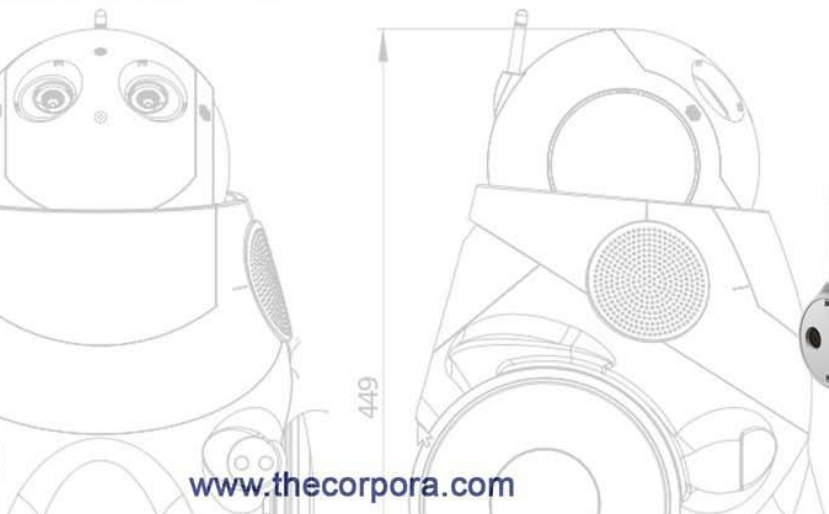
Thecorpora©
ROBOTIC COMPANY

Q.bo©

user´s guide



www.thecorpora.com

Q.bo User Guide

Contents

1. ArduQbo reference guide

ArduQbo is the driver implemented to communicate the Q.Board1 and Q.Board2 with the PC. It is prepared to work over ROS (www.ros.org) so it is needed to have ROS installed in your ubuntu distro to make it function properly.
It is recommended to follow the ROS tutorials to understand the ArduQbo guide.

1.1. Package Summary

This package is the ROS serial driver for the controller boards of QBO. It is needed to control the movement and the sensors of Q.bo.
It is recommended to use a configuration file to set the different parameters of the node.

1.2. qbo_arduqbo ROS Node

This node serves as the ROS driver for the controller boards of Q.bo. It is modular, it means than the systems that are going to be active can be configured with some ROS parameters.
The list of the controllers that can be activated is the following:

- Base controller
- Battery controller
- IMU controller
- Joint controller
- LCD controller
- Mics controller
- Mouth controller
- Nose controller
- Distance sensors controller

Each controller subscribes and/or publishes to different topics.

1.2.1. Base Controller

The base controller activates the movement of the base of the robot and the positioning system.

Published Topics

~odom (nav_msgs::Odometry)
Topic where the robot position in the world is published. The topic name can be changed with a ROS parameter

Subscribed Topics

~cmd_vel (geometry_msgs::Twist)

Topic where the Twist messages must be send to move the robot base. The topic name can be changed with a ROS parameter

1.2.2. Battery Controller

The battery controller activates the sending of the battery level in a ROS topic.

Published Topics

~battery_state (qbo_arduqbo::BatteryLevel)
Provides the battery level and the charging status. The topic name can be changed with a ROS parameter

1.2.3. IMU Controller

The IMU controller activates the sending of the IMU sensor readings in a ROS topic. The IMU sensor must be connected to the Q.board1

Published Topics

~imu_state (sensor_msgs::Imu)
Provides the inertial movement sensors readings of Q.bo to the ROS system. The topic name can be changed with a ROS parameter.
~imu_state/is_calibrated (std_msgs::Bool)
Indicates if the inertial sensors are calibrated or not.

Services

~imu_state/calibrate (std_srvs::Empty)
This service calibrates the inertial sensors of Q.bo.

1.2.4. Joint Controller

The joint controller activates the movement of the servos of the robot. It means the neck and eyes servos in Q.bo. The sensors are configured with a pair of ROS parameter described in the parameters section of this node.

Subscribed Topics

~cmd_joints (sensor_msgs::JointState)
Topic where the sensor_msgs::JointState messages must be send to move the servos of the robot. The topic name can be changed with a ROS parameter

1.2.5. LCD Controller

The LCD controller enables the sending of messages from the PC to the LCD screen of Q.bo.

Subscribed Topics

~cmd_lcd (qbo_arduqbo::LCD)

The messages sent to this topic are forwarded to the LCD screen of Q.bo. Only the first two lines of the LCD can be changed by the PC.

1.2.6. Microphone Controller

The microphone controller activates the sending of the mics levels in a ROS topic.

Published Topics

~mics_state (qbo_arduqbo::NoiseLevels)

Provides the microhpones' levels. The topic name can be changed with a ROS parameter

1.2.7. Mouth Controller

The Mouth controller enables the sending of mouth images from the PC to the mouth of Q.bo.

Subscribed Topics

~cmd_mouth (qbo_arduqbo::Mouth)

Topic where the qbo_arduqbo::Mouth messages must be send to set the mouth of the robot. The topic name can be changed with a ROS parameter

1.2.8. Nose Controller

The Nose controller enables the sending of the nose color from the PC to the nose of Q.bo.

Subscribed Topics

~cmd_nose (qbo_arduqbo::Nose)

Topic where the qbo_arduqbo::Nose messages must be send to set the nose color of the robot. The topic name can be changed with a ROS parameter

1.2.9. Distance Sensors Controller

The distance sensors controller publishes a set of sensor_msgs::PointCloud messages to the ROS system. It publishes one message for each activated sensor. The rate, frame and topic of the published messages is configured with the ROS parameters of the sensor controller.

1.3. ROS Messages

1.3.1 BatteryLevel

The ROS message BatteryLevel, is defined in the BatteryLevel.msg file located in the msg folder. It is published by qbo_arduqbo, indicated the battery level in volts and the charging status of the robot.
The content of *BatteryLevel.msg* is:

```
Header header
float32 level
uint8 status
```

1.3.2 LCD

The ROS message LCD, is defined in the LCD.msg file located in the msg folder. qbo_arduqbo subscribes to it. It provides the possibility to set different messages in the LCD screen of Q.bo.
The content of *LCD.msg* is:

```
Header header
string msg
```

The string msg must contain the message that is going to be set in the LCD screen.

