Webots User Guide

release 4.0.9

copyright © 2002 Cyberbotics Ltd. All rights reserved. www.cyberbotics.com

July 28, 2003

copyright © 2002 Cyberbotics Ltd. All rights reserved. All rights reserved

Permission to use, copy and distribute this documentation for any purpose and without fee is hereby granted in perpetuity, provided that no modifications are performed on this documentation.

The copyright holder makes no warranty or condition, either expressed or implied, including but not limited to any implied warranties of merchantability and fitness for a particular purpose, regarding this manual and the associated software. This manual is provided on an *as-is* basis. Neither the copyright holder nor any applicable licensor will be liable for any incidental or consequential damages.

This software was initially developed at the Laboratoire de Micro-Informatique (LAMI) of the Swiss Federal Institute of Technology, Lausanne, Switzerland (EPFL). The EPFL makes no warranties of any kind on this software. In no event shall the EPFL be liable for incidental or consequential damages of any kind in connection with use and exploitation of this software.

Trademark information

AiboTM is a registered trademark of SONY Corp. GeForceTM is a registered trademark of nVidia, Corp. JavaTM is a registered trademark of Sun MicroSystems, Inc. KheperaTM and KoalaTM are registered trademarks of K-Team S.A. LinuxTM is a registered trademark of Linus Torwalds. Mac OS XTM is a registered trademark of Apple Inc. MindstormsTM and LEGOTM are registered trademarks of the LEGO group. PentiumTM is a registered trademark of Intel Corp. Red HatTM is a registered trademark of Red Hat Software, Inc. Visual C++TM, WindowsTM, Windows 95TM, Windows 98TM, Windows METM, Windows NTTM, Windows 2000TM and Windows XPTM are registered trademarks of Microsoft Corp. UNIXTM is a registered trademark licensed exclusively by X/Open Company, Ltd.

Foreword

Webots is a three-dimensional mobile robot simulator. It was originally developed as a research tool for investigating various control algorithms in mobile robotics.

This user guide will get you started using Webots. However, the reader is expected to have a minimal knowledge in mobile robotics, in C programming and in VRML 2.0 (Virtual Reality Modeling Language).

The great innovation of the fourth version of Webots is that it includes a physics engine allowing to define complex mobile robots with various locomotion schemes. Predefined objects like shapes, sensors and servos allow users to create and run their own simulated robot. Thus, Webots is no longer limited to differentially wheeled robots.

The GUI of Webots version 3 has been replaced by wxWindows, an Open Source Free Software GUI toolkit.

If you have already developed programs using Webots 3, please read chapter 2 to update your programs to run with the new version.

We hope that you will enjoy working with Webots 4.

Thanks

Cyberbotics is grateful to all the people who contributed to the development of Webots, Webots sample applications, the Webots User Guide, the Webots Reference Manual, and the Webots web site, including Jordi Porta, Emanuele Ornella, Yuri Lopez de Meneses, Auke-Jan Ijspeert, Gerald Foliot, Allen Johnson, Michael Kertesz, Aude Billiard, and many others.

Moreover, many thanks are due to Prof. J.-D. Nicoud (LAMI-EPFL) and Dr. F. Mondada for their valuable support.

Finally, thanks to Skye Legon, who proof-read this guide.

Contents

1	Inst	alling W	/ebots	11
	1.1	Hardw	are requirements	11
	1.2	Installa	ation procedure	11
		1.2.1	RedHat Linux i386	11
		1.2.2	Windows 95, 98, ME, NT, 2000 and XP	12
		1.2.3	Mac OS X, version 10.2	12
	1.3	Registr	ration Procedure	13
		1.3.1	Webots license	13
		1.3.2	Registering	13
2	Upg	rading f	from Webots 3	17
	2.1	Contro	ller	17
		2.1.1	Controller includes	17
		2.1.2	Controller library	17
		2.1.3	Basic data type	17
		2.1.4	Khepera	18
		2.1.5	GTK+ GUI	18
	2.2	World		18
3	Gett	ing Stai	rted with Webots	21
	3.1	Launch	ning Webots	21
		3.1.1	On Linux	21
		3.1.2	On Mac OS X	21
		3.1.3	On Windows	21

CONTENTS

	3.2	Main Window: menus and buttons		
		3.2.1	File menu and shortcuts	22
		3.2.2	Edit menu	24
		3.2.3	Simulation menu and the simulation buttons	24
		3.2.4	Help menu	25
		3.2.5	Navigation in the scene	25
		3.2.6	Moving a solid object	26
		3.2.7	Selecting a solid object	26
	3.3	Scene '	Tree Window	27
		3.3.1	Buttons of the Scene Tree Window	27
		3.3.2	VRML nodes	29
		3.3.3	Webots specific nodes	30
		3.3.4	Principle of the collision detection	30
		3.3.5	Writing a Webots file in a text editor	31
	3.4	Citing	Webots	31
4	Tuto	orial: M	odeling and simulating your robot	33
4	Tuto 4.1	orial: M My firs	odeling and simulating your robot st world: kiki.wbt	33 33
4	Tuto 4.1	orial: M My firs 4.1.1	Odeling and simulating your robot St world: kiki.wbt	33 33 33
4	Tuto 4.1	orial: M My firs 4.1.1 4.1.2	odeling and simulating your robot st world: kiki.wbt	 33 33 33 37
4	Tuto 4.1	orial: M My firs 4.1.1 4.1.2 4.1.3	odeling and simulating your robot st world: kiki.wbt	 33 33 33 37 47
4	Tuto 4.1 4.2	orial: M My firs 4.1.1 4.1.2 4.1.3 My se	odeling and simulating your robot st world: kiki.wbt	 33 33 33 37 47 48
4	Tuto 4.1 4.2 4.3	orial: M My firs 4.1.1 4.1.2 4.1.3 My se My thi	odeling and simulating your robot st world: kiki.wbt	 33 33 37 47 48 49
4	Tuto 4.1 4.2 4.3	orial: M My firs 4.1.1 4.1.2 4.1.3 My se My thi 4.3.1	odeling and simulating your robot st world: kiki.wbt	 33 33 37 47 48 49 49
4	Tuto 4.1 4.2 4.3	 My first 4.1.1 4.1.2 4.1.3 My see My thi 4.3.1 4.3.2 	odeling and simulating your robot st world: kiki.wbt	 33 33 33 37 47 48 49 49 49 49
4	Tuto 4.1 4.2 4.3	orial: M My firs 4.1.1 4.1.2 4.1.3 My se My thi 4.3.1 4.3.2 4.3.3	odeling and simulating your robot st world: kiki.wbt	 33 33 33 37 47 48 49 49 49 56
4	Tuto 4.1 4.2 4.3	My first 4.1.1 4.1.2 4.1.3 My see My thi 4.3.1 4.3.2 4.3.3 ot and S	odeling and simulating your robot st world: kiki.wbt	 33 33 33 37 47 48 49 49 49 56 59
4	Tuto 4.1 4.2 4.3 Rob 5.1	orial: M My firs 4.1.1 4.1.2 4.1.3 My se My thi 4.3.1 4.3.2 4.3.3 ot and S Overvi	odeling and simulating your robot st world: kiki.wbt	 33 33 33 37 47 48 49 49 49 56 59 59
4	Tuto 4.1 4.2 4.3 Rob 5.1 5.2	orial: M My firs 4.1.1 4.1.2 4.1.3 My se My thi 4.3.1 4.3.2 4.3.3 ot and S Overvi Setting	odeling and simulating your robot st world: kiki.wbt Environment Robot A simple controller cond world: a kiki robot with a camera rd world: pioneer2.wbt Environment Controller Supervisor Controllers ew up a New Controller	 33 33 33 37 47 48 49 49 49 56 59 59 59
5	Tuto 4.1 4.2 4.3 Rob 5.1 5.2 5.3	orial: M My firs 4.1.1 4.1.2 4.1.3 My se My thi 4.3.1 4.3.2 4.3.3 ot and S Overvi Setting Webots	and simulating your robot st world: kiki.wbt Environment Robot A simple controller cond world: a kiki robot with a camera rd world: pioneer2.wbt Environment Robot with 16 sonars Controller Supervisor Controllers ew yup a New Controller	 33 33 33 37 47 48 49 49 49 56 59 59 60

CONTENTS

		5.3.2	From the point of view of Webots	60
		5.3.3	Synchronous versus Asynchronous controllers	60
	5.4	Readir	g Sensor Information	61
	5.5	Contro	Olling Actuators	61
	5.6	Going	further with the Supervisor Controller	62
6	Tuto	orial: U	sing the Khepera TM robot	63
	6.1	Hardw	are configuration	63
	6.2	Runnii	ng the simulation	64
	6.3	Under	standing the model	66
		6.3.1	The 3D scene	66
		6.3.2	The Khepera model	67
	6.4	Progra	mming the Khepera robot	68
		6.4.1	The controller program	68
		6.4.2	Looking at the source code	68
		6.4.3	Compiling the controller	71
	6.5	Transf	erring to the real robot	71
		6.5.1	Remote control	71
		6.5.2	Cross-compilation and upload	72
	6.6	Worki	ng extension turrets	73
		6.6.1	The K213 linear vision turret	73
		6.6.2	The Gripper turret	73
	6.7	Suppor	rt for other K-Team robots	74
		6.7.1	Koala TM	74
		6.7.2	Alice TM	76
7	Tuto	vial. II	sing the LECO Mindstorms TM robots	77
/	7 1	Buildi	ng up the Pover robot	77
	7.1	Webot	s model of the Pover robot	00
	7.2	Transf	aring to the real Power robot	100
	1.5	7 2 1		100
		720		100
		722	Cross compilation and unload	100
		1.3.3 7.2 A		101
		1.3.4		101

8	ALife Contest				
	8.1	Previo	us Editions	103	
	8.2	Rules		103	
		8.2.1	Subject	103	
		8.2.2	Robot Capabilities	104	
		8.2.3	Programming Language	105	
		8.2.4	Scoring Rule	105	
		8.2.5	Participation	106	
		8.2.6	Schedule	106	
		8.2.7	Prize	106	
	8.3	Web Si	ite	106	
	8.4	How to	Enter the Contest	107	
		8.4.1	Obtaining the software	107	
		8.4.2	Running the software	108	
		8.4.3	Creating your own robot controller	108	
		8.4.4	Submitting your controller to the ALife contest	109	
		8.4.5	Analysing the performance and improving your competing controller	110	
	8.5	Develo	opers' Tips and Tricks	112	
		8.5.1	Practical issues	112	
		8.5.2	Java Security Manager	112	
		8.5.3	Levels of Intelligence	113	
9	Prac	ctical W	ork: Robot Soccer	115	
	9.1	Setup		115	
	9.2	Rules		116	
	9.3	Progra	mming	116	
	9.4	ions	116		
		9.4.1	Modifiying the soccer field	117	
		9.4.2	Modifying the robots	117	
		9.4.3	Modifying the match supervisor	117	

Chapter 1

Installing Webots

1.1 Hardware requirements

Webots is available for RedHat Linux i386, Mac OS X, Windows 95, Windows 98, Windows ME, Windows NT, Windows 200 and Windows XP. Other versions of Webots for other UNIX systems (Debian Linux i386, Solaris, Linux PPC, Irix) may be available upon request.

OpenGL hardware acceleration is supported on Windows, Mac OS X and in some Linux configurations. It may also be available on other UNIX systems.

1.2 Installation procedure

To install Webots, you must follow the instructions corresponding to your computer / operating system listed below:

1.2.1 RedHat Linux i386

Webots will run on RedHat Linux distributions, starting from RedHat 7.2. Webots may run on other Linux distributions. For example, it can be easily installed on Debian Linux, using the alien command to translate the rpm package into a deb package before installation. If you do use Red Hat Linux, please refer to your Linux distribution documentation to get the Webots rpm package installed.

- 1. Log on as root
- 2. Insert the Webots CD-ROM, mount it (this might be automatic) and install the following packages

```
mount /mnt/cdrom
rpm -Uvh /mnt/cdrom/linux/lib/mpeg_encode-1.5b-4.i386.rpm
# mpeg_encode is useful only if you want to export MPEG movies
rpm -Uvh /mnt/cdrom/linux/webots/webots-4.0.9-1.i386.rpm
rpm -Uvh /mnt/cdrom/linux/webots/webots-kros-1.1.0-1.i386.rpm
# webots-kros is useful only if you want to cross-compile
# controllers for the Khepera robot
```

You may need to use the --nodeps or the --force if the rpm fails to install the packages.

