

gb robotics

ရာတာ User manual

Please read these instructions before use. Do not discard: keep for future reference. www.qbrobotics.com

Dear customer

Thank you for purchasing our product. To receive more complete service, please visit our website <u>www.qbrobotics.com</u>

Designed to be controlled like conventional servomotors, the qbmove can be changed in position and stiffness and so it can substitute small conventional servomotors used for constructing any mechanism or robotic system.

The qbmove advanced is the first affordable, muscle-like building block for constructing soft robots. An actuator that is small and lightweight, the qbmove is unique in its ability to endow robots with natural motion, i.e. that resembles the elegance and deftness of human motion, a radical contrast to the traditional rigid and mechanical motion of robots built with common servo-mechanisms.

The qbmove can tuned to be soft or rigid, depending on the task at hand. If the robot needs to guarantee safety during impact, the actuator can be set to be soft. While if precision is the greater need, the actuator can be set to be more rigid. Optimizing energy or speed performance can be achieved by alternating between soft and rigid settings. These combinations can be applied to tasks that are very complex, e.g. grasping or throwing objects, hammering, drawing, and so on.

The product before being packed has been tested with care.

Release 2.20 22/09/2020

Summary

1	Abc 1 1	out this	s document	3 3
	1.1	Symb	ols and designations	ך ג
2	Safe	⊃tv		л Л
2	2.1	Safety	y instructions	4
	2.2	Enviro	onmental conditions	4
	2.3	EC Di	rectives on product safety	5
3	Intr 3.1	oduct What	ion to qbmove kit 's in the box?	6 7
	3.2	Techr	nical data	8
	3.2.	1	Agonistic/Antagonistic VSA1	2
	3.2.	2	Mathematical model1	.4
	3.3	Electr	rical connections1	5
	3.3.	1	Qbally1	.5
4	Me	chanic	al assembly1	.7
	4.1	Flat F	lange assembly1	.9
	4.1.	1	Assembly sequence	0
	4.1.	2	ERNI cable routing in the Flat Flange	2
	4.2	Base	Flange assembly2	3
	4.2.	1	Assembly sequence 2	4
	4.3	C Flar	nge assembly2	5
	4.3.	1	Assembly sequence	6
	4.3.	2	ERNI cable routing in the C Flange2	9
	4.4	Doub	le Flat Flange assembly	0
	4.4.	1	Assembly sequence 3	1
	4.5	Gripp	er assembly3	3
	4.5.	1	Assembly sequence	4
	4.6	Tool t	tips3	5
	4.6.1		Payload evaluation and assembly examples	5
	4.6.	2	Distances between flange and actuator 3	7
5	Sof	tware		8
	5.1	Instal	ling the drivers	9
	5.2	qbmc	ove GUI	9
	5.2.	1	Installation procedure 3	9

	5	5 2	2	Main Lavout 4	n
	-		2	Pacie Tab	о 2
	3).Z.	5	DdSIC TdD	2
	5.3		Simul	ink package4	4
	5	5.3.	1	Installation4	4
	5	5.3.	2	Using the library4	4
	5	5.3.	3	Delta robot example	1
	5.4		qbAP	I software protocol	2
	5	5.4.	1	Compiler Installation5	2
	5	5.4.	2	Integrating the functions	3
	5	5.4.	3	Basic Functions	3
	5	5.4.	4	Code Examples	8
	5.5		ROS		1
	5	5.5.	1	Installation	1
	5	5.5.	2	Usage	2
	5	5.5.	3	Delta robot example	0
	5	5.5.	4	ROS packages overview7	2
6	Т	٢o	ublesh	looting7	4
	6.1		Qbmo	ove output shaft doesn't move smoothly7	5
	6.2		One o	or more qbmoves don't activate7	5
	6.3		Blue I	ED is off when the robot is powered7	6
	6.4		White	e LED is off when you use the robot7	6
	6.5		Clicki	ng on "Scan Ports" on GUI results in no port shown7	6
7	C	Con	nmissi	oning and Maintenance7	8
	7.1		Comr	nissioning7	9
	7.2		Main	tenance and warranty7	9
8	A	٩р	pendix		0
	8.1 VSA p		VSA p	apers 8	0
	8.2		qbmc	ve papers8	1
9	F	Rec	ord of	documented revisions	2

1 About this document

This documentation serves for safety-relevant operations on and with the servo-actuators. It contains safety instructions which must be observed.

The user should assume all responsibility for any accident caused by their careless handling of the product. Attention must be paid to the following safety instructions. Please read through this user's guide and make sure you fully understand all the instructions before assembling and operating this product.