1.3.3 NoiseLevels

The ROS message NoiseLevels, is defined in the NoiseLevels.msg file located in the msg folder. It is published by qbo_arduqbo, indicated the noise levels detected in the mics inputs of the Q.board2. Its values are the 10bit ADC readings of this noise levels.
The content of *NoiseLevels.msg* is:

```
Header header
uint16 m0
uint16 m1
uint16 m2
```

1.3.4 Mouth

The ROS message Mouth, is defined in the Mouth.msg file located in the msg folder. qbo_arduqbo subscribes to it. It provides the possibility to change the mouth shape of Q.bo.

The content of *Mouth.msg* is:

```
Header header
bool[20] mouthImage
```

Each element of the mouthImage array corresponds to a led of the mouth. If the value is True the led turns on and if the value is false the led turns off. The following table shows the led matrix and the element index of each led. It is assumed in the table that you see the led matrix from the front.

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19

1.3.5 Nose

The ROS message Nose, is defined in the Nose.msg file located in the msg folder. qbo_arduqbo subscribes to it. It provides the possibility to change the color of the Q.bo's nose. It only accept 2 different colors.
The content of *Nose.msg* is:

```
Header header
uint8 color
```

A value of 0 turns off the nose, a value of 2 puts a blue light on the nose and a value of 4 puts a green light on the nose.

1.4. ROS Services

The qbo_arduqbo package provides a service to test the different sensors and actuators available in Q.bo.

1.4.1. Test

The content of the Test.srv file is:

```
int8 SRFcount
int16[] SRFAddress
bool Gyroscope
bool Accelerometer
bool LCD
bool Qboard3
```

```
bool Qboard1
bool Qboard2
```

The test service checks the existence of the distance sensors controllers, of the IMU system, of the LCD screen and the boards Q.board1, Q.board2 and Q.board3.

1.5. ROS Parameters

The node accepts the following parameters:

~port1 (string, default: ”/dev/USB0”)
This is the string that indicates the serial port where the first controller board is connected.
~port2 (string, default: ”/dev/USB1”)
This is the string that indicates the serial port where the second controller board is connected.
~dmxPort (string, default: ”/dev/USB2”)
This is the string that indicates the serial port where the Dynamixel controller is connected.
~timeout1 (float, default: 0.01)
This is the maximum time in seconds that the driver waits for the Q.Board connected to the ~port1 to answer a command.
~timeout2 (float, default: 0.01)
This is the maximum time in seconds that the driver waits for the Q.Board connected to the ~port2 to answer a command.
~baud1 (integer, default: 115200)
This is the baudrate of the serial communication in the port1.
~baud2 (integer, default: 115200)
This is the baudrate of the serial communication in the port2.
~rate (float, default: 15.0)
This parameter sets the publication rate of the joint states message.
~controlledservos (dictionary, default: {})
This a dictionary that sets the configuration of the servos whose speed is needed to be controlled
~uncontrolledservos (dictionary, default: {})
This a dictionary that sets the configuration of the servos whose speed is not needed to be controlled
~dynamixelservo (dictionary, default: {})
This a dictionary that sets the configuration of the Dynamixel servos
~joint_states_topic (string, default: “joint_states”)
This parameters sets the name of the topic where the joint states messages are going to be sent.

~controllers (dictionary, default: {})

This is a dictionary that sets the controllers that are going to be active in the system and its configuration.

1.6. Servos dictionary

The elements of the parameters ~controlledservos, ~uncontrolledservos and ~dynamixelservo must be dictionaries. The element name must be the same as the joint name of the servo in the urdf model of the robot (if it exists). Following an example:

```
uncontrolledservos: {
   left_eyelid_joint: {id: 3, invert: true, max_angle_degrees: 180.0,
    min_angle_degrees: -180.0, range: 360.0, neutral: 1500 },
    right_eyelid_joint: {id: 4, max_angle_degrees: 180.0,
min_angle_degrees: -180.0, range: 360.0, neutral: 1500},
}
dynamixelservo:                                                    {
   head_pan_joint:  {id:  1,  invert:  false,  max_angle_degrees:  66.0,
min_angle_degrees:  -66.0,  range:  300.0,  ticks:  1024,  neutral:  570},
   head_tilt_joint:  {id:  2,  invert:  false,  max_angle_degrees:  20.0,
min_angle_degrees:  -37.8,  range:  300.0,  ticks:  1024,  neutral:  512},
}
```

Two servos are defined in the parameter ~uncontrolledservos and two in the parameter ~dynamixelservo. Its elements are:

- **id** (integer, default: -1): Identification number for the servo in the Q.board2 or identification number of the Dynamixel servo in the Dynamixel bus
- **invert** (boolean, default: false): Changes the rotation direction of the servo
- **max_angle_degrees** (float, default: 180.0): Maximum angle that is going to be sent to the servo in degrees
- **min_angle_degrees** (float, default: -180.0): Minimum angle that is going to be sent to the servo in degrees
- **range** (float, default: 180.0): Angle range of the servo
- **tics** (integer, default: 1800): Number of possible positions of the servo in its range
- **neutral** (integer, default: 1500): This value is used to set the 0 degree value of the servo

The ~uncontrolledservos parameter can be changed to ~controlledservos. In its case, a software estimation of the servo position is performed.

1.7. Controllers dictionary

The elements of the parameter ~controllers must be dictionaries. The element name is used as a representative name for the controller. Each controller must include the element type.
Only a fixed number of types are allowed in the driver. Each of them has its own specific parameters to set some options of the controller.
An example of the ~controllers parameter is shown below:

```
controllers: {
   "j_con": {
       type: joint_controller,
       rate: 15.0,
       topic: /cmd_joints
   },
   "b_con": {
       type: base_controller,
       rate: 15.0,
       topic: /cmd_vel,
       odom_topic: /odom,
       tf_odom_broadcast: true
   }
}
```

In the example two controllers are configured, the base controller and the joint controller. Following the different controller types and their specific parameters:

Base controller

The following parameters can be set.

