MOBILE ROBOT MODELING, SIMULATION AND PROGRAMMING

MASTER PROJECT, 2012-13, SECTION OF MICROENGINEERING

BIOROBOTICS LABORATORY (BioRob)
CYBERBOTICS

Mansolino David
15th March 2013

Professor:
Auke Jan Ijspeert

Supervisor:
Olivier Michel
Context

The aim of this project was to fully integrate the DARwIn-OP in Webots. Webots is a simulator for mobile robots and DARwIn-OP is an open source miniature humanoid robot platform.

Cross-Compilation

The native robot API being very different from Webots API, a cross-compilation tool has been made in order to allow the use of controllers made in simulation on the real robot without any need of modifications.

Remote-Control

A remote-control has been developed in order to allow running controllers on the computer from Webots and interact with the robot. This remote-control allows getting the state of the sensors and sending commands to the actuators of the real robot in real time from a controller running in Webots.

Robot window

A specific robot window has been made. This window helps the user to start the remote-control mode and automatize the process of transferring a controller to the robot and cross-compiling it.

Simulation model

Thanks to the documentation and a test battery, the numerical model of the robot has been calibrated very accurately. Furthermore a library containing the main functionalities of the robot API (walking algorithm, image processing, motion playback, etc.) has been write in order to be used with Webots API.

This integration has been divided in three main steps:

1) Creation of a simulation model of the robot.
2) Creation of a cross-compilation tool.
3) Creation of a remote-control tool.

Robot window

The whole project has been fully integrated in Webots and is now distributed with each version.
## Contents

1 **Introduction**  
   1.1 Context ................................................. 3  
   1.2 Goals of the project ................................. 3  
   1.3 State at the start of the project ................. 3  
   1.4 Organisation of the report ....................... 3  

2 **Materials and Methods**  
   2.1 Webots .................................................. 5  
   2.2 Git ....................................................... 5  
   2.2.1 Github ................................................ 5  
   2.3 DARwIn-OP .............................................. 6  
   2.3.1 Hardware ............................................ 6  
   2.3.2 Robotis API ......................................... 7  
   2.3.2.1 Dynamixel Firmware ................................ 7  
   2.3.2.2 CM730 Firmware .................................. 7  
   2.3.2.3 Framework ......................................... 8  

3 **Calibration**  
   3.1 Improvement of the model .............................. 9  
   3.1.1 Camera ............................................... 9  
   3.1.2 Servos ............................................... 9  
   3.1.3 Physic .............................................. 11  
   3.1.4 Leds ............................................... 11  
   3.1.5 Buttons ............................................ 12  
   3.1.6 Adaptation of the model .......................... 12  
   3.1.7 Asymmetry in the shoot ......................... 13  
   3.2 Improvement of the library ............................ 13  
   3.2.1 Gait Manager ...................................... 13  
   3.2.2 Motion Manager .................................... 15  
   3.2.3 Motion Manager in asynchronous mode ........... 15  
   3.2.4 Vision Manager ..................................... 16  

4 **Cross-Compilation**  
   4.1 Robot .................................................... 17  
   4.2 Led ..................................................... 18  
   4.2.1 RGB Led ........................................... 18  
   4.2.2 Back LED ........................................... 19  
   4.3 Accelerometer and Gyroscope ....................... 19  
   4.4 Camera ............................................... 20  
   4.5 Servomotor ............................................ 22  
   4.6 Speaker ............................................... 24  
   4.7 Keyboard .............................................. 25  
   4.8 Improvements of the communication speed .......... 26  

5 **Revision/Validation of the model**  
   5.1 Gyroscope .............................................. 27  
   5.2 Accelerometer ......................................... 28  
   5.2.1 Physical model ..................................... 29
5.3 Servomotors

5.3.1 Speed
5.3.2 Damping
5.3.3 P controller
5.3.4 Torque feedback
5.3.5 Max torque Available
5.3.6 Apply torque

5.4 Gait
5.4.1 Isolation of the balance algorithm

5.5 Results of the improvements

6 Remote-control

6.1 Communication protocol
6.1.1 Packet computer -> robot
6.1.2 Packet robot -> computer
6.1.3 Servo
6.1.4 Camera

6.2 Camera speed improvements

7 Robot Window

7.1 SSH communication
7.2 Data compression
7.3 Cross-compilation
7.3.1 Updating the Framework
7.3.2 Installing Webots API
7.3.3 Uninstalling Webots API

7.4 Remote-control

8 User aspect

8.1 Interactions with users
8.2 Examples
8.3 User guide
8.4 Demonstration video

9 Conclusion

9.1 Extension of the Framework possibilities
9.2 Reuse in other projects
9.3 Personal opinion

References

A Servos limits
B Masses
C Example of configuration (.ini) files
D Walking parameters
E Motions Files
F  IDs of the servos

G  Walking tests
   G.1  Accelerometer
   G.2  Ground stiffness
   G.3  Inertia
   G.4  Foot size
   G.5  Balance gain
   G.6  Servos trajectory
   G.7  Feet angle

H  User guide
# List of Figures

1. Hardware structure of the DARwIn-OP. ............................................... 6
2. DARwIn-OP in a stable initial position. ............................................ 10
3. Back of the DARwIn-OP with the seven LEDs and the three buttons. .... 11
4. Window used to catch keyboard input on the real robot. ..................... 25
5. Result of the test of the gyroscope, values of the three axes of the gyroscope in function of the time during the test. ......................... 27
6. Result of the test of the accelerometer (Left: view of the three axes / Right: zoom on the axes X and Y). ........................................... 28
7. Result of the test of the accelerometer after correction (Left: view of the three axes / Right: zoom on the axes X and Y). ....................... 28
8. Value of the three axes of the accelerometer when the robot is falling forward (left) and backward (right). ........................................ 29
9. Result of the test of the speed of the servos for six different speeds. ...... 31
10. Result of the test of the speed of the servos for six different speeds, with the new model of the servos with the maximum acceleration set. ... 32
11. Movement of the arm for the damping test ........................................ 32
12. Position of the arm in function of the time during the damping test....... 33
13. Position of the arm in function of the time during the damping test with the use of both dampingConstant and staticFriction ......................... 34
14. Movement of the arm for the test of the P controller. .......................... 35
15. Result of the test of the P controller, angle of the arm in function of the time for different values of P ............................................. 36
16. Result of the test of the torque feedback, torque feedback in function of the position of the second servo of the shoulder .................... 37
17. Result of the test of the maximum torque available, position of the servo in function of the MotorForce ............................................. 38
18. Result of the test of applying a torque to a servo, position of the servo in function of the torque ...................................................... 39
19. Comparison of the values of the gyro while walking between simulation and real robot................................................................. 40
20. Test of the balance algorithm outside of the walking algorithm, values of the gyroscope in function of the time with and without balance (real robot). ...................................................... 41
21. Test of the balance algorithm outside of the walking algorithm, values of the gyroscope in function of the time with and without balance (simulated robot). ...................................................... 41
22. Test of the speed of the robot when walking .................................... 42
23. Test of the speed of the robot when walking side wise ........................ 43
24. Structure of the packet computer→robot for remote-control. The number of chars used to stock each part is indicated below by the number of circles. The parts in red are mandatory and therefore present in each packet. The parts in orange are present only if the sensor is requested. And finally the parts in green are also presents only if the sensor/actuator is requested but they can be present several times. ............................................. 46
6.2 Structure of the packet robot->computer for remote-control. The number of chars used to stock each part is indicated below by the number of circles. The parts in red are mandatory and therefore present in each packet. The parts in orange are present only if the sensor is requested. And finally the parts in green are also present only if the sensor is requested but they can be present several times.

7.1 Tab Transfer of the robot window.

7.2 Button used to send a controller to the real robot and run it thanks to the cross-compilation tool.

7.3 Sending of a controller to the robot for cross-compilation procedure. Green boxes symbolize actions done by Webots. Blue boxes symbolize actions done by the robot (even if the command is still send to the robot by Webots). Violet boxes symbolize a call to another procedure which is explained in the following of this chapter.

7.4 This window informs the user that the robot is not in the stable position and ask him what he want to do.

7.5 Button used to stop a controller running on the real robot.

7.6 Framework update procedure.

7.7 Wrapper installation procedure.

7.8 Button used to remove Webots API from the robot.

7.9 This window appears when the user want to uninstall Webots API from the robot.

7.10 Button used to start remote-control.

7.11 Starting remote-control procedure.

7.12 Small window asking the user to wait until remote-control server has been started.

D.1 Walking : x offset parameters

D.2 Walking : y offset parameters

D.3 Walking : z offset parameters

D.4 Walking : roll offset parameters

D.5 Walking : pitch offset parameters

D.6 Walking : yaw offset parameters

D.7 Walking : hip pitch offset parameters

D.8 Walking : period time parameters

D.9 Walking : dsp ratio parameters

D.10 Walking : step forward back ratio parameters

D.11 Walking : foot height parameters

D.12 Walking : swing right left parameters

D.13 Walking : swing top down parameters

D.14 Walking : pelvis offset parameters

G.1 Comparison of the values of the gyroscope while walking between simulation and real robot.

G.2 Comparison of the values of the gyroscope and the accelerometer while walking, simulation (left) and real robot (right).

G.3 Values of the gyroscope and accelerometer while walking with a small stiffness of the robot and the ground.

G.4 Values of the gyroscope and accelerometer while walking without setting the inertia matrix.
G.5 Values of the gyroscope and accelerometer while walking with the surface of each foot divided by 4. .................................................. XIV
G.6 Values of the gyroscope and accelerometer while walking when increasing the gains of the balance algorithm. ............................. XV
G.7 Values of the gyroscope and accelerometer and position of all the servos of one leg while walking. ........................................ XV
G.8 Values of the gyroscope and accelerometer while walking with different feet offset angles. .............................................. XVI
Thanks

I want to thank Olivier Michel for having hosted me in his company, and for his great help and pertinent ideas.

Prof. Auke Jan Ijspeert for having followed the project and even coming to Cyberbotics see the advancement of it.

Chase from Robotis, for the time spent for testing and the information provided about the robot.

All the users that helped me to improve the project thanks to their questions and outside viewpoint.

I want also to thank particularly Fabien Rohrer for his help and the answer to the numerous questions I asked him.
1 Introduction

1.1 Context

This project has been made at Cyberbotics and is focused on the DARwIn-OP, which is an open source miniature humanoid robot platform with advanced computational power. Thanks to the cooperation with Robotis a copy have been acquired by Cyberbotics.

This project aims at fully integrate the DARwIn-OP robot in Webots simulator. The goal is twice, first, to include the DARwIn-OP in Webots as a standard robot, but also thanks to tests made on the robot to try to make a simulation model as realistic as possible and therefore to test the boundaries of the simulation and try to push them.

1.2 Goals of the project

The first goal of the project was to improve (complete and correct) the simulation model of the robot, while comparing its behavior with the one from the real robot. The second goal was to develop a tool capable of transferring a controller from simulation to the real robot. Two ways of transferring a controller have been developed. The first one called the cross-compilation tool is a reimplementation of the Webots API on the real robot. This allows copying the source code on the real robot and recompiling it without having to do any modifications. The second one, called remote-control, consists of running the controller in Webots like for the simulation, but instead of being connected to the simulated robot, Webots connects the outputs/inputs of the controller to the sensors/actuators of the real robot.

1.3 State at the start of the project

A simulation model of the DARwIn-OP was already present in Webots before the start of the project, but this model was not complete and based only on documentation of the robot and not on tests done on the real robot. This model has been greatly improved during this project. Furthermore, the goal was not only to focus on the simulation dimension of the robot, but also to provide an easy way to transfer controller previously done in the simulator to the real robot, which was not possible to do at all before this project.

1.4 Organisation of the report

This report will starts by a brief presentation of the technologies used during the project (mainly the robot and Webots). Then the improvements done on the simulation model of the robot will be explained and followed by the cross-compilation tool. Then a chapter will explains the tests done in order to compare the real robot and the simulated one, these tests have led again at an improvement of the simulated model of the robot. In the next chapter the working of the second way of transferring a controller to the real robot (remote-control) will be explained. The robot window will then be presented; this window simplifies the transfer of controllers to the robot. Finally different user aspects experienced during this project will be presented.
2 Materials and Methods

2.1 Webots

Webots is an interactive three-dimensional mobile robot simulator. It was originally developed at the Laboratoire de Micro-Informatique (LAMI) of the Swiss Federal Institute of Technology, as a research tool for investigating various control algorithms in mobile robotics. Webots is now developed by Cyberbotics, which is the leading company in mobile robot prototyping and simulation software.

Webots uses Open Dynamics Engine (ODE) in order to create physic realistic simulation. A few actuators and sensors are available and also most of the popular robots. It is also easily possible to create custom sensors, actuators, robots or environment. Each of these robots controllers can be written and compiled thanks to the built-in IDE, in a few programming languages. For some of these robots, the controllers written for controlling the simulated model of the robot can be transferred directly to the real robot.

2.2 Git

Git is a source code control system like subversion, but in contrary to subversion Git is decentralized, it means that each person that forks a project is the leader of his own local project. It is therefore possible to push and pull branches between different users without going through the original project. Furthermore git simplifies and promotes the use of branches. By the use of branches it is possible to work in parallel on several parts of the project in the same time without any risk of conflict.

2.2.1 Github

Github is a web service that allows hosting git project. It includes several tools to simplify the communication between the project members (private message, code commenting, simple forum, etc.). It also permits to create a small wiki and a web page for each project. Furthermore it generates automatically graphs of the evolution of the project.

Git is an open source software and Github is free for open source projects. The use of Git and Github in this project has also permitted (by the use of the pull-request) the supervisor of the project to have a check to the code before to integrate it into Webots source code, and if needed to make comments or ask for specific modifications. This has allowed to save a lot of time and to adopt a good working method during the whole project. Furthermore it has allowed some Webots users to follow the progression of the project and even to make some comments/improvements.
2.3 DARwIn-OP

The Darwin-op is an open source miniature humanoid robot platform with advanced computational power. The name DARwIn-OP comes from Dynamic Anthropomorphic Robot with Intelligence Open Platform. It is developed and manufactured by Robotis (a Korean robot manufacturer) in collaboration with the University of Pennsylvania.

The DARwIn-OP is mainly used by universities and research centers for educational and research purpose. It has a total of 20 degrees of freedoms:

- 2 in the head.
- 3 in each arm.
- 6 in each leg.

This robot is available at a fairly low price and is based on open source components (both hardware and software). It has also been used in the RoboCup international competition with some success.

2.3.1 Hardware

Like it is represented on figure 2.1. The DARwIn-OP has two controllers. The main one is a SBC-fitPC2i on which Ubuntu 9.10 is running. It is equipped with a CPU of 1.6GHz, 1Gb of RAM and a flash disk of 4GB. It has also a HDMI video out, 2 USB ports, an Ethernet port, 2 audio ports (in and out) and Wi-Fi. The sub-controllers is a CM-730, manufactured by Robotis.

![Figure 2.1: Hardware structure of the DARwIn-OP.](image-url)
Most of the sensors and actuators are connected to the sub-controller. The main controller just has to send or ask the value to the sub-controller and the sub-controller is in charge of reading the state of the requested sensor or applying the specified value to the actuator. Only a HD Logitech C905 Camera of 2M pixels is connected directly to the main controller.

The following sensors are connected to the sub-controller:

- A 3 axes gyroscope.
- A 3 axes accelerometer.
- 3 push buttons.
- 2 microphones.
- Optionally 8 pressure meters (4 under each foot).

The DARwIn-OP has also several leds (rgb and normal ones), in particular the rgb led of the eyes and on the top of the head. Furthermore it has a speaker and 20 actuators. The actuators are servomotors Dynamixel MX-28 (also produced by Robotis). Dynamixel MX-28 are servomotors of high quality, made of Maxon motor and equipped with absolute encoder. They have a gear of 1:193 which allows in combination with the encoder a resolution of 0.088°. Furthermore they can produce a high torque of 2.5Nm and a speed of 54rpm despite their small weight of only 72g.

2.3.2 Robotis API

Robotis provided an API in order to control the DARwIn-OP. This API has the three following components:

2.3.2.1 Dynamixel Firmware  The Dynamixel servomotor is the only part of the DARwIn-OP which is not open source, the firmware is also provided already compiled and the source code is not available. It is however possible to create a custom firmware from scratch, but for this project no change will be done on the Dynamixel (except maybe to update them with the last version of the firmware). Indeed the firmware of the Dynamixel is well developed and continually updated by Robotis.

2.3.2.2 CM730 Firmware  The firmware to control the sub-controller is provided by Robotis, it updates the value of each sensors (Gyroscope, Accelerometer, Buttons, Servo states, etc.) and send the command to the actuators (Leds and Servos) each 6ms. This firmware is open source and it is therefore possible to change it, but this is mainly useful in the case of branching new sensors/actuators to it, which is not the case in this project.
2.3.2.3 Framework The framework is the part that is used to program the robot from the main controller. This part is open source and several examples are provided in order to show the different possibilities provided by the robot and this framework. This framework is constituted of several distinct modules:

Communication with the sub-controller This module uses a kind of UART protocol in order to communicate with the CM730. Several minor changes have been made to this module during the project in order to improve it.

Motion This module is used to execute some specifics motions and has three sub-modules. The first one called Walking is a walking algorithm and controls each servo except the two of the head. The second one called Head controls only the head; this sub-module is often used at the same time as the walking one. The last module called Action is used to play motion pre-registered in a file called motion_file, it controls all the servos.

Vision This module has two parts. The first one is used to control the camera and acquire images. The second one consists of several tools of image processing (conversion of the image in different color formats, color tracking, etc.). Several improvements have been made to this module during this project in order to speed up the acquisition of images and to render this module more compatible with Webots API.

Other This last module contains different tools used by the other modules, like different mathematical tools and tools to read/write files.

All the changes made to this framework during the project will be discussed and explained in more details in section 4.
3 Calibration

This part concerns only the simulation. A previous numerical model of the DARwIn-OP has already been made before this project. But when this model was made, the real robot was not available, so the previous model was based only on data found in the DARwIn-OP's documentation. Furthermore because the real robot was not available, it was not possible to compare the model with the real robot. Therefore some calibration were not in agreement with the real robot, some devices were not present on the model and not all the functionalities present on Robotis Framework were available in the simulation.

Now that the real robot is present at Cyberbotics office, the goal was to correct and improve this numerical model of the robot, and also to try to implement all the key functionalities of Robotis Framework in simulation.

