Powered by Webots™ 7

fast prototyping and simulation of mobile robots

- Realistic dynamical model
- Cross-compilation and transfer
- Remote control
- Framework compatible for key functionnalities

DARwIn-OP

ROBOTIS

CYBERBOTICS

professional mobile robot simulation

Contents

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 be university and research center for educational and research purpose. It has a total of 20 degrees of freedoms:

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

This robot is very interesting because of it's low price and because it is open source (both hardware and software). For example this robot is often used for the RoboCup and even win it sometimes.

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 your parameters for example), which would be impossible to do with the real robot.

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

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. This is available both in simulation and on the real robot, and here again this is done in just one click.

Easy of use Webots greatly simplifies the programming of the robot. Indeed, Webots API is much more simple to understand and use than ROBOTIS API, moreover a very complete documentation is available for Webots.

2 Simulation model

The simulation model of DARwIn-OP has been made as close as possible to the real one. It is equiped with the following sensors and actuators :

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

The accelerometer return values between 0 and 1024 corresponding to values between -3 [g] to +3 [g] like on the real robot. For the gyro, it return again values between 0 and 1024, corresponding to values between -1600 [deg/sec] and +1600 [deg/sec], here again in respect of the values of 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].

Concerning the LEDs, their is 2 RBG LEDs, called *HeadLed* and *EyeLed*, these two LEDs are each made of two separated parts, one on the head of the robots and one other small part on the back panel of the robot. There is also three other LEDs no RGB on the back panel of the robot, they are called *BackLedGreen*, *BackLedBlue* and *BackLedRed*.

The name of the 20 servos are the following :

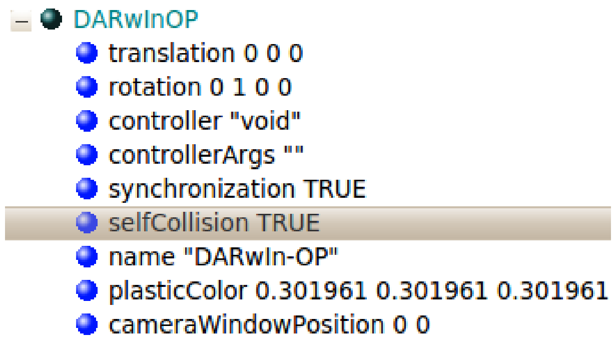
ID	Name	ID	Name	ID	Name	ID	Name
1	ShoulderR	2	ShoulderL	3	ArmUpperR	4	ArmUpperL
5	ArmLowerR	6	ArmLowerL	7	PelvYR	8	PelvYL
9	PelvR	10	PelvL	11	LegUpperR	12	LegUpperL
13	LegLowerR	14	LegLowerL	15	AnkleR	16	AnkleL
17	FootR	18	FootL	19	Neck	20	Head

The corresponding position of each servos can be seen in annex D.
Each of the 20 servos have the following configuration:

maxForce	2.5	$N * m$
acceleration	55	rad/s^2
maxVelocity	12.26	rad/s
dampingConstant	0.002	
staticFriction	0.025	$N * m$

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 1). 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.



DARwInOP
translation 0 0 0
rotation 0 1 0 0
controller "void"
controllerArgs ""
synchronization TRUE
selfCollision TRUE
name "DARwIn-OP"
plasticColor 0.301961 0.301961 0.301961
cameraWindowPosition 0 0

Figure 1: 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.

[1]Reference Manual available at : `http://www.cyberbotics.com/reference/`

3 Managers

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

3.1 Gait Manager

This manager implement the class *DARwInOPGaitManager* and allows to use the walking algorithm of the Framework.

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

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

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

```
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 influence the length of the foot step forward, it can take any value between -1 and 1.
- Y influence the length of the foot step in the side direction, it can take any value between -1 and 1.
- A influence the angle of the gait and permit also to rotate during the walk, it can take any value between 0 and 1.
- If MoveAimOn is set, it allowed to rotate around something in particular 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 order to prevent the robot from falling while walking.

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

```
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.

It must be noticed that the gait manager need to know the position of each servos and the values of the gyro to run. It is therefore essential to enable the gyro and the position feedback of each servos 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 class *DARwInOPMotionManager* and allows to play predefined motion stored in the file *motion_4096.bin*, the main motions and the corresponding id of the motion file are explained in annex B.

It is also possible to add custom motions to this file by using the tools *Action Editor* [2] provided by ROBOTIS.

The constructor of *DARwInOPMotionManager* is the following :

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

He only need a pointer to the robot to which it is applied. Then the following method can be used to play a motion :

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

This method only need the id of the motion that must be played.