1.2.2 Windows 95, 98, ME, NT, 2000 and XP

- 1. Uninstall any previous release of Webots or Webots-kros, if any, from the Start menu, Control Panel, Add / Remove Programs. or from the Start menu, Cyberbotics, Uninstall Webots or Uninstall Webots-kros.
- 2. Insert the Webots CD-ROM and open it.
- 3. Go to the windows\webots directory on the CD-ROM.
- 4. Double click on the webots-4.0.9_setup.exe file.
- 5. Follow the installation instructions.
- 6. Optionally, double click on the webots-kros-1.1.0_setup.exe file to install the cross-compiler for the Khepera robots.

In order to be able to compile controllers, you will need to install a C/C++ development environment. We recommend to use Dev-C++ which is provided on the Webots CD-ROM (in the windows/utils directory) as well as from the Bloodshed.net¹ web site. Dev-C++ is an integrated development environment (IDE) for C/C++ with syntax highlighting running on Windows. It includes the MinGW distribution with the GNU GCC compiler and utilities. This software is distributed under the terms of the GNU public license and hence is free of charge.

You may also choose to use Microsoft Visual C++TM if you own a license of this software.

1.2.3 Mac OS X, version 10.2

- 1. Insert the Webots CD-ROM and open it.
- 2. Go to the mac:webots directory on the CD-ROM.
- 3. Double click on the webots-4.0.9.tar.gz file.

http://www.bloodshed.net

1.3. REGISTRATION PROCEDURE

4. This will uncompress the Webots folder. You may move this folder to your applications directory or wherever you would like to install Webots.

In order to be able to compile controllers, you will need to install the Apple Mac OS X Developer tools, included in the Mac OS X installation CD-ROMs. File editing and compilation using Webots Makefiles can be achieved through these Apple tools. You will probably use the Project Builder application to edit the source codes of the Webots controllers and the Terminal application for invoking make from the directory in which your controller gets compiled.

The CodeWarriorTM development environment is not supported for the development of controllers (although it may also work).

1.3 Registration Procedure

1.3.1 Webots license

Starting with Webots 4, a new license system has been introduced to facilitate the use of Webots.

When installing Webots, you will get a license file, called webots.key, containing your name, address, serial number and computer ID. This encrypted file will enable you to use Webots according to the license you purchased. This file is strictly personal: you are not allowed to provide copies of it to any third party in any way, including publishing that file on any Internet server (web, ftp, or any other server). Any copy of your license file is under your responsibility. If a copy of your license file is used by an unauthorized third party to run Webots, then Cyberbotics may engage legal procedures against you. Webots licenses are (1) non-transferable and (2) non-exclusive. This means that (1) you cannot sell or give your Webots license to any third party, and (2) Cyberbotics and its official Webots resellers may sell or give Webots licenses to third parties.

If you need further information about license issues, please send an e-mail to:

<license@cyberbotics.com>

Please read your license agreement carefully before registering. This license is provided within the software package. By using the software and documentation, you agree to abide by all the provisions of this license.

1.3.2 Registering

After installing Webots, you will need to register your copy of Webots to get a license file called webots.key allowing you to use all the features of Webots corresponding to the license you purchased.

Simple registration

In order to proceed, launch Webots on the computer on which you would like to install the license file. Go the **Register** menu item of the **Help** menu of Webots and follow the instructions. If this computer is connected to the Internet, everything will run smoothly, fill in the requested form and you will shortly receive the webots.key license file via e-mail. Otherwise, you will have to fill in a form² on the website of Cyberbotics (see figure 1.1). You will then receive an e-mail containing the webots.key file corresponding to your license.

🔞 Cyberbotics: Webots 4 Registration - Mozilla 🦄			- • ×
[▼] <u>F</u> ile <u>E</u> dit <u>V</u> iew <u>G</u> o <u>B</u> ookmarks <u>T</u> ools <u>W</u> indov	w <u>H</u> elp		
🖸 🚱 🙆 🔘 🕓 http://www.cy	berbotics.com/registration/webots4.html	• Q	0 , 0
Webots 4 Registrat	ion		
Registration Form			
Please fill in the following form:			
Title: First name: Last name: E-mail: Organization: Address: Zip: City: State: Country: System: Serial Number: ⁽¹⁾ Computer ID: ⁽²⁾ Reseller:	Mr. Mr.		
(1). The serial number of your Webots convis written on the	he CD-ROM If you purchased Webots STD the serial numb	er was sent to	you by e-mail
 (2): The computer ID is the MAC address of your main network adapter. It appears in the help -> register menu of Webots. 			

Figure 1.1: Webots registration page

Please take care to properly fill in each field of this form. The *Serial Number* is the serial number of your Webots package which is printed the CD-ROM under the heading *S*/*N*:. The computer ID is given by Webots in the **Register** menu item of the **Help** menu.

²http://www.cyberbotics.com/registration/webots4.html

1.3. REGISTRATION PROCEDURE

After completing this form, click on the **Submit** button. You will receive shortly thereafter an e-mail containing your personal license file webots.key which is needed to install a registered copy of Webots as described below.

Registering several computers

If you need to register several computers, it may be convenient to register all the computers in the same webots.key license file. Hence this unique license file could be copied across all the computers needing a Webots license.

In order to proceed, just enter several computer IDs on the web form, corresponding to all the computers you want to register. The computer IDs have to be seperated by a simple space character.

Copying the license file

Once you received it by e-mail, just copy the webots.key license file into the resources directory of your Webots installation.

Under Linux, copy your personal webots.key file into the /usr/local/webots/server/resources directory where Webots was just installed.

Under Mac OS X, copy your personal webots.key file into the Webots:resources directory where Webots was just installed.

Under Windows, copy your personal webots.key file into the C:\Program Files\Webots\resources directory where Webots was just installed.

CHAPTER 1. INSTALLING WEBOTS

Chapter 2

Upgrading from Webots 3

If you have already worked with Webots 3, your existing programs may need to be modified for use with Webots 4.

2.1 Controller

2.1.1 Controller includes

The include path of the device files is the same as in Webots 3.2, i.e.:

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

2.1.2 Controller library

The name of the controller library to which you will link your executable files changed to "Controller.dll" on Windows and "libController.so" on Linux. It is not any more "GtkController.dll" or "libGtkController.so". Please update your project files or make files appropriately.

2.1.3 Basic data type

Since GTK+ and glib are not used any more, the data type coming from the glib are not recognized any more. Webots 4 uses the more standard C data types. Moreover, a couple of other Webots specific data type have slightly changed to become more consistant with the rest of the API. Please replace the following data types in your programs:

Webots 3	Webots 4
gchar	char
guchar	unsigned char
guint8	unsigned char
gint8	char
gint	int
guint	unsigned int
gint32	int
guint32	unsigned int
gint16	short
guint16	unsigned short
gfloat	float
gdouble	double
gpointer	void *
device_tag	DeviceTag
node_ref	NodeRef

Table 2.1: Data types changes between Webots 3 and Webots 4

2.1.4 Khepera

If you developed controllers specifically for the Khepera robot, the khepera_live function should be replaced by robot_live. Moreover, the #include <transfer/khepera.h> should be removed. Please check also that the model field of the DifferentialWheels node corresponding to the Khepera robot is "Khepera" or "Khepera II", so that it will be automatically recognized by Webots as a Khepera controller for remote control and code upload.

2.1.5 GTK+ GUI

The GTK+ graphical user interface is not any more integrated within the controller library. Hence, you are free to use any graphical user interface library, like wxWindows, GTK+, Motif, MFC, etc. To use for example GTK+, you have two options: (1) you can perform all the graphical user interface operations inside a separate thread, or (2) you can setup the GUI in the same thread and call repeatly the gtk_main_iteration_do function inside the main loop. An example of using wxWindows as the GUI of a controller in provided in the wxgui controller sample.

2.2 World

Very few changes were introduce in the Webots worlds that breaks the compatibility. However, since Webots 4 uses a different collision detection engine, boundingObject made up of

2.2. WORLD

Extrusion or IndexedFaceSet have to be changed to composite objects made up of Box, Sphere and / or Cylinder. A composite object is a Group node containing Transform nodes as children. These Transform nodes should contain the primitive shapes previously enumerated and positioned appropriately.

Chapter 3

Getting Started with Webots

To run a simulation in Webots, you need two things:

This chapter gives an overview of the basics of Webots, including the display of the world in the main window and the structure of the .wbt file appearing in the scene tree window.

Robot and Supervisor controllers will be explained in detail later on in this book.

3.1 Launching Webots

3.1.1 On Linux

From an X terminal, type webots to launch the simulator. You should see the world window appear on the screen (see figure 3.1).

3.1.2 On Mac OS X

Open the directory in which you uncompressed the Webots package and double-click on the Webots icon. You should see the world window appear on the screen (see figure 3.1).

3.1.3 On Windows

From the **Start** menu, go to the **Program Files** — **Cyberbotics** menu and click on the **Webots 4.0.9** menu item. You should see the world window appear on the screen (see figure 3.1).



Figure 3.1: Webots main window

3.2 Main Window: menus and buttons

The main window allows you to display your virtual worlds and robots described in the .wbt file. Four menus and a number of buttons are available.

3.2.1 File menu and shortcuts

The **New** menu item opens a new default world representing a chessboard of $10 \ge 10$ plates on a surface of $1 \le x \le 10$ plates on the surface of $1 \le x \le 10$ plates on the surface of $1 \le 10$ plates of $10 \le 10$ plates of $10 \le 10$



3.2. MAIN WINDOW: MENUS AND BUTTONS

The File menu will also allow you to perform the standard file operations: **Open...**, **Save** and **Save As...**, respectively, to load, save and save with a new name the current world.

The following buttons can be used as shortcuts:





The Revert item allows you to reload the most recently saved version of your .wbt file.

The following button can be used as a shortcut:



The **Export VRML** item allows you to save the .wbt file as a .wrl file, conforming to the VRML 2.0 standard. Such a file can, in turn, be opened with any VRML 2.0 viewer. This is especially useful for publishing a world created with Webots on the Web.

The **Make Movie...** item allows you to create a MPEG movie under Linux and Mac OS X or an AVI movie under Windows. As movies are created on a 30 frame per second basis, you should adapt the basic simulation step and the refresh display parameters in the general preferences to obtain a movie running at real time. Setting the basic simulation step to 32 ms and the refresh display each 1 ms should produce movies runnig close to real time.

The **Screenshot...** item allows you to take a screenshot of the current view in Webots. It opens a file dialog to save the current view as a PNG image.

The **Preferences** item pops up a window with the following panels:

• General: The Startup mode allows you to choose the state of the simulation when Webots is launched (stop, run, fast; see the Simulation menu).

The Basic simulation step parameter defines the duration of the simulation step executed by Webots. It is expressed in milliseconds. Setting this value to a high value will accelerate the simulation, but will decrease the accuracy of the simulation, especially for physics simulation and collision detection This value is also used when the **Step** button is pressed.

The Refresh display parameter is multiplicated to the basic step value to define how frequently the 3D display of the main window is refreshed in normal Run mode.

• Rendering: this tab controls the 3D rendering in the simulation window.

Checking the Display sensor rays check box displays the distance sensor rays of the robot(s) as red lines.

Checking the Display lights check box displays the lights (PointLight in the world so that they can be moved more accurately).

• Files and paths: The default .wbt world which is open when launching Webots and the user directory are defined here. The user directory should contain at least a worlds, controllers, and objects directories where Webots will be looking for files.

3.2.2 Edit menu

The **Scene Tree Window** item opens the window in which you can edit the world and the robot(s). A shortcut is available by double-clicking on a solid in the world. A solid is a physical object in the world.

3.2.3 Simulation menu and the simulation buttons

In order to run a simulation a number of buttons are available corresponding to menu items found under the **Simulation** menu:



Stop: interrupt Run or Fast modes.

Step: execute one simulation basic step. The duration of such a step is defined in the preferences of Webots and can be adjusted to suit your needs.

Run: execute simulation steps until the **Stop** mode is entered. In run mode, the 3D display of the scene is refreshed every n basic step, where n is defined in the Webots preferences.



Fast: same as Run, except that no display is performed (Webots PRO only). only.

The **Fast** mode performs a very fast simulation mode suited for heavy computation (genetic algorithms, vision, learning, etc.). However, as the world display is disabled during a **Fast** simulation, the scene in the world window remains blank until the **Fast** mode is stopped. This feature is available only with Webots PRO.

The World View / Robot View item allows you to switch between two different points of view:

- World View: this view corresponds to a fixed camera standing in the world.
- **Robot View**: this view corresponds to a mobile camera following a robot.