The examples and diagrams in this manual are included solely for illustrative purpose. Because of the many variables and requirements associated with any installation, qbrobotics s.r.l. cannot assume responsibility or liability for actual use based on the examples and diagrams.

No patent liability is assumed by qbrobotics s.r.l. with respect to use of information, circuits, equipment, or software described in this manual.

Reproduction of the contents of this manual, in whole or in part, without written permission of qbrobotics s.r.l., is prohibited.

1.1 Using this document

The documentation must always be complete and in a perfectly readable state.

Keep the document accessible to the operating and, if necessary, maintenance personnel at all times.

Pass the document to any subsequent owner or user of the product.

1.2 Symbols and designations

	WARNING : identifies information about practice or circumstances that can lead to personal injury. Attentions help you identify a hazard, avoid a hazard, and recognize the consequence.
IMPORTANT	Identifies information that is critical for successful application and understanding of the product.

2 Safety

2.1 Safety instructions



IMPORTANT SAFETY INSTRUCTIONS – Read Before Using!

- Operational, maintenance, and service requirements are covered in the instruction manual. Read the entire instruction manual before device use.
- Check that all the contents is intact after removing it from the packaging.
- Keep away from children and pets. Always set off or unplug when not in use.
- Never use aerosol products, petroleum-based lubricants or other flammable products on or near the actuators.
- Do not use any damaged power cable, plug, or loose outlet. It may cause fire or damage to people.
- Do not use if damaged or defective. Do not disassemble the actuators.
- Do not insert any objects between moving parts.
- Do not place your face too close to the robot.
- Disconnect the power supply before cleaning or maintenance.
- Disrespect of these precautions can affect safety of the device.

2.2 Environmental conditions

- Select the installation location so that clean dry air is available for cooling the motor and has unobstructed access to flow around the drive.
- Do not keep or operate the actuators in a place of high temperature or humidity.
- Select a supply voltage that is within the defined tolerance range.
- This product is not waterproof. Never operate the product in a wet place.
- For indoor use only.

2.3 EC Directives on product safety

- The following EC directives on product safety must be observed.
- If the product is being used outside the UE, international, national and regional directives must be also observed.

Machinery Directive (2006/42/EC)

Because of their small size, no serious threats to life or physical condition can normally be expected from electric miniature drivers. Therefore, the Machinery Directive does not apply to our products. The products described here are not "incomplete machines", so installation instructions are not normally issued by qbrobotics.

Low Voltage Directive (2014/35/EU)

The Low Voltage Directive applies for all electrical equipment with a nominal voltage of 75 to 1500 V DC and 50 to 1000 V AC. The products described in this device manual do not fall within the scope of this directive, since they are intended for lower voltages.

3 Introduction to qbmove kit

This chapter shows what the qbmove kit includes, all the technical information about the qbmove Advanced and its mechanical connection.

Following, the electrical connections between the devices.

6

3.1 What's in the box?



Figure 3-1 ① qbmove Advanced; ② Flat Flange; ③ Base Flange; ④ C-Flange; ⑤ Handle; ⑥ qbgripper; ⑦ qbally; ⑧ ERNI cable 200mm; ⑨ ERNI cable 300mm; ⑩ ERNI cable 1500mm; ⑪ USB cable; ⑫ User manual.

Screws and nuts are provided with each kind of flange:

- Flat Flange: n°8 screws M2x8 ISO1207; n°4 nuts M3 ISO4032 A2; n°4 screws M3x8 ISO4762;
- Base Flange: n°8 screws M2x8 ISO1207; n°4 nuts M3 ISO4032 A2; n°3 screws M3x10 ISO10642;
- C-Flange: n°16 screws M2x8 ISO1207;
- Other components: n°8 screw M3x10 ISO4762; n°8 nuts M3 ISO4032 A2.

Make sure that the following items are included in the package. If there are any items missing, contact the supplier.

