.0.0 (WORLD3), there is a big difference. WORLD1 has effectively the worst results, what is normal since the centers of mass were not at the correct positions. I made WORLD2 while I was upgrading Webots 4.0.27 to take care of the centers of mass, and wanting to know if I was heading in the good direction. Its results are pretty much the same than the one from WORLD3, which is the first Webots 5.0.0 world, and the only one that is perfectly respecting the HOAP-2 specifications. It is supposed to be the exact reproduction of the real HOAP-2, but there is an average error of 10.12[°], which is huge. In fact all the tests that make the robot fall in front have bad results (difference between 0.1 and 0.7[rd] with the real robot). On the contrary, the backward moves, and on the left and right sides, are much better, with only small differences with the real robot. The left and right side moves have pretty good results, what seems to indicate the world is well balanced in these directions. So to improve the results, I designed WORLD4, by trying to add some artificial backlash. The results are then a little bit better than WORLD3, but there is still an average error of 7.25[°]. The problem with this artificial backlash I added, is that it is just a very simple approximation of the effect of the backlash. And for the tests that are on the leg5 or leg3 joints, the results become then almost equal to the ones from WORLD3, as we are changing the angles that were set to new initial positions. This world was then just a little test, to see if that could improve the results, but it has not been really conclusive. And as soon as the robot is moving, like in the dynamical moves from section 9, these default angles would change, and it is impossible to do it this way. So I tried many other worlds, always by moving the center of mass of the torso, as it is the robot heaviest part, and by this fact the one that is the most effective on the robot balance. I only moved it in the front and rear direction, since the results in the left and right direction were already pretty good. I was in fact trying to find the best approximation of the real robot, what is really not an easy task. In WORLD5 and 6, I displaced the center of mass in front. WORLD6 is an impossible one (at least physically), as the body center of mass is not contained into the body, what is off course impossible. But it is this world that gave me the best results, with an average error of only 5.05[°]. I have not find a better world that could give me better results than this WORLD6.

9 Reproduction of dynamical moves

To perform statical tests[26] in order to check that a model is accurate is already a good thing. But it is even better to verify the accuracy with dynamical moves. So I used some of the csv trajectory files provided with the robot, and I got also some new csv files by asking to Mr. Nagashima, my contact at Fujitsu. Then it was really easy to use these csv files on Webots, using the "csv2webots" controller I developed[27].

It is pretty hard to realize what these moves look like, just by looking at a few pictures, and by reading my descriptions. You should look at my home page to see these movies[1]. I tried all of the moves, with the many different worlds I had, to see which of them was the closer one to the real robot. I did not use the WORLD4, as it would be absolutely useless for dynamical tests.

9.1 The m01 walk

The HOAP-2 robot was supplied with a walk move (m01.csv). It is a simple and slow forward motion. For the real robot, as you can see in the left picture of figure 33, I just put it on a table, and let it walk, even if I kept hanging it, not wanting to break the robot, in case of a fall. The real robot walked about 60.5[cm] in 16.1[s], what represents a speed of 0.0375[m/s] (0.1353[km/h]). But this value can vary a lot, since it depends on the kind of floor on which the robot is walking. The table I used here was totally flat and perfectly horizontal, but the robot was a little bit slipping. I tried a softer but a lot less slippery ground, but the robot could not walk on this surface, so I have no other results than this one for the real robot.

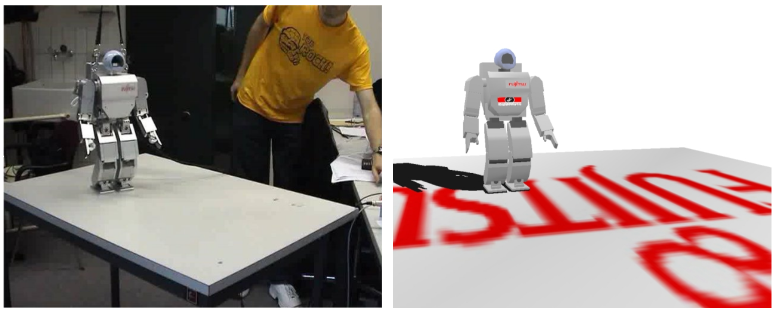


Figure 33: HOAP-2 and its Webots model performing a walk

9.1.1 Results

In simulation, with the robot in the original configuration (WORLD3 in section 8), the robot walks in a strange manner, not really smooth, just like if it was not able to put its feet correctly on the ground to get a good balance. But it walks, and does not fall. I measured a distance of 40.5[cm], what represents a speed of 0.0252[m/s] (0.0906[km/h]). The difference with the real robot is then

[26]See section 8.
[27]See section 7.

of 0.0123[m/s], which means that my simulated robot covers 1.2[cm] less every second than the real robot. This result off course depends a lot on the world I use. By changing the position of the body center of mass, I was able to improve the results. By using WORLD5 from section 8, I got 0.0288[m/s], and by using the WORLD6, I got 0.0276[m/s]. I then continued with new worlds, by continuing displacing the torso CoM. And I found a good position (see figure 34), that made the robots walk at 0.03[m/s], in an almost smooth manner. So the smallest difference I got is 7.54[mm] less a second. I made a test of posture

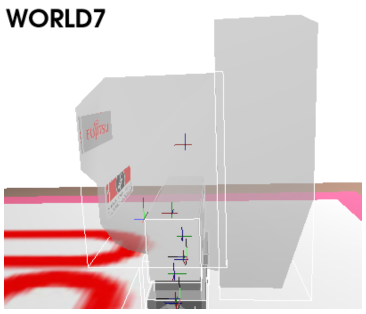


Figure 34: Torso center of mass position from the WORLD7

protocol on this new world, but the result was not better than the WORLD3, unfortunately.