The default view is the world view. If you want to switch to the **Robot View**, first select the robot you want to follow (click on the pointer button then on the robot), and then choose **Robot View** in the **Simulation** menu. To return to the **World View** mode, reselect this item.

A speedometer (see figure 3.2) allows you to observe the speed of the simulation on your computer. It is displayed in the bottom right hand side of the main window and indicates how fast the

3.2. MAIN WINDOW: MENUS AND BUTTONS

simulation runs compared to real time. In other words, it represents the speed of the virtual time. If the value of the speedometer is 2, it means that your computer simulation is running twice as fast as the corresponding real robots would. This information is relevant both in **Run** mode and **Fast** mode.



Figure 3.2: Speedometer

The basic simulation time step can be chosen from the preferences window. It is expressed in virtual time milliseconds. The value of this time step defines the duration of the time step executed during the **Step** mode. This step is multiplicated by the refresh parameter to define how frequently the display is refreshed. The refresh parameter can be changed from the preferences window.

In **Run** mode, with a time step of 64 ms and a fairly simple world displayed with the default window size, the speedometer will typically indicate approximately 0.5 on a Pentium II / 266 Mhz without hardware acceleration and 12 on a Pentium III / 500 Mhz with an nVidia Geforce II MX graphics card.

3.2.4 Help menu

In the **Help** menu, the **About...** item opens the About . . . window, displaying the license information.

The Introduction item is a short introduction to Webots (HTML file). You can access the User Guide and the Reference Manual with the User Guide and Reference Manual items (PDF files). The Web site of Cyberbotics item lets you visit our Web site.

3.2.5 Navigation in the scene

The view of the scene is generated by a virtual camera set in a given position and orientation. You can change this position and orientation to navigate in the scene using the mouse buttons. The x, y, z axes mentioned below correspond to the coordinate system of the camera; z is the axis corresponding to the direction of the camera.

• *Rotate viewpoint*: To rotate the camera around the *x* and *y* axis, you have to set the mouse pointer in the 3D scene, press the left mouse button and drag the mouse:

if you clicked on a solid object, the rotation will be centered around the origin of the local coordinate system of this object.

if you clicked outside of any solid object, the rotation will be centered around the origin of the world coordinate system.

- *Translate viewpoint*: To translate the camera in the *x* and *y* directions, you have to set the mouse pointer in the 3D scene, press the right mouse button and drag the mouse.
- Zoom / Tilt viewpoint: set the mouse pointer in the 3D scene, then:

if you press both left and right mouse buttons (or the middle button) and drag the mouse vertically, the camera will zoom in or out.

if you press both left and right mouse buttons (or the middle button) and drag the mouse horizontally, the camera will rotate around its z axis (tilt movement).

if you use the wheel of the mouse, the camera will zoom in or out.

3.2.6 Moving a solid object

In order to move an object, hold the shift key down while using the mouse.

- Translation: Pressing the left mouse button while the shift key is pressed allows you to drag solid objects on the ground (*xz* plan).
- Rotation: Pressing the right mouse button while the shift key is pressed rotates solid objects: a first click is necessary to select a solid object, then a second press-and-drag rotates the selected object around its *y* axis.
- Lift: Pressing both left and right mouse buttons, the middle mouse button, or rolling the mouse wheel while the shift key is pressed allows you to lift up or down the selected solid object.

3.2.7 Selecting a solid object

Simply clicking on a solid object allows you to select this object. Selecting a robot enables the choice of **Robot View** in the **simulation** menu. Double-clicking on a solid object opens the scene tree window where the world and robots can be edited. The selected solid object appears selected in the scene tree window as well.

3.3 Scene Tree Window

As seen in the previous section, to access to the Scene Tree Window you can either choose **Scene Tree Window** in the **Edit** menu, or click on the pointer button and double-click on a solid object.

The scene tree contains all information necessary to describe the graphic representation and simulation of the 3D world. A world in Webots includes one or more robots and their environment.

The scene tree of Webots is structured like a VRML file. It is composed of a list of nodes, each containing fields. Fields can contain values (text string, numerical values) or nodes.

Some nodes in Webots are VRML nodes, partially or totally implemented, while others are specific to Webots. For instance the Solid node inherits from the Transform node of VRML and can be selected and moved with the buttons in the World Window.

This section describes the buttons of the Scene Tree Window, the VRML nodes, the Webots specific nodes and how to write a .wbt file in a text editor.



Figure 3.3: Scene Tree Window

3.3.1 Buttons of the Scene Tree Window

The scene tree with the list of nodes appears on the left side of the window. Clicking on the + in front of a node or double-clicking on the node displays the fields inside the node, and similarly

expands the fields. The field values can be defined on the top right side of the window. Five editing buttons are available on the bottom right side of the window:





Paste after

These three buttons let you cut, copy and paste nodes and fields. However, you can't perform these operations on the three first nodes of the tree (WorldInfo, Viewpoint and Background). These nodes are mandatory and don't need to be duplicated. Similarly, you can't copy the Supervisor node because only one supervisor is allowed. Please note that when you cut or copy a robot node, like a DifferentialWheels or Supervisor node, the controller field of this node is reset to "void".

Delete: This button allows you to delete a node. It appears only if a node is selected. If a field is selected, the **Default Value** button appears instead.

Default Value: You can click on this button to reset the default value(s) of a field. A field with values must be selected in order to perform this button. If a node is selected, the Delete button replaces it.



Transform: This button allows you to transform a node into another one.

Insert after: With this button, you can insert a node after the one currently selected. This new node contains fields with default values, which you can of course modify to suit your needs. This button also allows you to add a node to a children field. In all cases, the software only permits you to insert a coherent node.



Insert Node: Use this to insert a node into a field whose value is a node. You can insert only a coherent node.

Export Node: Use this button to export a node into a file. Usually, nodes are saved in your objects directory. Such saved nodes can then be reused in other worlds.

Import Node: Use this button to import a previously saved node into the scene tree. Usually, saved nodes are located in the Webots objects directory or in your own objects directory. The Webots objects directory already contains a few nodes that can be easily imported.

3.3.2 VRML nodes

A number of VRML 2.0 nodes are partially or completely supported in Webots.

The exact features of VRML 2.0 are the subject of a standard managed by the International Standards Organization (ISO/IEC 14772-1:1997).

You can find the complete specifications on the official VRML Web site: \texttt{http://www.vrml.org}. The VRML nodes supported in Webots are the following:

- Appearance
- Background
- Box
- Color
- Cone
- Coordinate
- Cylinder
- DirectionalLight
- ElevationGrid
- Fog
- Group
- ImageTexture
- IndexedFaceSet
- IndexedLineSet
- Material
- PointLight
- Shape
- Sphere
- Switch
- TextureCoordinate

- TextureTransform
- Transform
- Viewpoint
- WorldInfo

The Reference Manual gives a comprehensive list of nodes with associated fields.

3.3.3 Webots specific nodes

In order to implement powerful simulations including mobile robots with two-wheel differential steerings, a number of nodes specific to Webots have been added to the VRML set of nodes.

VRML uses a hierarchical structure for nodes. For example, the Transform node inherits from the Group node, such that, like the Group node, the Transform node has a children field, but it also adds three additional fields: translation, rotation and scale.

In the same way, Webots introduces new nodes which inherit from the VRML Transform node, principally the Solid node. Other Webots nodes (DifferentialWheels, DistanceSensor, Camera, etc.) inherit from this Solid node.

The Reference Manual gives a complete description of all Webots nodes and their respective fields.

3.3.4 Principle of the collision detection

The collision detection engine is able to detect a collision between two Solid nodes. It calculates the intersection between the bounding objects of the solids. A bounding object (described in the boundingObject field of the Solid node) is a geometric shape or a group of geometric shapes which bounds the solid. If the boundingObject field is NULL, then no collision detection is performed for this Solid node. list of children of the Solid node are used to compute the bounding object.

The collision detection is mainly of use between a robot DifferentialWheels node) and an obstacle (Solid node), and between two robots. Two Solid nodes can never inter-penetrate each other; their movement is stopped just before the collision.

Example: a solid with a bounding box different from its list of children.

Let us consider the Khepera robot model. It is not exactly a Solid node, but the principle for the boundingObject is the same. Open the khepera.wbt file and look at the boundingObject field of the DifferentialWheels node. The bounding object is a cylinder which has been transformed. See figure 3.4.

3.4. CITING WEBOTS



Figure 3.4: The bounding box of the Khepera robot

3.3.5 Writing a Webots file in a text editor

It is possible to write a Webots world file (.wbt) using a text editor. A world file contains a header, nodes containing fields and values. Note that only a few VRML nodes are implemented, and that there are nodes specific to Webots. Moreover, comments can only be written in the DEF, and not like in a VRML file.

The Webots header is:

#VRML_SIM V4.0 utf8

After this header, you can directly write your nodes. The three nodes WorldInfo, Viewpoint and Background are mandatory.

Note: we recommend that you write your file using the tree editor. However it may be easier to make some particular modifications using a text editor (like using the search and replace feature of a text editor).

3.4 Citing Webots

When writing a scientific paper, or describing your project involving Webots on a web site, it is always appreciated to make a correct reference to Webots, mentionning Cyberbotics web site explicitely. In order to help you in such a task, we provide here some citation examples, including a BibTex entry that you can freely reuse in your own documents:

This project uses Webots¹, a commercial mobile robot simulation software developed by Cyberbotics Ltd.

This project uses Webots (http://www.cyberbotics.com), a commercial mobile robot simulation software developed by Cyberbotics Ltd.

The BibTex reference entry may look odd, as it is very different from a standard paper citation and we want the specified fields to appear in the normal plain citation mode of LaTeX.

```
@Misc{Webots,
  author={Webots},
  title={http://www.cyberbotics.com},
  note={Commercial Mobile Robot Simulation Software},
  editor={Cyberbotics Ltd.},
  url={http://www.cyberbotics.com}
}
```

Once compiled with LaTeX, it should display as follow:

References

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

Chapter 4

Tutorial: Modeling and simulating your robot

The aim of this chapter is to give you several examples of robots, worlds and controllers. The first world is very simple, nevertheless it introduces the construction of any basic robot, and explains how to program a controller. The second example will show you how to model a camera on this simple robot. The third example will show you how to build a virtual Pioneer 2TM robot from ActivMedia Robotics. The fourth part will explain how to work with robots from K-Team.

4.1 My first world: kiki.wbt

As a first introduction, we are going to simulate a very simple robot made up of a box, two wheels and two infra-red sensors (see figure 4.1), controlled by a program inspired by a Braitenberg algorithm, in a simple environment surrounded by a wall.

4.1.1 Environment

We just want to have a simple world with a surrounding wall. We will represent this wall using an Extrusion node. The coordinates of the wall are shown in figure 4.2.

First, go to the **File** menu, **New** item to open a new world. Then open the **Scene Tree Window** (from the **Edit** menu). We are going to change the lighting of the scene:

1. Select the PointLight node, and click on the + just in front of it. You can now see the different fields of the PointLight node. Select ambientIntensity and enter 0.6 as a value, then select intensity and enter 0.8, then select location and enter 0.5 0.5 0.5 as values. Press return.



Figure 4.1: The kiki robot

- 2. Select the PointLight node, copy and paste it. In this new PointLight node, type -0.5 0.5 0.5 in the location field.
- 3. Repeat this paste twice again with -0.5 0.5 -0.5 in the location field of the third PointLight node, and 0.5 0.5 -0.5 in the location field of the fourth and last PointLight node.
- 4. The scene is now better lit.

Secondly, let us create the wall:

- 1. Select the last Transform node and click on the insert after button.
- 2. Choose a Solid node.
- 3. Type "wall" in the name field.
- 4. Select the children field and **Insert after** a Shape node.
- 5. **Insert** a new Appearance node in the appearance field. **Insert** a new Material node in the material field of the Appearance node. Select the diffuseColor field of the Material node and choose a color to define the color of the wall.
- 6. Now insert an Extrusion node in the geometry field of the Shape.
- 7. Set convex to FALSE. Then, set the wall corner coordinates in the crossSection field. You will have to re-enter the first point (0) at the last position (10) to complete the last face of the extrusion.
- 8. In the spine field, write that the wall ranges between 0 and 0.1 along the y axis.



Figure 4.2: The kiki world

- 9. As we want our robot to detect the wall, we have to fill in the boundingObject field. The bounding object can have exactly the same shape as the wall, so **insert** a DEF at the level of geometry Extrusion: WALL. Then, in the boundingObject field, **insert** USE WALL.
- 10. Close the tree editor, save your file and look at the result.