3.1 Improvement of the model

The numerical model of the robot is contained in a file called proto, the proto of the DARwIn-OP is called 'DARwInOP.proto'. This file contains all that is needed in order to define the robot. Sensors, mechanical parts and actuators of the robot are defined in this file, as well as their name, configuration, position and even physic properties (color, mass, inertia, etc).

In this section all the improvements done on the proto will be explained. The data come from the documentation of the robot and from tests executed on the real robot.

3.1.1 Camera

A little correction was made on the value of the field of view; it has been corrected from 1.0472 to 1.023. This is not a big change but can still be very important in order to be as close as possible from the real robot in order to reproduce the exact same behavior.

The resolution of the camera of the numerical model is 160x120 pixels and for the real robot the standard resolution is 320x240 pixels. It has however been decided to leave the resolution smaller than on the real robot in order to speed up the simulation. But it must be noticed that it has also been made easy for the users to change it.

3.1.2 Servos

First of all, the position and direction of rotation of each servo has been checked. To do this, each servo has been tested one after the other, both in simulation and on the real robot, in order to see if they were corresponding and rotating in the same direction.

In order to configure the servos (torque, maximum speed, up and down limits, gain, etc.), a little program has been made for the real robot. This program read the actual state (actual position, maximum torque, actual torque, gain P/I/D, maximum speed) of the desired servo, and show them in the console. By the use of this program the configu-
ration of the servos of the numerical model of the robot has been corrected and updated.

The maximum torque of each servos (field \textit{maxForce}) has been corrected from 2.35\,Nm to 2.5\,Nm. The limits of each servos (limit before that the robot collide himself) has been set (they can be seen in appendix A). These limits were first set using the fields \textit{maxStop} and \textit{minStop}, but this was then changed in order to use the fields \textit{maxPosition} and \textit{minPosition}. The difference is that the first one set physical limits which are handle by ODE. When the limits are exceeded, ODE creates an opposed torque that brings the servo back to its limit value, but this can create some instabilities that can be seen by some little oscillations. In opposition, the second field specify software limits, which simply do not allow to put an angle greater than the limit, it is therefore very stable.

The field \textit{position} of each servos has been set, this field specify the initial position of each servos when the world is reverted. If this field is not set, the initial position of each servo is 0, which correspond in this case to the stand-up position. But the stand-up position is not a recommended position to switch on the real robot, because it is not a stable position when the servos are off. The sit down on the knee position (see figure 3.1) has been chosen because it is the only stable position (when all the servos are off) from which the robot can easily stand up. In order to set the initial position of all servos of the simulation model, the position of each servos of the real robot has been read when it is sit down and conversed to radian.

![Figure 3.1: DARwIn-OP in stable initial position.](image)

Finally it has been observed in simulation that when the torque of a servo was disabled, the arms were oscillating (like a pendulum) infinitely, contrarily to the real robot on which the arms stop quickly to move. This was due to the fact that there was not any damping or friction on the servo's model; in order to correct this, a little damping has been added to each servos thanks to the field \textit{dampingConstant} of the servos. This damping has been chosen sufficiently small in order to not perturb the motions but still big enough to stop the arm from oscillating.
3.1.3 Physics

Every solid has a Physics node, in this node everything concerning the physic parameters of the solid can be set. First, each mass and center of mass has been checked and corrected in accordance with the documentation and datasheet. Then the inertia of each solid has been added in order to be very precise on the physical behavior of the robot. The inertia is specified in the proto by the two following vectors:

\[
\begin{bmatrix}
I_{11} & I_{22} & I_{33}
\end{bmatrix}
\quad \text{and} \quad
\begin{bmatrix}
I_{12} & I_{13} & I_{23}
\end{bmatrix}
\]

These two vectors come from the inertia matrix:

\[
\begin{bmatrix}
I_{11} & I_{12} & I_{13} \\
I_{12} & I_{22} & I_{23} \\
I_{13} & I_{23} & I_{33}
\end{bmatrix}
\]

3.1.4 Leds

Two rgb-leds on the head of the robot were already present on the model, but there is also seven other smaller leds on the back of the robot. These leds are (refer to figure 3.2) starting from left, two monochromes leds of green color, three monochrome leds of (in order) red, blue and green, and finally the last two leds are rgb-leds.

![Figure 3.2: Back of the DARwIn-OP with the seven leds and the three buttons.](image)

None of these seven leds were present on the model. The two green leds represent the communication between the main computer and the CM730 sub-controller, one for each direction. They are not controllable by the user and it would be almost impossible to represent their real states (on or off) in the simulation, because there is no notion of communication between the main and the sub controllers in the simulation. For all these reasons it has been decided to not add these two leds to the numerical model. The three next leds are fully controllable by the user; therefore, they have been added to the model. Finally the two rgb-leds are always the same color as the two rgb-leds from the head, without the need (and possibility) of controlling them. In order to be as close as possible to the real robot it was decided to add them.
The first idea was to not add new LEDs, but just add a new shape to the LEDs of the head; they would then have formed a single LED, which would have been close as the idea of the real robot. But unfortunately this could not have been done, because the two parts of the shape would not have been referred to the same solid, which is impossible to do in Webots. The second idea was to add a second independent LED but this idea was abandoned, because it would have broken the idea that the user has to control only one LED and the other follows it. A good solution has finally been found, a specific material has been defined (by the DEF-USE mechanism) for each RGB-LED of the head. Then the two small LEDs on the back of the robot are represented by small solids (simple solid, not a LED) and the material previously defined for the head LEDs are used for these small solids. When changing the state of one of the RGB-LED of the head, the texture of the material used for the LED change in order to set the good color, and because the same material is used for the small solid on the back of the robot, the color change too, like if it is a second LED.

3.1.5 Buttons

Just below of the LEDs on the back of the robot there is three buttons like shown on figure 3.2. The right button is used to reset the CM730 sub-controller (but not the main controller). The two other buttons are available to the user; he can easily use them in his program. These buttons could have been symbolized on the numerical model by touch sensors, which are sensors used to measure the force applied to them. However, this is not really useful and convenient to use in simulation, so it has been decided to not add them.

3.1.6 Adaptation of the model

After all these calibrations of the numerical model, it appeared that the simulated model of the robot was not as stable as the real one, it was often falling during walking and especially when shooting. In order to have a behavior as close as possible from the real robot some previous calibration have been adapted in order to have a behavior matching perfectly with the real robot, at the cost of a distance of the data coming from the documentation of the robot.

First the center of mass of the body has been moved a little bit in order to increase the stability. It has been moved in direction of the back of the robot because it was always falling forward, and also lowered in order to be closer from the ground (which increase the stability too). The center of mass of the body has been chosen, because it is the central part of the robot and also because it represents 33.4% of the total mass (see appendix B). The size of the feet have also been increase of 25% in order to increase the contact surface and also the stability.
3.1.7 Asymmetry in the shoot

Finally it has been observed an asymmetry between the right and the left shoot. The left shoot was more stable than the right one; there were less laterals oscillations of the robot. The symmetry of the robot in the proto file has been checked, especially the masses, centers of mass and position of all solids. But nothing has been found, the model was perfectly symmetric. The two shoots have therefore been tested on the real robot and surprisingly the two shoots were different like in the simulation. In order to find the source of this difference, the two motion for the right and left shoot have been compared in the file storing the motions (motion_4096.bin). It appeared that the two motions were not simply the symmetry one of the other, but were two independent motions; this explains the difference between the two motions.

In conclusion, what seems first to be a problem of the simulation model, appeared finally to be perfectly normal and attests of a good matching between the real and the simulated robot.

3.2 Improvement of the library

A library has been developed in order to implement all the key functionalities of Robotis Framework in simulation. This library is divided in three parts called managers. Each manager implements a module of the Framework. The first one called Gait manager allows using the walking algorithm of Robotis Framework. The second one called Motion manager allows to play predefined motions stored in the motion_4096.bin file. The last one called Vision manager, contains some image processing tools, useful for example to find a colored ball in the camera image.

3.2.1 Gait Manager

This manager implements the DARwInOPGaitManager class and allows using the walking algorithm of the Framework. This manager was already present in the library before the start of this project, but has been improved in order to make the gait in simulation closer to the gait of the real robot, to extend its possibilities, to make it more compatible with the cross-compilation and finally to add the balance algorithm which use the gyroscope to prevent the robot from falling.

A lot of parameters are available in the Framework algorithm in order to tune the gait. But in order to make this manager easy to use by the user, it has been decided that only a subset of the parameters can be set with this manager. The other parameters are set to default values that are known to works fine, and the user can still change them if he wants just by changing the default values that are stored in configuration file *.ini. In appendix C an example of a config.ini file is shown and in appendix D all the gait parameters are explained. For this reason, the constructor of DARwInOPGaitManager is the following:

```cpp
DARwInOPGaitManager(webots::Robot *robot, const std::string &iniFilename);
```
The first parameter is the robot on which the algorithm applies and the second is the file name in which the default parameters are stored. The following methods are available in order to modify the main parameters in the controller:

```c
void setXAmplitude(double x);
void setYAmplitude(double y);
void setAAmplitude(double a);
void setMoveAimOn(bool q);
void setBalanceEnable(bool q);
```

These are the open parameters, they have the following impact on the gait:

- X influences the length of the foot step forward, it can take any value between -1 and 1.
- Y influences the length of the foot step in the side direction, it can take any value between -1 and 1.
- A influences the angle of the gait and allows also the robot to rotate during the walk, it can take any value between 0 and 1.
- If MoveAimOn is set, it allows the robot to rotate around something by inverting the sense of rotation, it can be very useful to turn around a ball in order to kick it in the right direction for example.
- If BalanceEnable is set, the gyroscope is used in the control loop to make the walking gait more robust.

Finally the following method can be used in order to run the algorithm:

```c
void start();
void step(int ms);
void stop();
```

Start and stop need to be used to stop/start the algorithm and step is used to run \( ms \) milliseconds of the algorithm.

Note that, in order to run, the gait manager needs to know the position of each servo and the values of the gyro. It is therefore essential to enable the gyro and the position feedback of each servo before to use it; if it is not the case, a warning will appears and they will automatically be enabled.
3.2.2 Motion Manager

This manager implements the *DARwInOPMotionManager* class and allows to play predefined motions stored in the file *motion_4096.bin*, the main motions and the corresponding id of the motion are explained in appendix E. Here again, this manager was already present in the library but has been improved during this project in order to correct some bugs (it was not possible to play long motions, motions that need to be played several times were played only once and it was not possible to play motions from any position), to add the possibility to play motion asynchronously and to render it more compatible with the cross-compilation.

The constructor of *DARwInOPMotionManager* object is the following:

```
DARwInOPMotionManager(webots::Robot *robot);
```

It only needs a pointer to the robot to which it applies. Then, the following method can be used to play a motion:

```
void playPage(int id);
```

This method only needs the id of the motion to be played, it stops the controller until the motion is finished.

3.2.3 Motion Manager in asynchronous mode

By default when starting a motion, the motion is run synchronously. That is the controller is stopped until the motion is finished. However, it is also possible to run a motion asynchronously. In that case, the motion is started but the execution flow of the controller is not stopped. This can be done by calling the method *playPage* with the second parameter set to false:

```
void playPage(int id, bool sync = true);
```

This will initiate the motion, but not run it, then in order to play ms second of the motion, the following method need to be called (before the robot step):

```
void step(int ms);
```

In order to know if the motion is finished, the following method can be called:

```
bool isMotionPlaying();
```
3.2.4 Vision Manager

This manager implements the DARwInOPVisionManager class which contains some image processing tools. This manager was not present at all before the start of this project. The constructor of this class needs the following parameters:

```c
DARwInOPVisionManager(int width, int height, int hue, int hueTolerance, int minSaturation, int minValue, int minPercent, int maxPercent);
```

The parameters are the following:

- The width of the image
- The height of the image
- The color hue of the target object to find
- The tolerance on the color hue of the target object to find
- The minimum color saturation of the target object to find
- The minimum color value of the target object to find
- The minimum percentage of color value in the image to validate the result
- The maximum percentage of color value in the image to validate the result

When an instance of this class is created, the getBallCenter method can be used to find the position of the target object:

```c
bool getBallCenter(double &x, double &y, const unsigned char * image);
```

This method returns true if the target was found, and false otherwise. If found, the x and y variables are set. The image pointer indicates the original image buffer. In order to find the position of the target object, this method proceeds to the following steps:

- Store the BGRA version of the image in a buffer
- Use this buffer to convert the image to HSV format
- Use the Finder class of the Framework to find the target object
- Extract and save the position of the target object

Once this method was called it is possible to know which pixels of the image are part of the target object by using this function:

```c
bool isDetected(int x, int y);
```

This method returns true if the pixel (x,y) is part of the target object and false otherwise.
4 Cross-Compilation

The cross-compilation tool is a wrapper that makes a bridge between Webots API and Robotis API, in order to allow the use of Webots controller on the real robot without any modification. This tool consists of a wrapper of all the Webots functions and is organized in different modules (each implemented in its own class). Each of this module represents one sensor or actuator except the module Robot which is used to mimic the working of Webots simulation.

In order to compile a controller with Webots API the controller must be compiled with a specific makefile called Makefile.darwin-op.

In the next sections, all the different modules will be explained and the working of their functions will also be explained. This chapter is very technical and in order to make it not too much annoying and long, some simplifications have been made, for example only the mains functions are explained (and only the public ones). But this chapter permits still to have a great overview and understanding of the cross-compilation tool, however for more details please see the source code. Now the following modules will be explained:

- Robot
- Led
- Accelerometer
- Gyroscope
- Camera
- Servomotor
- Speaker

4.1 Robot

This is the main module, it is required by every controller in order to work. At the beginning of each controller, the constructor of Robot is called, in this constructor, the physical robot is initialized, this is done by starting the connection with the CM730 card, checking that the firmware of the servomotors is up to date (too old firmware of the servomotors can cause several problems). Then all the devices (servos, gyro, leds, etc.) are created and initialized, the time step is loaded from the file 'config.ini' attached to the controller (if no file is present, no time step is specified or a time step smaller than 16ms is specified, a default time step of 16ms is used, more information on the choice of this value of 16ms in section 4.8), then the camera resolution is also loaded from the file 'config.ini' (here again if there is no file, a bad resolution or no resolution is specified, the default resolution of 320x240 is used, more information about this in section 4.4). Finally all the servos are slowly (in order to avoid any damage if starting from any wrong position) moved to the initial position (see figure 3.1 representing the initial position, which is the same as in simulation) and the eyes and head leds are set to green in order to indicate that the robot controller will now starts.
This module contains also a lot of different getters in order to obtain pointers on the different devices of the robot; like for example the one to obtain the accelerometer:

```cpp
Accelerometer *getAccelerometer(const std::string &name) const;
```

The use is exactly the same as in simulation, the name of the desired device has to be set as parameter and the function returns a pointer to the device. The names have been set to correspond exactly to the names used in the proto of the DARwIn-OP.

This module also contains the function `step` which is very important because as in simulation, it allows to run one time step of the controller. This function compute the time elapsed since the last time that it has been called, if the time is smaller than the time step, it waits the remaining time and then returns 0, if the time is greater, it returns immediately the number of milliseconds in excess. This function has then been completed by the other modules, but this will be explained in the following sections.

Finally the following small functions are also implemented in this module:

- `getName()` Returns always the string "darwin-op" (this function is present only for compatibility reasons).
- `getTime()` Returns the time in second since the controller was started.
- `getBasicTimeStep()` Returns the time step in milliseconds.

### 4.2 Led

The constructor of this device, use a static standard map in order to map the leds names to their ID, this is only done at the instantiation of the first led.

Then when an instance of the class `led` is created, the following methods are available for this led:

```cpp
set(int value);
int get();
```

The first one is used to set the led to a specific color (for rgb ones) or state (on/off for the back leds). And the second one is used to get the color or state of the led.

#### 4.2.1 RGB Led

The rgb leds are the easier ones, because setting or getting their color is simply done by writing/reading the two bytes on the CM730 corresponding to the desired led. For example `P_LED_EYE_L` and `P_LED_EYE_H` for the eye led. This can easily be done by using the following functions of the class CM730 from the Framework, which implements the communication with the CM730:

```cpp
int ReadWord(int id, int address, int *pValue, int *error);
int WriteWord(int id, int address, int value, int *error);
```
4.2.2 Back LED

For the back leds, the same principle remains, but the state of the three leds is encoded in only one byte ($P\_LED\_PANNEL$). So when setting or getting the state of only one of the three leds, a mask is applied in order to influence only one led and not the three in the same time. And here because only one byte (and not two like for the rgb leds) needs to be read/write, other functions of the class CM730 are used:

```c
int ReadByte(int id, int address, int *pValue, int *error);
int WriteByte(int id, int address, int value, int *error);
```

4.3 Accelerometer and Gyroscope

The accelerometer and gyroscope modules are very similar and they work the exact same way, for each one of these classes, the following methods are available:

```c
void enable(int ms);
void disable();
const double *getValues() const;
int getSamplingPeriod();
```

The functions `enable` and `disable` are empty functions but here again they are present for compatibility reasons with Webots API. The function `getSamplingPeriod` simply returns the time step of the controller, because the refreshing rate of these sensors is quicker than the minimum time step acceptable (16ms). Finally the most useful method is `getValues` which returns the values of the sensor. To get the values from the sensors, here again the corresponding two bytes of the sensor for the desired axis must be read from the CM730 (for example $P\_GYRO\_X\_L$ and $P\_GYRO\_X\_H$ for the axis X of the gyroscope), so for the function `getValues`, the values of the three axes are read and stored in a table and the address of this table is return by `getValues`.
4.4 Camera

This module implements the class Camera, when an instance of this class is created, the following methods are then available to the user:

```c
void enable(int ms);
void disable();
const unsigned char *getImage() const;
int getWidth() const;
int getHeight() const;
double getFov() const;
int getType() const;
double getNear() const;
int getSamplingPeriod();
static unsigned char imageGetRed(const unsigned char *image, int width, int x, int y);
static unsigned char imageGetGreen(const unsigned char *image, int width, int x, int y);
static unsigned char imageGetBlue(const unsigned char *image, int width, int x, int y);
static unsigned char imageGetGrey(const unsigned char *image, int width, int x, int y);
```

The methods `enable` use the class LinuxCamera from the Framework in order to initialize the camera then it creates a new thread which continuously takes an image (again by using the class LinuxCamera) and when the image is ready, it copies it in the image buffer and takes a new image. The function `disable` stops the thread and frees the image buffer.