9.2 The CPG walk

The second walk is a better and smoother one than he m01. The robot moves faster, and it has no problem to put its feet correctly on the ground. It scarcely slips, even if I performed this move on the same table than the m01. The robot covered the whole table in about 12.5[s]. So its speed is about 0.0844[m/s] (0.304[km/h]). For information, to date, Fujitsu engineers succeeded in making it walk at a speed of 1.44[km/h]. As a comparison, Honda ASIMO, that is 70[cm] taller, can reach 3.2[km/h].

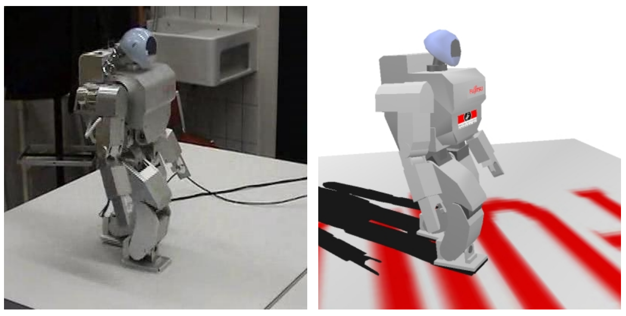


Figure 35: HOAP-2 and its Webots model performing the CPG walk

9.2.1 Results

Unfortunately, the world WORLD3 is not even able to perform the first step, and falls immediately on its back. So we get the same results than in the postures protocol, the weight seems to be situated a little bit to much in the back of the robot, and it is not balanced enough in the front. With WORLD5, it can only perform 2 step, before falling on its back. So WORLD5 is better than WORLD3, as we already realized during 8. With the physically impossible world WORLD6, the result is not better than for WORLD5. WORLD7 is also the best world I found for this move. The robot was then able to go all over the tatami without falling, and without slipping. The simulated robot traversed a distance of $1.355m$ in $17s$. So its speed is of $0.0797m/s$ ($0.287km/h$). So it gives us a difference of $4.7mm/s$, what is I think really impressive, as I was not expecting such a close value.

9.3 Standing up after lying on its back

At the beginning of this move, the robot is lying on its back. It then manages to stand up, by helping itself with both arms to raise it torso, and then by bringing its feet closer to its body, by lying its feet on the ground, and then by stretching its legs, still helping with its arms to keep balance.

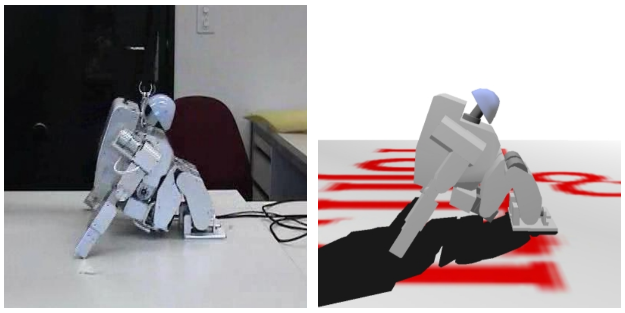


Figure 36: HOAP-2 and its Webots model standing up after lying on its back

9.3.1 Results

The WORLD3 model is not able to get up, since it can not get the balance to avoid falling backward when its arms are not touching the ground anymore. Both WORLD5 and WORLD6 are able to get up, as well as WORLD7, that runs perfectly, and its move is smooth. I also found some other worlds that allowed hoap2 to succeed in standing up (but that are not working for the CPG walk), and this time again, the posture protocol tests were not better than the one I already got.

9.4 Standing up after lying on its elbows

In this move, the robot starts lying on both elbows and feet. It then brings its arms closer to its feet, and then by a little folding of the knees, the robot stands up.

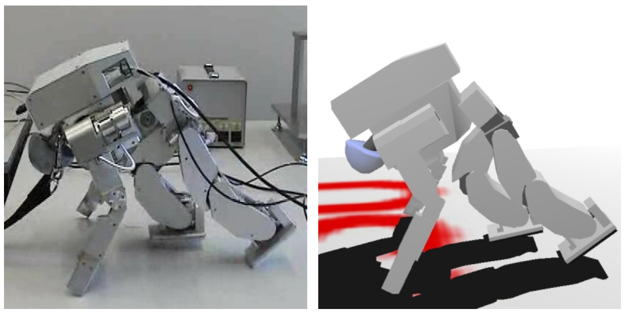


Figure 37: HOAP-2 and its Webots model standing up after lying on its elbows

9.4.1 Results

Once again, WORLD3 does not succeed in realizing this dynamical move. It seems that there is not much missing, but as soon as the robot stands on feet and hands, and then tries to get up, it falls, but in front this time. WORLD6 has a center of mass so badly situated, that the robot falls on its head, and is even worst than WORLD3. WORLD5, as well as WORLD7 are able to realize this move.

9.5 HOAP-2 performing sumo moves

This move is probably the most interesting and awesome one, as a dynamical test. The robot performs the kind of moves a real sumotori performs before a fight. At the beginning, it stands straight on its feet, and then starts to move, first by shifting smoothly on its right without even moving its feet. It then stands only on the right foot, the other leg standing in the air. Then it puts the left leg back on the ground, and then stands only on its right leg, performing even some kind of a big leg split(see figure 38). It then continues with some more moves, that are always so impressive. I have to say that even for the real robot, it was pretty hard to stay up at the end of the show. Sometimes, the robot fell. I had to try it many times, to successfully perform that move, as one can see on my web page.