The wall in the tree editor is represented in figure 4.3, while the same wall in the world editor is visible in figure 4.4



Figure 4.3: The wall in the tree editor



Figure 4.4: The wall in the world window
4.1.2 Robot

The aim of this subsection is to model the *kiki* robot. This robot is made up of a DifferentialWheels node, in which we find several children: a Transform node for the body, two Solid nodes for the wheels, two DistanceSensor nodes for the infra-red sensors and a Shape node with a texture.

The origin and the axis of the coordinate system of the robot and its dimensions are shown in figure 4.5.



Figure 4.5: Coordinate system and dimensions of the kiki robot

To model the body of the robot:

- 1. Open the scene tree window.
- 2. Select the last Solid node.
- 3. Insert after a DifferentialWheels node, give it a name: "kiki".

4. In the children field, first introduce a Transform node that will contain a box shape. In the new children field, **insert after** a Shape node. Choose a color, as described previously. In the geometry field, **insert** a Box node. Set the dimension of the box to [0.08 0.08 0.08]. Now set the translation values to [0 0.06 0] in the Transform node (see figure 4.6)



Figure 4.6: Body of the *kiki* robot: a box

To model the left wheel of the robot:

- 1. Select the previous Transform and insert after a Solid node in order to model the left wheel. Type "left wheel" in the name field, so that this Solid node is recognized as the left wheel of the robot and will rotate according to the motor command.
- 2. The axis of rotation of the wheel is x. The wheel will be made of a Cylinder rotated of pi/2 radians around the z axis. To obtain proper movement of the wheel, you must pay attention not to confuse these two rotations. consequently, you must add a Transform node to the children of the Solid node.
- 3. After adding this Transform node, introduce a Shape with a Cylinder in its geometry field. The dimensions of the cylinder are 0.01 for the height and 0.025 for the radius. Set the rotation to [0011.57]. Pay attention to the sign of the rotation; if it is false, the wheel will turn in the wrong direction.
- 4. In the Solid node, set the translation to $[-0.045 \ 0.025 \ 0]$ to position the left wheel, and set the rotation of the wheel around the *x* axis: $[1 \ 0 \ 0 \ 0]$.

38

- 5. Give a DEF name to your Transform: WHEEL; notice that you positioned the wheel in translation at the level of the Solid node, so that you can reuse the WHEEL Transform for the right wheel.
- 6. Close the tree window, look at the world and save it. Use the navigation buttons to change the point of view.

To model the right wheel of the robot:

- 1. Select the left wheel Solid node and insert after another Solid node. Type "right wheel" in the name field. Set the translation to $[0.045 \ 0.025 \ 0]$ and the rotation to $[1 \ 0 \ 0 \ 0]$.
- 2. In the children, **insert after** USE WHEEL. Press Return, close the tree window and save the file. You can examine your robot in the world editor, move it and zoom in on it.

The robot and its two wheels are shown in figure 4.7 and figure 4.8.

The two infra-red sensors are defined as two cylinders on the front of the robot body. Their diameter is 0.016 m and their height is 0.004 m. You must position these sensors properly so that the sensor rays point in the right direction, toward the front of the robot.

- 1. In the children of the DifferentialWheels node, **insert after** a DistanceSensor node.
- 2. Type the name "ir0". It will be used by the controller program.
- 3. Now, we will attach a cylinder shape to this sensor. In the children of the DistanceSensor node, insert after a Transform node. Give a DEF name to it: INFRARED, which you will use for the second IR sensor.
- 4. In the children of the Transform node, **insert after** a Shape node. Choose an appearance and **insert** a Cylinder in the geometry field. Type 0.004 for the height and 0.08 for the radius.
- 5. Set the rotation for the Transform node to [0 0 1 1.57] to adjust the orientation of the cylinder.
- In the DistanceSensor node, set the translation to position the sensor and its ray: [0.02 0.08 -0.042]. In the File menu, Preferences, Rendering, check the Display sensor rays box. In order to have the ray directed toward the front of the robot, you must set the rotation to [0 1 0 1.57].
- 7. In the DistanceSensor node, you must introduce some values of distance measurements of the sensors to the lookupTable field, according to figure 4.9. These values are:



Figure 4.7: Wheels of the kiki robot

lookupTable	[0	1024	Ο,
		0.05	1024	Ο,
		0.15	0	0]

8. To model the second IR sensor, select the DistanceSensor node and insert after a new DistanceSensor node. Type "ir1" as a name. Set its translation to [-0.02 0.08 -0.042] and its rotation to [0 1 0 1.57]. In the children, insert after USE INFRARED. In the lookupTable field, type the same values as shown above.

The robot and its two sensors are shown in figure 4.10 and figure 4.11.

Note: a texture can only be mapped on an IndexedFaceSet shape. The texCoord and texCoordIndex entries must be filled. The image used as a texture must be a .png or a .jpg file, and its size



Figure 4.8: Body and wheels of the kiki robot

must be $(2\hat{n}) * (2\hat{n})$ pixels (for example 8x8, 16x16, 32x32, 64x64, 128x128 or 256x256 pixels). Transparent images are not allowed in Webots. Moreover, PNG images should use either the 24 or 32 bit per pixel mode (lower bpp or gray levels are not supported). Beware of the maximum size of texture images depending on the 3D graphics board you have: some old 3D graphics boards are limited to 256x256 texture images while more powerful ones will accept 2048x2048 texture images.

To paste a texture on the face of the robot:

- 1. Select the last DistanceSensor node and insert after a Shape node.
- 2. In the appearance field, **insert** an Appearance node. In the texture field of this node, **insert** an ImageTexture node with the following URL: "kiki/kiki.png".
- 3. In the geometry field, insert an IndexedFaceSet node, with a Coordinate node in the coord field. Type the coordinates of the points in the point field.

[0.015	0.05	-0.041,
0.015	0.03	-0.041,
-0.015	0.03	-0.041,
-0.015	0.05	-0.041]



Figure 4.9: Distance measurements of the kiki sensors.

and insert after the coordIndex field the values 0, 1, 2, 3, -1.

- 4. In the texCoord field, insert a TexureCoordinate node. In the point field, enter the coordinates of the texture:
 - [0 0 1 0 1 1 0 1]

and in the texCoordIndex field, type 3, 0, 1, 2.

5. The texture values are shown in figure 4.12.

To finish with the DifferentialWheels node, you must fill in a few more fields:

- 1. In the controller field, type the name "simple". It will be used by the controller program.
- 2. The boundingObject field can contain a Transform node with a Box, as a box as a bounding object for collision detection is sufficient to bound the *kiki* robot. Insert a Transform node in the boundingObject field, with the translation set to [0 0.05 -0.002] and a Box node in its children. Set the dimension of the Box to [0.1 0.1 0.084].
- 3. In the axleLength field, enter the length of the axle between the two wheels: 0.09 (according to figure 4.5).
- 4. In the wheelRadius field, enter the radius of the wheels: 0.025.
- 5. Values for other fields are shown in figure 4.13 and the finished robot in its world is shown in figure 4.14.



Figure 4.10: The DistanceSensor nodes of the kiki robot



Figure 4.11: The kiki robot and its sensors



Figure 4.12: Defining the texture of the *kiki* robot



Figure 4.13: The other fields of the DifferentialWheels node



Figure 4.14: The kiki robot in its world

The kiki.wbt is included in the Webots distribution, in the worlds directory.

4.1.3 A simple controller

This first controller is very simple and thus named simple. The controller program simply reads the sensor values and sets the two motors speeds, in such a way that *kiki* avoids the obstacles.

Below is the source code for the simple.c controller:

```
#include <device/robot.h>
#include <device/differential wheels.h>
#include <device/distance sensor.h>
#define SPEED 100
static DeviceTag ir0,ir1;
void reset(void) {
 ir0 = robot_get_device("ir0");
 ir1 = robot_get_device("ir1");
  // printf("ir0=134531395 ir1=1108542220\n",ir0,ir1);
}
int main() {
  short left speed, right speed;
 unsigned short ir0_value, ir1_value;
 robot_live(reset);
 distance_sensor_enable(ir0,64);
  distance_sensor_enable(ir1,64);
  for(;;) { /* The robot never dies! */
    ir0_value = distance_sensor_get_value(ir0);
    ir1_value = distance_sensor_get_value(ir1);
    if (ir1_value>200) {
      left_speed = -20;
      right_speed = 20;
    } else if (ir0_value>200) {
      left speed = 20;
      right_speed = -20;
    } else {
      left_speed =SPEED;
      right_speed=SPEED;
    }
    /* Set the motor speeds */
    differential_wheels_set_speed(left_speed,right_speed);
```

```
robot_step(64); /* run one step */
}
return 0;
```

4.2 My second world: a *kiki* robot with a camera

The camera to be modeled is a color 2D camera, with an image 80 pixels wide and 60 pixels high, and a field of view of 60 degrees (1.047 radians).

We can model the camera shape as a cylinder, on the top of the *kiki* robot at the front. The dimensions of the cylinder are 0.01 for the radius and 0.03 for the height. See figure 4.15.



Figure 4.15: The kiki robot with a camera

Try modeling this camera. The kiki_camera.wbt file is included in the Webots distribution, in the worlds directory, in case you need any help.

A controller program for this robot, named camera is also included in the Webots distribution, in the controllers directory.

```
48
```

4.3 My third world: pioneer2.wbt

We are now going to model and simulate a commercial robot from ActivMedia Robotics: Pioneer 2-DXTM, as shown on the ActivMedia Web site: http://www.activrobots.com. First, you must model the robots environment. Then, you can model a Pioneer 2TM robot with 16 sonars and simulate it with a controller.

Please refer to the worlds/pioneer2.wbt and controllers/pioneer2 files for the world and controller details.

4.3.1 Environment

The environment consists of:

- a chessboard: a Solid node with an ElevationGrid node.
- a wall around the chessboard: Solid node with an Extrusion node.
- a wall inside the world: a Solid node with an Extrusion node.

This environment is shown in figure 4.16.

4.3.2 Robot with 16 sonars

The robot (a DifferentialWheels node) is made up of six main parts:

- 1. the body: an Extrusion node.
- 2. a top plate: an Extrusion node.
- 3. two wheels: two Cylinder nodes.
- 4. a rear wheel: a Cylinder node.
- 5. front an rear sensor supports: two Extrusion nodes.
- 6. sixteen sonars: sixteen DistanceSensor nodes.

The Pioneer 2 DXTM robot is depicted in figure 4.17.

Open the tree editor and add a DifferentialWheels node. Insert in the children field:

1. for the body: a Shape node with a geometry Extrusion. See figure 4.18 for the coordinates of the Extrusion.



Figure 4.16: The walls of the Pioneer 2TM robot world

- 2. for the top plate: a Shape node with a geometry Extrusion. See figure 4.19 for the coordinates of the Extrusion.
- 3. for the two wheels: two Solid nodes. Each Solid node children contains a Transform node, which itself contains a Shape node with a geometry Cylinder. Each Solid node has a name: "left wheel" and "right wheel". See figure 4.20 for the wheels dimensions.
- 4. for the rear wheel: a Transform node containing a Shape node with a geometry Cylinder , as shown in figure 4.21
- 5. for the sonar supports: two Shape nodes with a geometry Extrusion. See figure 4.22 for the Extrusion coordinates.
- 6. for the 16 sonars: 16 DistanceSensor nodes. Each DistanceSensor node contains a Transform node. The Transform node has a Shape node containing a geometry Cylinder. See figure 4.23 and the text below for more explanation.

Modeling the sonars:

The principle is the same as for the *kiki* robot. The sonars are cylinders with a radius of 0.0175 and a height of 0.002. There are 16 sonars, 8 on the front of the robot and 8 on the rear of the



Figure 4.17: The Pioneer 2 DX^{TM} robot

robot (see figure 4.23). The angles between the sonars and the initial position of the DEF SONAR Transform are shown in figure 4.24. A DEF SONAR Transform contains a Cylinder node in a Shape node with a rotation around the z axis. This DEF SONAR Transform must be rotated and translated to become the sensors FL1, RR4, etc.







Figure 4.19: Top plate of the Pioneer 2^{TM} robot



Figure 4.20: Wheels of the Pioneer 2^{TM} robot



Figure 4.21: Rear wheel of the Pioneer 2TM robot



0.184 < y < 0.234

Figure 4.22: Sonar supports of the Pioneer 2TM robot



Figure 4.23: Sonars location on the Pioneer 2TM robot



Figure 4.24: Angles between the Pioneer 2TM sonar sensors

Each sonar is modeled as a DistanceSensor node, in which can be found a rotation around the *y* axis, a translation, and a USE SONAR Transform, with a name (FL1, RR4, ...) to be used by the controller.