- **type** (string, must be: base_controller)
- **rate** (float, default: 15.0): Rate at which the odom message is going to be published
- **topic** (string, default: cmd_vel): Topic name where the controller subscribes to receive the Twist messages that command the robot movement
- **odom_topic** (string, default: odom): Topic name where the odom message is going to be published
- **tf_odom_broadcast** (boolean, default: true): If true, the tf transform from odom to base_link is publish

Battery controller

The following parameters can be set.

- **type** (string, must be: battery_controller)

- **rate** (float, default: 15.0): Rate at which the battery message is going to be published
- **topic** (string, default: battery_state): Topic name where the battery message is going to be published

IMU controller

The following parameters can be set.

- **type** (string, must be: imu_controller)
- **rate** (float, default: 15.0): Rate at which the imu message is going to be published
- **topic** (string, default: imu_state): Topic name where the imu message is going to be published

Joint controller

The following parameters can be set.

- **type** (string, must be: joint_controller)
- **rate** (float, default: 15.0): Rate at which the joint message is going to be published
- **topic** (string, default: battery_state): Topic name where the joint message is going to be published

LCD controller

The following parameters can be set.

- **type** (string, must be: lcd_controller)
- **topic** (string, default: cmd_lcd): Topic name where the controller subscribes to receive the LCD message that are sent to the LCD screen

Mics controller

The following parameters can be set.

- **type** (string, must be: mics_controller)
- **rate** (float, default: 15.0): Rate at which the noise level message is going to be published
- **topic**(string, default: cmd_mics): Topic name where the controller subscribes to receive the mic message that that change the microphone source of the Q.board2
- **mics_topic** (string, default: mics_state): Topic name where the noise levels message is going to be published

Mouth controller

The following parameters can be set.

- **type** (string, must be: mouth_controller)

- **topic** (string, default: cmd_mouth): Topic name where the controller subscribes to receive the mouth message that changes the mouth leds of Q.bo

Nose controller

The following parameters can be set.

- **type** (string, must be: nose_controller)
- **topic** (string, default: cmd_nose): Topic name where the controller subscribes to receive the nose message that change the nose color of Q.bo

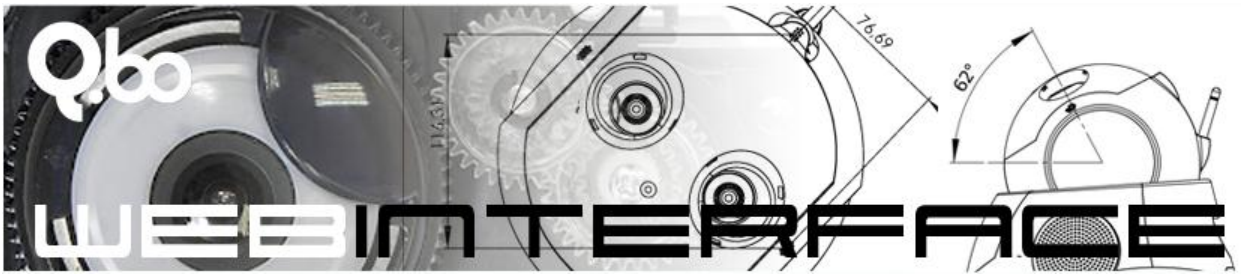
Distance sensor controller

The following parameters can be set.

- **type** (string, must be: sensors_controller)
- **rate** (float, default: 15.0): Rate at which the point cloud message with the sensor readings is going to be published
- **topic** (string, default: battery_state): Topic name where the point cloud message with the sensor readings is going to be published
- **sensors** (dictionary, default: {}): This element is where the distance sensors are configured. Following is explained with more detail the element contents.

5.2.9.1 Sensors dictionary

```
controllers: {
   "sens_con": {
       type: sensors_controller,
       rate: 5.0,
       topic: /distance_sensors_state,
       sensors: {
         front: {
           front_left_srf10: { type: srf10, address: 230, frame_id:
front_left_addon },

    front_right_srf10: { type: srf10, address: 226,
frame_id: front_right_addon }
         },
    floor: {
           floor_sensor: {type: gp2d12, address: 8, frame_id:
front_addon },
         }
       }
   }
}
```



2. Q.bo Webi reference guide

The Webi is a web interface control the robot remotely using a Web browser, and part of the OpenQbo software[1]. This web interface also gives information about the status of the systems inside the robot and launches some demo applications.

There are two ways to use the Webi. The first one is by connecting a keyboard, mouse and a screen to Q.bo. This is the administrator mode.

The second way to use the Webi is via WiFi. Q.bo needs to be connected to a WiFi network to use this mode.

Some functionalities can only be executed using the administrator mode.

In the administrator mode, the Webi can be accessed opening the web browser and entering the page: localhost:7070

In the remote mode the Webi can be accessed opening the web browser and entering the URL: "`http://<robot_ip>:7070`" where the <robot_ip> is the network IP of the robot. This IP is shown in the back LCD of the robot.

Important: it is recommendable to execute the Webi from the Google Chrome browser. The current version is completely compatible with this browser. The next version of the Q.bo Webi will have compatibility with other famous browsers such as the Internet Explorer, Opera, Safari and Mozzilla Firefox.

[1] The OpenQbo software provided with the robot is not covered by any guarantee that may be applicable to the hardware under the General Conditions of Sale. OpenQbo is a free software distribution based on the Ubuntu Operating System and as such is exempt from any guarantee. For any question regarding this, you may check the Ubuntu license. We preinstalled OpenQbo in the robot so that it can be operational the moment you open the packaging, but the "distro" does not deliver the full potential of the hardware nor was it our intention for that to happen. Feel free to program it any way you like and tell us in our wiki and in our forum what's the best you can get from Qbo.

