



- [Home](#)
- [Software](#)
- [Hardware](#)
- [Qbo Apps](#)
- [Forum](#)
- [Download](#)
- [Contribute](#)
- [Support](#)
- [Links](#)
- [Users](#)

qbo_arduqbo

1. General ROS Package Info

Author(s):
Miguel Angel Julian Aguilar
Type:
ROS Package
Dependencies:
qbo_cereal_port
Description:
This package is the ROS serial driver for the controller boards of Q.bo
Created:
2012-04-09
License:
GPLv2, LGPLv2.1
Tags:
[serial driver](#),
[driver](#)

2. 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.
The qbo_arduqbo node is launched at start up using the qbo_dynamixel_with_sensors.launch launch file.

3. 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.

3.1 Base Controller

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

Published Topics

~**odom** (nav_msgs::Odometry [http://web.archive.org/web/20160826132340/http://www.ros.org/doc/api/nav_msgs/html/msg/Odometry.html])

Topic were the robot position in the world is published. It uses sensors on the wheels to know the position of the robot in the external world with respect to its initial position.

The topic name can be changed with a [ROS parameter](#)

Command line example: print the odometry message

```
rostopic echo /odom
```

Python example: printing x et y position of the robot

```
#!/usr/bin/env python
import roslib; roslib.load_manifest('your_manifest')
import rospy

from nav_msgs.msg import Odometry

def callback(data):
    rospy.loginfo("odometry position: %f, %f"%(data.pose.pose.position.x, data.pose.pose.position.y))

rospy.init_node('print_odom')
rospy.Subscriber("/odom", Odometry, callback)
rospy.spin()
```

~**tf** (tf::tfMessage [<http://web.archive.org/web/20160826132340/http://ros.org/doc/groovy/api/tf/html/msg/tfMessage.html>])

Gives you the position of the robot in the world as a tf::tfMessage [<http://web.archive.org/web/20160826132340/http://ros.org/doc/groovy/api/tf/html/msg/tfMessage.html>]. Used by the odometry.

Command line example:

```
rostopic echo /tf
```

Subscribed Topics

~**cmd_vel** (geometry_msgs::Twist [http://web.archive.org/web/20160826132340/http://www.ros.org/doc/api/geometry_msgs/html/msg/Twist.html])

Topic were the Twist messages must be sent to move the robot base. The robot will move toward the direction contained in the message for 1 second. If another order is sent during this time, the previous order is canceled.

A Twist message is generic in ROS navigation and contains 3 linear speeds in m/s and 3 angular speeds in rad/s. As Q.bo is a wheeled platform, only the x linear speed and the z angular speed are used. The robot uses a PID controller to set the speeds to each wheel.

The topic name can be changed with a [ROS parameter](#)

Command line example: go forward

```
rostopic pub -1 /cmd_vel geometry_msgs/Twist [1,0,0] [0,0,0]
```

Service

~**qbo_arduqbo/stop_base** (qbo_arduqbo::BaseStop)

Service stopping the base.

Command line example:

```
rosservice call /qbo_arduqbo/stop_base 'test' true
```

3.2 Battery Controller

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

The *level* field gives you the battery level in volts (approx values for a standard Pro Evo Q.bo : max = 14.5, min = 12) and the *state* fields gives you information about the robot's status.

The state number has to be converted into a 6 bits binary number.

The 3 first bits are the battery state : 000 means that the battery is fully discharged, 001 means it is charging at constant current, 010 means that it is charging at constant voltage, 011 means that it is fully charged and 100 means that it is discharging.

The 4st bit is the external power state : plugged if 1, unplugged if 0

The 5st bit is the PC state: PC on if 1, PC off if 0

The 6st bit is the Q.Boards power: A value of 1 means that the Q.Boards power is on. A value of 0 means that it is off. It should be always on.

State decoding example:

```
If you read state 15:
15 in binary is 001111
001 : battery charging at constant current
1 : external power plugged
1 : PC on
1 : Q boards powered
```

Published Topics

~**battery_state** (qbo_arduqbo::BatteryLevel)

Command line example: print battery level and status

```
rostopic echo /battery_state
```

The topic name can be changed with a [ROS parameter](#)

3.3 IMU Controller

The IMU controller activates the sending of the IMU sensor readings in a ROS topic. The IMU sensor is on the Q.board4 and must be connected to the Q.board1

Published Topics

~imu_state/data (sensor_msgs::Imu [http://web.archive.org/web/20160826132340/http://www.ros.org/doc/api/sensor_msgs/html/msg/Imu.html])

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 [http://web.archive.org/web/20160826132340/http://www.ros.org/doc/api/std_msgs/html/msg/Bool.html])

Indicates if the inertial sensors are calibrated or not. A message is posted on this topic each time a calibration is finished.

Services

~imu_state/calibrate (std_srvs::Empty [http://web.archive.org/web/20160826132340/http://www.ros.org/doc/api/std_srvs/html/srv/Empty.html])

This service calibrates the inertial sensors of Q.bo. The robot should be on a flat surface and remain static during calibration.