Sonar name	translation	rotation
FL1	-0.027 0.209 -0.164	0101.745
FL2	-0.077 0.209 -0.147	0 1 0 2.094
FL3	-0.118 0.209 -0.11	0 1 0 2.443
FL4	-0.136 0.209 -0.071	0103.14
FR1	0.027 0.209 -0.164	0101.396
FR2	0.077 0.209 -0.147	0101.047
FR3	0.118 0.209 -0.116	0100.698
FR4	0.136 0.209 -0.071	0100
RL1	-0.027 0.209 0.253	010-1.745
RL2	-0.077 0.209 0.236	010-2.094
RL3	-0.118 0.209 0.205	010-2.443
RL4	-0.136 0.209 0.160	010-3.14
RR1	0.027 0.209 0.253	010-1.396
RR2	0.077 0.209 0.236	0 1 0 -1.047
RR3	0.118 0.209 0.205	010-0.698
RR4	0.136 0.209 0.160	0100

Table 4.1: Translation and rotation of the Pioneer 2TM DEF SONAR Transforms

To finish modeling the Pioneer 2^{TM} robot, fill in the remaining fields of the DifferentialWheels node as shown in figure 4.25.

4.3.3 Controller

The controller of the Pioneer 2TM robot is fairly complex. It implements a Braitenberg controller to avoid obstacles using its sensors. An activation matrix was determined by trial and error to compute the motor commands from the sensor measurements. However, since the structure of the Pioneer 2TM is not circular some tricks are used, such as making the robot go backward in order to rotate safely when avoiding obstacles. The source code of this controller is a good programming example. The name of this controller is pioneer 2.



Figure 4.25: Some fields of the Pioneer 2^{TM} DifferentialWheels node

Chapter 5

Robot and Supervisor Controllers

5.1 Overview

A robot controller is a program usually written in C, C++ or Java used to control one robot. A supervisor controller is a program usually written in C or C++ used to control a world and its robots.

5.2 Setting Up a New Controller

In order to develop a new controller, you must first create a controllers directory in your user directory to contain all your robot and supervisor controller directories. Each robot or supervisor controller directory contains all the files necessary to develop and run a controller. In order to tell Webots where your controllers are, you must set up your user directory in the Webots preferences. Webots will first search for a controllers directory in your user directory, and if it doesn't find, it will then look in its own controllers directory. Now, in your newly created controllers directory, you must create a controller subdirectory, let's call it simple. Inside simple, several files must be created:

- a number of C source files, like simple.c which will contain your code.
- a Makefile which can be copied (or inspired) from the Webots controllers directories. Note that Windows users have several alternatives to the Makefile: They can use a Dev-C++ project or a Microsoft Visual C++ project, as exemplified in the Webots controllers/braiten directory.

You can compile your program by typing make in the directory of your controller.

As an introduction, it is recommended that you copy the simple controller directory from the Webots controllers to your own controllers directory and then try to compile it.

Under Windows, if you use make and would like that your controller program opens up a DOS console to display printf messages, add the following line in your Makefile:

DOS_CONSOLE=1

5.3 Webots Execution Scheme

5.3.1 From the controller's point of view

Each robot controller program is built in the same manner. An initialization with the function robot_live is necessary before starting the robot. A callback function is provided to the robot_live function in order to identify the devices of the robot (see section 5.4). Then an endless loop (usually implemented as a for(ii) { } statement) runs the controller continuously until the simulator decides to terminate it. This endless loop must contain at least one call to the robot_step function which asks the simulator to advance the simulation time a given number of milliseconds, thus advancing the simulation. Before calling robot_step, the controller can enable sensor reading and set actuator commands. Sensor data can be read immediately after calling robot_step. Then you can perform your calculations to determine the appropriate actuator commands for the next step.

5.3.2 From the point of view of Webots

Webots receives controller requests from possibly several robots controllers. Each request is divided into two parts: an actuator command part which takes place immediately, and a sensor measuring part which is scheduled to take place after a given number of milliseconds (as defined by the parameter of the step function). Each request is queued in the scheduler and the simulator advances the simulation time as soon as it receives new requests.

5.3.3 Synchronous versus Asynchronous controllers

Each robot (DifferentialWheels or Supervisor) may be either synchronous or asynchronous. Webots waits for the requests of synchronous robots before it advances the simulation time; it doesn't wait for asynchronous ones. Hence an asynchronous robot may be late (if the controller is computationally expensive, or runs on a remote computer with a slow network connection). In this case, the actuator command occurs later than expected. If the controller is very late, the sensor measurement may also occur later than expected. However, this delay can be verified by the robot controller by reading the return value of the robot_step function (see the Reference Manual for more details). In this way the controller can adapt its behavior and compensate.

Synchronous controllers are recommended for robust control, while asynchronous controllers are recommended for running robot competitions where computer resources are limited, or for networked simulations involving several robots dispatched over a computer network with an unpredictable delay (like the Internet).

5.4 Reading Sensor Information

To obtain sensor information, the sensor must be:

- 1. *identified*: this is performed by the robot_get_device function which returns a handler to the sensor from its name. This needs to be done only once in the reset callback function, which is provided as an argument to the robot_live function. The only exception to this rule concerns the root device of a robot (DifferentialWheels or CustomRobot node) which doesn't need to be identified, because it is the default device (it always exists and there is only one of such device in each robot).
- 2. *enabled*: this is performed by the appropriate enable function specific to each sensor (see distance_sensor_enable for example). It can be done once, before the endless loop, or several times inside the endless loop if you decide to disable and enable the sensors from time to time to save computation time.
- 3. *run*: this is performed by the robot_step function inside the endless loop.
- 4. *read*: finally, you can read the sensor value using a sensor specific function call, like distance_sensor_get_value inside the endless loop.

5.5 Controlling Actuators

Actuators are easier to handle than sensors. They don't need to be enabled. To control an actuator, it must be:

- 1. *identified*: this is performed by the robot_get_device function which returns a handler to the actuator from its name. This needs to be done only once in the reset callback function, which is provided as an argument to the robot_live function. As with sensors, the only exception to this rule concerns the root device of a robot.
- 2. *set*: this is performed by the appropriate set function specific to each actuator (see differential_wheels for an example). It is usually called in the endless loop with different computed values at each step.
- 3. *run*: this is done by the robot_step function inside the endless loop.

5.6 Going further with the Supervisor Controller

The supervisor can be seen as a super robot. It is able to do everything a robot can do, and more. This feature is especially useful for sending messages to and receiving messages from robots, using the Receiver and Emitter nodes. Additionally, it can do many more interesting things. A supervisor can move or rotate any object in the scene, including the Viewpoint, change the color of objects, and switch lights on and off. It can also track the coordinate of any object which can be very useful for recording the trajectory of a robot. As with any C program, a supervisor can write this data to a file. Finally, the supervisor can also take a snapshot of the current scene and save it as a jpeg or PNG image. This can be used to create a "webcam" showing the current simulation in real-time on the Web!

Chapter 6

Tutorial: Using the KheperaTM robot

The goal of this chapter is to explain you how to use Webots with your Khepera robot. Khepera is a mini mobile robot developed by K-Team SA, Switzerland (www.k-team.com).

Webots can use the serial port of your computer to communicate with the Khepera robot.

6.1 Hardware configuration

- 1. Configure your Khepera robot in mode 1, for serial communication protocol at 9600 baud as described in figure 6.1.
- 2. Plug the serial connection cable between your Khepera robot and the Khepera interface.
- 3. Plug the Khepera Interface into a serial port of your computer (either COM1 or COM2, at your convenience).
- 4. Check the the Khepera robot power switch is OFF and plug the power supply to the Khepera Interface.

Note: Linux and Mac OS X users may want to redefine the COM1, COM2, COM3 and COM4 ports by setting WEBOTS_COM1, WEBOTS_COM2 WEBOTS_COM3 and/or WEBOTS_COM4 environment variables to point to the appropriate /dev device files.

On Linux, if these environment variables are not set, Webots will use respectively /dev/ttyS0, /dev/ttyS1, /dev/ttyS2 and /dev/ttyS3 for COM1, COM2 COM3 and COM4 (note the -1 difference). For example, if your laptop running Linux has no serial port, you may want to use a USB-RS232 converter, in which case it may be useful to type something like: export WEBOTS_COM1 /dev/ttyUSB0 to allow Webots to communicate with the Khepera through the USB port.



Figure 6.1: Khepera II mode selection

On Mac OS X, only COM1 has a default value which is set to "/dev/tty.USB Serial", corresponding to the default USB to serial converter (like the one installed by the USB232-P9 converters). Other USB to serial converters may require that you define the WEBOTS_COM1 environment variable to match their specific value. For example, the KeySpan USB to serial converter will need that you define WEBOTS_COM1 as "/dev/tty.USA28X1213P1.1". Please consult the documentation of your USB serial adapter to know the exact file name to be defined.

That's it. Your system is operational: you will now be able to simulate, remote control and transfer controllers to your Khepera robot.

6.2 Running the simulation

Launch Webots: on Windows, double click on the lady bug icon, on Linux, type webots in a terminal. Go to the **File Open** menu item and open the file named khepera.wbt, which contains a model of a Khepera robot (see figure 6.2) associated with a Khepera controller (see figure 6.3). If the Khepera controller window do not show up, press the **Step** button in the main window of Webots.

You can navigate in the scene using the mouse pointer. To rotate the scene, click on the left button and drag the mouse. To translate the scene, use the right button. To zoom and tilt, use the middle button. You may also use the mouse wheel to zoom in or out.

Using these controls, try to find a good view of the Khepera robot. You have probably noticed that clicking on an object in the scene would select it. Select the Khepera robot and choose the

6.2. RUNNING THE SIMULATION



Figure 6.2: Khepera example world

Simulation Robot View menu item. This way, the camera will follow the robot moves. Then, click on the **Run** button to start up the simulation. You will see the robot moving, while avoiding obstacles.

To visualize the range of the infra red distance sensors, go to the **File Preferences...** menu item to pop up the Preferences window. Then, check the **Display sensor rays** check box in the **Rendering** tab.

In the controller windows, the values of the infra-red distance sensors are displayed in blue, while the light measurement values are displayed in light green. You can also observe the speed of each motor, displayed in red and the incremental encoder values displayed in dark green (see figure 6.3).



Figure 6.3: Khepera Controls

6.3 Understanding the model

6.3.1 The 3D scene

In order to better understand what is going on with this simulation, let's take a closer look at the scene structure. Double click on an object in the scene, or select the **Edit Scene Tree Window** to open the scene tree window. If you double clicked on an object, you will see that object selected in the scene tree (see figure 6.4). Clicking on the little cross icon of an object name in the scene tree, will expand that object, displaying its properties.

We will not describe in details the Webots scene structure in this chapter. It is build as an extension of the VRML97 standard. For a more complete description, please refer to the Webots user guide and reference manuals. However, let's have a first overview.

You can see that the scene contains several objects, which we call nodes. You can play around with the nodes, expanding them to look into their fields, and possibly change some values. The WorldInfo node contains some text description about the world. The Viewpoint node defines the camera from which the scene is viewed. The Background node defines the color of the background of the scene which is blue in this world. The PointLight node defines a light which is visible from the light sensors of the robot. The light location can be displayed in the scene by checking **Display Lights** in the **Rendering** tab of the preferences window. The remaining nodes are physical objects and have a DEF name for helping identifying them.

The GROUND Transform is not a Solid which means no collision detection is performed

6.3. UNDERSTANDING THE MODEL



Figure 6.4: Scene tree window for the Khepera world

with this node. On the other hand, the WALL and BOX nodes are Solid nodes. They have a boundingObject field used for collision detection. Finally, the KHEPERA DifferentialWheels node defines the Khepera robot.

6.3.2 The Khepera model

As you can guess, a DifferentialWheels node defines any differentially wheeled robot. The parameters provided here correspond to the size and functionalities of a Khepera robot. For example, if you expand the children list, you will be able to find some shapes defining the body of the robot and a number of sensors, including distance and light sensors. Although on the Khepera robot, the light and distance sensors are the same device, they are divided into two logical devices in the Webots model. This makes the simulator more modular and generic. Moreover, you will notice that each device (DifferentialWheels, DistanceSensor, LightSensor, etc.) has a list of children defining either sub devices or 3D shapes.