[2]More informations about this tool at : `www.support.robotis.com/ko/product/darwin-op/development/tools/action_editor.htm`

3.3 Vision Manager

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

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

The parameters are the following :

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

To find the hue of your target and to understand the impact of the saturation and value you can refer to figure 2, for more information you can also find a lot of great documentation on internet about HSV colorspace.

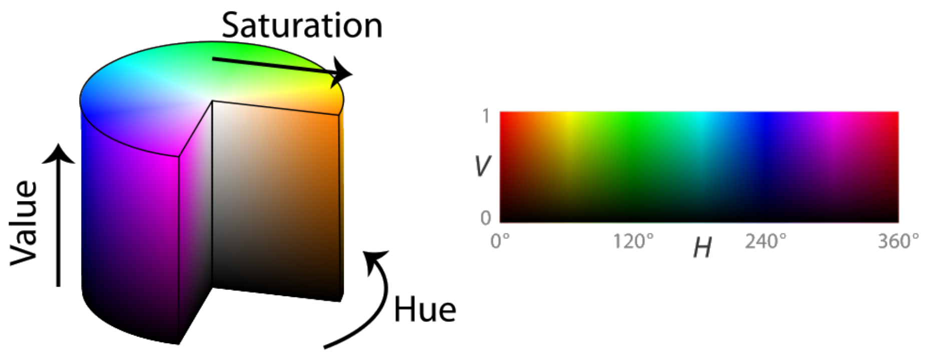


Figure 2: HSV colorspace

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

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

This method return true if the target has been found, and false otherwise. X and Y are references with which the result is transmitted. And image is a pointer to the image buffer. In order to find the position of the target, this method proceed to the following main steps:

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

When this method has been called it is possible to know which pixels of the image is part of the target by using this function :

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

This method return true if the pixel (x,y) is part of the target and false otherwise.

4 Examples

In this part we will see all the examples provided with Webots for the DARwIn-OP. 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 informations about the user interface in the corresponding chapter of the *User Guide* [3].

[3] User Guide available at : `www.cyberbotics.com/guide`

4.1 Symmetry

This example is very basic in order to explain the use of the servos.

It start 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 inverse of the position of each servos is applied on the servos of the left arm in order to mimic the motion of the right one.
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 to move.
It must be noticed 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 to the head, by adding this in your main loop:

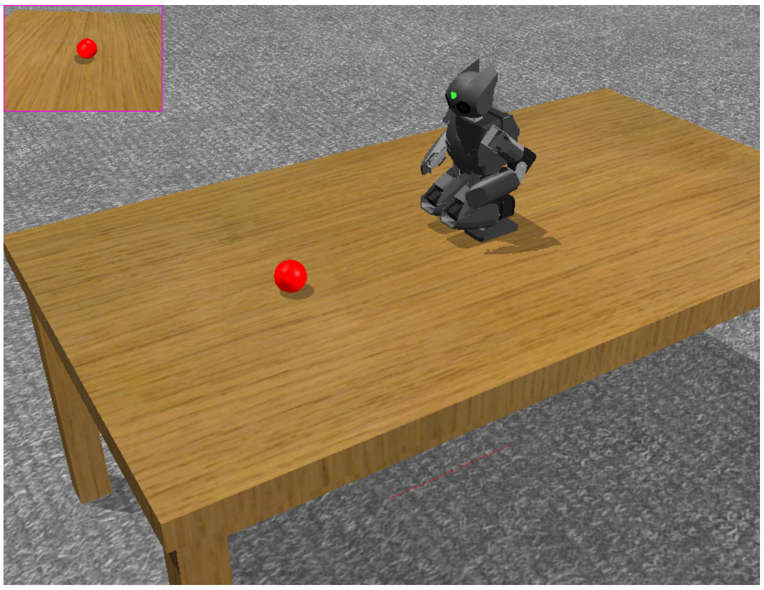
```
mServos[18]->setPosition(sin(getTime()));
```