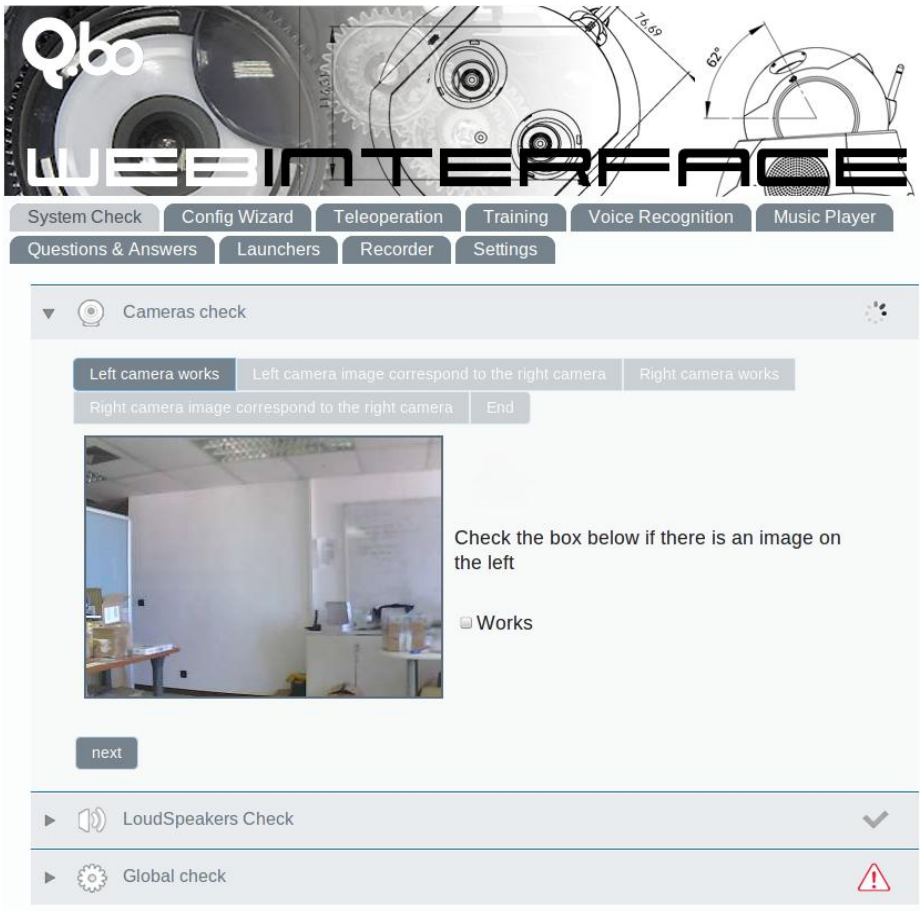
2.1. Login

Insert the following to connect to the robot:
Robot username: qbobot
Robot pass: QboPass
Webi pass: admin

2.2. System Check

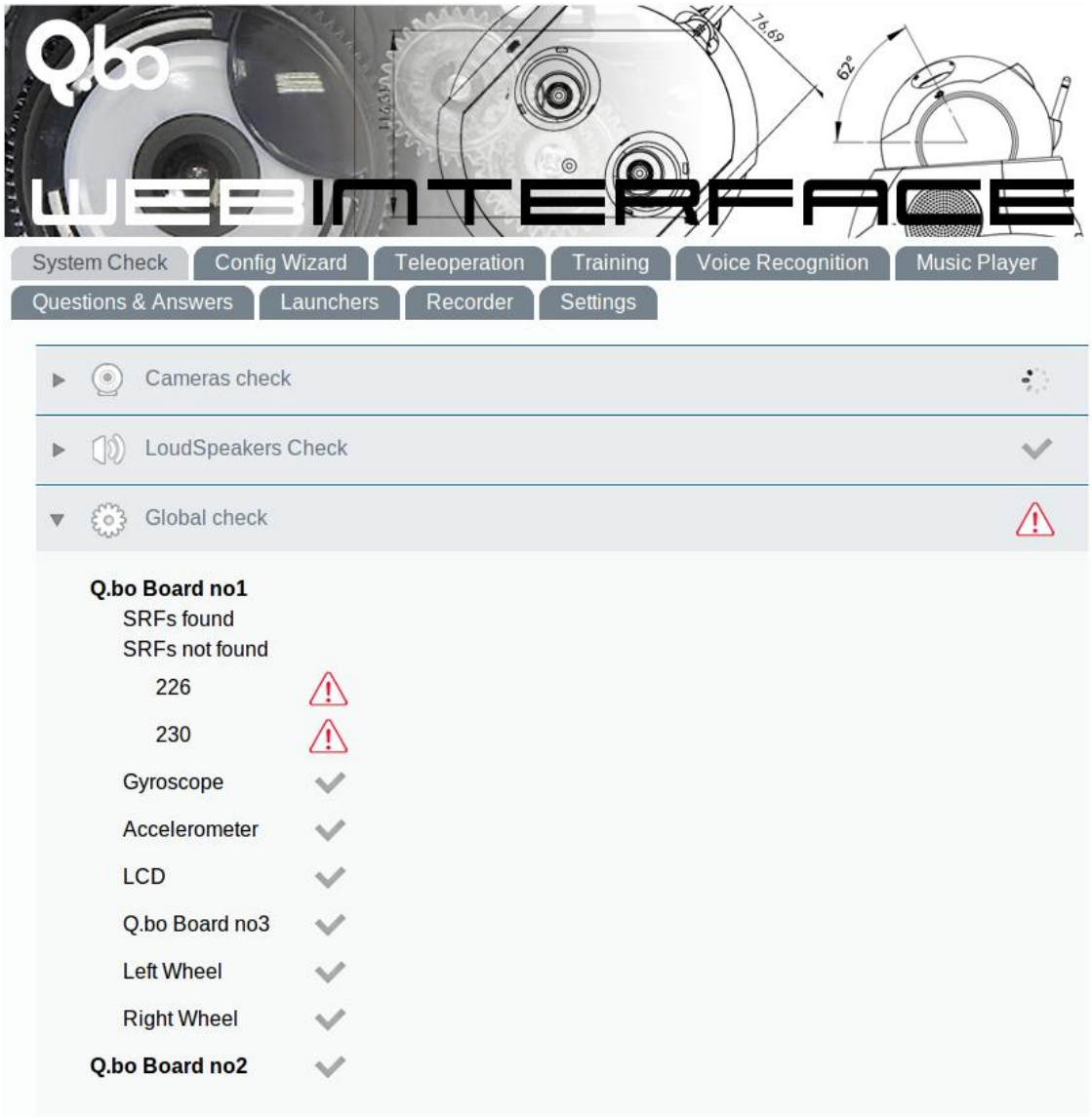
Cameras Check

The Cameras check allows you to test the correct function of Q.bo's cameras. You will only need to follow the instructions in the screen to finish the test.