Calibration of the IMU sensor is needed because gyroscope drift over time.

Command line example: calibrate inertial sensors

```
rosservice call /imu_state/calibrate
```

3.4 Joint Controller

The joint controller activates the movement of the servos of the robot. It means the neck (head_pan_joint for right-left and head_tilt_joint for up-down) and eyelids (left_eyelid_joint and right_eyelid_joint) of Q.bo.

If a new order is sent while an older one is not finished, the older is canceled, so you can do reactive control.

The goal position is a number in radian between the minimum and the maximum value for this servomotor.

The sensors are configured with a pair of ROS parameter described in the [parameters section](#) of this node.

Published Topics

~qbo_arduqbo/head_pan_joint (qbo_arduqbo::motor_state)

Topic containing the data about the head_pan (left-right) motor.

~qbo_arduqbo/head_tilt_joint (qbo_arduqbo::motor_state)

Topic containing the data about the head_tilt (up-down) motor.

Subscribed Topics

~cmd_joints (sensor_msgs::JointState [http://web.archive.org/web/20160826132340/http://www.ros.org/doc/api/sensor_msgs/html/msg/JointState.html])

Topic were the sensor_msgs::JointState [http://web.archive.org/web/20160826132340/http://www.ros.org/doc/api/sensor_msgs/html/msg/JointState.html] messages must be send to move the servos of the robot.

sensor_msgs/JointState is a classical ROS type to control servomotors. It contains 4 tables. The first table contains the names of the servos to command. The second one contains the goal positions. The third one contains velocity targets. The last one torque commands. Each of the three last tables should contain as much values as the number of names in the first table or be empty.

For Q.bo servomotors, only the position order is taken in account. A positive order on the pan motor will drive the head to the left. A positive order on the tilt motor will move the head down.

Command line example: move the head to the up and left.

The first table is a fake header. Be careful not to put spaces inside the tables.

```
rostopic pub /cmd_joints sensor_msgs/JointState [0,0,'test'] ['head_pan_joint','head_tilt_joint'] [0.5,-0.3] [] []
```

The topic name can be changed with a [ROS parameter](#)

Services

~qbo_arduqbo/unlock_motors_stall (std_srvs::Empty [http://web.archive.org/web/20160826132340/http://ros.org/doc/api/std_srvs/html/srv/Empty.html])

Service allowing to unlock the stall of the neck motors

Command line example:

```
rosservice call qbo_arduqbo/unlock_motors_stall
```

~qbo_arduqbo/head_tilt_joint/torqueEnable (qbo_arduqbo::TorqueEnable)

Service allowing to enable or disable the torque of the tilt motor.

Command line example: enable the torque of the tilt joint

```
rosservice call qbo_arduqbo/head_tilt_joint/torqueEnable true
```

~qbo_arduqbo/head_pan_joint/torqueEnable (qbo_arduqbo::TorqueEnable)

Service allowing to enable or disable the torque of the pan motor.

3.5 LCD Controller

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

Warning, the LCD screen is automatically updated with the PC state and the IP or hostname every second, and your message will disappear during this update.

Subscribed Topics

~cmd_lcd ([qbo_arduqbo::LCD](#))


The messages sent to this topic are forwarded to the LCD screen of Q.bo. There are 20 characters on each line, characters from the 21st will be skipped.

Only the first two lines of the LCD can be changed by the PC.

To write on the second line, your message should start with 1 (this caracter will not be displayed)

Command line example: print 'hello world' on the first line and 'I am alive' on the second one

```
rostopic pub /cmd_lcd qbo_arduqbo/LCD [0,0,'test'] "hello world"
rostopic pub /cmd_lcd qbo_arduqbo/LCD [0,0,'test'] "1I am alive"
```

 **Fix Me!** : The text is correctly set on the second line, but one caracter remains from the previous message (if previous message was abcdefghijklmnopqrst and I print '1hello', hellof is displayed. Workaround : send messages with spaces at the end.

3.6 Mics Controller

Gets the noise level of the three microphons of Q.bo

Published Topics

~mics_state ([qbo_arduqbo::NoiseLevels](#))

m0 is the right microphon

m1 is the front microphon

m2 is the left microphon

Command line example: print noise levels

```
rostopic echo /mics_state
```

3.7 Mouth Controller

Q.bo's mouth is a LED matrix of 4 lines of 5 LEDs (so 20 LEDs in total). The LEDs can be controlled one by one. The mouth is automatically activated when Q.bo speaks.

Subscribed Topics

~cmd_mouth ([qbo_arduqbo::Mouth](#))

This message contains a header and a list of 20 integers that should be 0 (LED off) or 1 (LED on), line by line.

Command line example: make Q.bo smile

```
rostopic pub -1 /cmd_mouth qbo_arduqbo/Mouth [0,0,'test'] [0,0,0,0,0,1,0,0,0,1,0,1,1,1,1,0,0,0,0,0]
```

3.8 Nose Controller

The nose of Q.bo is a RGB LED. The nose controller allows to chose the color of this LED.