The others methods work as follow:

- **getImage()** Returns a pointer to the image buffer.
- **getWidth()** Returns the width of the camera (set in the constructor of robot)
- **getHeight()** Returns the height of the camera (set in the constructor of robot)
- **getFov()** Returns always 1.0123
- **getNear()** Returns always 0.0
- **getSamplingPeriod()** If the time step is smaller than 30ms, it returns 30 (because camera refresh rate is 30ms) else it returns the time step.
- **imageGetRed()** Returns the red component of pixel (x, y)
- **imageGetGreen()** Returns the green component of pixel (x, y)
- **imageGetBlue()** Returns the blue component of pixel (x, y)
- **imageGetGrey()** Returns the mean of red, green and blue component of pixel (x, y)
After some tests of the camera in cross-compilation, it appears that enabling the camera was slowing down a lot the controller, because the new thread was very computationally expensive. This was due to the class LinuxCamera from the framework which had a bad algorithm to extract the image buffer from the buffer returned by the camera. The algorithm was following the following step:

- Extraction of the image buffer in the YUV format from the buffer returned by the camera.
- Vertical flip of the image in the YUV buffer.
- Horizontal flip of the image in the YUV buffer.
- Conversion of the buffer in the RGB format.
- Conversion of the buffer in the HSV format.

All these steps are very long due to the high number of pixels of the image. So in order to save computation power, the class LinuxCamera has been modified in order to avoid all these steps, because each step is a loop over each pixel. It has been improved in order to use only one loop (instead of five) which directly extract the buffer in BGRA format. The BGRA format has been choose instead of RGB because it is the format used by Webots. Thanks to this the thread uses a lot less power and allows the controller to run quicker.

The resolution of the camera can be set in the file ’config.ini’ and is loaded in the constructor of Robot (see appendix C for an example of configuration file). The resolutions supported by the camera of the robot have been tested and the working resolutions and their respective refresh time are reported in table 4.1.

<table>
<thead>
<tr>
<th>Width [pixel]</th>
<th>Height [pixel]</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>320</td>
<td>240</td>
<td>30</td>
</tr>
<tr>
<td>640</td>
<td>360</td>
<td>30</td>
</tr>
<tr>
<td>640</td>
<td>400</td>
<td>30</td>
</tr>
<tr>
<td>640</td>
<td>480</td>
<td>30</td>
</tr>
<tr>
<td>768</td>
<td>480</td>
<td>28</td>
</tr>
<tr>
<td>800</td>
<td>600</td>
<td>22.5</td>
</tr>
</tbody>
</table>

Table 4.1: Camera resolutions supported by the camera of the DARwIn-OP.
4.5 Servomotor

This is the bigger module and also the more complex one. All the following methods are available:

```c
void enablePosition(int ms);
void disablePosition();
void enableMotorForceFeedback(int ms);
void disableMotorForceFeedback();
int getSamplingPeriod();
int getType() const;
void setVelocity(double vel);
void setForce(double force);
void setMotorForce(double motor_force);
void setControlP(double p);
void setPosition(double position);
void setAcceleration(double force);
double getMotorForceFeedback() const;
double getPosition() const;
double getTargetPosition();
double getMinPosition();
double getMaxPosition();
```

The four first methods for enabling and disabling the position/motor force feedback are empty functions; they are present only for compatibility reasons. The `getSamplingPeriod()` method simply returns the time step, because the sampling period of the servos is 6ms which is smaller than the minimum time step accepted. Finally the method `getType()` returns always the value `WB_SERVO_ROTATIONAL` because all of the servos of the DARwIn-OP are rotational.

The next methods work all on the same principle, they convert and write the corresponding value on the specified servo by using the function `WriteWord` of the class `CM730`:

- **setVelocity**  Writes the value multiplied by $\frac{30}{0.114}$ (the factor $\frac{30}{\pi}$ is here in order to convert from rad/s to rpm and 0.114 is here because motor unit is 0.114rpm) on the register `P_MOVING_SPEED_L` of the servo.

- **setMotorForce**  Writes the value multiplied by $\frac{25}{1023}$ (the maximum force and $1/1023$ is the resolution) on the register `P_TORQUE_LIMIT_L` of the servo. If the value is bigger than 2.5 it simply writes 1023 on the register. If the value is 0, it writes 0 on the register `MX28::P_TORQUE_ENABLE` of the servo in order to completely release the torque.

- **setControlP**  Writes the value multiplied by 8 (because the resolution is 1/8) on the register `P_P_GAIN` of the servo.

- **setPosition**  The value is multiplied by $\frac{180}{\pi}$ (in order to convert from radian to degree) and then passed to the function `MX28::Angle2Value()` in order to convert it to the servomotor encoder position. Then a little security function is called in order to avoid self-collision (the function simply checks in some table if the value is not over or under some collision limits of the corresponding servo). Finally the value is written on the register `P.GOAL_POSITION_L` of the servo.
The function \texttt{setForce} is a little bit different, it starts by calling the function \texttt{setMotorForce} to set the desired force and then call the function \texttt{setPosition} to ask the servo to go to one of the collision limit (depending on the sign of the value send to \texttt{setForce}). The function \texttt{setAcceleration} is also different, because the servo do not offer the possibility to specify the desired acceleration (but this will be available in a future version of the firmware of the MX28). This function activates a mechanism that recalculates the desired speed and call the function \texttt{setVelocity} with the desired speed as argument at each time step (this has been added in the function \texttt{Robot::step}).

The function \texttt{getMotorForceFeedback} and \texttt{getPosition} are very similar they read the register \texttt{P_PRESENT_POSITION_L} and \texttt{P_PRESENT_LOAD_L} of the servo and convert the value respectively to newtons and radians. The function \texttt{getTargetPosition} simply return the last value set with the function \texttt{setPosition}.

Finally the last two functions \texttt{getMinPosition} and \texttt{getMaxPosition} simply returns the value of the upper or lower limit from the collision limits table.
4.6 Speaker

Speakers are not yet present in Webots, but as they will be added in a future version of Webots and they are used on the real robot by the framework, it has been decided to implement a module Speaker in cross-compilation. This module has the following public methods:

```c
void enable(int ms);
void disable();
void playFile(const char* filename);
void playFileWait(const char* filename);
void speak(const char * text, const char * voice = "en", int speed = 175);
void speakFile(const char * filename, const char * voice = "en", int speed = 175);
```

Here again the two first methods are present only for a future compatibility with Webots speakers, and they are empty functions. The methods playFile and playFileWait call some Framework functions in order to play an audio file (MP3 for example). The difference between these two methods is that the first one play the sound while the controller continue to run whereas the second one stop the controller until the sound is finished.

The two last methods have been added in order to add a new possibility of text to speech to the robot, which is not possible to do with the Framework. This two functions use espeak in order to generate sound directly from the text, or from the text contained in a file. Optionally, the voice can be chosen and also the number of words per minute.
4.7 Keyboard

In order to export all the functions concerning the keyboard present in Webots to the robot, a little library using X11 (standard graphical environment system on Linux) has been developed. This library can be used to open a small window which is used to get keyboard inputs. X11 development tool is not present on the robot, and because the idea is to change as less as possible the robot environment, it has been decided that this library will be distributed already compiled from a Linux 32bits, in order to not having to compile it on the robot (which would not be possible without installing X11 development tool on the robot). This small library has the following functions (contained in the Keyboard class):

```c
void createWindow();
void closeWindow();
void startListenKeyboard();
void resetKeyPressed();
int getKeyPressed();
```

The two first functions are used to create and destroy the window (see figure 4.1 for a preview of the window). The function startListenKeyboard is used to initialize the keyboard events, then, each time a key is pressed, its state (which is stored in a table) is updated, the function resetKeyPressed is used to reset the states table. And finally the function getKeyPressed returns one of the keys pressed (the next call of this function returns the next key pressed and not the same one, if no key have been pressed or the function has been called more times than the number of keys pressed, the function returns 0).

![Webots Cross-Compilation: Keyboard inputs](image)

This window is used to catch keyboard input by the controller.
Please do not close it and do not unfocus it.
It will be closed automatically when the controller terminates.

Figure 4.1: Window used to catch keyboard input on the real robot.

This library is used in the module Robot. The function keyboardEnable has been added to the robot class, this function called the two functions createWindow and startListenKeyboard, similarly the function keyboardDisable calls the function closeWindow. The function keyboardGetKey can be used to get the key pressed, this function simply called the function getKeyPressed. And finally in the robot step the call of the function (only if keyboard is enabled) resetKeyPressed has been added.
4.8 Improvements of the communication speed

The cross-compilation in this state worked well, but was very slow. It appears after some tests that the problem was due to the communication between the two controllers (the fitPC and the CM730 card), indeed the communication is not very quick, each writing/reading of a word (2 bytes) takes approximately 1ms. So when the controller has to send a lot of commands, this slow down a lot the controller speed. For example sending a position for each servos send 20 commands and takes therefore approximately 20ms, and this is only for setting position of the servos, the use of the accelerometer, gyro, led, other settings of the servos, etc. can quickly slow down a lot the controller.

In order to speed up the communication speed, two mechanisms have been used. The first one concerns the sending of commands to the servos (position, speed, motorForce and controlIP), when one of the functions setting one of the parameters is used, the command is now not directly send to the servo, but the value is stored in a variable. Then in the robot step, a function calling SyncWrite is called, thanks to this function, it is possible to send all the commands of all the twenty motors in only one packet (instead of one packet for each command and for each servos) to the CM730. This results in a huge gain of time, by the use of the function SyncWrite the time needed to send all the commands to all the servos is only 1-2ms (instead of 1ms for only one command for one servo).

The second mechanism concerns the reading of all the sensors (including the servos speed, position and load). Thanks to the function BulkRead, it is possible here again to receive in only one packet the value of all the sensors, so instead of asking the value directly when a sensors function is called, the function BulkRead is called at each robot step in order to update all the sensors values and store the values in the corresponding variables. Then when a function is asking for a sensor value, it is the value of the corresponding variable that is returned. Here again the time needed to receive all the sensors values thanks to the function BulkRead is only 1-2ms. But the function BulkRead had a little problem, indeed it permitted to read the state of all the sensors but concerning the servos, it permitted only to read the position and not the speed and load. So in order to include the reading of the speed and load, the function BulkRead from the Framework has been a little bit improved.

Thanks to these improvements, the controller can now easily respect a time step of 16ms (same time step as in simulation) whatever the number of sensors/actuators are used by the controller.
5 Revision/Validation of the model

In this part, the cross-compilation tool developed in the previous section will be used in order to run several tests controllers both in simulation and on the real robot. The resulting behaviors and sensors values from simulation and from the real robot will be compared. This will permits to validate the simulation model and the functioning of the cross-compilation tool if the results are similar in simulation and on the real robot. Otherwise, it will bring out a problem whether of the simulation model whether in the cross-compilation tool. After identifying the source of this mismatching between reality and simulation, this will be revised in order to make the simulation model as close as possible to the real one.

5.1 Gyroscope

The first device checked was the gyroscope. The test controller consists of first making the robot stand-up and then move the four servos of the hip in order to make the body of the robot swing (without any movements of the feet, see *GyroTest* video in the CD-ROM attached to this report, for a demo of this test). In the meantime, the values of the gyroscope are read and saved for both the three axes. The result of this test can be seen in the figure 5.1.

![Gyroscope Test Graph](image)

Figure 5.1: Result of the test of the gyroscope, values of the three axes of the gyroscope in function of the time during the test.

The first five seconds of the test have been cut from the graph in order to not take into account the transitional regime of the beginning of the test. The curve of the three axes are very similar on the real robot and in simulation, we can thus conclude that there is a good matching between the simulated model of the gyroscope and the real gyroscope. We can therefore validate the gyroscope, the orientation of all the three axes is correct and the output function is correct too.
5.2 Accelerometer

The controller test of the accelerometers was the same as the one of the gyroscope, despite the fact that it was the values of the three axes of the accelerometer and not of the gyroscope that were stored. Here again the result of this test can be seen in figure 5.2.

![Figure 5.2: Result of the test of the accelerometer (Left: view of the three axes / Right: zoom on the axes X and Y)](image)

Here again the transitional regime has been cut. The curves of the three axes are very similar on the real robot and in simulation, but we clearly see a symmetry of the axes X and Z around the value 512. 512 been the central value (when there is no acceleration), we can conclude that the axes X and Z are in the wrong direction in the simulation model. We can see the result of the same controller test after having corrected the accelerometer model in simulation in figure 5.3.

![Figure 5.3: Result of the test of the accelerometer after correction (Left: view of the three axes / Right: zoom on the axes X and Y)](image)

Now all the curves match perfectly. This validate the accelerometer.
5.2.1 Physical model

Now that the accelerometer has been validated, it is used in order to compare the physical model of the robot in simulation. For that purpose the controller test consists of first making the robot stand-up and then making it to bow slowly (see FallfrontTest and textitFallbackTest videos for a demo of this test), by increasing or decreasing the angle at the feet and storing the value of the three axes of the accelerometer in function of the angle of the feet. We can see in figure 5.4 the result of this test when falling forward (left) and when falling backward (right).

![Figure 5.4: Value of the three axes of the accelerometer when the robot is falling forward (left) and backward (right).](image)

We can clearly see that in the two cases the simulation model require a bigger angle to fall than the real robot. It means that the simulated robot is more stable than the real one, this is entirely normal, because it has been made more stable deliberately by moving the center of mass of the model down and increasing the size of the feet in order to not fall when doing a shoot. We can also see a bigger difference when falling forward, here again this is entirely normal, because the center of mass of the model has also been moved backward for increasing the stability.
5.3 Servomotors

A lot of tests have been made on the servomotors in order to verify that the characteristics set for the servomotors in simulation match with the real one and to verify the working of the cross-compilation tool. The following characteristics have been tested:

- The maximum speed and acceleration.
- The damping.
- The effect of the controller P.
- The maximum torque available.
- The torque feedback.
- The control of the servomotors by torque.

Each of this test have been made on only one of the twenty servomotors, and because they are all the same model, we expect that they have all a very similar behavior. In the next sections all the tests concerning the characteristics listed above will be explained and the results commented.

5.3.1 Speed

The first test was about the speed, this test has been made on the servo of the neck (servo 19 in appendix F), because this servo is the more free to rotate (big angle of rotation available, no risk of collision with another part of the body, small inertia, etc.). This test controller consists of making the robot watching to the left (at an angle of 90°) and as soon as it reach this angle watching to the right (at an angle of -90°), this is repeated several times while recording the time needed for each rotation. See SpeedTest video for a demo. This test controller has been run six times with six different speeds covering the whole range of speeds available by these servos. The result of this test are shown in figure 5.5.
Figure 5.5: Result of the test of the speed of the servos for six different speeds.

We can clearly see a difference between the real robot and the simulated one increasing with the speed. The real robot takes more time at high speed to make the same amount of rotation than the simulated one, so the simulated robot is too fast. We can first think that the maximum speed of the simulated servo is too high despite the fact that it is in agreement with its datasheet, but after that some more tests were conducted it appears that it was not the case. Indeed it was not a problem of maximum speed but a problem of acceleration, the maximum acceleration of the simulated servos was not set, and therefore the servo was reaching instantly the maximum speed unlike the real servos which need some time to reach the maximum speed. So when there was a lot of changes of direction because of the high speed, the impact of the acceleration was bigger and it is for this reason that the difference was bigger at high speed. On figure 5.6 the result of the test with the maximum acceleration set to $55 \text{ rad/s}^2$ is shown.
Figure 5.6: Result of the test of the speed of the servos for six different speeds, with the new model of the servos with the maximum acceleration set.

There is now a very good matching for the whole area of the speed. This validates both the speed and the acceleration of the simulated servomotors.

5.3.2 Damping

The next characteristic of the servos tested was the damping. This test has been done on the servos of the shoulder (servo 1 in appendix F), the test consists of putting the arm at $90^\circ$ (figure 5.7 right) and then releasing it (setting the motorForce to 0, see DampingTest video for a demo of this test).
In the chapter 3.1.2, the field `dampingConstant` of each servos has been tune in order to have visually the same movement in simulation and on the real robot. But here the position of the arm is recorded each 16ms until the arm stops (figure 5.7 left) for different values of damping. The result of the test can be seen in the following figure 5.8.

![Figure 5.8: Position of the arm in function of the time during the damping test.](image)

Wherever the value of the `dampingConstant` was, it has not been possible to find a good matching (number of oscillations before to stop, value of the overshoot and final position) between the model and the real robot. Indeed in Webots the only way to add damping to the servos was the use of `dampingConstant` which apply a resistant torque on the servos opposed to the moving direction and proportional to the servo speed. This was absolutely not sufficient to model all the different kinds of damping; it was for this reason that it was not possible to have a good matching between the two curve.

It has also been decided to try to improve the model of damping of Webots. A second parameter for the damping called `staticFriction` has been added by using a physic plugin (in order to not make any change to Webots sources in a first time). This new parameters apply a resistant torque on the servos, opposed to the moving direction but this time independent of the servo speed. Here again the damping test has been run for different combinations of values of `dampingConstant` and `staticFriction`. The best result has been found with a `dampingConstant` of 0.002 and `staticFriction` of 0.025 N * m, the result can be seen in the figure 5.9.
Even if the two curves are not exactly the same, the result is much better than with only one parameter. Of course it would have been possible to add again a new parameter to improve the damping model but it has been decided that this model was sufficiently realistic for our application.

In front of the good results of the new model of the damping implemented in the physic plugin, it has been decided to add this second parameter to Webots source code, the field staticFriction is now available in Webots for each servos since Webots version 7.0.3 thanks to Fabien Roher who integrates this physic plugin into Webots.
5.3.3 P controller

This test has been done on the second servos of the shoulder (servo 3 in appendix F) the test consists of putting the arm down (figure 5.10 left, servo is at angle -0.8 rad) setting a specific value for the P controller and then asking the servo to put the arm up (figure 5.10 right, servo is at angle 0.8 rad, see ControlPTTest video for a demo of this test). The position of the arm is then recorded each 16 ms.

![Figure 5.10: Movement of the arm for the test of the P controller.](image)

This test has been run several times with different values for the P controller both on the real robot and on the simulated one. The result can be seen in figure 5.11.
As we can see on figure 5.11 there is a huge difference of behavior of the P controller between the real robot and the simulated one. The bigger difference and also the most important is that on the real robot the arm never reach the target for small values of the P controller and the error decrease by increasing the P controller. But in simulation the arm reach always the target position but the time needed to reach this position decrease when the value of the P controller increase.