Webots recognizes this DifferentialWheels as a Khepera robot because its model field is set to "Khepera". Moreover, each sensor is named in a specific way in order to be recognized by Webots. For example, the distance sensor with a name set to "ds0" corresponds to the first infrared distance sensor. The Khepera interface recognized distance sensors named "ds0" to "ds7", light sensors named "ls0" to "ls7", camera sensor named "k213", and distance sensors named "fs0" to "fs2" (optional floor color sensors). This allows Webots to display the Khepera window when you double-click on the Khepera robot in the 3D world or when you choose the **Show Robot Window** menu item in the **Simulation** menu while the corresponding robot is selected.

The differential wheels model

The differential wheels model of a robot is defined by a number of parameters, including the axle length, the wheel radius, the maximum speed, maximum acceleration, the speed unit, slip noise and encoder noise. Values for these parameters are provided in this example to match approximately a Khepera robot. You may need to refine them if you need a very precise model. Please refer to the Webots user guide for a complete description of these parameters.

The sensor model

The distance sensors are simulated by computing the collision between a single sensor ray and objects in the scene. The response of the sensor is computed according to its lookupTable and modulated by the color of the object (since these sensors are of "infra-red" type, red objects are seen better than green ones). The lookupTable is actually a table of floating point values which is extrapolated to compute the response of the sensor. The first value is the distance expressed in meters (increasing the biggest distance value will make the sensor look further). The second value is the response read by the controller of the robot and the third value is the percentage of white noise associated to the distance and response, expressed in the range [0;1]. For a more complete discussion on the distance sensor model, please refer to the Webots user guide.

Light sensors are pretty similar to distance sensors. They also rely on a lookupTable for computing their return value according the measured value.

6.4 Programming the Khepera robot

6.4.1 The controller program

Among the fields of a DifferentialWheels node, you may have noticed the controller field. This field defines an executable program that will control the robot. By default executable programs are searched in the Webots controllers directory, but you can define another location in the Preferences **Files and paths** tab, under the **User path**: label. This path define a directory in webots will look for a worlds and a controllers directory. The controllers directory should contain subdirectories named after the names of the controllers (i.e., khepera in our case). This khepera directory should contain an executable file named khepera.exe on Windows or khepera on Linux. Moreover, along with the executable file, you will also find sources files and possibly makefiles or project files used to build the executable from the sources.

6.4.2 Looking at the source code

The source code of the example controller is located in the following file under the Webots directory:

68

```
controllers/khepera.c
It contains the following code:
#include <stdio.h>
#include <device/robot.h>
#include <device/differential_wheels.h>
#include <device/distance sensor.h>
#include <device/light_sensor.h>
#define FORWARD SPEED 8
#define TURN SPEED 5
#define SENSOR_THRESHOLD 40
DeviceTag ds1,ds2,ds3,ds4,ls2,ls3;
void reset(void) {
 ds1 = robot_get_device("ds1"); /* distance sensors */
 ds2 = robot get device("ds2");
 ds3 = robot_get_device("ds3");
 ds4 = robot_get_device("ds4");
 ls2 = robot_get_device("ls2"); /* light sensors */
 ls3 = robot_get_device("ls3");
}
int main() {
 short left_speed=0,right_speed=0;
 unsigned short ds1_value,ds2_value,ds3_value,ds4_value,ls2_value,ls3_value;
  int left_encoder,right_encoder;
 robot_live(reset);
 distance_sensor_enable(ds1,64);
 distance_sensor_enable(ds2,64);
 distance_sensor_enable(ds3,64);
 distance_sensor_enable(ds4,64);
  light_sensor_enable(ls2,64);
  light_sensor_enable(ls3,64);
 differential_wheels_enable_encoders(64);
  for(;;) { /* The robot never dies! */
    ds1_value = distance_sensor_get_value(ds1);
    ds2_value = distance_sensor_get_value(ds2);
    ds3_value = distance_sensor_get_value(ds3);
    ds4_value = distance_sensor_get_value(ds4);
    ls2_value = light_sensor_get_value(ls2);
    ls3_value = light_sensor_get_value(ls3);
    if (ds2_value>SENSOR_THRESHOLD &&
```

```
ds3_value>SENSOR_THRESHOLD) {
    left_speed = -TURN_SPEED; /* go backward */
    right_speed = -TURN_SPEED;
  }
  else if (ds1_value<SENSOR_THRESHOLD &&
           ds2_value<SENSOR_THRESHOLD &&
   ds3_value<SENSOR_THRESHOLD &&
           ds4_value<SENSOR_THRESHOLD) {
    left_speed = FORWARD_SPEED; /* go forward */
    right_speed = FORWARD_SPEED;
  }
  else if (ds3_value>SENSOR_THRESHOLD ||
           ds4_value>SENSOR_THRESHOLD) {
    left_speed =-TURN_SPEED; /* turn left */
    right_speed = TURN_SPEED;
  }
  if (ds1_value>SENSOR_THRESHOLD ||
      ds2_value>SENSOR_THRESHOLD) {
    right_speed=-TURN_SPEED; /* turn right */
    left_speed=TURN_SPEED;
  }
  left_encoder = differential_wheels_get_left_encoder();
  right_encoder = differential_wheels_get_right_encoder();
  if (left_encoder>9000)
   differential_wheels_set_encoders(0,right_encoder);
  if (right_encoder>1000)
   differential_wheels_set_encoders(left_encoder,0);
  /* Set the motor speeds */
  differential_wheels_set_speed(left_speed,right_speed);
  robot_step(64); /* run one step */
}
return 0;
```

This program is made up of two functions: a main function, as in any C program and function named reset which is a callback function used for getting references to the sensors of the robot. A number of includes are necessary to use the different devices of the robot, including the differential wheels basis itself.

The main function starts up by initializing the library by calling the kheperalive function, passing as an argument a pointer to the reset function declared earlier. This reset function will be called each time it is necessary to read or reread the references to the devices, called device tags. The device tag names, like "ds1", "ds2", etc. refer to the name fields you can see in the scene tree window for each device. The reset function will be called the first time from the

}

6.5. TRANSFERRING TO THE REAL ROBOT

kheperalive function. So, from there, you can assume that the device tag values have been assigned.

Then, it is necessary to enable the sensor measurements we will need. The second parameter of the enable functions specifies the interval between updates for the sensor in millisecond. That is, in this example, all sensor measurements will be performed each 64 ms.

Finally, then main function enters an endless loop in which the sensor values are read, the motor speeds are computed according to the sensor values and assigned to the motors, and the encoders are read and sometimes reset (although this make no special sense in this example). Please note the robot_step function at the end of the loop which takes a number of milliseconds as an argument. This function tells the simulator to run the simulation for the specified amount of time. It is necessary to include this function call, otherwise, the simulation may get frozen.

6.4.3 Compiling the controller

To compile this source code and obtain an executable file, a different procedure is necessary depending on your development environment. On Linux, simply go to the controller directory where the khepera.c resides, and type make. On Windows, you may do exactly the same if you are working with Cygwin. If you use Dev-C++ or Microsoft Visual C++, you will need to create a project file and compile your program from your Integrated Development Environment. Template project files for both Dev-C++ and Visual C++ are available in the braiten controller directory.

Once compiled, reload the world in Webots using the **Revert** button (or relaunch Webots) and you will see your freshly compiled run in Webots.

6.5 Transferring to the real robot

6.5.1 Remote control

The remote control mode consists in redirecting the inputs and outputs of your controller to a real Khepera robot using the Khepera serial protocol. Hence your controller is still running on your computer, but instead of communicating with the simulated model of the robot, it communicates with the real device via connected to the serial port.

To switch to the remote control mode, your robot needs to be connected to your computer as described in section 6.1. In the robot controller window, select the **COM** popup menu corresponding to the serial port to which your robot is connected. Then, just click on the **simulation** popup menu in the controller window and select **remote control** instead. After a few seconds, you should see your Khepera moving around, executing the commands sent by your controller. The controller window now displays the sensor and motor values of the real Khepera. You may press the simulation **stop** to stop the real robot. The **run** will restart it. The **step** button is helpful to run the real robot step by step. To return to the simulation mode, just use the popup menu previously used to start the remote control mode. You may remark that it is possible to change the baud rate for communicating with the robot. The default value is 57600 baud, but you may choose another value from the popup menu.

Important: If you change the baud rate with the popup menu, don't change the mode on the Khepera robot, since the baud rate is changed by software. The mode on the Khepera robot should always remain set to 1 (i.e., serial protocol at 9600 bauds).

6.5.2 Cross-compilation and upload

We assume in this subsection, that you have installed the webots-kros package provided with Webots.

Cross-compiling a controller program creates a executable file for the Khepera micro-controller from your C source file. In order to produce such an executable, you can use either the Makefile.kros file (for the Khepera robot) or the Makefile.kros2 file (for the Khepera II robot). These files are provided within the khepera controller directory. From Linux, just type:

make -f Makefile.kros

for Khepera, or:

make -f Makefile.kros2

for Khepera II.

From Windows, launch the Webots-kros application and follow the instructions. In both cases you see the following messages telling you that the compilation is progressing successfully:

```
Compiling khepera.c into khepera.s
Assembling khepera.s into khepera.o
Linking khepera.o into khepera.s37
khepera.s37 is ready for Khepera (II) upload
```

It may be necessary to remove any previous khepera. o which may conflict with the one generated by the cross-compiler. In order to do so, you can type:

```
make -f Makefile.kros clean
```

Finally, to upload the resulting khepera.s37 executable file onto the Khepera robot, click on the **upload** button in the controller window. Please note that you don't need to change the mode of the Khepera robot since the upload mode is activated by software from the mode 1. The green LED of your Khepera should switch on while uploading the program. It lasts for a few seconds or minutes before completing the upload. Once complete, the robot automatically executes the new program.

72
6.6 Working extension turrets

6.6.1 The K213 linear vision turret

The example world khepera_k213.wbt contains a complete working case for the K213 linear vision turret. The principles are the same as for the simple Khepera example, except that additional functions are used for enabling and reading the pixels from the camera. The function camera_get_image returns an array of unsigned characters representing the image. The macro camera_image_get_grey is used to retrieve the value of each pixel. As seen on figure 6.5, the camera image is displayed in the controller window as grey levels and as an histogram.



Figure 6.5: Khepera K213 controls

6.6.2 The Gripper turret

figure 6.6 shows the khepera_gripper.wbt example. In this example a model of a Khepera is equipped with a Gripper device. It can grab red cylinders, carry them away and put them down. From a modeling point of view, the Gripper turret is made up of two Webots devices:

- A Servo node which represents the servo motor controlling the height of the gripper (rotation).
- A Gripper node which represents the gripping device: the two fingers.

These devices can be configured to match more precisely the real one or to try new designs. For example, it is possible to configure the maximum speed and acceleration of the Servo node, simply by changing the corresponding fields of that node in the scene tree window.





When clicking on a Khepera robot equipped with a gripper turret. The Khepera window poping up shows the gripper device (see figure 6.7). It shows the rotation of the gripper arm, the aperture of the grips, the presence of an object within the grips and the resistivity of a gripper object. If you have a real gripper mounted on a Khepera robot, it can be remote controlled by Webots.

6.7 Support for other K-Team robots

6.7.1 KoalaTM

The Webots distribution contains an example world with a model of a Koala robot. This robot is much bigger than the Khepera and has 16 infra-red sensors, as seen on figure 6.8. The example can be found in worlds/koala.wbt.



Figure 6.7: Khepera gripper controls



Figure 6.8: The Koala robot

6.7.2 AliceTM

An example of Alice robot is also provided. Alice is much smaller than Khepera and has two to four infra-red sensors. In our example, we have only two infra-red sensors (see figure 6.9). The example can be found in worlds/alice.wbt.



Figure 6.9: The Alice robot

Chapter 7

Tutorial: Using the LEGO MindstormsTM robots

In this chapter, you will learn how to use Webots with the LEGOTM MindstormsTM robots. The LEGOTMMindstormsTM is a series of LEGOTM products allowing to build robots from LEGOTM bricks. A special brick called RCX is used to control the robot. This brick contains a microcontroller chip, a LCD display, a buzzer, 3 sensor inputs and 3 actuator outputs. Available sensors include touch sensors, light sensors, rotation sensors, temperature sensors. Actuators include motors and lights. The basic box, called "Robotics Invention System" includes two motors, two touch sensors and one light sensor. This chapter will be based on this basic box. However, Webots is not limited to this basic box and you could easily go beyond this chapter by creating much more complex virtual robots based on advanced LEGOTM MindstormsTM elements.

The first section describes step by step instructions to build up the Rover robot. This robot will be used thoughout this tutorial.

The second section describes the Webots model corresponding to the Rover robot. It explains how to program its controller in Java and how to compile it.