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 illustrate 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);
```

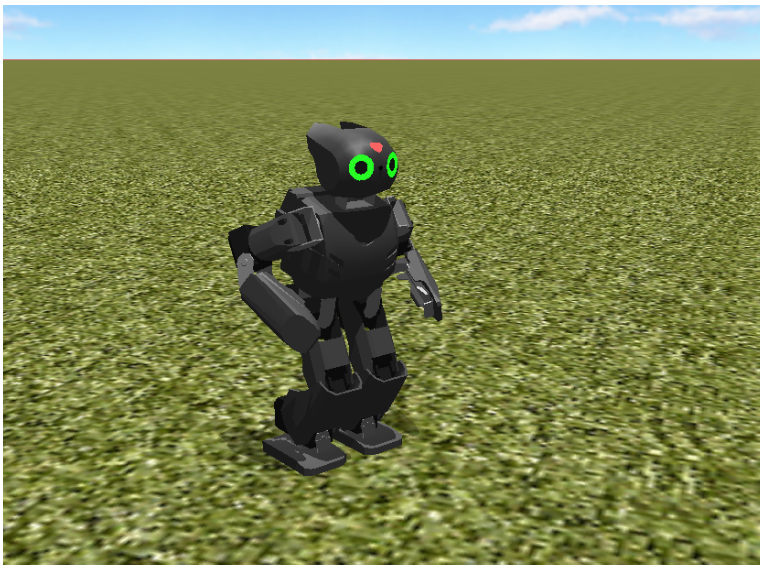
Here the color is set in hexadecimal. The format is R8G8B8: The most significant 8 bits (left hand side) indicate the red level (between 0x00 and 0xFF). Bits 8 to 15 indicate the green level and the least significant 8 bits (right hand side) indicate the blue level. For example, 0xFF0000 is red, 0x00FF00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

4.3 Walk

This example illustrate 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 enter an infinite loop. The first things done in the loop is to check if the robot has not fallen, this is done by using the accelerometer. Then if the robot has fallen, the motion manager is used in order to make the robot to stand up. Then the keyboard is read, if the key *Page Up* is pressed it make the robot start to walk, contrary if key *Page Down* is pressed the robot stop to walk. Then the keys up/down/right/left are used to make the robot turn and move forward/backward, severals key can be used in the same time.

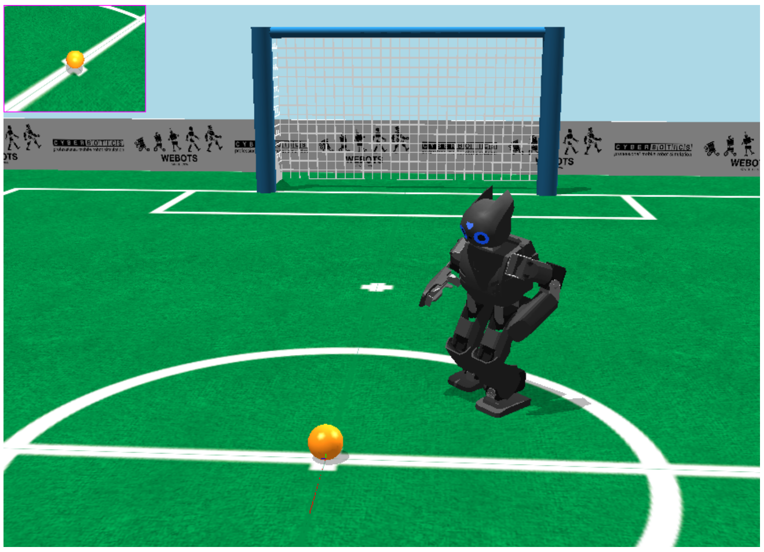
Try to add some more action by using more key. You can for example use the keys *KEYBOARD_NUMPAD_LEFT* and *KEYBOARD_NUMPAD_RIGHT* to make a left/right shoot (page 13 and 12 in motion manager).

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

This example work in cross-compilation but you will need to connect an usb keyboard to the robot. It is recommended to test this example with remote control in order to use the computer's keyboard instead.