3.2 Technical data

Mechanical and electrical characteristics of the qbmove Advanced:



Figure 3-2 qbmove Advanced. (1) output shaft; (2) daisy-chain RS485; (3) USB port.

operating data				
(quar	(unit)	(value)		
mechanical				
Continuous C	utput Power	[W]	33	
Nominal	Torque	[Nm]	5,5	
Nomina	l Speed	[rad/s]	5,5	
Peak T	orque	[Nm]	6,8	
Maximu	n Speed	[rad/s]	6,33	
Maximum	Stiffness	[Nm/rad]	83,5	
Minimum	Stiffness	[Nm/rad]	0,5	
Nominal Stiffness	no load	[s]	0,25	
Variation Time	max torque	[s]	0,25	
Maximum El	astic Energy	[J]	0,88	
Maximum	Hysteresis	[°]	5	
Maximum Doflaction	Max stiffness	[°]	6	
	Min stiffness	[°]	50	
Active Rota	tion Angle	[°]	±180	
Angular Resolution		[°]	360/32768	
Wei	ght	[kg]	0,45	
	electrical			
Voltage	Voltage Supply			
Nominal	[A]	1,8		
Maximum Current		[A]	3	
Starting Current		[A]	13	
control data				
Nominal cu	rrent (USB)	[A]	0,26	
Electrical protocol		RS485/USB	-	





operating data			
(quantity)	(unit)	(value)	
Sensor a – Motor 1 Posi	tion		
Resolution	[°]	360/32768	
Range	[°]	0-360	
Sensor b – Motor 2 Posi	tion		
Resolution	[°]	360/32768	
Range	[°]	0-360	
Sensor c – Motor 1 Curr	rent		
Resolution	[A]	5/1638	
Range	[A]	5	
Sensor d – Motor 2 Current			
Resolution	[A]	5/1638	
Range	[A]	5	
Sensor e – Input Voltage			
Resolution	[V]	25/1638	
Range	[V]	0-25	





The qbmove is a back drivable actuator. When the robot is off, a sufficiently high external load can move the output shaft.

Do not use the Peak torque more than few seconds.

Dimensions of the flanges.















Figure 3-8 Examples of connection.

3.2.1 Agonistic/Antagonistic VSA

The qbmove Advanced embeds the features of a servo motor and, moreover, the possibility of adjusting the output shaft stiffness.

The Figure 3-9 shows a scheme of the agonistic/antagonistic VSA principle, implemented in our actuator. Basically, there are two motors connected at the output shaft by non-linear springs, so the output position "x" and the stiffness " σ " depend on the motors' positions "q₁" and "q₂", on the torque " τ " and on the mathematical model of the system.



Figure 3-9 Agonist/Antagonistic VSA model.

All these quantities are each other connected, then, to evaluate a proper mathematical model experimental data are needed.

Using a pendulum structure, a sinusoidal movement with different frequencies, and different stiffness presets, the obtained results can be synthetized in the graphic below; where the deflection " δ " is the difference between the angular position of the output shaft and the equilibrium position $\delta = x - (q_1+q_2)/2$ and the quantity $(q_1 - q_2)/2$ is the stiffness preset.



Figure 3-10 Experimental Torque Deflection Characteristic.

Using fitting functions (see the next paragraph) the relations between deflection, torque and stiffness are represented by the following graphics.



Figure 3-11 Torque - Deflection characteristic.



Figure 3-12 Torque - Stiffness characteristic.

Moreover, also considering the motors' speed, our actuator has a 3d workspace. The figures below show the Torque – Speed characteristic (Figure 3-13) and the VSA workspace (Figure 3-14).



Figure 3-13 Torque - Speed characteristic.



Figure 3-14 Torque - Speed - Stiffness Workspace.

3.2.2 Mathematical model

The following mathematical functions and their parameters describe the mathematical model of our agonistic/antagonistic VSA.

			value	Unit
q_1 x q_2	(🕤) (🕤)	k1	0,0026	[Nm]
		<i>a</i> 1	8.9995	[1/rad]
(motor 1) (output motor 2)		<i>k</i> ₂	0.0011	[Nm]
Silait Mark	× ×	<i>a</i> ₂	8.9989	[1/rad]
line		2		

equilibrium point	$X_e = \frac{(q_1 + q_2)}{2}$
output stiffness	$\sigma = a_1 k_1 \cosh(a_1 (x - q_1)) + a_2 k_2 \cosh(a_2 (x - q_2))$
output torque	$\tau = k_1 \sinh(a_1(x - q_1)) + k_2 \sinh(a_2(x - q_2))$
elastic energy	$H = \frac{k_1 (\cosh(a_1(x-q_1)) - 1)}{a_1} + \frac{k_2 (\cosh(a_2(x-q_2)) - 1)}{a_2}$

3.3 Electrical connections

In this guide you will find explanation regarding power connection of a single qbmove units as well as qbmove chains. Power connection is made using the qbally component (seeFigure 3-15).



Figure 3-15: qbAlly1



The supply voltage must be 24V!