Finally, the last section explains how to cross-compile the Java controller you used for simulating the Rover in Webots. Once cross-compiled, your controller can be uploaded into a real Rover robot!

7.1 Building up the Rover robot

One of the most interesting model that can be build straight out the "Robotics Invention System" box is the Rover robot. This robot is described in this section. It has a two differential wheels drive system, a light sensor looking down to the ground and two touch sensors.

The following tables describe the construction of the Rover robot, first the bumper, then the rear wheel, the eyes, the body and the antennas.

In the following tables, the numbers in parentheses are the length of the axles.

Warning : the yellow elastic of the bumper is not represented ; The connectors' wires are not represented ; the real antennas are not exactly the same as the ones on the pictures.



Figure 7.1: The Rover robot

































Step	Pieces	Modeling the body
22	Connectors with wire : 1- Left bumper 2- Light sensor 3- Right bumper A- Left motor B- Right motor	
23	Eyes	





7.2 Webots model of the Rover robot

Webots already includes a model for the Rover robot you just built. So, you won't have to rebuild a virtual copy of this robot. The world file containing this model is named rover.wbt and depicted in figure 7.2. This file lies in the Webots worlds directory.



Figure 7.2: The Rover model in Webots

Before opening this file in Webots, Windows and Linux users should check that have properly installed java on their computer. The java -version command should answer this question.

Once you have launched Webots and opened the rover.wbt world, press the stop button to stop the simulation and study carefully the scene. Open the scene tree window by double-clicking on the robot. The scene is very simple. It countains a surrounding wall, a textured ground displaying a track and a Rover robot. Let's open the DifferentialWheels node corresponding to the Rover robot. Looking at its children list will reveal the robot is equipped with one distance sensor (looking down) and a couple of touch sensors, i.e., the bumpers. The two wheels are implemented as Solid nodes with "left wheel" and "right wheel" as names to allow the simulator to make them rotate when necessary. Finally the controller field of this DifferentialWheels node is set to "Rover". The fact the name of the controller begin with a capital letter means that the robot is programmed using the Java langage. If you press the run button, the Rover robot will start moving on, following the track drawn on the floor, as programmed in its controller. Let's have a look at the Java controller for the Rover robot. This controller lies in the Rover subdirectory of the Webots controllers directory. It contains a single Java source file named Rover.java and a Makefile file which are used for the compilation. To compile your controller, just type make in the Rover directory and it will produce a Rover.class java binary file that is used by Webots to control the robot.

Now, have a look at the source code. Open the Rover.java in your favorit text editor and try to understand what it contains. Useful comments should help you understand some details. If you are familiar with Java you will very easily understand everything since it is a very simple example. Basically, it gets references to the distance sensor and the touch sensors, enable these sensors for measurements each 64 milliseconds and enter an endless loop in which it perform a simple line following algorithm using only the distance sensor looking down to read the color of the floor. You may modify this program, recompile it and see how your modified version performs.

7.3 Transfering to the real Rover robot

7.3.1 leJOS

Now that you have a simulation model running as you like, it is time to transfer to the real robot to see if it behaves the same. In order to proceed, you will need to install the leJOS software. The leJOS software is a replacement firmware for the LEGOTMMindstormsTM RCX brick. It is a Java Virtual Machine (JVM) that fits into the 32KB memory on the RCX hence allowing you to program your RCX in Java. The leJOS software is included on the Webots CD-ROM. Windows users will find a Windows version in the lib subdirectory of the windows directory. Macintosh and Linux users will find a source version the lib subdirectory of the common directory. The documentation, including installation instructions, is located in the common doc robots rcx directory. Please take some time to read this documentation to undestand how leJOS works. leJOS is also available from the leJOS web site¹.

7.3.2 Installation

Once you installed leJOS, as described in the installation instructions, you will have to upload the leJOS firmware into the RCX brick, replacing the LEGOTM operating system. Please follow the leJOS instructions to perform this installation. Note that you can easily revert to the LEGOTM operating system using the LEGOTMCD-ROM. Finally, you will have to set the LEJOS_HOME environment variable to point to the location where leJOS was installed. It is also necessary to add the leJOS bin directory into your PATH environment variable, so that you can use the leJOS tools from the command line.

100

¹http://www.lejos.org

7.3. TRANSFERING TO THE REAL ROVER ROBOT

7.3.3 Cross-compilation and upload

If everything was installed properly, cross-compilation and upload should be an easy task. Be sure that your robot is ready to receive a leJOS program. Go to the Rover controller directory and simply type make -f Makefile.lejos to launch the cross-compilation and upload processes. Note that it may be necessary to perform a make clean just before to remove any class file used for simulation. The cross-compilation process uses a different class file. Upload should happen just after cross-compilation and you should be able to run your controller on the real Rover robot.

7.3.4 How does it work ?

The Makefile.kros links your controller with a special Java wrapper class named Controller. This class lies in the Webots lib directory, in the RCXController.jar archive. It is a simple wrapper class between Webots Java API and leJOS API. Thanks to this system, the same Java source code can be used for both simulated robots and real robots. However, you should read carefully the limitations of leJOS Java implementation to avoid using Java features or libraries that are not supported by leJOS.

Chapter 8

ALife Contest

A programming contest based on Webots in organized permanently on the Internet. The web site of the contest¹ may provide more up to date information about it than this manual. ALife stands for "Artificial Life".

8.1 **Previous Editions**

This is actually the third edition of the ALife contest. Two editions were organized in 1999 and 2000. Each competition gathered about 10 teams worldwide made up of one to three individuals. The winners were respectively Keith Wiley from the University of New Mexico, USA and Richard Szabo from Budapest University, Hungary.

8.2 Rules

8.2.1 Subject

Two robots are roaming a maze-like environment (see figure 8.1), looking for energy. Energy is provided by chargers (see figure 8.2). However, chargers are scattered all around the environment and it is not so easy for the robots to find them. Moreover, once used by a robot, a charger will be unavailable for a while (see figure 8.3). Hence, the robot will have to go away and look for another charger. A robot will die if it fails finding an available charger before it runs out of energy. Then, the remaining robot will be declared the winner of the match.

The world configuration is choosen randomly for each match. A number of world configurations is provided within the Webots package, They are named alife.wbt, alife1.wbt, alife2.wbt, etc. Please note that the initial position and orientation of the robots may also be choosen randomly.

http://www.cyberbotics.com/contest/



Figure 8.1: The world used in the contest

8.2.2 Robot Capabilities

All robots have the same capabilities. They are based on a model of Khepera robot equipped with a K6300 color matrix vision turret. Hence each robot has a differential wheels basis with incremental encoders, eight infra-red sensors for light and distance measurement, and a color matrix camera plugged on the top of the robot, looking in front. The resolution of this camera was scaled down to 80x60 pixels with a color depth of 32 bits. As you may have already understood, analyzing the camera image is a crucial issue in developing an efficient robot controller and you probably need to perform vision based navigation, using landmarks and mapping.



Figure 8.2: A charger full of energy



Figure 8.3: An empty charger

8.2.3 Programming Language

For the contest, the robots can be programmed in Java only. This ensures that the binaries carry no viruses or cheating systems. Hence, the executables files (.class files) can be easily shared among competitors without disclosing source code. Beware, that very good Java decompilers exists and that it may be possible for a cheating competitors to restore your code from your .class. He will just miss your comments... You may protect your Java code from such piracy by obfuscating it using a Java code obfuscator. This will make the code resulting from Java decompilation very difficult to understand, and practically unusable. Free even open source Java source code obfuscators may be found on the Internet.

They is no limit on the computation time a robot can use. However, since the simulator runs approximately in real time without any synchronization with the robots, robots performing extensive computations may miss some sensor information or react too late in some critical situations.

8.2.4 Scoring Rule

Once submitted on the web site, your robot will be appended at the bottom of the hall of fame. Then, it will engage matches each round. If n robots are presents in the hall of fame, n-1 matches will be played each round. The first match will confront the last robot (bottom rank in the hall of fame) to the last but one robot (rank #n-1). If the last robot wins, the two robots will swap their positions in the hall of fame, making the last robot win one position and the last but one robot fall down to the bottom position. Otherwise, nothing is changed. Then, the new last but one robot (which may have just changed) will play against the last but two robot. If the last but one robot robot wins, they will swap their positions, otherwise nothing occurs. And so on until we reach the top of the hall of fame. This way a robot can theoretically climp up from the bottom to the top position within a single round. However, a robot can loose only one rank per round. This is to encourage new competitors to submit their robots and have a chance to climb up the hall of fame rapidly. A round will be played every day during the contest.

It is always possible to introduce a new version of an existing robot controller, by simply uploading the versions of the .class files, erasing any previous ones. When a new version of a robot controller is introduced in the contest, its position in the hall of fame remains unchanged. The next matches are run using the new version.

8.2.5 Participation

The contest is open to any people from any country. Competitors may choose run for themself or to represent their university or company. However, although competitors can update their robot controller by submitting new versions, only a single robot controller per competitor is allowed. If someone submits several robot controllers with different names into the contest, this person and the corresponding robot controllers will be banned out the contest.

8.2.6 Schedule

The contest started on July 1st 2002. From this date, competitors could download all the contest material and develop their robot controller. Matches between resulting controllers are held continuously from the middle of the summer until the end of the competition, on May 1st 2003. It is possible to enter the contest at any time before May 1st, 2003.

8.2.7 Prize

The winner of the contest will be the robot ranked at first position on May 1st, 2003. The authors of this robot will receive a Khepera II robot and a Webots PRO package (see figure 8.4).

8.3 Web Site

The web site of the contest² allows you to view matches running in real time, to view the results, especially the hall of fame that contains the ranking of the best robots with their score. It is also possible to visit the home page of each robot engaged in the contest, including a small description of the robot's algorithm, the flag of the robot and possibly the e-mail of the author. You can even download the Java binary controller (.class files) of the some robots. This can be useful to understand why a robot performs so well and to confront on your computer your own robot against a possibly better one.

²http://www.cyberbotics.com/contest/



Figure 8.4: First prize: a Khepera II robot and a Webots PRO package.

8.4 How to Enter the Contest

If you are willing to challenge the other competitors of the contest, here is the detailed procedure on how to enter the ALife contest. You will need either a Windows or a Linux machine to program your robot controller.

8.4.1 Obtaining the software

All the software for running the contest may be obtained free of charge.

- The Webots software to be used for the contest is available from the Webots download page³. This is an evaluation version of Webots which contains all the necessary material to develop a robot controller for the contest, except the Java environment. Follow the instructions on the Webots download page to install the Webots package.
- The Java 2 Standard Edition (J2SE) Software Development Kit (SDK) may be downloaded from Sun web site⁴ for free. Please use the version 1.4 of the SDK. Follow the instructions from Sun to install the SDK.

³http://www.cyberbotics.com/products/webots/download.html ⁴http://java.sun.com/j2se/1.4/download.html

8.4.2 Running the software

Launch Webots and open the world named alife.wbt. Click on the **run** to start the simulation. You will see two robots moving around in the world. Each robot is controlled by a Java program named respectively ALife0 and ALife1 located in the Webots controllers directory. You may enter their directory and have a look a the source code of the programs.

8.4.3 Creating your own robot controller

The simplest way to create your own robot controller is to start from the existing ALife0 or ALife1 controllers.

Installation

It is safer and cleaner to install a local copy of the material you will need to modify while developing your intelligent controller. Here is how to proceed:

- 1. Create a working directory which you will store all your developments. Let's call this directory my_alife. It may be in your Linux home directory or in your Windows My Documents directory or somewhere else.
- 2. Enter this directory and create two subdirectories called controllers and worlds.
- 3. Copy the file alife.wbt from the Webots worlds directory to your own worlds you just created. Copy also the the alife directory and all its contents from the Webots worlds directory to your own worlds directory. You may replace the images Alife0.png and Alife1.png in the alife directory by your own custom images. These images are actually texture flags associated to the robots. Their size must be 64x64 pixels with 24 or 32 bits depth. They should not represent a green rectangle, possibly faking the face of a charger and hence confusing the opponent. If a flag appears to be a charger fake, it will be removed.
- 4. Copy the whole ALife0 directory from the Webots controllers directory to your own controllers directory you just created. Repeat this with the ALife1 directory. This way you could modify the example controllers without loosing the original files.
- 5. In order to indicate Webots where the files are, launch Webots, go to the File menu and select the **Preferences...** menu item to open the Preferences window. Select the Files **and paths** tab. Set alife.wbt as the Default world and indicate the absolute path to your my_alife directory, which may be /home/myname/my_alife on Linux or C:\My Documents\my_alife on Windows.