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 used 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 your 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 *CXX_SOURCES* section).

This example work in cross-compilation. But we recommend you to test it on a soft ground and away from any source of danger (stairs, hot surface ...), because the robot will move a lot and it is not excluded that it fall sometimes.

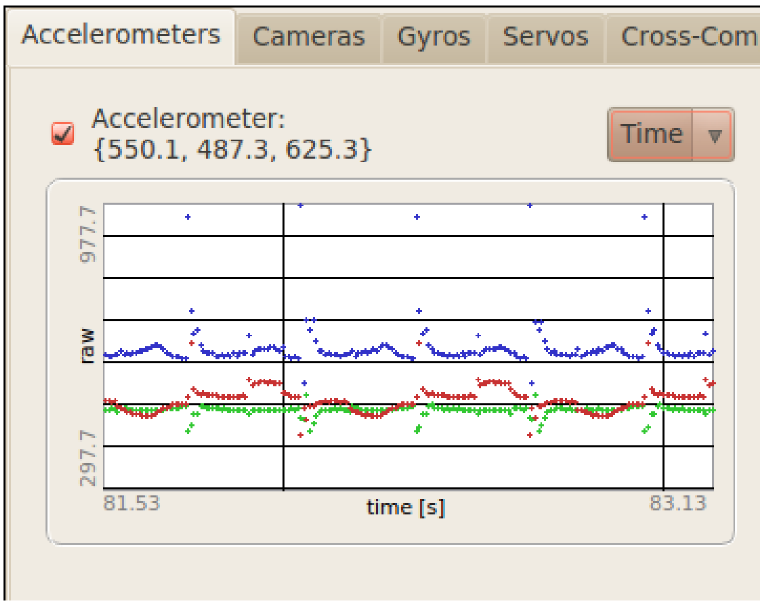
5 Robot Window

When double clicking on the robot, a new window appears. This window is called *Robot Window*, it has severals tabs who permits to do a lot of different things. The first four tabs concerns only the simulation and 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 case PEU probable of something going wrong with the *Robot Window* (freeze, bad behavior, ...) you can at any time restart it by pressing the revert button of the simulation.

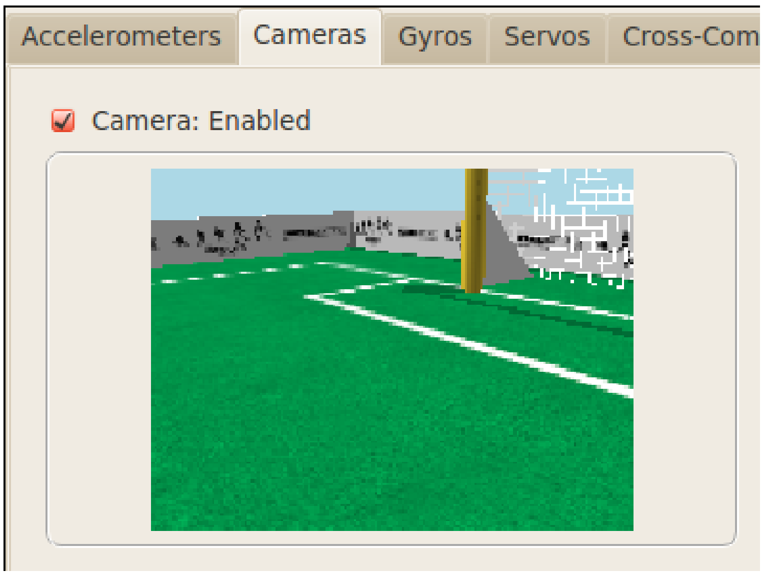
Accelerometers This tab can be used to investigate the values of the accelerometer while the controller is running. If the checkbox is checked, the values of the accelerometer are shown and plotted on the graph in real time. Four different types of graph can be plot. The first three are one axis in function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

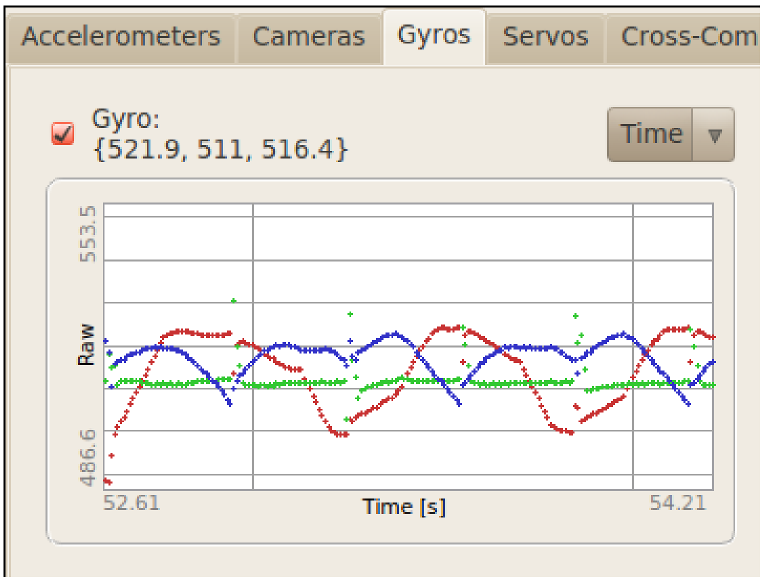


You can at any time click on the graph to adjust the scale to the data currently plotted.

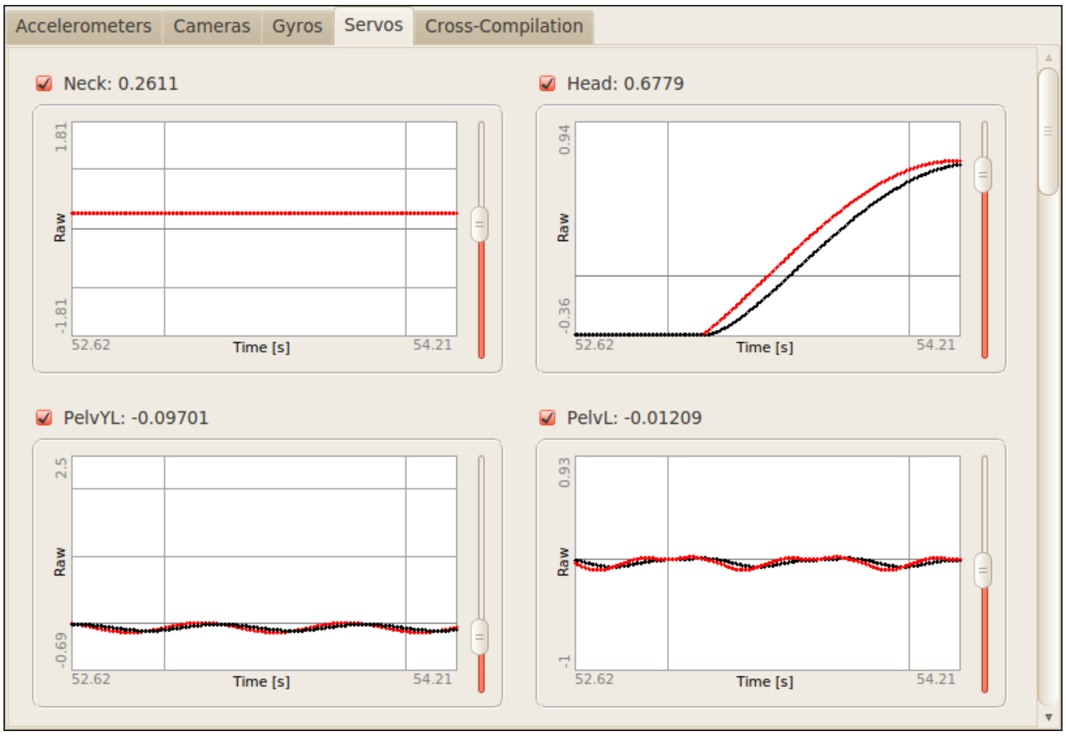
Cameras This tab is very simple, if the checkbox is checked, the picture of the camera is shown and updated in real time.



Gyro This tab is very similar to the accelerometer tab but for the gyro. If the checkbox is checked, the values of the gyro are shown and plotted on the graph in real time. Here again the four different types of graph can be plot.