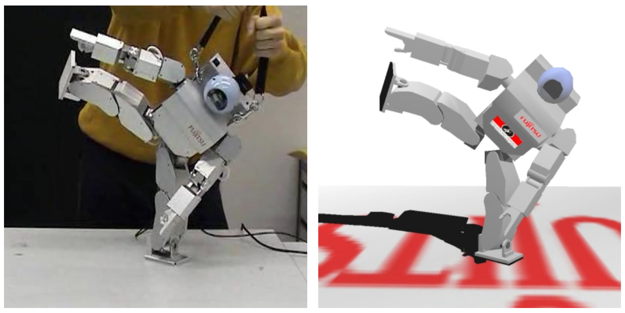


Figure 38: HOAP-2 and its Webots model performing sumo moves

9.5.1 Results

As one could expect it, WORLD3 is not able to play the sumotori. But it is not that bad, since it is able to execute the first moves, during about 30[s]. But at this moment, the robot is executing a move where it bends pretty fast both hips, knees and ankles, in order to stay as if it was seated in the air. And WORLD3 is not able to stay in this position. In fact, none of the worlds I already used that far were able to perform from the beginning to the end these fighting moves, and all fell at approximately the same time. So I designed a new world, with a torso center of mass that was even more shifted in front than the WORLD7 (see figure 39). The WORLD8 does not have allowed either to improve the postures protocol from section 8.

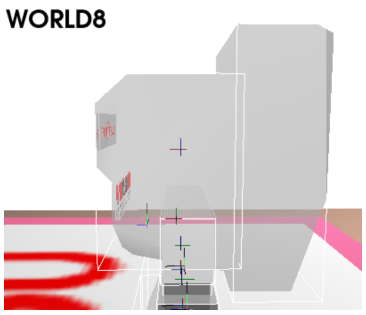
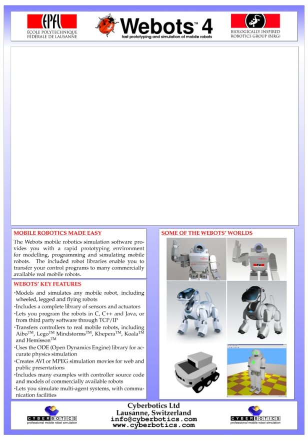


Figure 39: Torso center of mass position from the WORLD8

10 Humanoids 2004 conference


ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
Webots™ 4
fast prototyping and simulation of mobile robots
BIOLOGICALLY INSPIRED ROBOTICS GROUP (BIRG)

MOBILE ROBOTICS MADE EASY

The Webots mobile robotics simulation software provides you with a rapid prototyping environment for modelling, programming and simulating mobile robots. The included robot libraries enable you to transfer your control programs to many commercially available real mobile robots.

WEBOTS' KEY FEATURES

·Models and simulates any mobile robot, including wheeled, legged and flying robots
·Includes a complete library of sensors and actuators
·Lets you program the robots in C, C++ and Java, or from third party software through TCP/IP
·Transfers controllers to real mobile robots, including Aibo™, Lego™ Mindstorms™, Khepera™, Koala™ and Hemisson™
·Uses the ODE (Open Dynamics Engine) library for accurate physics simulation
·Creates AVI or MPEG simulation movies for web and public presentations
·Includes many examples with controller source code and models of commercially available robots
·Lets you simulate multi-agent systems, with communication facilities

SOME OF THE WEBOTS' WORLDS

Cyberbotics Ltd
Lausanne, Switzerland
info@cyberbotics.com
www.cyberbotics.com

Figure 40: Webots poster at Humanoids 2004 conference

The Humanoids 2004 conference[28] took place at Santa Monica, California, USA. Olivier sent me there to represent Cyberbotics Ltd.. My work there consisted in presenting a poster about the Webots software, its models of humanoid robots, and of course my work with the HOAP-2. I designed a poster containing some text about the software and its great possibilities. I also added a few pictures, representing some real robots, opposed to their models on Webots. I also left a 4/3 sized blank space, allowing me to broadcast Webots videos in it. This has been a really good experience for me. I spoke with many people from leading fields in humanoid robotics, and assisted to many important and interesting conferences, under the Californian sun...
You can find a bigger version of this poster in the appendix 12.2.

[28]http://www.humanoids.ws/humanoids04/

11 Further developments and conclusion

The first further development that should be apported to this project, is the implementation of a cross compiler for the HOAP2, that would allow the transfer to the real robot of the controllers designed on Webots. Webots already includes transfer systems for a number of existing real robots including Khepera, Hemisson, LEGO Mindstorms, Aibo, etc, and developing a cross-compilation system for HOAP-2 would allow the users to recompile their Webots controller on the real robot, what is an important feature that a good simulator should have. Because effectively, the final goal of a simulation is very often the realization of the same behavior on the real robot once that the user is satisfied with his simulation. The user could then develop its Webots controller for the hoap2 world, do simulation for some times, and then recompile directly exactly the same controller on HOAP-2, and to watch in the real world how the robot is behaving. Knowing that HOAP-2 runs on real-time Linux, it should not be a difficult task to develop this cross compilator.

With the controllers I developed, one is already able to run csv files in the simulation, and to get the same motion than the real robot. So it could be useful to have a cross compiler, to reproduce the simulation's controllers on the real robot.

Then one could also implement the HOAP-2 foot sensors, because they are really important, for example for the bipedal locomotion, and it could allow to perform ZMP calculation to simulate smooth walking motions.

It could also be interesting to design more precisely the HOAP-2 hands. In my current model, I have not designed the hands precisely, since I was not really interested in the hands shapes and their use. But it could be a good thing to design them, in order to perform grasping simulations that could easily be reproduced with the real robot.