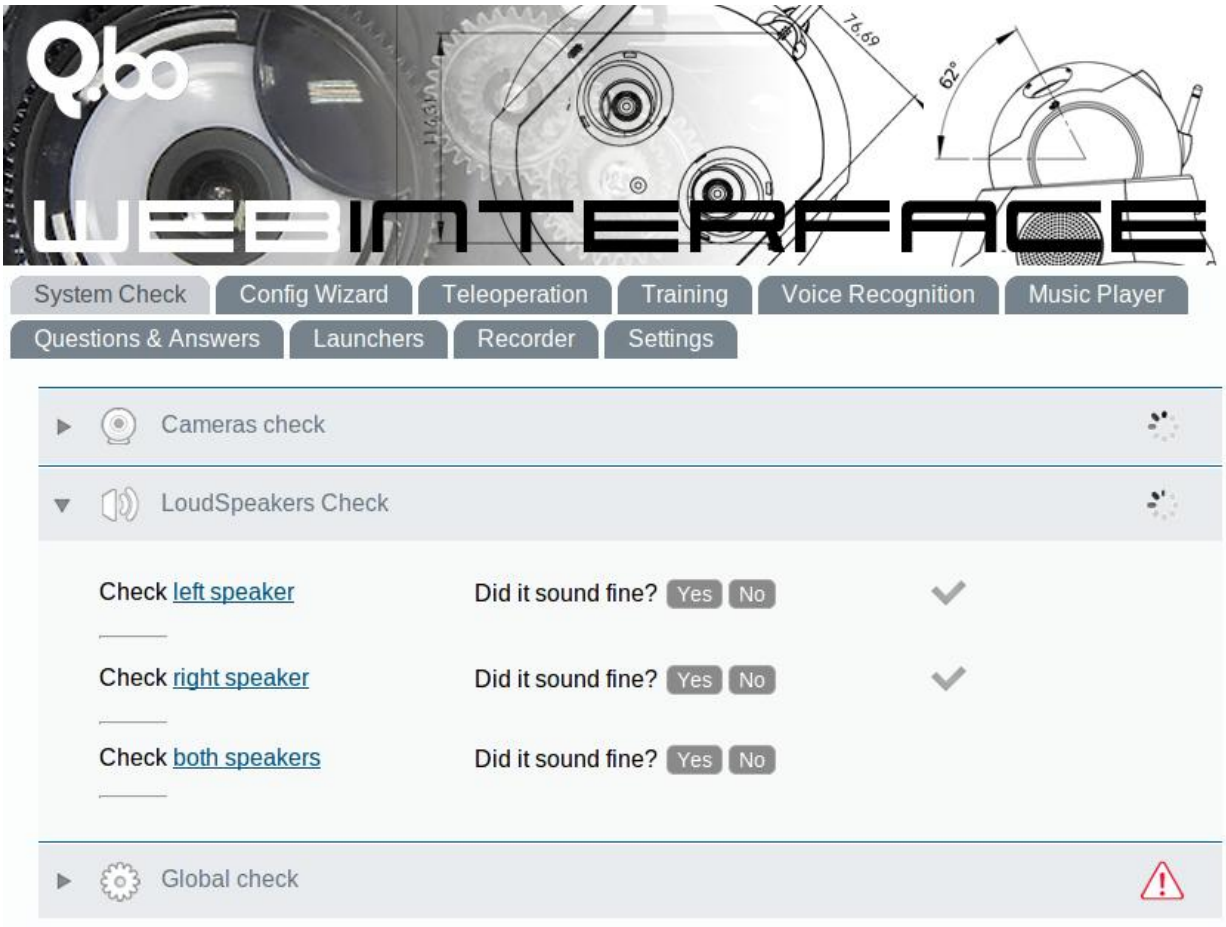
Global Check

The global check shows if all the Q.bo sensors and devices are working properly.