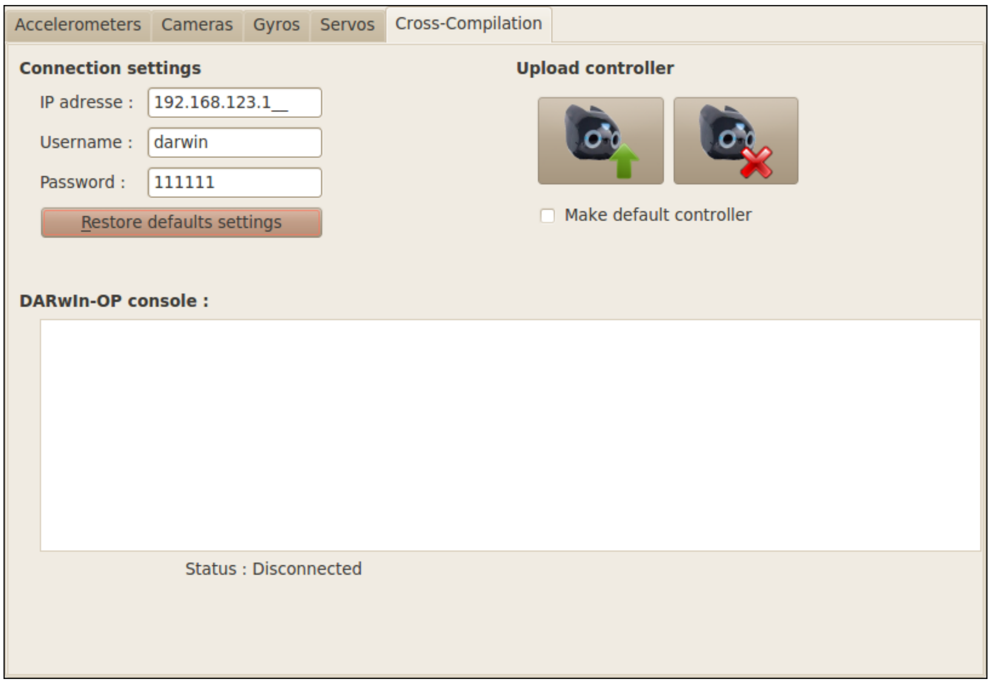


Servos Finally this last tab can be used to see and influence the state of each servos. The use of each servos in the robot window can separately be set by checking/unchecking the corresponding checkbox of the servos. If the checkbox is checked, the value of the servo is shown and plotted in function 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 the tab *Transfer* of the robot window (figure 3).



IP adresse : 192.168.123.1__
Username : darwin
Password : 111111
Status : Disconnected

Figure 3: Tab Transfer of the robot window.

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 command *ifconfig* on the robot, the IP address is the *inet addr* of wlan0 (warning, the adress can sometimes change without any specific reasons). The second parameters 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 parameters 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 in order 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 button *Restore default settings* (Alt+r).

Before you can send your controller to the real robot you have to change the file Makefile.darwin-op to suit to your project. If you have added new file 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*.

Before to send the controller you will also need to complete the *Robot Config* section of the file *config.ini*. You have two parameters to complete :

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 set. 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.

Width [pixel]	Height [pixel]	FPS
320	240	30
640	360	30
640	400	30
640	480	30
768	480	28
800	600	22.5

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 :