Concerning my project, I must say that I found very interesting the way Olivier and I have interacted, with the big talks we had, trying to find a way to improve Webots that would satisfy both of us, me as a user needing new functionalities to be able to design my new world, and Oliver as a software developer wanting to keep his software as competitive as possible, and ascendantly compatible. It has also been very interesting to write my own code to adapt Webots, in order to better suit my needs. Unfortunately, there has been times during which I was waiting for Olivier to release a new Webots version, and there was not much things I could do during these waiting times. So my schedule has been oscillating between days during which I could not make good progress in my work (I used this time by writing my report, performing tests on the real robot, and reading documents), and had to occupy myself, and weeks during which I just could not stop working, because of all the things I was then able to do, and had time to get back. So there has been a lot of drawbacks with these Webots bugs, that made me waist a few days of work, for example when I was waiting for a Webots version on which I could set my centers of mass, version that I finally upgraded by myself, what was not scheduled in my job description.

Concerning the evaluation of the current version of the simulator, I do not think that the results are excellent. They have always been improving since the beginning of my Thesis, but I think the results are still not satisfactory. WORLD3 is the world corresponding exactly to both HOAP-2 instruction manual, and inspection report. But this world results are not very impressive, as we have seen. The problem is also that I have not found a world that was able to reproduce correctly every moves, and that got good results in the posture protocols. So maybe the problem comes from a bug that I would have missed in Webots, but I am not sure of that, since I performed lots of tests to be sure that it was correct.

There is also a big interrogation point on the accuracy and precision of the inspection report. It is conceivable that it could have some errors in it. That could be a way of explaining some bad results I had. But it will be really difficult to discover if there is effectively errors in the robot specifications.

I think also that the most obvious reason of the bad results I have, is this backlash problem. And it will be really difficult to take care of this on Webots, since it is almost impossible to simulate correctly and efficiently the backlash. As I already said, it depends on the motor construction, its wear, etc. and is not even constant for a single motor. I think it could be possible to have some randomly generated backlash, but that would then be almost impossible to make it correspond to a real motor. And I think that measure the backlash of every motor, to then set it on Webots, is not efficaciously feasible.

This project allowed to bring important modifications to Webots, because it is obvious that the concept of center of mass is mandatory for a physical simulator. Thanks to the tests I performed, I think that it is working properly. And I think that this hoap2 world could be a great motivation to buy a Webots license, for a team or a school wanting to do humanoid robotics, since a Webots license cost about 25 times less than a real HOAP-2, and is a lot less restricting than the real HOAP-2.

I really much enjoyed working on this simulator, and to play with this wonderful robot. And I will continue working on the HOAP-2 for one more year, but this time by doing this not on the side of the development of a simulation, but with the real robot, since I will perform an industrial internship at Fujitsu Laboratories, in Kawasaki, Japan, where I will probably work on Motion Learning Method using CPG/NP.

References

[1] P. Cominoli.
Diploma thesis related web page, with movies, pictures and sources
www.birg.epfl.ch
links: "BIRG INDEX: Student Project", "STUDENTS: P. Cominoli"

[2] Fujitsu Automation Co.,Ltd.
HOAP-2 Instruction Manual.
http://www.automation.fujitsu.com/en/products/products09.html
Fujitsu, Kawasaki, Japan.

[3] P. Cominoli.
Development of a physical simulation of a real humanoid robot.
Semester Project at BIRG laboratory
Swiss Federal Institute of Technology, Summer 2004.

[4] Webots.
http://www.cyberbotics.com
Commercial Mobile Robot Simulation Software.

[5] J.J. Kuffner, S. Kagami, M. Inaba, and H. Inoue.
Graphical simulation and high-level control of humanoid robots.
In Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'00),
Takamatsu, Japan, November 2000.

[6] R. A. Brooks.
Artificial Life and Real Robots.
Toward a Practice of Autonomous Systems: European Conference on Artificial Life,
Paris, France, MIT Press, December 1991

[7] S. Mojon.
Realization of a Physic Simulation for a Biped Robot.
Semester Project at BIRG laboratory
Swiss Federal Institute of Technology, Summer 2003.

[8] T. Tjahjowidodo, F. Al-Bender, H. Van Brussel K.U.Leuven.
Identification of Backlash in Mechanical Systems.
Department Mechanical Engineering Celestijnenlaan 300 B, B-3001, Heverlee, Belgium.

[9] C. Schell, R. Muschong, J. Rhinehart, P. Chang, A. Zeid, H. Khalil, C. J. Radcliffe, S. Seshagir.
Friction and Backlash Compensation of an Electronic Steering System.
Michigan State University College Of Engineering, Spring 2000.

[10] A. Robertsson, J. Bengtsson, P. Alriksson.
Nonlinear Control and Servo Systems Lecture 8: Backlash, Quantization.
Nonlinear Control and Servo Systems course 2005.
Lund Institute of Technology, Sweden.

[11] F. Kanehiro, K. Fujiwara, S. Kajita, K. Yokoi, K. Kaneko, H. Hirukawa, Y. Nakamura, K. Yamane.
Open architecture humanoid robotics platform.
ICRA '02. IEEE International Conference on , Volume: 1 , 11-15 May 2002
Robotics and Automation, 2002. Proceedings.

[12] K. Kaneko, F. Kanehiro, S. Kajita, H. Hirukawa, T. Kawasaki, M. Hirata, K. Akachi, T. Isozumi.
Humanoid robot HRP-2.
ICRA '04. 2004 IEEE International Conference on , Volume: 2 , April 26-May 1, 2004
Robotics and Automation, 2004. Proceedings.