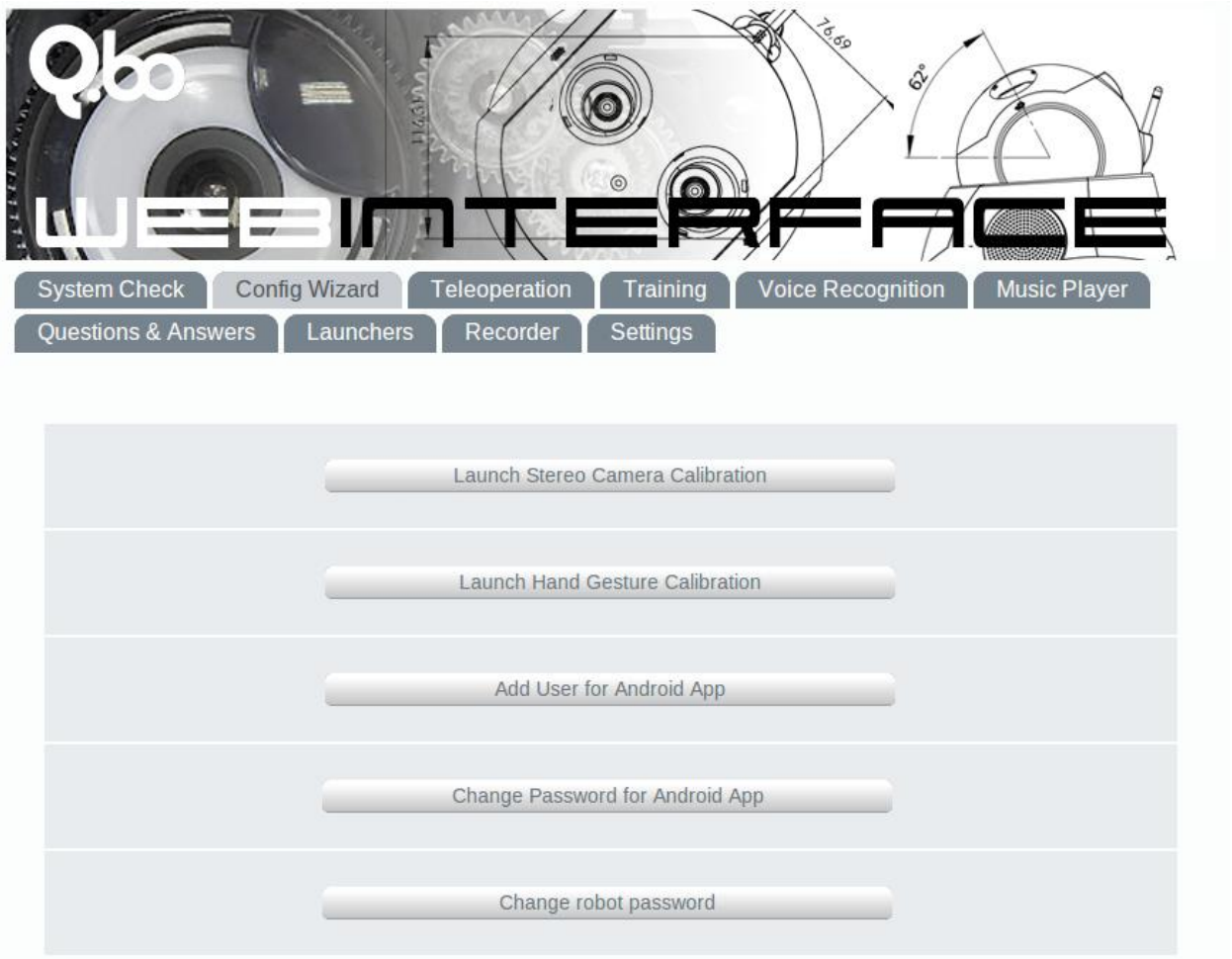
LoudSpeakers Check

Here you can see if the stereo loudspeakers of Q.bo are working properly. Click on the speakers links to launch a test sound and verify if the loudspeakers work correctly.



2.3. Config Wizard

The Configuration Wizard let you launch configuration programs for Q.bo, such as the camera calibration, as well as setting the users for the Android application and change the Webi password.



Launch Stereo Camera Calibration

Here you can re-calibrate the cameras to obtain more accurate results in stereoscopic tasks. Just follow the instructions that appear on the console that is launched by this button. Note: this can only be executed from a browser executed in the Q.bo robot.

Launch Hand Gesture Calibration

By using this calibration you can select your own gestures to control the gesture music player. Just follow the instructions that appear on the console that is launched by this button. Note: this can only be executed from a browser executed in the Q.bo robot.

Add User for Android App

You can add new users to the teleoperation Android application of Q.bo.

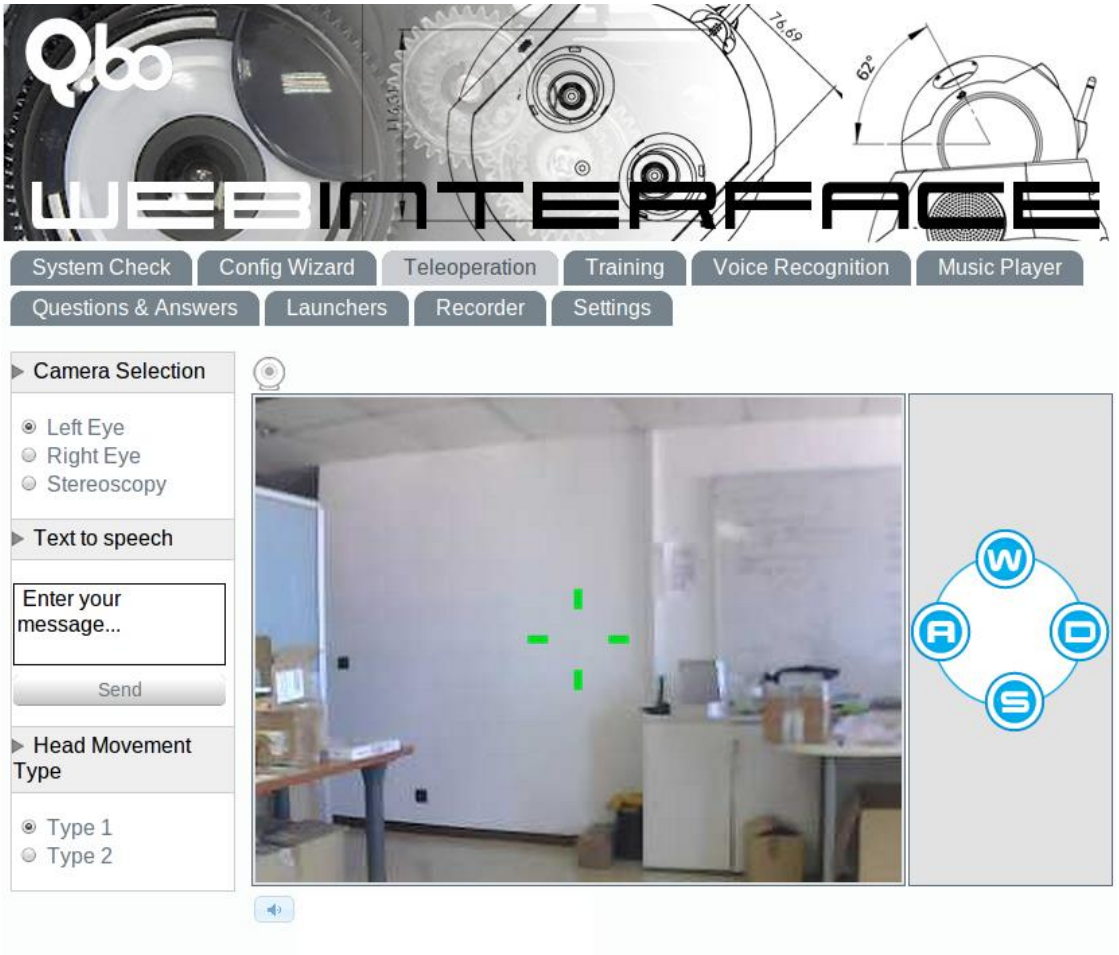
Change Password for Android App

Here you will be able to change the password for the users of the teleoperation Android application.