Webots will then connect to the robot, if any error appear 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 times and some step are longer than other, so be patient please, this is 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 6.1) the controller will be initialized (head and eyes LED in red) and then started (head and eyes LED in green).

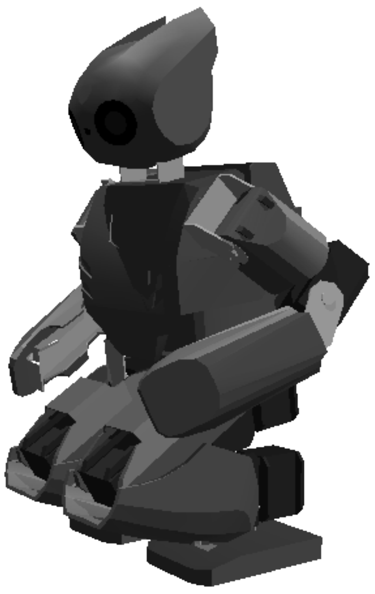


Figure 4: Start position of the robot. The robot is sit down (same start position that in simulation).

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 :

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. But 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 from the robot

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

This will restore your robot like it was before installing Webots files to 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 been update at the sending of the first controller (don't worry if you use severals 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 lasts 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 tool *Firmware Installer*[4] from ROBOTIS.

[4]More informations about this tool at : `www.support.robotis.com/ko/product/darwin-op/development/tools/firmware_installer.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 this two function :

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

Filename is the path to an audio file (MP3 for example). The function *playFile* play the file without stopping the controller while the function *playFileWait* stop the controller as long as the audio file is not finished.

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

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

And do not forget to include *webots/Speaker.hpp*.