This difference can be explained by the fact that on the real robot the P controller influences the torque while in simulation it influences the speed. After verification in the documentation of Webots and of the servos, the hypothesis has been confirmed.

It would have been possible to correct this on the cross-compilation tool in order to make the P controller influence on the speed instead of the torque. But it would have complicated the code, because this would have required to recalculate the speed at each time step while the control on the torque is automatically done by the servos. Furthermore a control on the torque is more adapted for this kind of robot. For all these reasons it has been decided to not make any adjustment on this, maybe in a future version of Webots a control on the torque will be available in simulation. In the meantime a difference will remain, but this difference is visible only for small values of the P controller, and because high values are commonly used, this difference should not cause any problem.
5.3.4 Torque feedback

This test has been done on the second servo of the shoulder (servo 3 in appendix F) like in the previous test, it consist of putting the arm down (figure 5.10 right, servo is at angle 0.8 rad) and slowly increase the angle of the servo until that the arm is up (figure 5.10 left, servo is at angle -0.8 rad), the torque feedback is continually recorded in function of the arm angle. The result of this test is visible on figure 5.12

![Graph showing torque feedback](image)

Figure 5.12: Result of the test of the torque feedback, torque feedback in function of the position of the second servo of the shoulder

The first try of this test is represented in blue for both simulation and real robot. We can see a symmetry between the two curves, indeed there was a problem of sign of the torque feedback on some of the servos on the real robot.

The result after correction of the cross-compilation tool is also shown on figure 5.12 in red. Now the two curves are similar, the torque feedback is more accurate in simulation than on the real robot but this is normal because there is no torque sensors on the servos, so it is not possible to have a good accuracy.
5.3.5 Max torque Available

Here again this test was made on the second servos of the shoulder (servo 3 in appendix F), it consists similarly to the previous ones of putting the arm down (figure 5.10 left, servo is at angle -0.8 rad), setting the MotorForce to 0 and then asking the servo to put the arm up (figure 5.10 right, servo is at angle 0.8 rad). The MotorForce is slowly increased and the position of the servo is recorded in function of the MotorForce. The result of this test is shown on figure 5.13.

![Figure 5.13: Result of the test of the maximum torque available, position of the servo in function of the MotorForce](image)

There is a small difference between the curve of the real robot and the one from the simulated robot, this can be caused by some small resistances of the servo not modeled in simulation or some masses or centers of mass of the arm not exactly correct in simulation. However the difference is sufficiently small for our application.
5.3.6 Apply torque

This test is very similar from the previous one, instead of setting the target position and increasing the maximum torque available when the arm is down, it was directly a torque in the upper direction that was applied and the resulting position of the arm was recorded. Result is shown in figure 5.14.

![Position of the arm in function of the force applied (real robot)](image)

![Position of the arm in function of the force applied (simulation)](image)

Figure 5.14: Result of the test of applying a torque to a servo, position of the servo in function of the torque.

Here again the difference between the two curves can be caused by some small resistances of the servo not modeled in simulation or some masses or centers of mass of the arm not exactly correct in simulation.
5.4 Gait

Because when enabling the balance in simulation we had strange behavior, we decided to also compare the gait between simulation and the real robot. For this, the values of the gyroscope are recorded while the robot is walking. The gyroscope has been chosen because the balance algorithm is based on the values of the axes X and Y of the gyroscope.

Like it can be seen on figure 5.15 there is a huge difference between simulation and real robot:

![Figure 5.15: Comparison of the values of the gyro while walking between simulation and real robot.](image)

To find the origin of this, a lot of different tests have been made. They are all explained and commented in appendix G. But unfortunately none of these tests gave us the reason of this difference or any way to correct it.

5.4.1 Isolation of the balance algorithm

In order to find if the problem of the mismatching of the values coming from the gyroscope while walking were due to the walking algorithm or to the balance algorithm, the balance algorithm of the walk has been added on the test controller of gyroscope (move the four servos of the hip in order to make the body of the robot swing, like for the test of the gyroscope). The result of this test can be seen in the following figures 5.16 and 5.17.
Figure 5.16: Test of the balance algorithm outside of the walking algorithm, values of the gyroscope in function of the time with and without balance (real robot).

On the real robot, when the balance algorithm is off (top of figure 5.16) we have as expected the same result as in figure 5.1 (bottom). But when the balance algorithm is off, the swing decrease and therefore also the values of the gyroscope, because the balance algorithm stabilizes the robot.

Figure 5.17: Test of the balance algorithm outside of the walking algorithm, values of the gyroscope in function of the time with and without balance (simulated robot).

On the simulated robot, the behavior is very similar to the real one. We can also conclude that the problem of the gait do not come from the implementation of the balance algorithm in simulation. The problem should come from intrinsic parameters of the simulation (physical engine, collisions, inertia, etc.) that influence the motion of the robot when walking and therefore distort the values measured by the gyroscope.

This resulting into problems for the lateral balance (front/back balance seems to works despite the strange values coming from the gyroscope), it has been decided to let the balance algorithm but to put the gains for the lateral balance (in the config.ini file) to 0 by default. In order that the lateral balance is unactivated by default, but if the user wants to try it, he can do it easily.
5.5 Results of the improvements

In this chapter a lot of modifications have been done on the simulated model in order to make it closer to the real robot. The simulated model after and before these corrections have been compared in order to evaluated the impact off all these modifications.

The first test concerns the center of mass of the body which has been moved (compared to the real one) and the surface of the feet which have been increased in order to make the robot more stable when playing motions. By putting the center of mass to its correct position, the robot was still unstable, but however the difference between the center of mass of the real robot and the one of the simulated model have been reduced of 50% and the robot is still stable for all the motions. For the surface of the feet, the original size could have been restored without any loss of stability. This is a proof that these modifications have improved the model and decreases the difference with the real robot.

The second test concerns the walking algorithm. The speed of the robot has been tested when walking normally and side wise. The results of this test are shown on figure 5.18 and 5.19.

![Figure 5.18: Test of the speed of the robot when walking.](image)

The figure 5.18 shows that the speed of the robot when walking has been a little bit increase with the new model of the robot.
Figure 5.19: Test of the speed of the robot when walking side wise.

On figure 5.19 we can see that the speed when walking side wise has been greatly improved by the new model of the robot.
6 Remote-control

For the remote-control, a small server has been done. This server runs on the robot, then Webots can connect to this small server and communicate with it. The cross-compilation tool previously created is then used in order to easily read sensors value or write command to the actuators when Webots sends such commands to the server.

On Webots side, a remote-control library has been made in order to allow running the DARwIn-OP model of Webots in mode remote-control. This library uses a wrapper to redefine most of the functions used to control the DARwIn-OP, it also implements the communication protocol with the robot and is in charge of the timing management.

The DARwIn-OP has been the first complex robot for which a Webots remote-control has been developed at Cyberbotics. The development of this remote-control has permitted to point out some problems of Webots when running in remote-control mode. For example the servos were not present in Webots in mode remote control, and also the keyboard had some strange behaviors. Thanks to the help of Fabien Roher all these little errors have been fixed in Webots source code.

On the client side (Webots side) the communication is implemented using Qt QTcpSocket. Qt been extensively used and distributed with Webots, using Qt for the communication is the easier way and also ensures to be cross platform. On the server side (robot side) Qt could not have been used because it is not installed by default on the robot and here again in order to make as less modifications as possible on the robot, it has been decided to not install it. Instead of QTcpSocket, the basic Linux sockets are used on the robot, because here the use of a platform dependent implementation is not a problem.

The communication protocol being the bigger part of the remote-control, its working will now be explained in more detail.

6.1 Communication protocol

The communication protocol between the computer and the robot for the remote-control will now be explained. Two kinds of packet are send between the robot and the computer, the first one the packet computer->robot is used to send command to the actuators of the robot (leds and servos) and to ask for sensors values. The second one robot->computer is send by the robot in response to the first one and contains only the values of the sensors (gyro, accelerometer, camera and servo feedbacks).
6.1.1 Packet computer -> robot

This packet is sent to the robot by the computer and is constituted exclusively of chars. Its configuration is represented on figure 6.1.

![Figure 6.1: Structure of the packet computer->robot for remote-control. The number of chars used to stock each part is indicated below by the number of circles. The parts in red are mandatory and therefore present in each packet. The parts in orange are present only if the sensor is requested. And finally the parts in green are also present only if the sensor/actuator is requested but they can be present several times.](image)

The packet starts by a \textit{W} in order to indicate to the robot that this packet is a valid packet sent by Webots (any packet non-starting by \textit{W} will be ignored by the remote-server).

Then the characters \textit{A G C} ask to the remote-server to send respectively, the values of the accelerometer, the gyroscope and the image of the camera. Of course they are asked only if needed by Webots (the sensor is requested).

Then for each leds whose state has been changed since last time step, the character \textit{L} is added, this character is followed by the corresponding index of the led (coded on only one char) and the state (coded on 3 chars, because led colors are coded on 24 bits in Webots). Then for each servo requested (both for feedback or command), a \textit{S} is added followed by the servo number and the following characters:

\begin{itemize}
    \item \textit{p} followed by the value if a command on the servo position is requested.
    \item \textit{v} followed by the value if a command on the servo velocity is requested.
    \item \textit{a} followed by the value if a command on the servo acceleration is requested.
    \item \textit{m} followed by the value if a command on the servo motor force is requested.
    \item \textit{c} followed by the value if a command on the servo control P is requested.
    \item \textit{f} followed by the value if a command on the servo force is requested.
    \item \textit{P} if the position feedback is requested.
    \item \textit{F} if the force feedback is requested.
\end{itemize}

Finally the packet is ended with a \textbf{\texttt{\textbackslash 0}}.
6.1.2 Packet robot -> computer

This packet is sent to the computer by the robot and is here again constituted exclusively of chars. Its configuration is represented on figure 6.2.

Figure 6.2: Structure of the packet robot->computer for remote-control. The number of chars used to stock each part is indicated below by the number of circles. The parts in red are mandatory and therefore present in each packet. The parts in orange are present only if the sensor is requested. And finally the parts in green are also present only if the sensor is requested but they can be present several times.

Here again, the packet starts by a W, for the same reasons than before. Then the size of the packet is encoded on four chars (size of an integer).

Then if the accelerometer is requested, the values of the three axes are added to the packet (each value being encoded on four chars, because the output values of the accelerometer are integers). The exact same mechanism is used for the gyroscope. Then if the camera is requested, the size of the image buffer is encoded on four chars, followed by the entire image buffer. Finally the values of the position and/or force feedback are added for each servo for which it is requested. And here again the packet is ended by \(0\).

6.1.3 Servo

In order to speed up the communication speed by decreasing the size of the packets send between the robot and the computer, the values (both of the commands and the results of the servos feedbacks) are transformed to integers before to be sent and then re-transformed in doubles on the other side, because an integer take half of the space than a double. In the following, the conversion for each command/result is explained, this conversion has been optimized in order to lose as less as possible of precision.

**Position command** The position is converted from radian to encoder units (which is of type integer), so there is absolutely no loss of precision.

**Velocity command** The speed is converted from rad/s to motor speed units (which is 0.053rpm), so here again there is absolutely no loss of precision.

**Acceleration command** The acceleration is multiplied by 100'000 in order to have a resolution of \(10^{-6} \text{ rad/s}^2\).

**MotorForce command** The force is convert from N/m to motor torque units (which is 0.0025N/m), so here again there is absolutely no loss of precision.

**ControlP command** The controlP is multiplied by 1000 in order to have a resolution of \(10^{-3}\) (which is better than the resolution on the real robot).

**Force command** The force is converted from N/m to motor torque units (which is 0.0025N/m).
**Position feedback** The position is converted from radian to encoder unit (which are of type integer).

**Force feedback** The force is converted from N/m to motor torque units (which is 0.0025N/m).

### 6.1.4 Camera

In remote-control, the camera supported the following resolution (they are smaller than in cross-compilation in order to not slow down the communication speed):

<table>
<thead>
<tr>
<th>Width [pixel]</th>
<th>Height [pixel]</th>
</tr>
</thead>
<tbody>
<tr>
<td>320</td>
<td>240</td>
</tr>
<tr>
<td>160</td>
<td>120</td>
</tr>
<tr>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>40</td>
<td>60</td>
</tr>
<tr>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>20</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 6.1: Camera resolutions supported in remote-control.

Any combination of height and width is also available; there is no need to choose resolutions from the same line. When starting the remote-server, the height and width zoom factors (relative to a resolution of 320x240) must be passed as arguments. Then the robot takes the image at a size of 320x240 and use the height and width zoom factors to put in the packet only the pixels needed in order to construct an image of the desired resolution.
6.2 Camera speed improvements

The image buffer of the camera being the bigger part of the packet robot->computer and especially for high resolutions, the packet become so large that it slows down the communication and the time step of 16ms could not be respected anymore. In order to decrease a little bit the size of the buffer, it has been tried to send only the BGR components of the buffer and to skip the A component, then on Webots side the buffer can easily be reconstructed because the A component is always 1.

<table>
<thead>
<tr>
<th>Resolution</th>
<th>160x120</th>
<th>320x240</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time step</td>
<td>image size</td>
</tr>
<tr>
<td>BGRA</td>
<td>24.81 ms</td>
<td>76.8k</td>
</tr>
<tr>
<td>BGR</td>
<td>18.67 ms</td>
<td>57.6k</td>
</tr>
</tbody>
</table>

Table 6.2: Comparison of the image size and of the time step for two different resolutions. The time step is a mean over 2000 time steps. In order to have always the same image for the comparison, the camera image buffer is redirected from an image file.

Like we can see on table 6.2 by removing the A component, the time step is reduced of approximately 0.75% from the previous one. This can be explained by the fact that 0.25% of the data are removed. By doing this, the communication speed speeds up, but remains still far from the time step of 16ms for high resolutions.

In order to increase again the communication speed, the image buffer is compressed into jpeg format using the libjpeg. In order to find the compromise between the time spent for compression/decompression and the time spent to send the data, several tests have been made with different camera resolutions and compression qualities. The results of these tests are shown in table 6.3.

<table>
<thead>
<tr>
<th>Resolution</th>
<th>160x120</th>
<th>320x240</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time step</td>
<td>compression time</td>
</tr>
<tr>
<td>Compression at 100%</td>
<td>16.41 ms</td>
<td>7.89 ms</td>
</tr>
<tr>
<td>Compression at 80%</td>
<td>12.45 ms</td>
<td>5.33 ms</td>
</tr>
<tr>
<td>Compression at 65%</td>
<td>12.50 ms</td>
<td>5.23 ms</td>
</tr>
<tr>
<td>Compression at 50%</td>
<td>12.74 ms</td>
<td>4.98 ms</td>
</tr>
<tr>
<td>Compression at 100%</td>
<td>56.18 ms</td>
<td>23.24 ms</td>
</tr>
<tr>
<td>Compression at 80%</td>
<td>29.65 ms</td>
<td>17.12 ms</td>
</tr>
<tr>
<td>Compression at 65%</td>
<td>23.42 ms</td>
<td>17.14 ms</td>
</tr>
<tr>
<td>Compression at 50%</td>
<td>22.78 ms</td>
<td>16.64 ms</td>
</tr>
</tbody>
</table>

Table 6.3: Comparison of the image size and of the time step for different quality of the jpeg compression. Each time is a mean over 2000 time step. In order to have always the same image for the comparison, the camera image buffer is redirected from an image file.
The results are now close from the time step of 16ms required, but most of the time (especially at high resolutions) is spent for the compression of the image buffer on the robot. In order to speed up the compression time on the robot, a special library optimized for fast compression and called *libjpeg-turbo* has been used. The results with this new library are shown in table 6.4.

<table>
<thead>
<tr>
<th>Image Size</th>
<th>Compression at 100%</th>
<th>Compression at 80%</th>
<th>Compression at 65%</th>
<th>Compression at 50%</th>
</tr>
</thead>
<tbody>
<tr>
<td>160x120</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time Step</td>
<td>13.1 ms</td>
<td>9.6 ms</td>
<td>10.2 ms</td>
<td>10.3 ms</td>
</tr>
<tr>
<td>Compression Time</td>
<td>4.76 ms</td>
<td>2.12 ms</td>
<td>2.49 ms</td>
<td>2.24 ms</td>
</tr>
<tr>
<td>Decompression Time</td>
<td>1.12 ms</td>
<td>0.75 ms</td>
<td>0.78 ms</td>
<td>0.82 ms</td>
</tr>
<tr>
<td>Image Size</td>
<td>20k</td>
<td>5.6k</td>
<td>4.3k</td>
<td>3.6k</td>
</tr>
<tr>
<td>320x240</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time Step</td>
<td>42.08 ms</td>
<td>17.96 ms</td>
<td>15.30 ms</td>
<td>14.40 ms</td>
</tr>
<tr>
<td>Compression Time</td>
<td>11.88 ms</td>
<td>8.19 ms</td>
<td>7.72 ms</td>
<td>7.30 ms</td>
</tr>
<tr>
<td>Decompression Time</td>
<td>4.19 ms</td>
<td>2.85 ms</td>
<td>2.93 ms</td>
<td>2.81 ms</td>
</tr>
<tr>
<td>Image Size</td>
<td>63.2k</td>
<td>15.1k</td>
<td>11.3k</td>
<td>9.3k</td>
</tr>
</tbody>
</table>

Table 6.4: Comparison of the image size and of the time step for different quality of the jpeg compression using the library libjpeg-turbo for the image compression on the robot. Each time is a mean over 2000 time steps. In order to have always the same image for the comparison, the camera image buffer is redirected from an image file.

Finally the time step of 16ms is respected. A compression quality of 65% is used for image bigger than 160x120 and a compression quality of 80% is used for image of 160x120 and smaller.
7 Robot Window

A specific robot window has been made for the DARwIn-OP, this window has the following tabs:

**Accelerometer** This tab is used to view in real-time the values of the three axes of the accelerometer (both in simulation and remote-control).

**Gyro** This tab is used to view in real-time the values of the three axes of the gyroscope.

**Servo** This tab is used to view in real-time the current and target position of each of the 20 servos, the target position can also be changed thanks to sliders.