Change robot password

Here you will be able to change the Webi password requested in the login.

2.4. Teleoperation



Vision screen

This window shows what Q.bo sees. It is possible to move Q.bo's head by clicking with the mouse left button in the window and holding it. There are two different types of head movements.

Head Movement Type

Type 1: In this type of movement the head will return to the origin after releasing the mouse left button.
Type 2: The head movement's speed is proportional to the distance of the cursor to the center.

Camera Selection

With the camera selection, the image shown in the vision screen will be the corresponding to the left camera, the right camera or a stereoscopic image to see in 3D.

Text to speech

Q.bo will reproduce with its own voice what you type in the message box.

Base Movement

You can move Q.bo by using the buttons of the Webi or the “wasd” keys as if it was a videogame.

2.5. Training

In the Training Section, you can ask Q.bo to recognize Faces and Objects as well as to make him learn them. You can select between the Face or Object recognition mode using the radio buttons on the top.

Face Recognition

Q.bo will track with its head the nearest person it sees. Click "**Guess What is This**" so that Q.bo can answer you. The answer will appear on the right side of the screen.

To make Q.bo learn a new face, make sure the person is being tracked by the robot and then click on "**Train Yourself**", insert the name of the person and validate. A timer will appear and the Q.bo will start capturing images and learning the person. When the process is done, Q.bo will let you know.

Object Recognition

Q.bo will track with its head the nearest objects it detects. Click "**Guess What is This**" so that Q.bo can tell you the name of the objects it is detecting. The answer will appear on the right side of the screen.

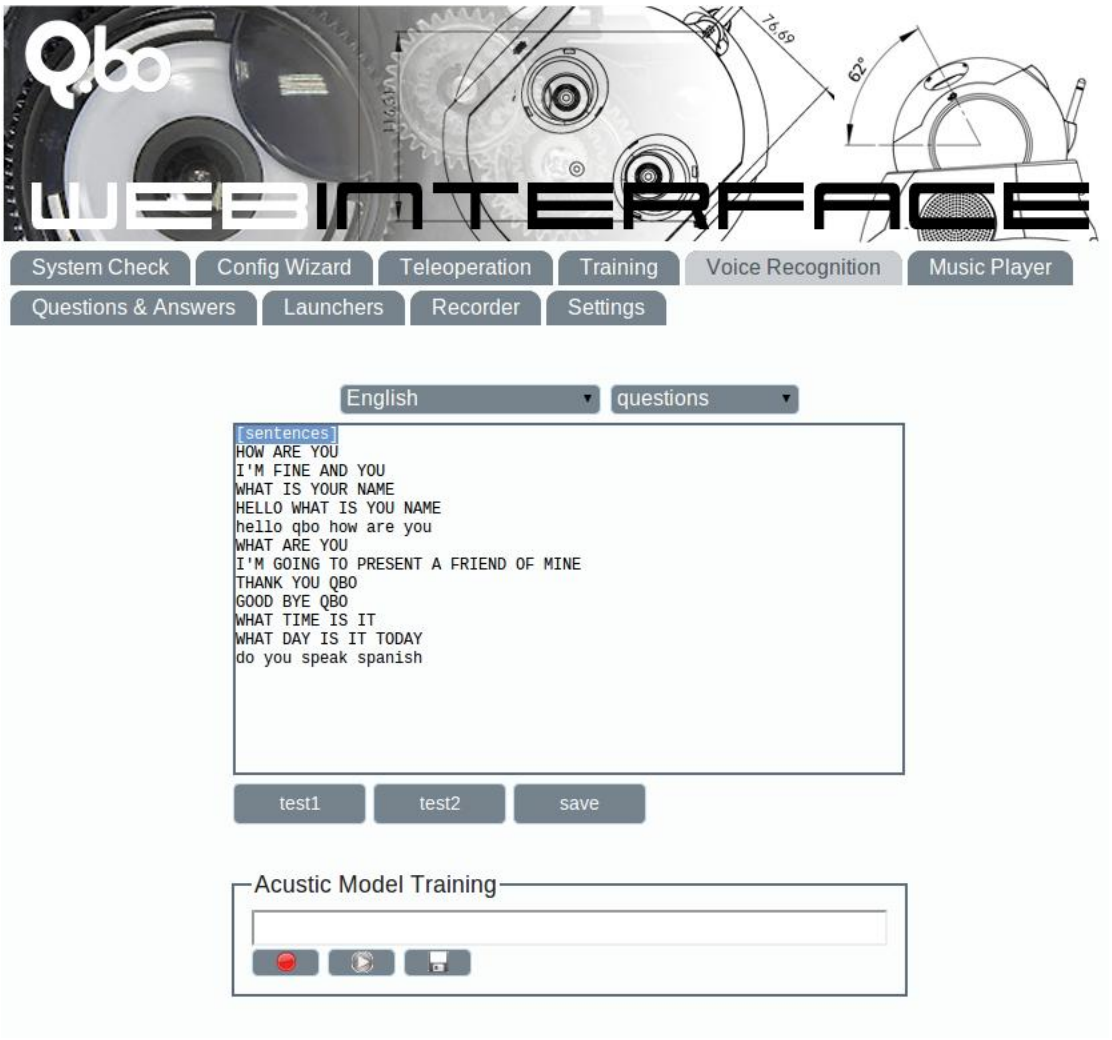
To make Q.bo learn a new object, make sure the object is being tracked by the robot and then click on "**Train Yourself**", insert the name of the object and validate. A timer will appear and the Q.bo will start capturing images and learning the appearance of the object. When the process is done, Q.bo will let you know.