[13] L. Geppert.
Qrio, the robot that could.
Spectrum, IEEE , Volume: 41 , Issue: 5 , May 2004

[14] M.J. Riezenman.
Robots stand on own two feet.
Spectrum, IEEE , Volume: 39 , Issue: 8 , Aug. 2002

[15] A. Brossard.
Control of the humanoid robot HOAP-2. (in French)
Diploma Thesis at ASL laboratory.
Swiss Federal Institute of Technology, Winter 2005.

[16] http://www.plyojump.com/

[17] ASIMO web site.
http://asimo.honda.com/

[18] K. Hirai, M. Hirose, Y. Haikawa, T. Takenaka.
The development of Honda humanoid robot.
1998 IEEE International Conference on , Volume: 2 , 16-20 May 1998
Robotics and Automation, 1998. Proceedings.

[19] Yobotics, Inc.
http://yobotics.com/
Firm specialized in robotic design, consulting, and research.

[20] Roboworks.
http://www.newtonium.com
3D modeling and simulation software.

[21] SD Fast.
http://www.sdfast.com
Physically-based simulation of mechanical systems.

[22] Fujitsu Web site.
www.fujitsu.com
Automation technology.

[23] Open Dynamics Engine.
http://ode.org/
Open source, high performance library for simulating rigid body dynamics.

12 Appendix

12.1 HOAP-2 inspection report

Table : Mass Property of the LINK belong to joint data A-1

RLEG_JOINT[1]	MASS = 3.93880e-02 KILOGRAM CENTER OF GRAVITY with respect to RJ1_LINK_CS coordinate frame: X Y Z 4.05183e-07 -3.83869e+00 2.19162e+01 MM INERTIA at CENTER OF GRAVITY with respect to RJ1_LINK_CS coordinate frame: (KILOGRAM * MM^2) INERTIA TENSOR: Ixx -Ixy -Ixz 2.77458e+01, 2.48926e-06, 1.08060e-06 -Iyx Iyy -Iyz 2.48926e-06, 9.57690e+00,-1.29806e+00 -Izx -Izy Izz 1.08060e-06,-1.29806e+00, 2.37104e+01
RLEG_JOINT[2]	MASS = 1.72696e-01 KILOGRAM CENTER OF GRAVITY with respect to RJ2_LINK_CS coordinate frame: X Y Z -3.28966e+00 5.01030e-01 3.47170e+00 MM INERTIA at CENTER OF GRAVITY with respect to RJ2_LINK_CS coordinate frame: (KILOGRAM * MM^2) INERTIA TENSOR: Ixx -Ixy -Ixz 6.31338e+01,-6.18885e-01, 1.01226e+00 -Iyx Iyy -Iyz -6.18885e-01, 5.55820e+01, 3.28855e-01 -Izx -Izy Izz 1.01226e+00, 3.28855e-01, 4.51904e+01
RLEG_JOINT[3]	MASS = 4.38575e-01 KILOGRAM CENTER OF GRAVITY with respect to RJ3_LINK_CS coordinate frame: X Y Z -7.18179e+01 7.58280e+00 -1.42801e+00 MM INERTIA at CENTER OF GRAVITY with respect to RJ3_LINK_CS coordinate frame: (KILOGRAM * MM^2) INERTIA TENSOR: Ixx -Ixy -Ixz 2.31713e+02, 1.44579e+01, 1.32330e+01 -Iyx Iyy -Iyz 1.44579e+01, 4.72676e+02,-9.52495e+00 -Izx -Izy Izz 1.32330e+01,-9.52495e+00, 4.35755e+02
RLEG_JOINT[4]	MASS = 2.85982e-01 KILOGRAM CENTER OF GRAVITY with respect to RJ4_LINK_CS coordinate frame: X Y Z -4.29165e+01 1.17240e+01 -2.82151e+00 MM INERTIA at CENTER OF GRAVITY with respect to RJ4_LINK_CS coordinate frame: (KILOGRAM * MM^2) INERTIA TENSOR: Ixx -Ixy -Ixz 1.84076e+02,-3.29998e+00, 4.55079e+00 -Iyx Iyy -Iyz -3.29998e+00, 2.95317e+02,-8.29468e+00 -Izx -Izy Izz 4.55079e+00,-8.29468e+00, 2.25932e+02
RLEG_JOINT[5]	MASS = 1.71128e-01 KILOGRAM CENTER OF GRAVITY with respect to RJ5_LINK_CS coordinate frame: X Y Z 3.24289e+00 4.65209e+00 -7.32177e-01 MM INERTIA at CENTER OF GRAVITY with respect to RJ5_LINK_CS coordinate frame: (KILOGRAM * MM^2) INERTIA TENSOR: Ixx -Ixy -Ixz 5.94587e+01,-4.08202e-01,-7.20727e-02 -Iyx Iyy -Iyz -4.08202e-01, 4.59388e+01,-5.54424e-01 -Izx -Izy Izz -7.20727e-02,-5.54424e-01, 5.11179e+01
RLEG_JOINT[6]	MASS = 1.36753e-01 KILOGRAM CENTER OF GRAVITY with respect to RJ6_LINK_CS coordinate frame: X Y Z -2.48732e+01 7.85960e-02 3.82498e+00 MM INERTIA at CENTER OF GRAVITY with respect to RJ6_LINK_CS coordinate frame: (KILOGRAM * MM^2) INERTIA TENSOR: Ixx -Ixy -Ixz 1.55016e+02,-7.14057e-02, 1.70683e+01 -Iyx Iyy -Iyz -7.14057e-02, 1.39643e+02, 3.80602e-01 -Izx -Izy Izz 1.70683e+01, 3.80602e-01, 5.97968e+01