**Camera** This tab is used to view in real-time the image of the camera.

**Transfer** This tab is used to interact with the real robot (installation of Webots API on the robot, sending of controller thanks to the cross-compilation tool, starting remote-control mode), the tab is shown in figure 7.1.

![Figure 7.1: Tab Transfer of the robot window.](image)

The four first tabs are the same as in the generic robot window of Webots, and they can be easily reused thanks to the library `qt_utils` provided by Webots. But the last one is specific to the DARwIn-OP. This tab being the more interesting and also the only one made in this project, its operation will now be described in more details.
7.1 SSH communication

In order to communicate with the robot (send file, execute command, read standard and error outputs, etc.), it has been decided to use the SSH protocol. Indeed this protocol is very simple and still allows to do all the necessary actions and furthermore, it is already present on the robot. This allows us to start communicating with the robot without having to previously install anything on the robot (no need of user operation). On Webots side, several libraries has been investigated in order to implement the SSH protocol.

The first idea was to use an extension of Qt creator called Utils which fully integrates the SSH protocol, because Qt is already distributed with Webots. But this extension is very large and because it is an extension, it was not included in the part of Qt distributed with Webots, and adding it would have added a lot of files and also greatly increased the size of Webots. For this reason it has been decided to give up the use of this library and investigate for another one lighter.

It has finally been decided to use the library libssh. This library is very light (only about 25K-30K lines of code), and even if it is not as simple to use as some other bigger libraries (as Utils for example), it implements all the functionalities needed for the project (SSH, SFTP, authentication, channels, etc.).

Furthermore, libssh is already present on most of the Linux and Mac platforms. On Windows, this library is not present by default, and none working compiled library for Windows has been found. The library has therefore been compiled from sources, but some modifications have been made on the library in order to compile it for Windows. By default the libssh has a dependency on openssl in order to crypt the communication, but this dependency was causing some trouble on Windows because some module were missing. It has therefore been replaced by libgcrypt which was easier to compile and use on Windows.

In order to have a clean code and to simplify the use of this library, a class called SSH has been created. This class implement all the functionalities needed for this project like for example:

- Open an SSH connexion with another computer
- Management of secured connexion authentication
- Send files to another computer
- Read files from another computer
- Execute command on another computer
- Read standard and error output of another computer
- Start programs on another computer (even in graphic mode)
- etc.
7.2 Data compression

In order to compress the files before to send them to the robot the *libtar* library was first used. Indeed this library can compress multiple files in a tar archive, but it can also compress directory (including sub-directories) and add files to an existing library. Here again this library is very widespread on Linux and Mac platforms, but unfortunately it is not fully compatible with Windows.

The use of libtar has therefore been abandoned. Instead the *libzip* library has been chosen, this library has the advantage of been cross-platform, but it does not offer the possibility to compress entire directory but only files and it is also more complex to use. In order to simplify its use, a new class called ZIP has been created, this class use libzip and Qt in order to create a very simple interface which allows to compress files and also directories with the possibility to specify if directories must be compressed recursively or not (include sub-directories or not).
7.3 Cross-compilation

All the tab transfer is then implemented in a class called \textit{Transfer}. This class being large enough, only the key functionalities will be explained here thanks to diagrams (for more precision you can of course see the source code).

This following button is used to send a controller thanks to the cross-compilation tool.

Figure 7.2: Button used to send a controller to the real robot and run it thanks to the cross-compilation tool.

When this button is pressed, the followings steps will be executed in a new thread.
Both the Framework and the Wrapper have a version file, which consist of a simple text file in which the version is written in the format X.X.X. The SFTP channel is used to read these two files and if they are not present or if the version contained in Webots is more up to date, the Framework and/or Wrapper is automatically updated/installed.

In order to send all the files of the controller, the entire controller directory is compressed in an archive. This archive is created in the temporary directory of the operating system and has the following name `webots_darwin_ProcessPID_controller.zip`. The PID of the process is added in the name in order to avoid problems if several instances of Webots are used in the same time.
During the compilation of the controller, the outputs of the terminal are shown in the robot window, this is especially useful in case of compilation errors (furthermore the errors are displayed in red).

Before to start the controller, the controller check_start_position is executed in order to verify if the robot is in a stable position. This controller tests the position of all the servos of the legs and also the orientation of the robot thanks to the accelerometer. If the robot is not close to the stable position a window like the one from figure 7.4 informs the user and ask him what he want to do. This is a security in order to avoid starting the robot from any unknown position which can maybe damage the robot.

![Stability check](image)

Figure 7.4: This window informs the user that the robot is not in the stable position and ask him what he want to do.

Furthermore during the whole process of sending and compiling the controller a progress bar indicates the progression to the user. And as soon as a SSH connection has been successfully open, the connection settings are saved in order to reload them the next time the robot window is opened.

Then when the controller is running the button from figure 7.2 is replaced by the following one:

![Stop button](image)

Figure 7.5: Button used to stop a controller running on the real robot.

When this button is then pressed, the thread is stopped, the controller process is killed, the controller’s files are removed from the robot, the SSH connection is closed, and finally the button and status label are refreshed.
7.3.1 Updating the Framework

When the function for updating the Framework is called, the following steps are executed:

![Diagram of Framework update procedure]

Figure 7.6: Framework update procedure.

Like for the controller, the archive is placed in the temporary directory and contain the PID of the controller. At the end of the procedure, the wrapper version file is deleted. This is done in order to force the update and especially the recompilation of the wrapper.
7.3.2 Installing Webots API

When the function for installing Webots API is called, the following steps are executed:

- Creation of the archive of the Wrapper files
- Send the archive by SFTP to the robot
- Deletion of the archive
- Extraction of the files from the archive and displacement of them in the directory /darwin/Linux/project/webots
- Compilation of the controller check_start_position
- Compilation of the remote-control server
- Compilation of the Wrapper library
- Compilation of the managers library
- Creation of a backup of the file /etc/rc.local
- Replacement of the file /etc/rc.local by the one from the Wrapper

Figure 7.7: Wrapper installation procedure.

Here again the archive is placed in the temporary directory and contain the PID of the controller. At the end of the procedure, the file /etc/rc.local is backed-up and replace by a custom one. This file is in charge of automatically starting the demo program of Robotis at the startup of the robot, by replacing it by a custom one, the demo program is not anymore started and furthermore the default program of the wrapper (if any) is started instead.
7.3.3 Uninstalling Webots API

It is also possible to completely uninstall Webots API from the robot by the use of the following button:

![Figure 7.8: Button used to remove Webots API from the robot.](image)

When pressing this button, the following warning window appears:

![Figure 7.9: This window appears when the user want to uninstall Webots API from the robot.](image)

If the user answer yes, the following steps are executed:

- Creation of a new SSH connection.
- Restoration of the default file rc.local in order to restore the demo program as default program when starting the robot.
- Deletion of the entire directory /darwin/Linux/project/webots.
- Closure of the SSH connection.
7.4 Remote-control

This following button is used to start the remote-control mode in Webots and the remote-server on the robot:

![Remote-control button](image)

Figure 7.10: Button used to start remote-control.

When this button is pressed, the following steps are executed:

![Flowchart](image)

Figure 7.11: Starting remote-control procedure.

During the whole procedure of starting the remote-control server, a small window (see figure 7.12) ask the user to wait until remote-control has been started and also give an estimation of the remaining time. This window also sticks the main robot window in
order to deny every other actions. If the Cancel button is pressed, the procedure is then stopped and the connection closed.

Figure 7.12: Small window asking the user to wait until remote-control server has been started.

Then here again the button 7.10 is replaced by the button from figure 7.5. When pressing on this button the remote-control server is stopped, the SSH connection is closed, and the simulation is put again in mode \texttt{WB\_MODE\_SIMULATION}.
8 User aspect

The result of this project been very user-oriented, this project contains also a large part of communication with users, documentation writing, creation of examples of use, creation of advertisement, etc.

It has also been taken care that it will works on every platforms. The project has been mainly done on Linux but have also been tested on other platforms and some modifications/improvements have also been made in order to support these other platforms. The goal was to be compatible with all the platforms supported by Webots. Currently it has been tested with success on Linux, Mac and Windows, for both 32bits and 64bits version.

All along the duration of the project each times a new version of Webots was released the last modifications about the DARwIn-OP were integrated in Webots, this has permitted to have feedback from users.

8.1 Interactions with users

The project been on Github, the source code can be seen by everyone and is also available at any time (even the part still not integrated in Webots). Thanks to this, early in the development stage of the project some users started using it and their number growth all along the project.

This add a lot of work because some users had a lot of questions, but it was also a good means of testing and debugging the project, indeed most of the errors were quickly reported by users. Some users even bring some ideas on how to improve/correct some parts of the project.

8.2 Examples

In order to give a starting point to new users, different simple examples have been made and are now distributed with Webots. The examples are the following:

Symmetry  This example is the most simple, it explains how to use the servos and interact with the mouse on the simulation.

Visual Tracking  This example illustrates the use of the camera (and in the same time of the vision manager) and of the rgb leds.

Walk  This example illustrates the use of the gait and motion manager and also how to use the keyboard to interact with the simulation.

Soccer  This is the more complex example, it consists in a simple soccer player and uses almost all the sensors and actuators of the DARwIn-OP.
8.3 User guide

In order help the user to use the DARwIn-OP with Webots, a user guide has been written. This user guide is very complete and explains all the aspect of this project including:

- The simulation model and all of its devices.
- How to use the managers.
- Explanations of the example and ideas of how to improve them.
- The different tabs of the robot window.
- The cross-compilation tool.
- The remote-control tool.

This guide is concentrated mainly on the DARwIn-OP with Webots and not on Webots in general and must therefore be used in conjunction with Webots user guide. This user guide can be seen in appendix H.

8.4 Demonstration video

In order to make some advertisement, a demonstration video have been made, this video illustrates all the aspect of the project (simulation, remote-control and cross-compilation) and is visible on the CD-ROM.
9 Conclusion

The DARwIn-OP is now one of the more complex and best calibrated robots of Webots. During its calibration some mismatching between the real and simulated world has been point out, but most of them have been corrected (or at least improved) this has also permitted to improve not only the model of the DARwIn-OP but also to add/correct some part of the simulator. Furthermore the possibility to transfer the controller from simulation to the real robot (thanks to cross-compilation or remote-control) proves a realistic behavior of the robot in simulation, and demonstrates a high level of realism of the simulator.

9.1 Extension of the Framework possibilities

The use of this project and Webots extend a lot the basic possibilities provided by the Framework. Additionally of the possibilities of simulation offer by Webots, the cross-compilation tool gives an alternative way of programming the robot using Webots API instead of Robotis API. Using the cross-compilation tools gives also several advantages from the use of the Framework, indeed all the functionalities offered by the Framework are also available in the cross-compilation tool (some of them are even improved) and other useful possibilities has been added (text-to-speech functions and use the keyboards to interact with the robot for example).

Furthermore Webots API is well documented (which is not the case of Robotis API), used by a fairly large community (possibility to ask for other users help on forum for example) and is common to every robot (at least in Webots simulations). Using a computer for programming and not do it directly on the robot can also be much more convenient for the user. Finally the Framework do not provided any way of doing remote-control, which is very useful in order to make quick tests or to take advantage of the computing power of a computer.

9.2 Reuse in other projects

Some parts of this project can simply be reused in other projects. For example the SSH class of the robot window can be used in any other projects who need to use the ssh protocol without any problem, because it has not any dependency on Webots API but only on libssh. The ZIP class too, because it only has dependency on libzip and Qt, and provided a very easy way of compressing files and folders.

Furthermore the remote-control and cross-compilation tools give a very good example and starting point for developing a new remote-control or cross-compilation tool for any other robots present in Webots. The DARwIn-OP running under Linux, these ones and also the robot window can be adapted to any other robots using Linux without making too much modifications. Moreover Robotis sells also all the electronic hardware of the DARwIn-OP for using it in order to make other robots. Any robot made with this hardware could use with a very few modifications the remote-control, cross-compilation and robot window developed in this project.
9.3 Personal opinion

I found very interesting to work in a real company in which the way of thinking and working is not always the same as in course or other projects in laboratories at the EPFL. Working in a company had also allowed me to familiarize myself with aspects of the work that were usually not present in previous projects made at the EPFL, like for example contact with clients, promotion of the project, or even the way of thinking user interface.

One other aspect that differs from previous projects, done at the EPFL, was the fact that often at the EPFL we try to do one prototype which should works on one particular system and then maybe if it works well try to adapt it to other systems. But in company this is not possible, it has to work on every systems (every operating systems for this project) and all possible cases of use (it should for example handle the case were several instances of Webots are running and trying to communicate with the robot in the same time, it should be able to works if the robot is connected by wifi or ethernet, etc.). For example the robot window was working fine on Linux and Mac, but I spent several weeks in order to adapt it for Windows.

It was also interesting for me, coming from microengineering, to work in a company mainly based on computer science. At the beginning it was sometimes a little bit hard for me to adapt myself to this field. But after it appears very complementary and because of my master in robotics and autonomous system I already had good knowledge about robotic.
References

Here are referenced all the documents that help me during this project and/or can be interesting for the reader in order to go further.

Webots User Guide  This guide is very useful in order to have a quick overview and understanding of Webots, it can be found at the following address:
  http://www.cyberbotics.com/guide

Webots Reference Manual  This manual gives a deeper explanation of the use of Webots, especially on the API, it can be found at the following address:
  http://www.cyberbotics.com/reference

DARwIn-OP Manual  This is the manual of the robot, it mainly explains how to use the robot and especially the Framework, it can be found at the following address:

DARwIn-OP documentation  All the datasheets of the robot and also the source code of the robot are available at the following address:
  http://sourceforge.net/projects/darwinop

Github depository  The depository of this project is available at this address:
  https://github.com/darwinop/webots-cross-compilation

Qt documentation  Qt being extensively used in the robot window and the remote-control, its great documentation was very helpful:
  http://qt-project.org/doc

Xlib documentation  For the creation of the small window and capture of keyboard inputs on the robot, the following websites have been very helpful:
  http://user.xmission.com/~georgeps/documentation/tutorials/Xlib_Beginner.html
  http://tronche.com/gui/x/xlib

Libssh documentation  For the use of libssh, the following website has been used in order to get information about the API:
  http://api.libssh.org/master/modules.html

Libzip documentation  For the use of libzip, the following website has been used in order to get information about the API:
  http://www.nih.at/libzip

Libjpeg documentation  For the jpeg compression and decompression the following websites have been used:
  http://apodeline.free.fr/DOC/libjpeg/libjpeg.html
  http://www.libjpeg-turbo.org/
## A Servos limits

<table>
<thead>
<tr>
<th>ID</th>
<th>Lower limit [encoder]</th>
<th>Upper limit [encoder]</th>
<th>Lower Limit [rad]</th>
<th>Upper limit [rad]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>191</td>
<td>4096</td>
<td>-2.84860</td>
<td>3.14159</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>3905</td>
<td>-3.14159</td>
<td>2.84860</td>
</tr>
<tr>
<td>3</td>
<td>1605</td>
<td>3548</td>
<td>-0.67955</td>
<td>2.30097</td>
</tr>
<tr>
<td>4</td>
<td>583</td>
<td>2549</td>
<td>-2.24728</td>
<td>0.76852</td>
</tr>
<tr>
<td>5</td>
<td>970</td>
<td>2804</td>
<td>-1.65363</td>
<td>1.15969</td>
</tr>
<tr>
<td>6</td>
<td>1278</td>
<td>3109</td>
<td>-1.18117</td>
<td>1.62755</td>
</tr>
<tr>
<td>7</td>
<td>470</td>
<td>2480</td>
<td>-2.42062</td>
<td>0.66268</td>
</tr>
<tr>
<td>8</td>
<td>1600</td>
<td>3680</td>
<td>-0.68722</td>
<td>2.50346</td>
</tr>
<tr>
<td>9</td>
<td>1390</td>
<td>2708</td>
<td>-1.00936</td>
<td>1.01243</td>
</tr>
<tr>
<td>10</td>
<td>1400</td>
<td>2650</td>
<td>-0.99402</td>
<td>0.92346</td>
</tr>
<tr>
<td>11</td>
<td>889</td>
<td>2340</td>
<td>-1.77788</td>
<td>0.44792</td>
</tr>
<tr>
<td>12</td>
<td>1724</td>
<td>3145</td>
<td>-0.49701</td>
<td>1.68278</td>
</tr>
<tr>
<td>13</td>
<td>2038</td>
<td>3513</td>
<td>-0.01534</td>
<td>2.24728</td>
</tr>
<tr>
<td>14</td>
<td>571</td>
<td>2067</td>
<td>-2.26567</td>
<td>0.02915</td>
</tr>
<tr>
<td>15</td>
<td>1239</td>
<td>2947</td>
<td>1.24099</td>
<td>1.37905</td>
</tr>
<tr>
<td>16</td>
<td>1142</td>
<td>2845</td>
<td>-1.38978</td>
<td>1.22258</td>
</tr>
<tr>
<td>17</td>
<td>1603</td>
<td>2728</td>
<td>-0.68262</td>
<td>1.04311</td>
</tr>
<tr>
<td>18</td>
<td>1385</td>
<td>2435</td>
<td>-1.01703</td>
<td>0.59365</td>
</tr>
<tr>
<td>19</td>
<td>790</td>
<td>3150</td>
<td>-1.92975</td>
<td>1.69045</td>
</tr>
<tr>
<td>20</td>
<td>1815</td>
<td>2660</td>
<td>-0.35742</td>
<td>0.93879</td>
</tr>
</tbody>
</table>

Table A.1: Limits of each servos before to collide the robot.
## B Masses

<table>
<thead>
<tr>
<th>Name</th>
<th>Number</th>
<th>Mass [gr]</th>
<th>Mass total [gr]</th>
<th>Percent of total mass</th>
</tr>
</thead>
<tbody>
<tr>
<td>Body</td>
<td>1</td>
<td>976</td>
<td>976</td>
<td>33.4</td>
</tr>
<tr>
<td>Head</td>
<td>1</td>
<td>158</td>
<td>158</td>
<td>5.4</td>
</tr>
<tr>
<td>Neck</td>
<td>1</td>
<td>24</td>
<td>24</td>
<td>0.8</td>
</tr>
<tr>
<td>Shoulder</td>
<td>2</td>
<td>26</td>
<td>52</td>
<td>1.8</td>
</tr>
<tr>
<td>Upper Arm</td>
<td>2</td>
<td>168</td>
<td>336</td>
<td>11.5</td>
</tr>
<tr>
<td>Lower Arm</td>
<td>2</td>
<td>59</td>
<td>118</td>
<td>4.0</td>
</tr>
<tr>
<td>Upper hip</td>
<td>2</td>
<td>27</td>
<td>54</td>
<td>1.8</td>
</tr>
<tr>
<td>Lower hip</td>
<td>2</td>
<td>167</td>
<td>334</td>
<td>11.5</td>
</tr>
<tr>
<td>Thigh</td>
<td>2</td>
<td>119</td>
<td>238</td>
<td>8.1</td>
</tr>
<tr>
<td>Tibia</td>
<td>2</td>
<td>70</td>
<td>140</td>
<td>4.8</td>
</tr>
<tr>
<td>Ankle</td>
<td>2</td>
<td>167</td>
<td>334</td>
<td>11.5</td>
</tr>
<tr>
<td>Foot</td>
<td>2</td>
<td>79</td>
<td>158</td>
<td>5.4</td>
</tr>
</tbody>
</table>

Table B.1: Mass of each parts of the DARwIn-OP.
C Example of configuration (.ini) files

This is a typical example of configuration file (.ini):

```
[Walking Config]
x_offset = -10.0;
y_offset = 5.0;
z_offset = 20.0;
roll_offset = 0.0;
pitch_offset = 0.0;
yaw_offset = 0.0;
hip_pitch_offset = 13.0;
period_time = 600.0;
dsp_ratio = 0.1;
step_forward_back_ratio = 0.28;
foot_height = 40.0;
swing_right_left = 20.0;
swing_top_down = 5.0;
pelvis_offset = 3.0;
arm_swing_gain = 1.5;
balance_knee_gain = 0.3;
balance_ankle_pitch_gain = 0.9;
balance_hip_roll_gain = 0.0;
balance_ankle_roll_gain = 0.0;

[Robot Config]
time_step = 16.0;
camera_width = 320.0;
camera_height = 240.0;
```

This file must be in the same directory that the controller.
D Walking parameters

This appendix explains all the parameters that can be set in the configuration file (.ini) to tune the gait.