Use Q.bo Cloud option: This option allows Q.bo to share learned objects with the Q.bo Cloud and use objects stored in the Q.bo Cloud. To activate this option, click on the check box "**Use Q.bo Cloud**"
Note: When you teach new objects and share them with the Q.bo Cloud, they will also be stored locally.

2.6. Questions & Answers

You can make a Question->Answer rule in the Q.bo robot by selecting the Question in the dropdown list, writing the answer you want Q.bo to respond and clicking on “Add”. The new rule should appear the in the rule list. You can set several answers for the same question, which will be chosen randomly.

2.7. Voice Recognition



Here you can find the following options:

1. One for adding new sentences to be understood by your robot. Select your language and a group of sentences, for example, *questions*. Then write your sentence under [sentences] label.

More complex sentences can be created by using other labels. For example, lets say you want the robot to recognize "*Call Peter*" and "*Call Maria*". One way to do it would be writing both sentences under [*sentences*] label, like this:

```
[sentences]
Hello Peter
```

```
Hello Maria
```

But there is a better way to do this sort of things. Just create your own label, like [*names*], and then use it under [*sentences*] label, as shown below:

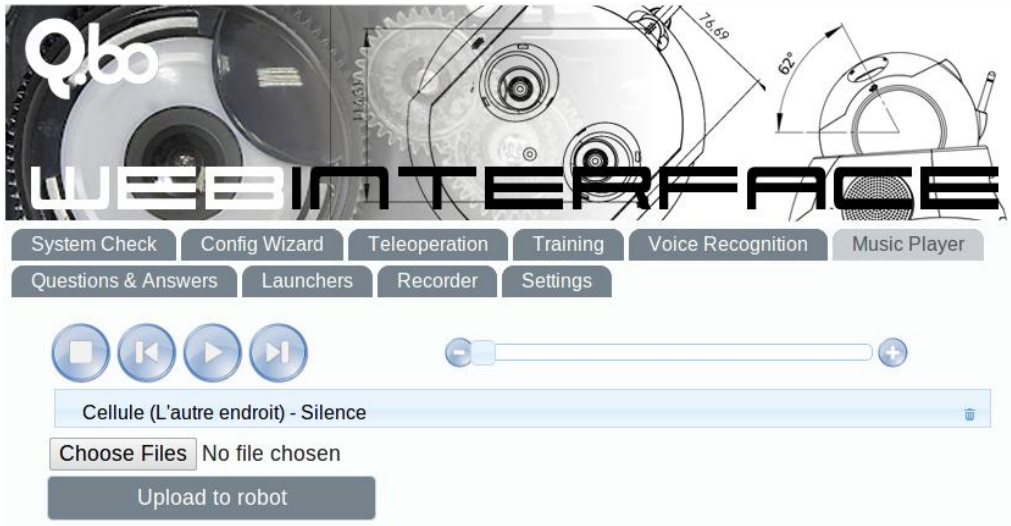
```
[sentences]
Hello {names}

[names]
Peter
Maria
```

Then you have to pass two tests in order to check if the acoustic model can recognize it. If you succeed with this you can then save it and you robot will recognize the new sentences.

1. The other option is for collaborating to improve the acoustic model by recording any sentence you want and writing its transcription. Just press the “rec” button, say something and when you finish click on “stop”. You will then get a transcription of what the acoustic model outputs of what you said. If it is correct, just press the “save” button, otherwise write exactly what you said and save it and the transcription will be uploaded to our servers.

2.8. Music Player

This is a web music player made for Q.bo. You can upload music from your PC which will be stored in Q.bo. Then, you can play this music from the robot’s loudspeakers.



2.9. Launchers

Android APK
This button launches the program that enables the communication with the android devices.

Play Music
This button launches the gesture music player.

Face Recognition
This button launches the face recognition interface.

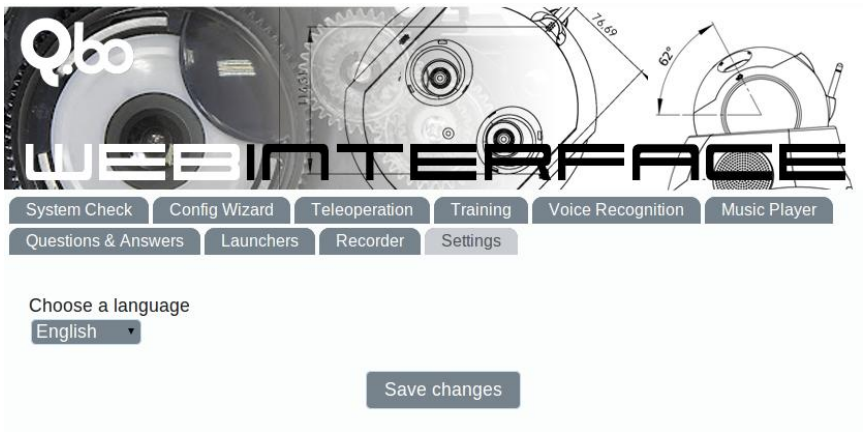
Random Move
This button makes Q.bo move randomly avoiding obstacles.

Start Conversation
This button launches the Questions & Answers program

2.10. Recorder

Through the Recorder tab you will be able to remotely record what Q.bo sees and listens, and also play it in any moment and see it through the Web browser.

2.11. Settings



In “Settings” you can set the language of the Webi.

3. Contact us

You can find an answer to many of your questions about Q.bo in the FAQ list from the following link: www.thecorpora.com
It is also possible to contact directly with the technical support team in the following e-mail direction: technical_support@thecorpora.com

There is also a community called OpenQbo in which you can find all the information regarding to the construction and programming of Q.bo, with a documentation of the software, a wiki and a forum. You can also read more about the already developped Q.bo applications:
www.openqbo.org