RARM_JOINT[1]	MASS = 1.99031e-01 KILOGRAM
	CENTER OF GRAVITY with respect to RHJ1_LINK_CS coordinate frame:
	X Y Z 5.69887e-02 3.59362e+01 1.10127e+01 MM
	INERTIA at CENTER OF GRAVITY with respect to RHJ1_LINK_CS coordinate frame: (KILOGRAM * MM^2)
	INERTIA TENSOR:
	Ixx -Ixy -Ixz 1.42087e+02,-5.75840e-02,-7.51591e-02
	-Iyx Iyy -Iyz -5.75840e-02, 6.52462e+01, 3.39010e+01
	-Izx -Izy Izz -7.51591e-02, 3.39010e+01, 1.17938e+02
RARM_JOINT[2]	MASS = 2.01439e-01 KILOGRAM
	CENTER OF GRAVITY with respect to RHJ2_LINK_CS coordinate frame:
	X Y Z -1.00180e-02 -9.91178e+00 -2.28432e+00 MM
	INERTIA at CENTER OF GRAVITY with respect to RHJ2_LINK_CS coordinate frame: (KILOGRAM * MM^2)
	INERTIA TENSOR:
	Ixx -Ixy -Ixz 1.53251e+02, 2.71682e-01, 2.07022e-01
	-Iyx Iyy -Iyz 2.71682e-01, 1.35628e+02,-1.53672e+01
	-Izx -Izy Izz 2.07022e-01,-1.53672e+01, 7.07822e+01
RARM_JOINT[3]	MASS = 2.20645e-01 KILOGRAM
	CENTER OF GRAVITY with respect to RHJ3_LINK_CS coordinate frame:
	X Y Z 7.84674e-02 -2.96507e+00 -3.23713e+01 MM
	INERTIA at CENTER OF GRAVITY with respect to RHJ3_LINK_CS coordinate frame: (KILOGRAM * MM^2)
	INERTIA TENSOR:
	Ixx -Ixy -Ixz 1.22386e+02, 9.41629e-02,-1.85832e-02
	-Iyx Iyy -Iyz 9.41629e-02, 1.09558e+02,-8.27732e-01
	-Izx -Izy Izz -1.85832e-02,-8.27732e-01, 5.65823e+01
RARM_JOINT[4]	MASS = 1.67552e-01 KILOGRAM
	CENTER OF GRAVITY with respect to RHJ4_LINK_CS coordinate frame:
	X Y Z -5.29589e+000 4.62381e+01 5.06259e+00 MM
	INERTIA at CENTER OF GRAVITY with respect to RHJ4_LINK_CS coordinate frame: (KILOGRAM * MM^2)
	INERTIA TENSOR:
	Ixx -Ixy -Ixz 2.80658e+02,-2.40293e+00,-1.29926e+00
	-Iyx Iyy -Iyz -2.40293e+00, 3.40780e+01, 1.85149e+01
	-Izx -Izy Izz -1.29926e+00, 1.85149e+01, 2.70963e+02

LLEG_JOINT[1]	MASS = 3.93880e-02 KILOGRAM CENTER OF GRAVITY with respect to LJ1_LINK_CS coordinate frame: X Y Z 4.05183e-07 -3.83869e+00 2.19162e+01 MM INERTIA at CENTER OF GRAVITY with respect to LJ1_LINK_CS coordinate frame: (KILOGRAM * MM^2) INERTIA TENSOR: Ixx -Ixy -Ixz 2.77458e+01, 2.48926e-06, 1.08060e-06 -Iyx Iyy -Iyz 2.48926e-06, 9.57690e+00,-1.29805e+00 -Izx -Izy Izz 1.08060e-06,-1.29805e+00, 2.37104e+01
LLEG_JOINT[2]	MASS = 1.72696e-01 KILOGRAM CENTER OF GRAVITY with respect to LJ2_LINK_CS coordinate frame: X Y Z -3.28965e+00 -6.32384e-01 3.47170e+00 MM INERTIA at CENTER OF GRAVITY with respect to LJ2_LINK_CS coordinate frame: (KILOGRAM * MM^2) INERTIA TENSOR: Ixx -Ixy -Ixz 6.30681e+01, 2.93348e-01, 1.01227e+00 -Iyx Iyy -Iyz 2.93348e-01, 5.55820e+01,-3.50678e-01 -Izx -Izy Izz 1.01227e+00,-3.50678e-01, 4.51248e+01
LLEG_JOINT[3]	MASS = 4.38537e-01 KILOGRAM CENTER OF GRAVITY with respect to LJ3_LINK_CS coordinate frame: X Y Z -7.19284e+01 7.57808e+00 1.34158e+00 MM INERTIA at CENTER OF GRAVITY with respect to LJ3_LINK_CS coordinate frame: (KILOGRAM * MM^2) INERTIA TENSOR: Ixx -Ixy -Ixz 2.31801e+02, 1.49663e+01,-1.45158e+01 -Iyx Iyy -Iyz 1.49663e+01, 4.73478e+02, 1.04350e+01 -Izx -Izy Izz -1.45158e+01, 1.04350e+01, 4.36352e+02
LLEG_JOINT[4]	MASS = 2.85827e-01 KILOGRAM CENTER OF GRAVITY with respect to LJ4_LINK_CS coordinate frame: X Y Z -4.29459e+01 1.17224e+01 2.97258e+00 MM INERTIA at CENTER OF GRAVITY with respect to LJ4_LINK_CS coordinate frame: (KILOGRAM * MM^2) INERTIA TENSOR: Ixx -Ixy -Ixz 1.82049e+02,-2.84645e+00,-4.79234e+00 -Iyx Iyy -Iyz -2.84645e+00, 2.91216e+02, 8.37920e+00 -Izx -Izy Izz -4.79234e+00, 8.37920e+00, 2.22810e+02
LLEG_JOINT[5]	MASS = 1.71128e-01 KILOGRAM CENTER OF GRAVITY with respect to LJ5_CS coordinate frame: X Y Z 3.24289e+00 4.65210e+00 7.81358e-01 MM INERTIA at CENTER OF GRAVITY with respect to LJ5_CS coordinate frame: (KILOGRAM * MM^2) INERTIA TENSOR: Ixx -Ixy -Ixz 5.94459e+01,-4.08202e-01, 4.99529e-01 -Iyx Iyy -Iyz -4.08202e-01, 4.59261e+01, 6.50510e-01 -Izx -Izy Izz 4.99529e-01, 6.50510e-01, 5.11179e+01
LLEG_JOINT[6]	MASS = 1.36753e-01 KILOGRAM CENTER OF GRAVITY with respect to RJ6_LINK_CS coordinate frame: X Y Z -2.48732e+01 7.85960e-02 3.82498e+00 MM INERTIA at CENTER OF GRAVITY with respect to RJ6_LINK_CS coordinate frame: (KILOGRAM * MM^2) INERTIA TENSOR: Ixx -Ixy -Ixz 1.55016e+02,-7.14057e-02, 1.70683e+01 -Iyx Iyy -Iyz -7.14057e-02, 1.39643e+02, 3.80602e-01 -Izx -Izy Izz 1.70683e+01, 3.80602e-01, 5.97968e+01