**X offset** is the offset of the feet in the X direction. Unit is in millimeter.

![Figure D.1: Walking : x offset parameters](image)

**Y offset** is the offset of the feet in the Y direction. Unit is in millimeter.

![Figure D.2: Walking : y offset parameters](image)
Z offset is the offset of the feet in the Z direction. Unit is in millimeter.

Figure D.3: Walking: z offset parameters

Roll offset is the angle offset at the feet along X axis. Unit is in degree.

Figure D.4: Walking: roll offset parameters
**Pitch offset** is the angle offset at the feet along Y axis. Unit is in degree.

![Figure D.5: Walking: pitch offset parameters](image)

**Yaw offset** is the angle offset of the leg along Z axis. Unit is in degree.

![Figure D.6: Walking: yaw offset parameters](image)

**Hip pitch offset** is the tilt of DARwIn-OP’s body. It uses a special unit of the motor corresponding to 2.85 degree.

![Figure D.7: Walking: hip pitch offset parameters](image)
**Period time** is the time required for DAwrIn-Op to complete two full steps (left and right foot). Unit is in millisecond.

![Diagram showing period time parameters]

**Figure D.8: Walking : period time parameters**

**DSP ratio** is the ratio between the time when both feet are on the ground to only one foot (either left or right) is on the ground.

![Diagram showing DSP ratio parameters]

**Figure D.9: Walking : dsp ratio parameters**
**Step forward back ratio** is the differential distance according to X direction, between DARwIn-OP’s left and right foot during walk. Unit is in millimeter.

![Illustration of Step forward back ratio](image1)

**Figure D.10: Walking : step forward back ratio parameters**

**Foot height** is the maximum height of the foot during the step. Unit is in millimeter.

![Illustration of Foot height](image2)

**Figure D.11: Walking : foot height parameters**

**Swing right left** is the left and right Swaying of DARwIn-OP’s body during walking. Unit is in millimeter.

![Illustration of Swing right left](image3)

**Figure D.12: Walking : swing right left parameters**
**Swing top down** is the up and down swaying of DARwIn-OP’s body during walking. Unit is in millimeter.

![Swing top down parameters](image1)

**Pelvis offset** is angle offset at the pelvis along X axis. It uses a special unit of the motor corresponding to 2.85 degree.

![Pelvis offset parameters](image2)

**Arm swing gain** is the gain that influences the movement of the arm during walking.

**Balance knee gain** is the gain at the knee level for the front/back balance

**Balance ankle pitch gain** is the gain at the ankle level for the front/back balance.

**Balance hip roll gain** is the gain at the hip level for the lateral balance. Since the lateral balance does not work very well in simulation, we recommend you to set this parameter to 0.

**Balance ankle roll gain** is the gain at the ankle level for the lateral balance. Since the lateral balance does not work very well in simulation, we recommend you to set this parameter to 0.
## Motions Files

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Description</th>
<th>Recommended initial position</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ini</td>
<td>Move to standing position</td>
<td>Standing up</td>
</tr>
<tr>
<td>2</td>
<td>OK</td>
<td>Nods head</td>
<td>Standing up</td>
</tr>
<tr>
<td>3</td>
<td>no</td>
<td>Shakes head</td>
<td>Standing up</td>
</tr>
<tr>
<td>4</td>
<td>hi</td>
<td>Tilts forward</td>
<td>Standing up</td>
</tr>
<tr>
<td>6</td>
<td>talk1</td>
<td>Holds out his hand</td>
<td>Standing up</td>
</tr>
<tr>
<td>9</td>
<td>walkready</td>
<td>Prepares to walk</td>
<td>Standing up</td>
</tr>
<tr>
<td>10</td>
<td>f up</td>
<td>Gets up</td>
<td>Lying face against the ground</td>
</tr>
<tr>
<td>11</td>
<td>b up</td>
<td>Gets up</td>
<td>Lying back on the ground</td>
</tr>
<tr>
<td>12</td>
<td>rk</td>
<td>Right shoot</td>
<td>Standing up</td>
</tr>
<tr>
<td>13</td>
<td>lk</td>
<td>Left shoot</td>
<td>Standing up</td>
</tr>
<tr>
<td>15</td>
<td>sit down</td>
<td>Sits</td>
<td>Standing up</td>
</tr>
<tr>
<td>16</td>
<td>stand up</td>
<td>Stands up</td>
<td>Seated</td>
</tr>
<tr>
<td>17</td>
<td>mull1</td>
<td>Gets balanced on the head</td>
<td>Standing up</td>
</tr>
<tr>
<td>23</td>
<td>d1</td>
<td>Does yes with the arm</td>
<td>Standing up</td>
</tr>
<tr>
<td>24</td>
<td>d2</td>
<td>Applaud</td>
<td>Standing up</td>
</tr>
<tr>
<td>27</td>
<td>d3</td>
<td>Does yes with the arm and head</td>
<td>Standing up</td>
</tr>
<tr>
<td>29</td>
<td>talk2</td>
<td>Holds out his hand</td>
<td>Standing up</td>
</tr>
<tr>
<td>31</td>
<td>d4</td>
<td>Stretches in front and rear</td>
<td>Standing up</td>
</tr>
<tr>
<td>38</td>
<td>d2</td>
<td>Wave with the hand</td>
<td>Standing up</td>
</tr>
<tr>
<td>41</td>
<td>talk2</td>
<td>Presents himself</td>
<td>Standing up</td>
</tr>
<tr>
<td>54</td>
<td>int</td>
<td>Applaud louder</td>
<td>Standing up</td>
</tr>
<tr>
<td>57</td>
<td>int</td>
<td>Applaud</td>
<td>Standing up</td>
</tr>
<tr>
<td>70</td>
<td>rPASS</td>
<td>Performs a pass with the right foot</td>
<td>Standing up</td>
</tr>
<tr>
<td>71</td>
<td>lPASS</td>
<td>Performs a pass with the left foot</td>
<td>Standing up</td>
</tr>
<tr>
<td>90</td>
<td>lie down</td>
<td>Lies on the front</td>
<td>Standing up</td>
</tr>
<tr>
<td>91</td>
<td>lie up</td>
<td>Lies on the back</td>
<td>Standing up</td>
</tr>
<tr>
<td>237</td>
<td>sitdown</td>
<td>Jumps up and down</td>
<td>Standing up</td>
</tr>
<tr>
<td>239</td>
<td>sitdown</td>
<td>Jumps up and down quickly</td>
<td>Standing up</td>
</tr>
</tbody>
</table>

Table E.1: Motions stored in the motion files.

This are the defaults motions stored in the motion files `motion_4096.bin` and `motion_1024.bin`. The two files store the same motions, but for different angular resolutions of the servos. The resolution 4096, is the best one and the more up to date, for this reason the file `motion_4096.bin` has been privileged during this project.

It is also possible to add custom motions by using the tools *Action Editor* provided by Robotis.

---

[^E1]: More informations about this tool at: [www.support.robotis.com/ko/product/darwin-op/development/tools/action_editor.htm](http://www.support.robotis.com/ko/product/darwin-op/development/tools/action_editor.htm)
IDs of the servos
G Walking tests

This appendix gather all the tests related to the walking algorithm. All these tests have been made in order to find the origin of the strange values returned by the gyroscope while walking. Since the balance algorithm of the walk is based on the values returned by the gyroscope while walking, these strange values have a bad effect on the walk when the balance is enabled.

The goal of these tests was to find the origin of the mismatching of the values returned by the gyroscope while walking between the real robot and the simulated one. For recall the figure G.1 which was already present in section 5.4 illustrates this mismatching.

![Figure G.1: Comparison of the values of the gyroscope while walking between simulation and real robot.](image)

G.1 Accelerometer

The values of the accelerometer have also been recorded while the robot was walking, the results are represented in figure G.2.

![Figure G.2: Comparison of the values of the gyroscope and the accelerometer while walking, simulation (left) and real robot (right).](image)

On figure G.2 the moments where the robot has the two feet on the ground are represented by the horizontal lines. In simulation the values of the accelerometer seems less stable, there is more shocks.
G.2 Ground stiffness

In the last test it appears that there is more shocks in the simulation. In order to try to decrease these shocks and to see if it has an impact on the gait, the stiffness of the ground and of the robot has been decreased.

Different values of stiffness have been tested without any encouraging result. In figure G.3 the result of this test is shown with the smaller stiffness acceptable before to lose the stability of the robot.

G.3 Inertia

In order to see if the walk is influenced by some wrong inertia matrixes, the inertia matrixes of the proto have been removed. Indeed if the inertia matrixes are not set, Webots calculate them itself from the bounding box. The inertia matrix calculated are not supposed to be as precise as the one coming from the documentation but are enough good to see if the inertia has a great impact on the gait.
Figure G.4: Values of the gyroscope and accelerometer while walking without setting the inertia matrix.

The result from figure G.4 are very similar from the one with the inertia matrix set. So the problem seems not to come from the inertia.

G.4 Foot size

In order to see if the shape or size of the foot has an impact on the values returned by the gyroscope while walking, the surface of the feet has been divided by two.

Figure G.5: Values of the gyroscope and accelerometer while walking with the surface of each foot divided by 4.

Here again despite the great change done at the feet, the curves of the gyroscope values are not greatly affected.
G.5 Balance gain

In order to test if the impact of the balance algorithm can be improved in simulation by adjusting the different gains influencing the algorithm, different tests have been run by increasing or decreasing these gains.

Figure G.6: Values of the gyroscope and accelerometer while walking when increasing the gains of the balance algorithm.

On figure G.6 the gains have been increased a lot. Here the curves change a little bit, but the gain have been increased a lot and the robot is not very stable at all.

G.6 Servos trajectory

In order to compare the trajectory of the servos of the legs, the position of each servos of one leg have been recorded while walking both in simulation and on the real robot.

Figure G.7: Values of the gyroscope and accelerometer and position of all the servos of one leg while walking.

On figure G.7 the balance algorithm seems to have a bigger impact on the trajectory of the servos of the real robot than on the simulated one. Indeed, on the real robot the sinusoidal of each servo is much more deformed. But it has been pointed in the previous test that increasing the gain for the balance does not resolve the problem.
G.7 Feet angle

Finally the last test concerns the angle that made the feet with the ground. In the documentation of the robot it is said that there is no offset angle and this seems to be logical. But in order to see if a small angle offset at the feet level can have a big impact on the gait, different offset angles have been tested.

Figure G.8: Values of the gyroscope and accelerometer while walking with different feet offset angles.

On figure G.8 small offset angles at the feet level seems to have a very small impact on the values returned by the gyroscope while the robot is walking, and bigger ones render the robot unstable.
H User guide

The following is the entire user guide. Some parts of this user guide are very similar from this report and other are more focused on users. This user guide is made in order to give a good overview of the DARwIn-OP with Webots to beginner user of Webots, but it still have to be used in conjunction with Webots user guide.
- Realistic dynamical model
- Cross-compilation and transfer
- Remote control
- Framework compatible for key functionalities
Foreword

This document will explain you how is it possible to program the DARwIn-OP robot using Webots. Webots allows to program both the virtual robot model and the real robot by crosscompiling programs, or by remote-controlling the robot.

In the first chapters, all the features of the simulation model of the DARwIn-op will be presented and the examples included in Webots will be explained.

Then, in the following chapters, the possibilities of interaction with the real robot (real-time sensors viewing, cross- compilation and controller installation) will be explained.

We hope that you will enjoy working with Webots, and that it will greatly simplify your work with the DARwIn-OP.
# Contents

1 DARwIn-OP .......................... 1

2 Simulation model .................... 2

3 Managers ............................ 5
   3.1 Gait Manager ........................ 5
   3.2 Motion Manager ...................... 6
      3.2.1 Motion Manager in mode step-by-step .......................... 7
   3.3 Vision Manager ...................... 8

4 Examples ............................ 10
   4.1 Symmetry ............................ 11
   4.2 VisualTracking ........................ 12
   4.3 Walk ................................. 13
   4.4 Soccer .............................. 14

5 Robot Window ....................... 15

6 Cross-compilation ................. 18
   6.1 Send a controller to the robot .................................. 19
   6.2 Permanently install a controller to the robot .................. 21
   6.3 Uninstall Webots files from the robot .......................... 21
   6.4 Dynamixel MX28 firmware ....................................... 21
   6.5 Using speaker ........................................... 22
   6.6 Using keyboard ......................................... 23

7 Remote control ..................... 24
   7.1 Camera resolution ........................................ 25
   7.2 Controller speed ....................................... 25

8 Known Bugs .......................... 26
   8.1 Lateral balance ......................................... 26
   8.2 Controller P ......................................... 26
   8.3 Text-to-speech warning message .............................. 26

9 Bibliography ....................... 27

A Walking parameters .......... I

B Motions files .................... VII

C Audio files available .......... VIII

D Voices available ............. IX
1 DARwIn-OP

The Darwin-op is an open source miniature humanoid robot platform with advanced computational power. The name DARwIn-OP means Dynamic Anthropomorphic Robot with Intelligence-Open Platform. It is developed and manufactured by ROBOTIS (a Korean robot manufacturer) in collaboration with the University of Pennsylvania.

The DARwIn-OP is mainly used by universities and research centers for educational and research purpose. It has a total of 20 degrees of freedoms:

- 2 in the head.
- 3 in each arm.
- 6 in each leg.

This robot is available at a fairly low price and is based on open source components (both hardware and software). It has been used in the RoboCup international competition with some success.

The DARwIn-OP robot has been fully integrated into Webots in collaboration with ROBOTIS. By using DARwIn-OP in conjunction with Webots you will have the following benefits compared to the use of ROBOTIS API directly on the real robot:

**Simulation**  You will be able to test your controller in simulation, without any risk of damaging the robot. You will also be able to run automatically a lot of different simulations in a very small amount of time (to tune up parameters for example), which would be impossible to do with the real robot.

**Cross compilation**  When your controller is doing fine in simulation, you will be able to send and run it on the real robot without changing anything to your code, just by pressing a button in the robot window.

**Remote control**  To debug or understand your controller’s behavior, you will be able to see in real time the state of all the sensors and actuators on the computer screen. This is available both in simulation and on the real robot, and here again this is done in just one click. You will also be able to run your controller on the computer, but instead of sending commands to and reading sensor data from the simulated robot, it sends commands to and reads sensor data from the real robot.

**Ease of use**  Webots greatly simplifies the programming of the robot. Indeed, Webots API is simple to understand and to use and come with a complete documentation.
2 Simulation model

The simulation model of DARwIn-OP was design to be as close as possible to the real one. It is equipped with the following sensors and actuators:

- 20 servos
- 5 LEDs (including 2 RGB ones)
- A 3 axes accelerometer
- A 3 axes gyroscope
- A camera

The accelerometer returns values between 0 and 1024 corresponding to values between -3 [g] to +3 [g] like on the real robot. For the gyro, it returns also values between 0 and 1024, corresponding to values between -1600 [deg/sec] and +1600 [deg/sec], here again similarly to the values returned by the real robot. Their respective names are Accelerometer and Gyro.

The camera is a RGBA camera and has a basic resolution of 160x120 pixels, but it can be changed to any value. The horizontal field of view is 1.0123 [rad].

Each of the 2 RGB LEDs, called HeadLed and EyeLed, is split in two separated parts, one on the head of the robot and one other small part on the back panel of the robot. There are also three other standard LEDs on the back panel of the robot, they are called BackLedGreen, BackLedBlue and BackLedRed.

The name of the 20 servos are the following:

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>ID</th>
<th>Name</th>
<th>ID</th>
<th>Name</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ShoulderR</td>
<td>2</td>
<td>ShoulderL</td>
<td>3</td>
<td>ArmUpperR</td>
<td>4</td>
<td>ArmUpperL</td>
</tr>
<tr>
<td>5</td>
<td>ArmLowerR</td>
<td>6</td>
<td>ArmLowerL</td>
<td>7</td>
<td>PelvYR</td>
<td>8</td>
<td>PelvYL</td>
</tr>
<tr>
<td>9</td>
<td>Pelv</td>
<td>10</td>
<td>PelvL</td>
<td>11</td>
<td>LegUpperR</td>
<td>12</td>
<td>LegUpperL</td>
</tr>
<tr>
<td>13</td>
<td>LegLowerR</td>
<td>14</td>
<td>LegLowerL</td>
<td>15</td>
<td>AnkleR</td>
<td>16</td>
<td>AnkleL</td>
</tr>
<tr>
<td>17</td>
<td>FootR</td>
<td>18</td>
<td>FootL</td>
<td>19</td>
<td>Neck</td>
<td>20</td>
<td>Head</td>
</tr>
</tbody>
</table>

The corresponding position of each servo can be seen in figure 1. Each of the 20 servos has the following configuration:

<table>
<thead>
<tr>
<th>maxForce</th>
<th>2.5</th>
<th>N * m</th>
</tr>
</thead>
<tbody>
<tr>
<td>acceleration</td>
<td>55</td>
<td>rad/s²</td>
</tr>
<tr>
<td>maxVelocity</td>
<td>12.26</td>
<td>rad/s</td>
</tr>
<tr>
<td>dampingConstant</td>
<td>0.002</td>
<td></td>
</tr>
<tr>
<td>staticFriction</td>
<td>0.025</td>
<td>N * m</td>
</tr>
</tbody>
</table>
Figure 1: Position of the servos

For more information on the use of all of these sensors/actuators refer to the Reference Manual of Webots \(^1\).