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

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

Severals audio files are already present on the robot in the folder */darwin/Data/mp3/*, you can freely use them like this :

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

The annex C reference all the audio files available.

7 Remote control

A Walking parameters

This annex explain 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 direction X. Unit is in millimeter.

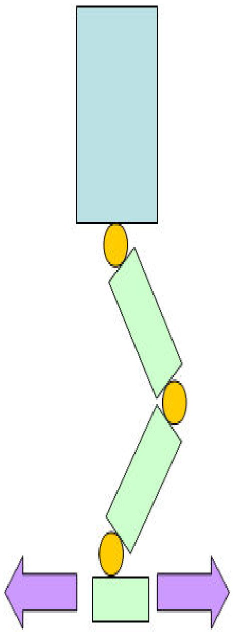


Figure 5: Walking : x offset parameters

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

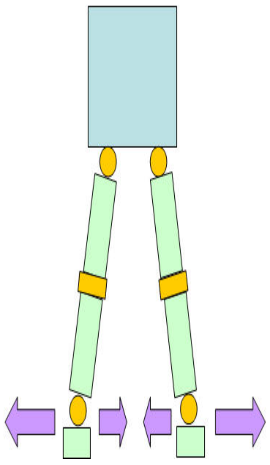


Figure 6: Walking : y offset parameters

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

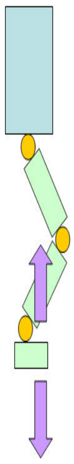


Figure 7: Walking : z offset parameters

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

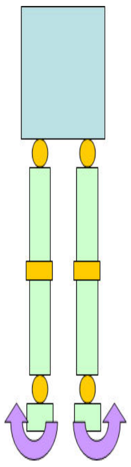


Figure 8: Walking : roll offset parameters

Pitch offset is the angle offset at the feet along Y axis. Unit is in degree.

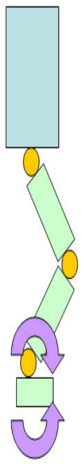


Figure 9: Walking : pitch offset parameters

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

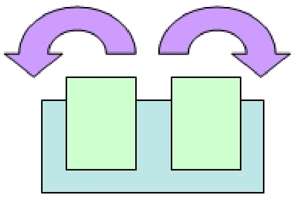


Figure 10: Walking : yaw offset parameters

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

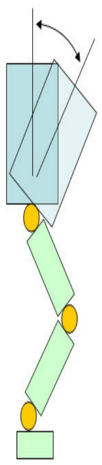


Figure 11: Walking : hip pitch offset parameters

Period time is the time required for DArwIn-Op to complete two full steps (left and right foot). Unit is in millisecond.

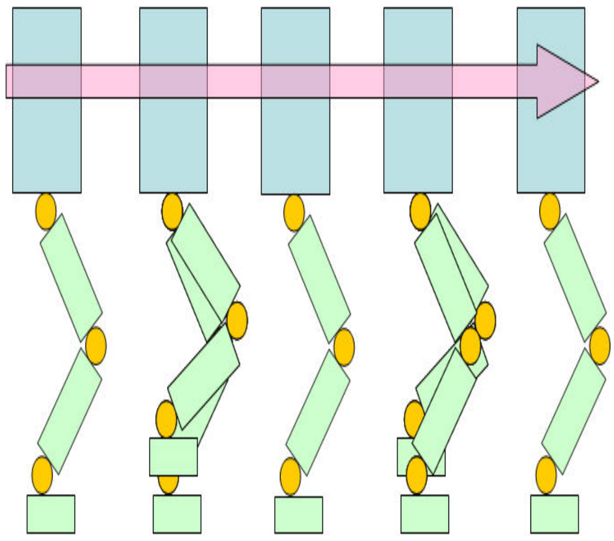


Figure 12: Walking : period time parameters

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.