LARM_JOINT[1]	MASS = 1.99031e-01 KILOGRAM CENTER OF GRAVITY with respect to LHJ1_LINK_CS coordinate frame: X Y Z -5.69861e-02 3.59362e+01 -1.10127e+01 MM INERTIA at CENTER OF GRAVITY with respect to LHJ1_LINK_CS coordinate frame: (KILOGRAM * MM^2) INERTIA TENSOR: Ixx -Ixy -Ixz 1.42087e+02, 5.75829e-02,-7.51502e-02 -Iyx Iyy -Iyz 5.75829e-02, 6.52462e+01,-3.39010e+01 -Izx -Izy Izz -7.51502e-02,-3.39010e+01, 1.17938e+02
LARM_JOINT[2]	MASS = 2.01439e-01 KILOGRAM CENTER OF GRAVITY with respect to LHJ2_CS coordinate frame: X Y Z -1.02610e-01 -9.91178e+00 -2.28305e+00 MM INERTIA at CENTER OF GRAVITY with respect to LHJ2_CS coordinate frame: (KILOGRAM * MM^2) INERTIA TENSOR: Ixx -Ixy -Ixz 1.53253e+02,-1.60475e-01, 2.49619e-01 -Iyx Iyy -Iyz -1.60475e-01, 1.35627e+02,-1.53617e+01 -Izx -Izy Izz 2.49619e-01,-1.53617e+01, 7.07801e+01
LARM_JOINT[3]	MASS = 2.18287e-01 KILOGRAM CENTER OF GRAVITY with respect to LHJ3_LINK_CS coordinate frame: X Y Z -2.62502e-02 2.94852e+00 -3.26378e+01 MM INERTIA at CENTER OF GRAVITY with respect to LHJ3_LINK_CS coordinate frame: (KILOGRAM * MM^2) INERTIA TENSOR: Ixx -Ixy -Ixz 1.20095e+02, 2.35113e-01,-2.30691e-01 -Iyx Iyy -Iyz 2.35113e-01, 1.07647e+02, 8.96281e-01 -Izx -Izy Izz -2.30691e-01, 8.96281e-01, 5.56209e+01
LARM_JOINT[4]	MASS = 1.67592e-01 KILOGRAM CENTER OF GRAVITY with respect to LHJ4_LINK_CS coordinate frame: X Y Z -5.29948e+00 4.62509e+01 -5.06689e+00 MM INERTIA at CENTER OF GRAVITY with respect to LHJ4_LINK_CS coordinate frame: (KILOGRAM * MM^2) INERTIA TENSOR: Ixx -Ixy -Ixz 2.81004e+02,-2.39305e+00, 1.29841e+00 -Iyx Iyy -Iyz -2.39305e+00, 3.40730e+01,-1.84659e+01 -Izx -Izy Izz 1.29841e+00,-1.84659e+01, 2.71298e+02