3.3.1 Qbally

The qbally is the simplest qbally available. It provides a handily interface between your power supply and the 6 poles ERNI cable used to create daisy chain connection between qbmoves units. With the qbally you can power a single qbmove unit (Figure 3-16) or a short chain (up to 5) of qbmoves as shown in Figure 3-17

The qbmove is provided of two LEDs:

- White LED: it indicates that the logic circuit is powered;
- Blue LED: it indicates that the power circuit is powered.



Figure 3-16: Connection to a qbMove

IMPORTANT

If you connect the qbmove to the power supply and the USB cable isn't connected, both LEDs are on. Because to electrically power the power circuit, it also means to power the logic one.

IMPORTANT Each qbmove you connect to your system **must have a unique ID**.



Figure 3-17: Serial connection of four qbmoves using a qbally1

It is advisable not to connect more than four qbmoves for each chain, due to the current capacity of the ERNI cable.

4 Mechanical assembly

This chapter provides the user with the required procedures needed to assemble different kind of mechanical connection between qbmoves and, furthermore, some technical advice for a correct use of the platform.

ob robotics

Encoding of components for mechanical assembly



Figure 4-4:Parts from left to right - 2001, 3001, 3002



Figure 4-3: Parts from left to right - 1001, 1002, 1003 ,1004



Figure 4-2: Parts from left to right - 1005, 1006



Figure 4-1: Parts from left to right - 3003, 3005, 3007, 3008

Assembly code	Description	Unified code
3001	Bearing 25x32x4	MR 6705
3003	Screw M2x8	ISO 1207 – UNI 6107
3005	Screw M3x8	ISO 4762 – UNI 5931
3007	Screw M3x10 SV	ISO 10642
3008	Nut M3	UNI 5588 – DIN 934 (PG)

Table 4-1: Unified	l components	codes
--------------------	--------------	-------

Required tools for the assembly are listed below.

- Flathead screwdriver
- 2 mm Allen wrench
- 2.5 mm Allen wrench

4.1 Flat Flange assembly





q-brobotics

Figure 4-5: Two qbMoves connected using a flat flange

Required components:



Figure 4-6: Required components for a flat flange

4.1.1 Assembly sequence

Use of the CORE component.



Figure 4-8: Anchors Opening

In Figure 4-8 arrows represent points to push in order to open the locking mechanism.



Figure 4-7: Flat Flange

Assemble the 1002 parts with the 1001 central core, using the two pins as reference. Make sure to the orientation of the two components is the same and with the anchor teeth on the same side of the counterbores of the central component.

Assemble the Flat flange on the output pulley of the qbmove, by the eight screws 3003, as shown in Figure 4-9.



Figure 4-9: Flat Flange assembly procedure on a qbMove

Now a second qbmove can be connected to the first one by snapping the flanges, as shown in Figure 4-10.



Figure 4-10: Two qbMoves connected with a flat flange

4.1.2 ERNI cable routing in the Flat Flange

The ERNI cable, which allows the connection of 2 qbmoves, can be placed inside the flat flange in two ways; First one is shown in Figure 4-11.



Figure 4-11: First possible ERNI cable routing inside a flat flange

Another way is shown in Figure 4-12.



Figure 4-12: Second possible ERNI cable routing inside a flat flange

4.2 Base Flange assembly

This type of flange is used to connect one qbmove to your desired frame.



Figure 4-13: qbMove connected to a Base Flange

Required components:





4.2.1 Assembly sequence

Preparation of the Base Flange

Assemble the 1002 parts with the 1004 parts using two 3007 screws and two 3008 nuts, as shown in Figure 4-15.



Figure 4-15: Add-on assembly procedure

Assembly of the Base Flange



Figure 4-16: Base Flange

Assemble the two parts obtained with the 1001 central core, using the two pins as references. Make sure to the orientation of the two components is the same and with the anchor teeth on the same side of the counterbores of the central component.

Place the removable component from the side of the elastic anchors of the Core in correspondence of the first notches, as shown in Figure 4-16

Snap the Base flange on one of the free faces of the qbmove (Figure 4-18) then, using the provided holes, screw the entire assembly wherever you need.



Figure 4-18: Base Flange connected to a qbMove

4.3 C Flange assembly



Figure 4-17: qbMoves connected by a C-Flange

Required components:



Figure 4-19: Required components for a C-Flange

4.3.1 Assembly sequence

Preparation of a C Flange



Figure 4-20: Free pulley on qbMove assembly procedure

Assemble the 3001 bearing on the free 2001 pulley, on the opposite side of the eight treaded holes. Assemble the obtained component with the qbmove, to reveal the spot, remove the plastic bottom cap with a little flathead screwdriver.