The physical model is very realistic and self collision check is available. To activate the self collision, expand DARwIn-OP in the scene tree and set selfCollision field to true (see figure 2). Use the self collision check only if you need it, because it is very computationally costly and can therefore significantly slow down the simulation speed.

Figure 2: Scene tree of the DARwIn-OP.

The following sensors/actuators are not present on the simulation model:

- The three buttons on the back of the robot are not present because they have no interest in the simulation.
- The microphones are not present in simulation because sound is not yet supported in Webots.
- The speakers are not present too because sound is not yet supported in Webots, but this will certainly be added soon.
3 Managers

A library is provided in order to implement all the key functionalities of Robotis Framework in simulation. This library is divided in three parts called managers. Each manager implements a module of the Framework. The first one called Gait manager allows you to use the walking algorithm of the Robotis Framework. The second one called Motion manager allows you to play predefined motions stored in the motion_4096.bin file. The last one called Vision manager, contains some image processing tools, useful for example to find a colored ball in the camera image.

3.1 Gait Manager

This manager implements the DARwInOPGaitManager class and allows you to use the walking algorithm of the Framework.

A lot of parameters are available in the Framework algorithm to tune the gait. But in order to make this manager easy to use, only a subset of the parameters can be set. The other parameters are set to default values that are known to works fine. It is however possible to change them if needed, by changing the default values that are stored in a *.ini configuration file. In appendix A, all the parameters of the gait are explained. The C++ constructor of DARwInOPGaitManager object is the following:

```cpp
DARwInOPGaitManager(webots::Robot *robot, const std::string & iniFilename);
```

The first parameter is the robot on which the algorithm applies and the second is the file name in which the default parameters are stored. The following method are available in order to modify the main parameters in your controller:

```cpp
void setXAmplitude(double x);
void setYAmplitude(double y);
void setAAmplitude(double a);
void setMoveAimOn(bool q);
void setBalanceEnable(bool q);
```

These are the open parameters, they have the following impact on the gait:

- **X** influences the length of the foot step forward, it can take any value between -1 and 1.
- **Y** influences the length of the foot step in the side direction, it can take any value between -1 and 1.
- **A** influences the angle of the gait and allows also the robot to rotate during the walk, it can take any value between 0 and 1.
• If MoveAimOn is set, it allows the robot to rotate around something by inversing the sense of rotation, it can be very useful to turn around a ball in order to kick it in the right direction for example.

• If BalanceEnable is set, the gyroscope is used in the control loop to make the walking gait more robust.

Finally the following method can be used in order to run the algorithm:

```cpp
void start();
void step(int ms);
void stop();
```

Start and stop need to be used to stop/start the algorithm and step is used to run ms milliseconds of the algorithm.

Note that, in order to run, the gait manager needs to know the position of each servo and the values of the gyro. It is therefore essential to enable the gyro and the position feedback of each servo before to use it, if it is not the case, a warning will appear and they will automatically be enabled.

### 3.2 Motion Manager

This manager implement the `DARwInOPMotionManager` class and allows you to play a predefined motion stored in the `motion_4096.bin` file. The main motions and their corresponding ids are listed in appendix B.

It is also possible to add custom motions to this file by using the `Action Editor` tool.

The constructor of `DARwInOPMotionManager` object is the following:

```cpp
DARwInOPMotionManager(webots::Robot *robot);
```

It only needs a pointer to the robot to which it applies. Then, the following method can be used to play a motion:

```cpp
void playPage(int id);
```

This method only needs the id of the motion to be played.

---

2More informations about this tool provided by ROBOTIS is available at: [www.support.robotis.com/ko/product/darwin-op/development/tools/action_editor.htm](http://www.support.robotis.com/ko/product/darwin-op/development/tools/action_editor.htm)
3.2.1 Motion Manager in mode step-by-step

By default when starting a motion, the motion is run synchronously. That is the controller is stopped until the motion is finished. But it is also possible to run a motion asynchronously, in that case, the motion is started but the execution flow of the controller is not stopped. This can be done by calling the method \textit{playPage} with the second parameter set to false:

\begin{verbatim}
void playPage(int id, bool sync = true);
\end{verbatim}

This will initiate the motion, but not run it, then in order to play ms second of the motion, the following method need to be called (before the robot step):

\begin{verbatim}
void step(int ms);
\end{verbatim}

In order to know if the motion is finished, the following method can be called:

\begin{verbatim}
bool isMotionPlaying();
\end{verbatim}

A typical use of the motion manager in mode step-by-step will be the following:

\begin{verbatim}
MotionManager->playPage(1, false);
while(MotionManager->isMotionPlaying()) {
    MotionManager->step(mTimeStep);
    /*
    * Do something,
    * like image processing for example
    */
    myStep();
}
\end{verbatim}
3.3 Vision Manager

This manager implements the DARwInOPVisionManager class. The constructor of this class is the following:

```cpp
DARwInOPVisionManager(int width, int height, int hue, int hueTolerance,
    int minSaturation, int minValue, int minPercent, int maxPercent);
```

The parameters are the following:

- The width of the image
- The height of the image
- The color hue of the target object to find
- The tolerance on the color hue of the target object to find
- The minimum color saturation of the target object to find
- The minimum color value of the target object to find
- The minimum percentage of color value in the image to validate the result
- The maximum percentage of color value in the image to validate the result

To find the color hue of the target object and to understand the impact of the saturation and value you can refer to figure 3, for more information you can also find a lot of great documentation on the Internet about HSV colorspace.

![HSV colorspace](image)

Figure 3: HSV colorspace
When an instance of this class is created, the `getBallCenter` method can be used to find the position of the target object:

```c
bool getBallCenter(double &x, double &y, const unsigned char * image);
```

This method returns true if the target was found, and false otherwise. If found, the `x` and `y` variables are set. The image pointer indicates the original image buffer. In order to find the position of the target object, this method proceeds to the following steps:

- Store the BGRA version of the image in a buffer
- Use this buffer to convert the image to HSV format
- Use the `Finder` class of the Framework to find the target object
- Extract and save the position of the target object

Once this method was called it is possible to know which pixels of the image are part of the target object by using this function:

```c
bool isDetected(int x, int y);
```

This method returns true if the pixel `(x,y)` is part of the target object and false otherwise.
4 Examples

In this part we will see all the examples provided with Webots for the DARwIn-OP model. We will describe how they work, how to use them and what can be done with them. All the examples can be found in WEBOTS_HOME/projects/robots/darwin-op/worlds.

The following buttons are the main ones used to control the simulation (they all are situated on top of the 3D view):

- **Open world** is used to open another example.
- **Revert** is used to reload the example file and restart the simulation.
- **Run** is used to start the simulation at real time speed.
- **Stop** is used to stop the simulation.

You will also need to use the following buttons to edit the examples (they are situated on top of the text editor):

- **Open file** is used to open a new file in the text editor.
- **Save file** is used to save the current file.
- **Build** is used to build the current project.
- **Clean** is used to clean all the compilation files of the current project.

You can find more information about the user interface in the corresponding chapter of the *User Guide*

---

4.1 Symmetry

This example is very basic and explains the use of the servos.

It starts by setting the motor force of the three servos of the right arm to zero in order to completely release this arm. Then, in an infinite loop, the position of the previous three servos is read and displayed. Finally, still in the loop, the opposite position of each servo of the right arm is applied to the corresponding servo of the left arm in order to mimic the motion of the right arm.

In order to move the right arm which is free in simulation, select the robot, then press Ctrl+Alt and left click on the arm, then without releasing the left button move the mouse. This will apply a force (symbolized by an arrow) which will make the arm move.

Note that it is very important to activate the position feedback of the servos in order to read their position. In this example, this is done in the constructor.

You can also try to add an oscillation of the head, by adding this in your main loop:

```cpp
mServos[18]->setPosition(sin(getTime()));
```

Then save the file, press the build button and finally revert the simulation to start the new controller.

This example is well suited for the cross-compilation and we recommended that you start by testing the cross-compilation tool by using this example.
4.2 VisualTracking

This example illustrates the use of the camera (including the vision manager) and of the RGB LEDs.

In the infinite loop the vision manager is used to find the red ball. Then, if the ball has been found the head led is set to green and otherwise to red. Then, again, if the ball has been found the position of the two servos of the head is corrected to watch in the direction of the ball. To move the ball in simulation, press Ctrl+Shift and move the ball with the left button of the mouse pressed on it.

Try to change the color of the LED by changing this line:

```
mHeadLED->set(0xFF0000);
```

Here the color is set in hexadecimal. The format is R8G8B8: The most significant 8 bits (left hand side) indicate the red level (between 0x00 and 0xFF). Bits 8 to 15 indicate the green level and the least significant 8 bits (right hand side) indicate the blue level. For example, 0xFF0000 is red, 0x00FF00 is green, 0x0000FF is blue, 0xFFFF00 is yellow, etc.

Try also to use the other RGB LED, this is done simply be exchanging `mHeadLED` by `mEyeLED`.

Here again this example is well suited for cross-compilation. You can adjust the color of the ball by changing the value in the constructor of DARwInOPVisionManager if your ball has a different color.

This example can also be used as a tool to tune the parameters of the vision manager in order to fit your application.
4.3 Walk

This example illustrates the use of the gait and motion manager, the use of the keyboard, and also the use of the accelerometer.

At the beginning of the controller, the motion manager is used to make the robot stand up, then the controller enters an infinite loop. The first thing done in the loop is to check if the robot has not fallen down, this is achieved by using the accelerometer. Then if the robot has fallen down, the motion manager is used to make the robot to stand up. Then, the keyboard is read, if the space bar is pressed the robot start/stop to walk. Then, the keys up/down/right/left are pressed to make the robot turn and move forward/backward, several keys can be pressed at the same time.

Try to add some more action by using more keys. You can for example use the KEYBOARD.NUMPAD.LEFT and KEYBOARD.NUMPAD.RIGHT keys to make a left/right shoot (page 13 and 12 in motion manager). You can also use normal keys like 'A' instead if you prefer.

You can also use another key to make the robot walk quicker or slower (change the XAmplitude sent to the gait manager, values must be between -1 and 1).

This example works in cross-compilation but you will need to connect a USB keyboard to the robot. Otherwise, it is recommended to test this example with the remote control in order to use the computer’s keyboard instead.

This example can also be used to explore and test all the parameters of the gait.
4.4 Soccer

This is a very complete example which used the three managers and almost all of the sensors.

The controller is a very simple soccer player. It relies on most of the tools used in the previous example. We recommend you to study it by yourself and of course to improve it.

To extend this controller you can add new files to the project, but do not forget to also add them to the makefile (add the cpp files to the \texttt{CXX\_SOURCES} section). This example is also a good starting point for developing a more complicated controller.

This example works in cross-compilation. But we recommend you to test it on a soft ground and away from any source of danger (stairs, hot surface, etc.), because the robot will move a lot and it is not excluded that it falls down from time to time.
5 Robot Window

When you double click on the robot, a new window appears. This window is called Robot Window, it has several tabs allowing you to perform different things. The first four tabs concern the simulation and remote control. They will be describe here, the last tab is used to interact with the real robot and will therefore be describe in the next sections.

In the unlikely case of something going wrong with the Robot Window (freeze, bad behavior, etc.) you can at any time restart it by pressing the revert button of the simulation.

Accelerometers  This tab can be used to investigate the values of the accelerometer while the controller is running. If the checkbox is checked, the values of the accelerometer are shown and plotted on the graph in real time. Four different types of graph can be plot. The first three are one axis in function of an other, and the last one, plots the value of the three axes in function of the time. The corresponding colors are the following:

- Red for axis X
- Green for axis Y
- Blue for axis Z

You can click any time on the graph to adjust the scale of the data currently plotted.
**Cameras**  This tab is very simple, if the checkbox is checked, the picture of the camera is shown and updated in real time.

**Gyro**  This tab is very similar to the accelerometer tab but addresses the gyro. If the checkbox is checked, the values of the gyro are shown and plotted on the graph in real time. Here again four different types of graph can be plot.
Servos  Finally, this last tab can be used to see and influence the state of each servo. The use of each servo in the robot window can separately be set by checking/unchecking the corresponding checkbox of the servo. If the checkbox is checked, the value of the servo is shown and plotted in function of the time. On the graph, two different colors are used to distinguish the target value (in red) and the real value (in black). It is also possible to manually change the value of the servo by using the slider beside the graph.
6 Cross-compilation

To send your controller to the real robot and make it run on it, go to the Transfer tab of the robot window (figure 4).

![Transfer tab of the robot window.](image)

The first thing to do is to set the connections settings. The first setting is the IP address of the robot. If you use an Ethernet cable to connect to the robot, the IP address is 192.168.123.1. But if you use a wifi connection with the robot the address is not fixed, to know it, execute the `ifconfig` command on the robot, the IP address is the `inet addr` of wlan0 (warning, the address can sometimes change without any specific reason). The second parameter is the username with which you log-on on the robot, if you do not have explicitly changed it, the username is `darwin`. Finally the last parameter is the password corresponding to the username, here again, if you do not have explicitly changed it, the password is `111111`. Each time you connect successfully to the robot, all the settings are saved so that it is not necessary to set them each time you start the program. If you want to restore the default parameters of the connection, just click on the Restore default settings button (Alt+r).

Before you can send your controller to the real robot you have to change the Makefile.darwin-op file to suit to your project. If you have added new files to the project, do not forget to add them to the `CXX SOURCES` and if you have changed the project name, change also the `TARGET` value.
Before to send the controller you will also need to complete the *Robot Config* section of the *config.ini* file. You have two parameters to fill in:

**Time step** The time step in milliseconds must be specified in the field `time_step`, a minimal time step of 16ms is requested, if no time step (or a time step smaller than 16ms) is set, the default time step of 16ms is used. Warning: Depending on the complexity of you controller, a time step of 16ms can not always be respected. For example using the camera or the manager can slow done the speed, so enable them only if you really need them.

**Camera resolution** The horizontal and vertical resolution of the camera must be set in the fields `camera_width` and `camera_height`. Only the resolutions specified in table 1 are supported, if another resolution is set, the default resolution of 320x240 will be used.

<table>
<thead>
<tr>
<th>Width [pixel]</th>
<th>Height [pixel]</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>320</td>
<td>240</td>
<td>30</td>
</tr>
<tr>
<td>640</td>
<td>360</td>
<td>30</td>
</tr>
<tr>
<td>640</td>
<td>400</td>
<td>30</td>
</tr>
<tr>
<td>640</td>
<td>480</td>
<td>30</td>
</tr>
<tr>
<td>768</td>
<td>480</td>
<td>28</td>
</tr>
<tr>
<td>800</td>
<td>600</td>
<td>22.5</td>
</tr>
</tbody>
</table>

Table 1: Camera resolutions supported by the camera of the DARwIn-OP.

### 6.1 Send a controller to the robot

To test your controller on the real robot press the following button:

![Controller button](image)

Webots will then connect to the robot, if any error appears during the connection, the reason will be shown. If it is the first time you connect the robot with Webots, Webots will install all the files needed on the robot. This can take some time and some step are longer than other, so be patient please, this happens only on the first connection, the next ones will be shorter. You can also see in real time what is appening in the *DARwIn-OP console*. Webots will also stop the auto start of the demo program at the startup of the robot, but don’t worry the program is not suppressed and the auto start can easily be reinstalled (explanation follows).
Then the controller code itself is send to the robot. All the directory of the controller is send to the robot, so please put all the files needed by your controller in the same directory. The controller itself is then compiled for the robot and you can see the compilation in the DARwIn-OP console. If the compilation success and the robot is close to the start position (figure 5) the controller will be initialized (head and eyes LED in red) and then started (head and eyes LED in green).

![Figure 5: Start position of the robot. The robot is sit down (same start position that in simulation).](image)

It is recommended when testing a new controller whose behavior is not very certain to hold the robot by its handle.

To stop the controller press the following button:

![Figure 6: Stop button.](image)

This will stop the controller and clean all files previously sent to the robot.

You can also stop the controller by pressing the right button at the back of the robot. This will not entirely stop the controller but will at least avoid the robot from moving. It will also release the torque of all the servos.
6.2 Permanently install a controller to the robot

If you want to install the controller on the real robot, check the checkbox *Make default controller*. Then when you press the button to send the controller to the real robot, instead of running after the compilation, the controller is set to start automatically at the startup of the robot without any need of Webots or any computer. Warning: the robot still need to be in the start position when starting but their wont be any verification on the position, it is your responsibility to make sure that you always start the robot in this position (starting from an unknown position is not safe).

6.3 Uninstall Webots files from the robot

If you don’t need to use anymore Webots with your DARwIn-OP, you can uninstall all the files installed on the DARwIn-OP by Webots by pressing this button:

![Uninstall Webots files](image)

This will restore your robot like it was before installing the Webots files on it. Even the demo program will again automatically start at the startup of the robot. But if you send again a controller to the robot with Webots, all the files will again be installed. You can also use this button to reinstall all Webots files to the robot if you think something went wrong during the installation.

If you install a new version of Webots on your computer the Webots files on the robot will automatically be updated at the sending of the first controller (don’t worry if you use several version of Webots, an older version can not erase files from a newer version).

6.4 Dynamixel MX28 firmware

The cross-compilation has been optimized for the last firmware versions of the servos. You need to have at least version 27 of the firmware installed on all the servos, if this is not the case (on old DARwIn-OP robot for example) you will be informed when you will try to send a controller to the real robot. In order to update the firmware version please use the *Firmware Installer* tool\(^4\).