From there, you can modify the source code of the controllers in your controllers directory, recompile them and test them with Webots.
8.4. HOW TO ENTER THE CONTEST

Modifying and Compiling your controller

If you know a little bit of Java, it won't be difficult to understand the source code of the ALife0 and ALife1 controllers, which are stored respectively in the ALife0.java and ALife1.java. You may use any standard Java objects provided with the Java SDK. The documentation for the Controller class is actually the same as for the C programming interface, since all the methods of the Controller class are similar to the C functions of the Controller API described in the Webots Reference Manual, except for one function, robot_live which is useless in Java. Before modifying a controller, it is recommended to try to compile the copy of the original controllers.

To compile the ALife0 controller, just go to the ALife0 directory and type the following on the command line:

javac -classpath "C:\Program Files\Webots\lib\Controller.jar;." ALife0.java
on Windows.

javac -classpath "/usr/local/webots/lib/Controller.jar:." ALife0.javaon Linux.

If everything goes well, it should produce a new ALife0.class file that will be used by Webots next time you launch it (or reload the alife.wbt world).

Now, you can start developing! Edit the ALife0.java, add lines of code, methods, objects. You may also create other files for other objects that will be used by the ALife0 class. Test your controller in Webots to see if it performs well and improve it as long as you think it is necessary.

8.4.4 Submitting your controller to the ALife contest

Once you think you have a good, working controller for your robot, you can submit it to the online contest. In order to proceed, you will have to find a name for your robot. Let's say "MyBot" (but please, choose another name). Copy your ALife0.java to a file named MyBot.java. Edit this new file and replace the line:

```
public abstract class ALife0 extends Controller {
```

by:

public abstract class MyBot extends Controller {

Save the modified file and compile it using a similar command line as seen previously. You should get a MyBot.class file that you could not test, but that will behave the same way as ALife0.class.

Register to the contest from the main contest web page⁵, providing "MyBot" as the name of the robot. Then, upload all the necessary files in your MyBot directory. This includes the following:

⁵http.//www.cyberbotics.com/contest

- MyBot.class file and possibly some other .class files corresponding to other java objects you created (it is useless to upload the ALife0.class file)
- A text file named description.txt of about 10 lines that may include some HTML tags, like hyperlinks.
- A PNG image named flag.png that will be used as a texture to decorate your robot, so that you can recognize it from the webcam. This image should be a 64x64 pixels with a bit depth of 24 or 32. It should not represent a green rectangle, trying to fake the face of a charger, otherwise it will be cancelled.

That's it. Once this material uploaded, your robot will automatically enter the competition with an initial score of 10. A contest supervisor program will use you controller to run matches and update your score and position in the hall of fame. You can check regularly the contest web site to see how your robot performs.

8.4.5 Analysing the performance and improving your competing controller

Match movies

During each round, a number of match movies are generated. They are called SomeRobot_vs_SomeOtherRobot. and stored in the results directory of the contest home page. These files can be played back with Webots. Just download them and save them in the alife_playback directory which lies in the Webots controllers directory. Rename the file to match.dat (overwriting the existing match.dat file) and open the world named alife_playback.wbt with Webots. You should then see the match playback running. To know who was the winner in a .dat file, just look at the two bottom lines of the file. If the last line ends with 0, then the first robot wins (i.e., its name is displayed on the first line of the file). Otherwise the second robot wins.

Debug and error log

In order to debug your program, or at least to understand what went wrong or right during a round match, you can save data into a log file. This will help you developing your controller, especially on Windows where the DOS console closes immediately after a controller crashes and doesn't let you read the printed messages in this console. Moreover, it may also be useful to do it during the contest matches running on the match server to understand exactly how your controller behaved during a contest match. Your log file can be retrieved from the match server after the round completed as a zipped file.

To proceed, you first need to create such a log file and then log useful information using the println statement:

8.4. HOW TO ENTER THE CONTEST

```
import java.io.*;
. . .
 PrintStream log;
 FileOutputStream file;
  try {
    file = new FileOutputStream("log.txt");
    log = new PrintStream(file);
  } catch (Exception e) { }
    . . .
    log.println("My estimated coords: ("+x+","+y+") my state="+state);
    . . .
    log.println("My energy level: "+energy);
    . . .
  loq.close();
 file.close();
. . .
```

During each round, for each competitor using this log file facility, a log file called log.zip is stored in the controller directory of the competitors directory of the contest home page. This file is the compressed version of your log.txt file. It contains all the debug messages produced by your controller along the different matches of the last round. Please note that this log file will be visible by all the other competitors, so be cautious and don't reveal your secret algorithms. Also useful, in the results directory, a file called errors.zip contains the error log of the last round, which may be useful to detect if your controller crashed, producing a java exception. Note that these files are erased at the beginning of each new round and replaced by new ones corresponding to the new round.

Robot memory

It may be useful for your robot to store some data corresponding to knowledge aquired across the different matches. Such data should be saved regularly during a normal run or, if you prefer, just when the controller energy reaches a small value (like below 3), that is the match is about to complete. The data can be in turn re-read by the controller when it starts up a new match, to refresh its memory. Here is how to implement it:

```
import java.io.*;
// to create/write into the file
Random r = new Random();
try {
   DataOutputStream s=new DataOutputStream(new FileOutputStream("memory.dat"));
   s.writeInt(100); // save 100 int
   for (int i=0; i<100;i++) s.writeInt(r.nextInt(100));</pre>
```

```
// you should rather save some useful info here instead of random garbage!
} catch (Exception e) {
    e.printStackTrace(System.out);
}
// to read from that file
try {
    DataInputStream s =
    new DataInputStream(new FileInputStream("memory.dat"));
    int t = s.readInt(); // read the size of the data
    int[] a = new int[t];
    for(int i=0; i<t; i++) a[i] = s.readInt(); // read back my garbage
    for (int i=0; i<t; i++) System.out.print(a[i]+"\t" );
} catch (Exception e) {
    e.printStackTrace(System.out);
}</pre>
```

The memory.dat file of each competitor is also made available for download to all competitors on the contest web site. This file is stored at the same place as the log.zip file, that is, within the controller directory of the competitors directory on the contest web site.

8.5 Developers' Tips and Tricks

This section contains some hints to develop efficiently an intelligent robot controller.

8.5.1 Practical issues

The ALife0 example program display a Java image for showing the viewpoint of the camera, after some image processing. This is pretty computer expensive and you may speed up the simulation by disabling this display, which should be used only for debug. By the way, during contest matches, the Java security manager is set so that your Java controller cannot open a window or display anything.

8.5.2 Java Security Manager

To avoid cheating or viruses, a Java security manager is used for contest matches ran by the automatic contest supervisor. This security manager will prevent your Java controller from opening any file for writing or reading and doing any networking stuff.

8.5. DEVELOPERS' TIPS AND TRICKS

8.5.3 Levels of Intelligence

It is possible to distinguish a number of level in the complexity of the control algorithms. These level can be ranked as follow:

- 1. The robot is able to move and avoid obstacles. However, it does not use the camera information at all and will find chargers only by chance. This correspond to the ALife0 controller.
- 2. In addition to level 1, the robot is able to recognize if a full charger is in front of it, even far away. In this case, it will be able to adjust its movement to reach the charger if not obstacles are on the way. Otherwise, the robot will look into another direction for chargers.
- 3. In addition to level 2, the robot is able to move around obstacles preventing a movement toward a full charger.
- 4. In addition to level 3, the robot is able to perform an almost complete exploration of the world, reaching places difficult to reach for simpler robots (you will rapidly notice that some places are more difficult to reach than others, the problem is that these places may contain chargers...).
- 5. In addition to level 4, the robot is able to build a map of its environment (mapping), so that once a charger is found, it is placed on the map, thus facilitating the procedure for finding it back. After completing the map, the robot can efficiently navigate between chargers without loosing time to search for them.
- 6. In addition to level 5, the robot tries to chase its opponent, blocking it, preventing it to reach chargers or emptying chargers just before it arrives.

During the previous editions of the contest, the best competitors reached level 4 (and even one reached level 5 after the contest ended). We believe that reaching level 5 or 6 may lead to significant performance improvements and probably to the first place of the hall of fame...

Chapter 9

Practical Work: Robot Soccer

Robotics soccer has become an increasingly attractive research application for mobile robotics. Many contests are organized world wide, among them the most famous are probably the FIRA contest and the RoboCup contest. This chapter will get you started with a robot soccer application in Webots.

9.1 Setup

Webots contains a setup for robotics soccer as depicted in figure 9.1. This setup is freely inspired from the official FIRA Small League MiroSot Games Rules. It can be modified to suit your needs.



Figure 9.1: A soccer simulation in Webots: soccer.wbt

Each team is composed of three robots. Each robot has a controller program which is aware of

the the position and orientation of every robot in the soccer field. Each robot can drive its motors wheels to move in the soccer field. A supervisor process is responsible for counting the time. By default, a match lasts for 10 simulated minutes which may correspond to 1 minute if your computer is powerful and if you run the match without the real time option checked in. The supervisor process also counts the goals and reset the ball and the robots to their initial positions after a goal has been scored.

9.2 Rules

The rules are very simple: you have to drive your robots so that you score a maximum of goal within the 10 minutes of the match. There are no fouls, no penalty kick or free kick.

There is no obligation to have a goal keeper, you may decide to have three players all over the field, or to have one, two or even three goal keepers!

You cannot modify robots, i.e., change their shape, add sensors, etc.

9.3 Programming

In order to program your robot, a single controller program is used for each team. The soccer_blue controller program is used for the blue team while the soccer_yellow controller program is used for the yellow team. Each of these controller programs will be run as three concurrent processes. In each instance of these programs, a test is done to determine the number of the robot which can be 1, 2 or 3, according to the name of the DifferentialWheels node. One can also test the team color the same way. The provided examples shows how to distinguish the goal keeper (number 3) from the other players (numbers 1 and 2). Hence, it is possible to have a generic soccer.c source code and to compile it to either a soccer_blue.exe or a soccer_yellow.exe executable file. Please note that on Linux and Mac OS X, the .exe extension is not used.

In order to get starting programming a robot soccer team, you should have a look in details to the soccer_blue.c or soccer_yellow.c source codes. These examples shows how to obtain the x, y and orientation for each robot from the supervisor, as well as the coordinates of the ball. They contain useful macros for that. Moreover, they show how to program each independant robot according to its number. Finally, they show how to make a fairly intelligent goal keeper that will get placed according to the ball position. The behavior of players 1 and 2 is random in this example and it is up to you to make them more intelligent!

9.4 Extensions

This very simple robotics soccer system can be configured or extended according to your needs.

9.4.1 Modifiying the soccer field

It is possible to redesign the soccer field as you need. You can enlarge it, resize the goals, change the ground texture, etc. Moreover, you can change the ball properties, like its mass, its bounce parameter, etc. All these changes are possible from the scene tree window. For resizing the field, you will have to edit the coordinates of the components of the field. It will also be necessary to update the respective bounding objects accordingly.

For example, if you want to change the bounce parameter of the ball to make it bounce less, just double click on the ball, open the ball node in the scene tree window, open the physics node of the ball node and set the bounce parameter to 0.2 instead of 0.7. This will make the ball.

9.4.2 Modifying the robots

Similarly, it is possible to modify the robots. You can change the number of robot per team, add new sensors to the robots, like distance sensors or cameras, remove the receiver sensor if you want to prevent the robots to be aware of global coordinates provided by the supervisor. All these operation can be performed through the scene tree window, using copy and paste functions and editing the robots properties. This way, it is possible to turn the soccer robots into fully autonomous robots relying only on local information and not on global coordinates provided by the supervisor.

9.4.3 Modifying the match supervisor

If you would like to modify the rules, you will probably have to modify the match supervisor. This is a small C supervisor controller program called soccer_supervisor lying in the controllers directory. The match supervisor has only three functions: (1) it measures the time decreasing from 10 minutes to zero, (2) it count the goals, update the score and reset the robots and the ball after a goal and (3) it provides each robot with global coordinates and orientation for each robot and global coordinates for the ball. You may change any of these features, and add additional features, like fouls when a robot hits another robot.

For example, let's assume you want that the robots should not touch each other, otherwise a penalty kick is called. Your supervisor program should compute the distance between each robots of different teams. If this distance drops below the size of a robot, you call the penalty. Do to so, just set the ball and robots positions so that the robot which benefit of the penalty kick is ready to kick.

This way, it is possible to add many new rules, like prevent the goal keeper to leave its goal, etc.

9.4. EXTENSIONS