BODY_JOINT[1] *1) WAIST coordinate difinition *2)Loading item CPU unit : On Wireless LAN Unit : On Battery : On	MASS = 2.37622e+00 KILOGRAM CENTER OF GRAVITY with respect to BODY1_LINK_CS coordinate frame: X Y Z -6.29381e+01 2.80136e-01 6.71528e+01 MM INERTIA at CENTER OF GRAVITY with respect to BODY1_LINK_CS coordinate frame: (KILOGRAM * MM^2) INERTIA TENSOR: Ixx -Ixy -Ixz 8.05592e+03,-4.72994e+01,-7.02973e+02 -Iyx Iyy -Iyz -4.72994e+01, 9.24194e+03, 1.01937e+00 -Izx -Izy Izz -7.02973e+02, 1.01937e+00, 7.03308e+03
BODY_JOINT[1] *1) WAIST coordinate difinition *2)Loading item CPU unit : On Wireless LAN Unit : Off Battery : Off	MASS = 1.59374e+00 KILOGRAM CENTER OF GRAVITY with respect to BODY1_LINK_CS coordinate frame: X Y Z -4.86368e+01 1.19112e-02 8.60519e+01 MM INERTIA at CENTER OF GRAVITY with respect to BODY1_LINK_CS coordinate frame: (KILOGRAM * MM^2) INERTIA TENSOR: Ixx -Ixy -Ixz 4.77151e+03,-7.27252e+01, 5.49937e+02 -Iyx Iyy -Iyz -7.27252e+01, 5.33640e+03, 5.71186e+01 -Izx -Izy Izz 5.49937e+02, 5.71186e+01, 5.51773e+03
BODY_JOINT[1] *1) WAIST coordinate difinition *2)Loading item CPU unit : Off Wireless LAN Unit : Off Battery : Off	MASS = 1.48198e+00 KILOGRAM CENTER OF GRAVITY with respect to BODY1_LINK_CS coordinate frame: X Y Z -4.40262e+01 5.45960e-01 8.43444e+01 MM INERTIA at CENTER OF GRAVITY with respect to BODY1_LINK_CS coordinate frame: (KILOGRAM * MM^2) INERTIA TENSOR: Ixx -Ixy -Ixz 4.54319e+03,-2.41738e+01, 3.87040e+02 -Iyx Iyy -Iyz -2.41738e+01, 4.76232e+03, 3.64751e+01 -Izx -Izy Izz 3.87040e+02, 3.64751e+01, 4.94122e+03
BODY_JOINT[2]	MASS = 4.97680e-01 KILOGRAM CENTER OF GRAVITY with respect to BODY2_LINK_CS coordinate frame: X Y Z -2.57370e+01 5.56017e+00 -3.62639e-01 MM INERTIA at CENTER OF GRAVITY with respect to BODY2_LINK_CS coordinate frame: (KILOGRAM * MM^2) INERTIA TENSOR: Ixx -Ixy -Ixz 7.26374e+02, 2.81664e+01, 1.02147e+01 -Iyx Iyy -Iyz 2.81664e+01, 7.89590e+02,-5.05608e+00 -Izx -Izy Izz 1.02147e+01,-5.05608e+00, 2.74925e+02
HEAD_JOINT[1]	MASS = 2.08892e-02 KILOGRAM CENTER OF GRAVITY with respect to HEAD1_LINK_CS coordinate frame: X Y Z 2.06780e+00 5.98154e-01 5.68043e+01 MM INERTIA at CENTER OF GRAVITY with respect to HEAD1_LINK_CS coordinate frame: (KILOGRAM * MM^2) INERTIA TENSOR: Ixx -Ixy -Ixz 9.51077e+00,-7.41228e-02,-1.05256e+00 -Iyx Iyy -Iyz -7.41228e-02, 6.40490e+00,-3.02325e-01 -Izx -Izy Izz -1.05256e+00,-3.02325e-01, 4.97111e+00
HEAD_JOINT[2]	MASS = 6.69886e-02 KILOGRAM CENTER OF GRAVITY with respect to HEAD2_LINK_CS coordinate frame: X Y Z 5.29518e-01 -3.94438e+00 -2.35576e-01 MM INERTIA at CENTER OF GRAVITY with respect to HEAD2_LINK_CS coordinate frame: (KILOGRAM * MM^2) INERTIA TENSOR: Ixx -Ixy -Ixz 3.20353e+01,-5.04880e+00,-6.55641e-02 -Iyx Iyy -Iyz -5.04880e+00, 3.51472e+01, 4.54066e-02 -Izx -Izy Izz -6.55641e-02, 4.54066e-02, 2.59511e+01

12.2 Humanoids 2004 conference

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Webots™ 4

fast prototyping and simulation of mobile robots

BIOLOGICALLY INSPIRED ROBOTICS GROUP (BIRG)

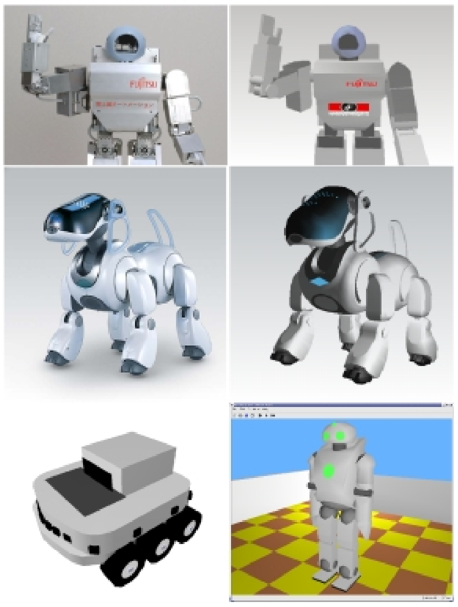
MOBILE ROBOTICS MADE EASY

The Webots mobile robotics simulation software provides you with a rapid prototyping environment for modelling, programming and simulating mobile robots. The included robot libraries enable you to transfer your control programs to many commercially available real mobile robots.

WEBOTS' KEY FEATURES

·Models and simulates any mobile robot, including wheeled, legged and flying robots
·Includes a complete library of sensors and actuators
·Lets you program the robots in C, C++ and Java, or from third party software through TCP/IP
·Transfers controllers to real mobile robots, including Aibo™, Lego™ Mindstorms™, Khepera™, Koala™ and Hemisson™
·Uses the ODE (Open Dynamics Engine) library for accurate physics simulation
·Creates AVI or MPEG simulation movies for web and public presentations
·Includes many examples with controller source code and models of commercially available robots
·Lets you simulate multi-agent systems, with communication facilities

SOME OF THE WEBOTS' WORLDS



CYBERBOTICS
professional mobile robot simulation

Cyberbotics Ltd
Lausanne, Switzerland
`info@cyberbotics.com`
`www.cyberbotics.com`

CYBERBOTICS
professional mobile robot simulation