Subscribed Topics

~cmd_nose ([qbo_arduqbo::Nose](#))

The nose message contains a header and an integer corresponding to the nose color: 0 for off, 1 for red or 2 for green.

Command line example: light the nose

```
rostopic pub -1 /cmd_nose qbo_arduqbo/Nose [0,0,'test'] 1
```

3.9 Distance Sensors Controller

The distance sensor controller manages all the distance sensors, infrared and ultrasounds. Q.bo has one vertical infrared distance sensor in front of it to check the distance to the floor (and avoid jumping from a table) and two horizontal ultrasonic sensors, one before each wheel to check for walls.

The distance sensors controller publishes a set of sensor_msgs::PointCloud [http://web.archive.org/web/20160826132340/http://www.ros.org/doc/api/sensor_msgs/html/msg/PointCloud.html] 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.

Subscribed Topics

~distance_sensor_state/floor_sensor (sensor_msgs::PointCloud [http://web.archive.org/web/20160826132340/http://www.ros.org/doc/api/sensor_msgs/html/msg/PointCloud.html])

Topic containing the distance seen by the floor sensor (vertical infrared distance sensor). For a robot on the floor, the distance is around 0.24 (in meters).

Command line example: read the floor sensor

```
rostopic echo /distance_sensors_state/floor_sensor
```

~distance_sensor_state/front_left_srf10 (sensor_msgs::PointCloud
[http://web.archive.org/web/20160826132340/http://www.ros.org/doc/api/sensor_msgs/html/msg/PointCloud.html])

Topic containing the distance seen by the left ultrasonic sensor. Maximum range is approx. 1 meter on the standard Evo Pro version.

~distance_sensor_state/front_right_srf10 (sensor_msgs::PointCloud
[http://web.archive.org/web/20160826132340/http://www.ros.org/doc/api/sensor_msgs/html/msg/PointCloud.html])

Topic containing the distance seen by the right ultrasonic sensor. Maximum range is approx. 1 meter on the standard Evo Pro version.

3.10 Diverse

Useful features

Services

~qbo_arduqbo/test_service (qbo_arduqbo::Test)

Test service giving you information about the connection with the different pieces of hardware.

This service is also useful to know if the arduqbo node is already started.

Python example: Before trying to communicate with arduqbo:

```
rospy.wait_for_service('qbo_arduqbo/test_service')
```

C++ example: Before trying to communicate with arduqbo:

```
ros::service::waitForService("qbo_arduqbo/test_service", -1);
```

4. ROS Messages

4.1 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 PC state.

The content of *BatteryLevel.msg* is:

```
Header header
float32 level
uint8 state
```

4.2 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.

4.3 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
```

4.4 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
uint8[20] mouthImage
```

4.5 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 3 different colors.

The content of *Nose.msg* is:

```
Header header
uint8 color
```

4.6 motor_state

The ROS message `motor_state`, is defined in the `motor_state.msg` file located in the `msg` folder. `qbo_arduqbo` publishes it.

It contains all useful data for the servomotors

The content of *motor_state.msg* is:

```
Header header
int32 id          # motor id
int32 goal        # commanded position (in encoder units)
int32 position    # current position (in encoder units)
int32 error       # difference between current and goal positions
int32 speed       # current speed (0.111 rpm per unit)
float64 load      # current load - ratio of applied torque over maximum torque
float64 voltage   # current voltage (V)
int32 temperature # current temperature (degrees Celsius)
bool moving      # whether the motor is currently in motion
```

5. ROS Services

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

5.1 Test

Content of the `Test.srv` file:

```
---
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.

5.2 BaseStop

Content of the `BaseStop.srv` file:

```
string sender
bool state
---
```

The `BaseStop` service is used to stop the base.

5.3 TorqueEnable

Content of the `TorqueEnable.srv` file:

```
bool torque_enable
---
```

The `TorqueEnable` service is used to enable the torque of the neck servomotors.

6. 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`)

parameter description

~timeout2 (float, default: `0.01`)

parameter description

~baud1 (integer, default: `115200`)

parameter description

~baud2 (integer, default: `115200`)

parameter description

~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. Not used in the standard Pro Evo version.

~uncontrolledservos (dictionary, default: {})

This a dictionary that sets the configuration of the servos whose speed is not needed to be controlled

~dynamixel servo (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.

6.1 Servos dictionary

The elements of the parameters **~controlledservos**, **~uncontrolledservos** and **~dynamixel servo** 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},
}
dynamixel servo: {
  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},
}
```

The two eyelids servos are defined in the parameter **~uncontrolledservos** and the two neck dynamixel servos in the parameter **~dynamixel servo**. 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
- **ticks** (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.

6.2 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:

6.2.1 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

6.2.2 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

6.2.3 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

6.2.4 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

6.2.5 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

6.2.6 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

6.2.7 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

6.2.8 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

6.2.9 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 were the distance sensors are configured. Following is explained with more detail the element contents.

6.2.9.1 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 },
      }
    }
  }
}
```


7. How to install

8. How to use

You could leave a comment if you were logged in.

Hide/Show