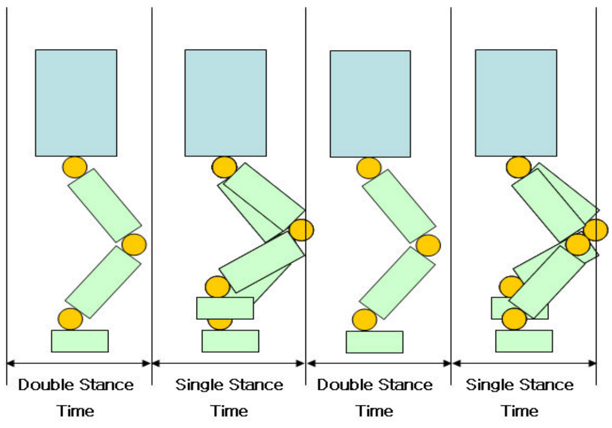


Figure 13: Walking : dsp ratio parameters

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.

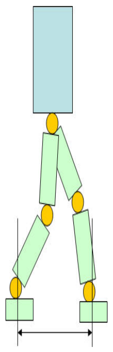


Figure 14: Walking : step forward back ratio parameters

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

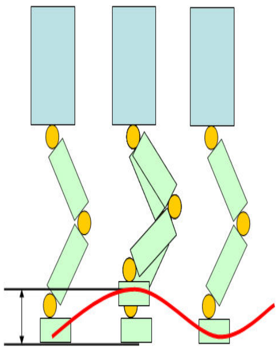


Figure 15: Walking : foot height parameters

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

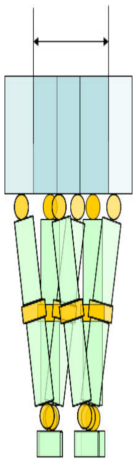


Figure 16: Walking : swing right left parameters

Swing top down is the up and down swaying of DARwIn-OP's body during walking. Unit is in millimeter.

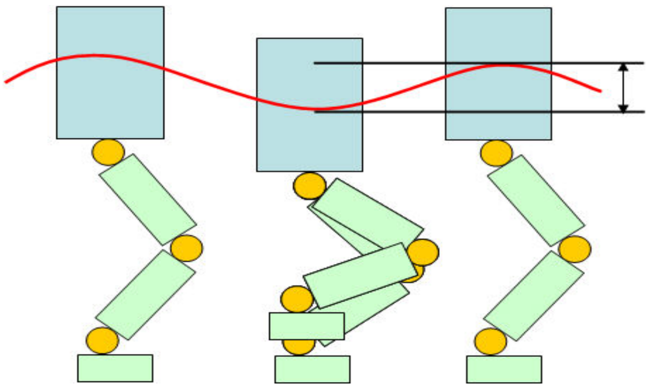


Figure 17: Walking : swing top down parameters

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

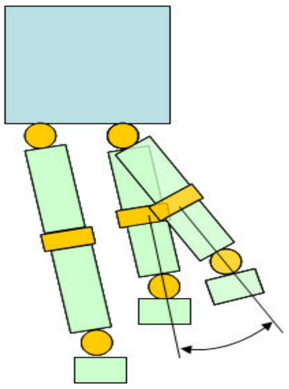


Figure 18: Walking : pelvis offset parameters

Arm swing gain is the gain that influence the movements 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 lateral balance do not work very well in simulation, we recommend you tu put this parameters to 0.

Balance ankle roll gain is the gain at the ankle level for the lateral balance. Since lateral balance do not work very well in simulation, we recommend you tu put this parameters to 0.

B Motions files

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

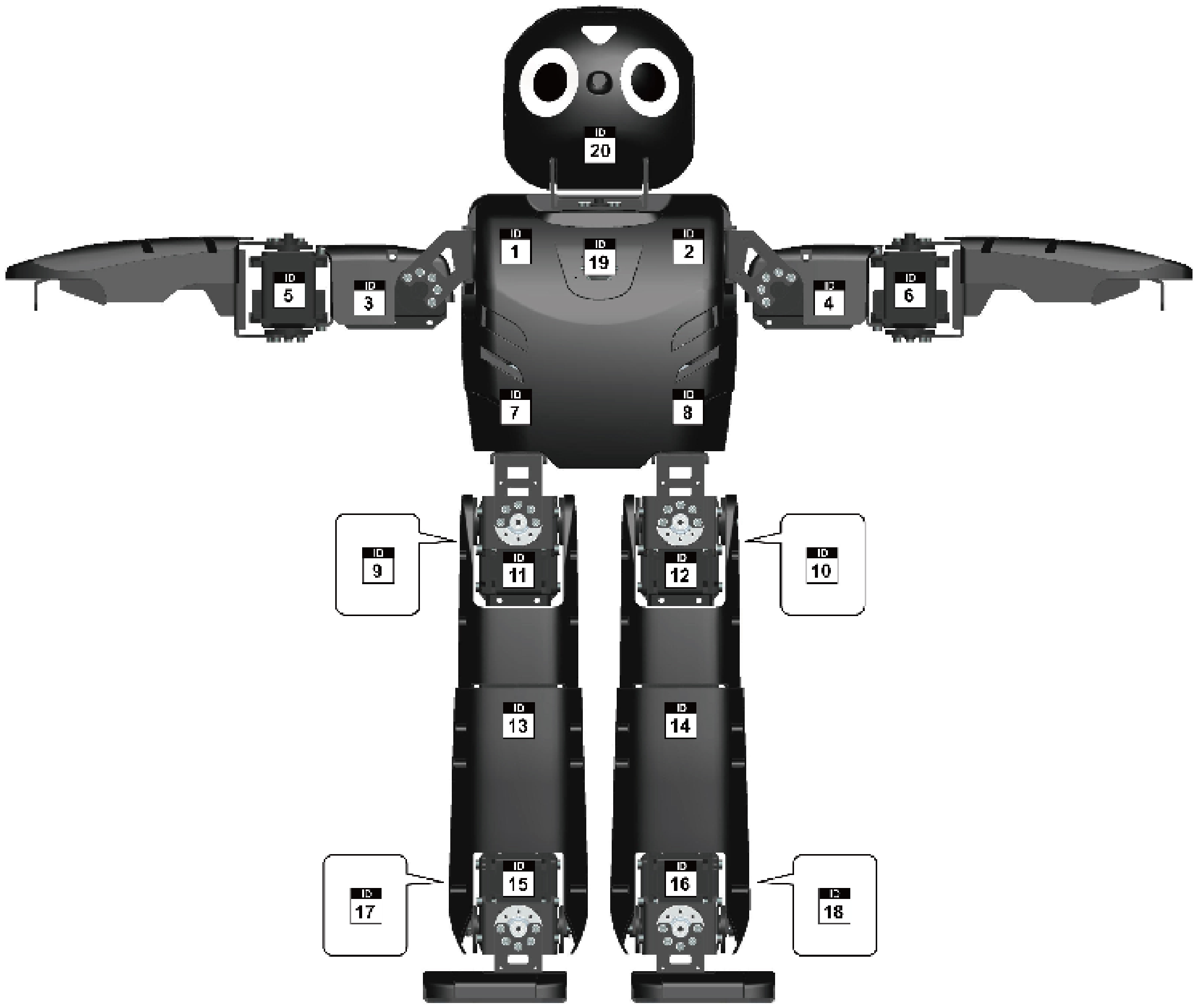
Table 2: Motions stored in the motions files.

C Audio files available

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

Table 3: Audio files already available on the robot in the directory /darwin/Data/mp3/

D Position of each servos



ID 20
ID 1
ID 19
ID 2
ID 5
ID 3
ID 4
ID 6
ID 7
ID 8
ID 9
ID 11
ID 12
ID 10
ID 13
ID 14
ID 17
ID 15
ID 16
ID 18