---

\(^4\)More informations about this tool from ROBOTIS at: [www.support.robotis.com/ko/product/darwin-op/development/tools/firmwareInstaller.htm](www.support.robotis.com/ko/product/darwin-op/development/tools/firmwareInstaller.htm)
6.5 Using speaker

As speaker are not present in Webots, it is not possible to use the speakers in simulation. In cross-compilation it is still possible to play sound by using these two functions:

```cpp
virtual void playFile(const char* filename);
virtual void playFileWait(const char* filename);
```

`filename` is the path to an audio file (MP3 for example). The `playFile` function plays the file without stopping the controller (parallel execution) while the `playFileWait` function stops the controller until the audio file playback is complete (serial execution).

In order to use them you have to write something similar to this:

```cpp
#include <webots/Speaker.hpp>

mSpeaker = getSpeaker("Speaker");
mSpeaker->enable(mTimeStep);
mSpeaker->playFile("hello.mp3"); // the file is in the same directory as the controller
```

Because Speaker are not yet present in simulation we recommend you to put all your code concerning the speaker within `#ifdef CROSSCOMPILATION` statements in order to keep the same code running in simulation and on the real robot. Here is an example:

```cpp
#ifdef CROSSCOMPILATION
mSpeaker = getSpeaker("Speaker");
mSpeaker->enable(mTimeStep);
#endif
```

Several audio files are already present on the robot in the `/darwin/Data/mp3/` folder, you can freely use them this way:

```cpp
mSpeaker->playFile("/darwin/Data/mp3/Introduction.mp3"); // this file is already on the robot, no need to send it.
```

The C appendix references all the audio files available.

You can also use this two text to speech functions of `Speaker` class:

```cpp
virtual void speak(const char * text, const char * voice = "en", int speed = 175);
virtual void speakFile(const char * filename, const char * voice = "en", int speed = 175);
```

In the first one you need to specify the path as an argument to the file containing the text and in the second one you can directly specify the text. You can also specify the voice you want to use (appendix D lists all the voices) and the speed in words per minute.
6.6 Using keyboard

The use of the keyboard is also available in cross-compilation. To use a keyboard you just have to connect an USB keyboard to the robot to one of the two USB ports available on the back of the robot (any wireless keyboard will also works).

Then when enabling the keyboard in your controller, a small window like the one depicted on figure 7 will show up on the real robot screen (if any connected).

![Figure 7: Small window used to capture the keyboard inputs in cross-compilation.](image)

This little window is used to capture the input of the keyboard. Please do not close this window or unset the focus on it (by clicking outside this window) or you wont be able to read the keyboard input anymore.
Remote control

Remote control is much more simpler to use than cross-compilation, you do not have to set the time step in any files, or to edit any specific makefile, the exact same controller that in simulation can be used for remote control (without even having to recompile it). Moreover, the remote-control mode allows you to visualize the state of the sensors and actuators of the real robot in real time. To use remote-control, open the robot window, go to the Transfer tab, as for cross-compilation you have to set the connection settings (the settings been the same as for cross-compilation, see previous chapter for more information). To start the remote control, stop and revert your simulation, put your robot in the stable position (see figure 5). Then press the following button:

![Remote control button](image.png)

A small window (similar of the one from picture 8) will appear and ask you to wait until the remote-control has been started. When this window disappears and the eyes of the robot switch from red to green, the remote-control has been sucessfully started. You can now easily start and stop your controller in remote-control mode by using the run and stop button of Webots (see chapter Examples for more details). Warning: if you revert the simulation it will stop the remote-control mode. In order to stop the remote-control (without reverting) simply press the stop button of the remote control (it has the same aspect from the one of the cross-compilation in figure 6).

![Remote control window](image2.png)

Figure 8: This small window asks you to wait until remote-control has started.

When the controller runs in remote-control mode, you can see in the other tabs of the robot window the values of the sensors of the real robot in real-time.
7.1 Camera resolution

In remote control, the camera’s resolutions supported are not the same as in cross-compilation, indeed they are smaller in order to not slow down too much the communication speed between Webots and the robot. All the resolutions available are specified in Table 2. Unlike from cross-compilation you do not have to specify the desired resolution in any file, the resolution is automatically send to the robot from Webots. So in order to adjust the resolution, just do the same way you would do it in the simulation (by editing `cameraWidth` and `cameraHeight` fields of the DARwIn-OP in the scene tree window).

<table>
<thead>
<tr>
<th>Width [pixel]</th>
<th>Height [pixel]</th>
</tr>
</thead>
<tbody>
<tr>
<td>320</td>
<td>240</td>
</tr>
<tr>
<td>160</td>
<td>120</td>
</tr>
<tr>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>40</td>
<td>60</td>
</tr>
<tr>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>20</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 2: Camera resolutions supported in remote-control.

Note that you do not need to choose a width and a height from the same line, any combination of height and width is valid (for example, you can use a resolution of 320x30).

7.2 Controller speed

Your controller is supposed to run at a speed of 1.0x whatever you chose to run the simulation at run in real-time or as fast as possible mode. It can still happen sometimes that the speed can not achieve a speed of 1.0x, especialy when using the camera at high resolution, the mode as fast as possible without graphics should resolve this problem.

If despite this you can not achieve a speed of 1.0x, it means that your connection with the robot is to slow. You should consider reducing camera resolution in order to increase the speed.
8  Known Bugs

8.1  Lateral balance
In simulation, in the DARwInOPGaitManager, the lateral balance does not work as expected. It is recommended to set `balance_hip_roll_gain` and `balance_ankle_roll_gain` to 0.0, this must be done in the ‘config.ini’ file associated with the controller.

8.2  Controller P
In simulation the P gain of the servo affects the speed but on the real robot it affects the torque. This can cause differences between simulation and reality in some specific cases. Especially when P is small.

8.3  Text-to-speech warning message
When using one of the two functions of text to speech from the Speaker module in cross-compilation, you might see the following message:

```
bt_audio_service_open : connect() failed : Connection refused (111)
```
You can simply ignore this message. This message is due to the fact that the robot is trying to communicate with a non-existent bluetooth device. You can supress this message by executing the following command on the robot:

```
sudo apt-get purge bluez-alsa
```
9 Bibliography

For any information about Webots, please visit Cyberbotics website: http://www.cyberbotics.com

Each time a new version of Webots is released the latest files of DARwIn-OP with Webots project are included, but you can always find the latest files of the project in the corresponding Github: https://github.com/darwinop/webots-cross-compilation

For any information about the DARwIn-OP robot please visit ROBOTIS website: http://support.robotis.com/ko/product/darwin-op.htm

DARwIn-OP being open source you can also find all the source files of the project here: http://sourceforge.net/projects/darwinop
A Walking parameters

This appendix explains all the parameters that can be set in the configuration file (.ini) to tune the gait.

X offset is the offset of the feet in the X direction. Unit is in millimeter.

![Figure 9: Walking: x offset parameters](image)

Y offset is the offset of the feet in the Y direction. Unit is in millimeter.

![Figure 10: Walking: y offset parameters](image)
**Z offset** is the offset of the feet in the Z direction. Unit is in millimeter.

![Figure 11: Walking : z offset parameters](image1)

**Roll offset** is the angle offset at the feet along X axis. Unit is in degree.

![Figure 12: Walking : roll offset parameters](image2)
**Pitch offset** is the angle offset at the feet along Y axis. Unit is in degree.

![Figure 13: Walking : pitch offset parameters](image)

**Yaw offset** is the angle offset of the leg along Z axis. Unit is in degree.

![Figure 14: Walking : yaw offset parameters](image)

**Hip pitch offset** is the tilt of DARwIn-OP’s body. It uses a special unit of the motor correspondig to 2.85 degree.

![Figure 15: Walking : hip pitch offset parameters](image)
**Period time** is the time required for DArwIn-Op to complete two full steps (left and right foot). Unit is in millisecond.

Figure 16: Walking : period time parameters

**DSP ratio** is the ratio between the time when both feet are on the ground to only one foot (either left or right) is on the ground.

Figure 17: Walking : dsp ratio parameters
**Step forward back ratio** is the differential distance according to X direction, between DARwIn-OP’s left and right foot during walk. Unit is in millimeter.

![Figure 18: Walking : step forward back ratio parameters](image)

**Foot height** is the maximum height of the foot during the step. Unit is in millimeter.

![Figure 19: Walking : foot height parameters](image)

**Swing right left** is the left and right Swaying of DARwIn-OP’s body during walking. Unit is in millimeter.

![Figure 20: Walking : swing right left parameters](image)
**Swing top down** is the up and down swaying of DARwIn-OP’s body during walking. Unit is in millimeter.

![Figure 21: Walking : swing top down parameters](image1)

**Pelvis offset** is angle offset at the pelvis along X axis. It uses a special unit of the motor corresponding to 2.85 degree.

![Figure 22: Walking : pelvis offset parameters](image2)

**Arm swing gain** is the gain that influences the movement of the arm during walking.

**Balance knee gain** is the gain at the knee level for the front/back balance.

**Balance ankle pitch gain** is the gain at the ankle level for the front/back balance.

**Balance hip roll gain** is the gain at the hip level for the lateral balance. Since the lateral balance does not work very well in simulation, we recommend you to set this parameter to 0.

**Balance ankle roll gain** is the gain at the ankle level for the lateral balance. Since the lateral balance does not work very well in simulation, we recommend you to set this parameter to 0.
## Motions files

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Description</th>
<th>Recommended initial position</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ini</td>
<td>Move to standing position</td>
<td>Standing up</td>
</tr>
<tr>
<td>2</td>
<td>OK</td>
<td>Nods head</td>
<td>Standing up</td>
</tr>
<tr>
<td>3</td>
<td>no</td>
<td>Shakes head</td>
<td>Standing up</td>
</tr>
<tr>
<td>4</td>
<td>hi</td>
<td>Tilts forward</td>
<td>Standing up</td>
</tr>
<tr>
<td>6</td>
<td>talk1</td>
<td>Holds out his hand</td>
<td>Standing up</td>
</tr>
<tr>
<td>9</td>
<td>walkready</td>
<td>Prepares to walk</td>
<td>Standing up</td>
</tr>
<tr>
<td>10</td>
<td>f up</td>
<td>Gets up</td>
<td>Lying face against the ground</td>
</tr>
<tr>
<td>11</td>
<td>b up</td>
<td>Gets up</td>
<td>Lying back on the ground</td>
</tr>
<tr>
<td>12</td>
<td>rk</td>
<td>Right shoot</td>
<td>Standing up</td>
</tr>
<tr>
<td>13</td>
<td>lk</td>
<td>Left shoot</td>
<td>Standing up</td>
</tr>
<tr>
<td>15</td>
<td>sit down</td>
<td>Sits</td>
<td>Standing up</td>
</tr>
<tr>
<td>16</td>
<td>stand up</td>
<td>Stands up</td>
<td>Seated</td>
</tr>
<tr>
<td>17</td>
<td>mul1</td>
<td>Gets balanced on the head</td>
<td>Standing up</td>
</tr>
<tr>
<td>23</td>
<td>d1</td>
<td>Does yes with the arm</td>
<td>Standing up</td>
</tr>
<tr>
<td>24</td>
<td>d2</td>
<td>Applaud</td>
<td>Standing up</td>
</tr>
<tr>
<td>27</td>
<td>d3</td>
<td>Does yes with the arm and head</td>
<td>Standing up</td>
</tr>
<tr>
<td>29</td>
<td>talk2</td>
<td>Holds out his hand</td>
<td>Standing up</td>
</tr>
<tr>
<td>31</td>
<td>d4</td>
<td>Stretches in front and rear</td>
<td>Standing up</td>
</tr>
<tr>
<td>38</td>
<td>d2</td>
<td>Wave with the hand</td>
<td>Standing up</td>
</tr>
<tr>
<td>41</td>
<td>talk2</td>
<td>Presents himself</td>
<td>Standing up</td>
</tr>
<tr>
<td>54</td>
<td>int</td>
<td>Applaud louder</td>
<td>Standing up</td>
</tr>
<tr>
<td>57</td>
<td>int</td>
<td>Applaud</td>
<td>Standing up</td>
</tr>
<tr>
<td>70</td>
<td>rPASS</td>
<td>Performs a pass with the right foot</td>
<td>Standing up</td>
</tr>
<tr>
<td>71</td>
<td>lPASS</td>
<td>Performs a pass with the left foot</td>
<td>Standing up</td>
</tr>
<tr>
<td>90</td>
<td>lie down</td>
<td>Lies on the front</td>
<td>Standing up</td>
</tr>
<tr>
<td>91</td>
<td>lie up</td>
<td>Lies on the back</td>
<td>Standing up</td>
</tr>
<tr>
<td>237</td>
<td>sitdown</td>
<td>Jumps up and down</td>
<td>Standing up</td>
</tr>
<tr>
<td>239</td>
<td>sitdown</td>
<td>Jumps up and down quickly</td>
<td>Standing up</td>
</tr>
</tbody>
</table>

Table 3: Motions stored in the motions files.
## Audio files available

<table>
<thead>
<tr>
<th>File</th>
<th>Length [sec]</th>
<th>Size [kB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autonomous soccer mode.mp3</td>
<td>1</td>
<td>29</td>
</tr>
<tr>
<td>Bye bye.mp3</td>
<td>1</td>
<td>18.4</td>
</tr>
<tr>
<td>Clap please.mp3</td>
<td>1</td>
<td>20.4</td>
</tr>
<tr>
<td>Demonstration ready mode.mp3</td>
<td>2</td>
<td>31.5</td>
</tr>
<tr>
<td>Headstand.mp3</td>
<td>1</td>
<td>19.2</td>
</tr>
<tr>
<td>Interactive motion mode.mp3</td>
<td>1</td>
<td>29.8</td>
</tr>
<tr>
<td>Introduction.mp3</td>
<td>16</td>
<td>258.8</td>
</tr>
<tr>
<td>Left kick.mp3</td>
<td>1</td>
<td>17.2</td>
</tr>
<tr>
<td>No.mp3</td>
<td>1</td>
<td>13.5</td>
</tr>
<tr>
<td>Oops.mp3</td>
<td>1</td>
<td>14.7</td>
</tr>
<tr>
<td>Right kick.mp3</td>
<td>1</td>
<td>18.4</td>
</tr>
<tr>
<td>Sensor calibration complete.mp3</td>
<td>2</td>
<td>36.4</td>
</tr>
<tr>
<td>Sensor calibration fail.mp3</td>
<td>2</td>
<td>37.2</td>
</tr>
<tr>
<td>Shoot.mp3</td>
<td>1</td>
<td>15.5</td>
</tr>
<tr>
<td>Sit down.mp3</td>
<td>1</td>
<td>20.4</td>
</tr>
<tr>
<td>Stand up.mp3</td>
<td>1</td>
<td>19.6</td>
</tr>
<tr>
<td>Start motion demonstration.mp3</td>
<td>2</td>
<td>34.3</td>
</tr>
<tr>
<td>Start soccer demonstration.mp3</td>
<td>2</td>
<td>34.3</td>
</tr>
<tr>
<td>Start vision processing demonstration.mp3</td>
<td>2</td>
<td>42.9</td>
</tr>
<tr>
<td>System shutdown.mp3</td>
<td>1</td>
<td>26.2</td>
</tr>
<tr>
<td>Thank you.mp3</td>
<td>1</td>
<td>17.2</td>
</tr>
<tr>
<td>Vision processing mode.mp3</td>
<td>1</td>
<td>28.2</td>
</tr>
<tr>
<td>Wow.mp3</td>
<td>1</td>
<td>17.6</td>
</tr>
<tr>
<td>Yes.mp3</td>
<td>1</td>
<td>16.8</td>
</tr>
<tr>
<td>Yes go.mp3</td>
<td>1</td>
<td>24.1</td>
</tr>
</tbody>
</table>

Table 4: Audio files already available on the robot in the directory `/darwin/Data/mp3/`
### D Voices available

<table>
<thead>
<tr>
<th>Name</th>
<th>Voice</th>
<th>Name</th>
<th>Voice</th>
</tr>
</thead>
<tbody>
<tr>
<td>af</td>
<td>afrikaans</td>
<td>bs</td>
<td>bosnian</td>
</tr>
<tr>
<td>ca</td>
<td>catalan</td>
<td>cs</td>
<td>czech</td>
</tr>
<tr>
<td>cy</td>
<td>welsh</td>
<td>da</td>
<td>danish</td>
</tr>
<tr>
<td>de</td>
<td>german</td>
<td>el</td>
<td>greek</td>
</tr>
<tr>
<td>en</td>
<td>english</td>
<td>eo</td>
<td>esperanto</td>
</tr>
<tr>
<td>es</td>
<td>esperanto</td>
<td>fi</td>
<td>finnish</td>
</tr>
<tr>
<td>fr</td>
<td>french</td>
<td>grc</td>
<td>greek</td>
</tr>
<tr>
<td>hi</td>
<td>hindi</td>
<td>hr</td>
<td>croatian</td>
</tr>
<tr>
<td>hu</td>
<td>hungarian</td>
<td>hy</td>
<td>armenian</td>
</tr>
<tr>
<td>id</td>
<td>indonesian</td>
<td>is</td>
<td>icelandic</td>
</tr>
<tr>
<td>it</td>
<td>italian</td>
<td>ku</td>
<td>kurdish</td>
</tr>
<tr>
<td>la</td>
<td>latin</td>
<td>lv</td>
<td>latvian</td>
</tr>
<tr>
<td>mk</td>
<td>macedonian</td>
<td>nl</td>
<td>dutch</td>
</tr>
<tr>
<td>no</td>
<td>norwegian</td>
<td>pl</td>
<td>polish</td>
</tr>
<tr>
<td>pt</td>
<td>brazil</td>
<td>pt-pt</td>
<td>portugal</td>
</tr>
<tr>
<td>ro</td>
<td>romanian</td>
<td>ru</td>
<td>russian</td>
</tr>
<tr>
<td>sk</td>
<td>slovak</td>
<td>sq</td>
<td>albanian</td>
</tr>
<tr>
<td>sr</td>
<td>serbian</td>
<td>sv</td>
<td>swedish</td>
</tr>
<tr>
<td>sw</td>
<td>swahili</td>
<td>ta</td>
<td>tamil</td>
</tr>
<tr>
<td>tr</td>
<td>turkish</td>
<td>vi</td>
<td>vietnam</td>
</tr>
<tr>
<td>zh</td>
<td>Mandarin</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Available audio voices