Figure 4-21: First flange wing assembly procedure

Assemble the 1003 component on the free pulley by using eight 3003 screws. Assemble the component obtained before and the second 1003 component with the Core, using the two pins as reference.



Figure 4-22: C-Flange assembly procedure

Place the removable 1003 component in correspondence of the first notches, as shown in Figure 4-22. Assembly of the C Flange with the QB-move



Figure 4-23: qbMove C-Flange assembly procedure

Place the qbmove 2 in the desired position and then close the C Flange.

Move the qbmove ① in the desired "zero" position and then fix it with the eight 3003 screws. NOTE: with the two flanges screwed to the qbmove ① it is still possible to open the flange to detach or change the orientation of the qbmove ②.

4.3.2 ERNI cable routing in the C Flange

The ERNI cable can be routed inside the C Flange as shown in Figure 4-24.



Figure 4-24: ERNI cable routing inside the C-Flange

4.4 Double Flat Flange assembly

This type of flange is used to rigidly connect two qbmoves.



Figure 4-25: qbMoves connected by a Double Flat Flange

Required components:



Figure 4-26: Required components to assemble a double flat flange

4.4.1 Assembly sequence

Preparation of the Double Flat Flange.



Figure 4-27: Removable component

Join two 1002 components and fix each other with two 3005 screws and two 3006 nuts, as shown in Figure 4-27.

The obtained components will be the removable elements of the Double Flat Flange.



Figure 4-28: Double flat flange

Assemble the two obtained components with the Core, using the two pins as reference. Place the removable component in correspondence of the first notches of the flanges, as shown in Figure 4-28.

Assembly the Double flat flange with the qbmove.

With the double flat flange, you can rigidly connect two qbmove with different orientation. An example is shown in Figure 4-29.



Figure 4-29: qbMove double flat flange assembly
4.5 Gripper assembly

Required items.



Figure 4-31: Required components for a qbMove gripper



Figure 4-30: qbMove gripper

4.5.1 Assembly sequence

Assemble the 1005 component on a *Flat Flange* using four 3008 nuts and two 3007 screws, as shown in figure below.



Figure 4-32: Fixed finger assembly procedure

Assemble the fixed finger on the qbmove closing the *Flat Flange* and locking it by two 3007 screws, as shown in Figure 4-32. Make sure to orient the qbmove as shown, in order to have the electrical connectors as indicated.



Figure 4-33: Fixed finger qbmove assembly procedure

Assemble the second finger, component 1006, using the 3003 screws, as shown in Figure 4-33. The qbmove must be equipped with a rear pulley, as like in a *C* Flange mounting.



Figure 4-34: Assembled gripper

4.6 Tool tips

In this paragraph, you can find some basic advices to correctly assemble a robot using the qbmove kit, avoiding unwanted behaviors of the assembled robot and let the user using it safely. For a smart assembly, you must read with attention the chapter 2, primarily the technical data.



The qbmove is a back drivable actuator. When the robot is off, a sufficiently high external load can move the output shaft.

4.6.1 Payload evaluation and assembly examples

The payload of the assembled robot will depend on the actuator's torque, on the masses, on the distances and accelerations.

Before starting the assembly, you must decide which type of robot you want to build and make proper mechanical considerations about the workspace, the dynamics and the payload. A wrong assembly can drastically reduce the robot's payload when near the limits of the workspace. Instead, a different kinematic can help lifting of higher loads.

Figure 4-35 shows some schematic examples of assembly. Usually the axis of the first joint is vertically placed to circularly explore the workspace of your robot, then the other actuators can be assembled in different ways.



Figure 4-35: Different types of assembly examples

Using the flanges in the kit, you can make the joints of you robot. For example: Base Flange to fix the first qbmove to a frame; Flat Flange for first and fourth joint of (2) and second for (4) and (5); and C Flange for other joints.

IMPORTANT Before assembly, make a correct evaluation of the joint's torque based on the technical data and external loads.



Each qbmove you connect to your system must have a unique ID.

4.6.2 Distances between flange and actuator

When you assemble any robot, you must be careful about the distances between flanges and actuator's carter (Figure 4-36) because during the use, cables or some object can make interference with relative movement of the components.

To avoid malfunctions after assembly, you must manually check the movement of each joint to verify if they work properly.



Figure 4-36: Critical distances between qbmove's carter and C-Flange (first two images) and Flat Flange (last picture on the right)

IMPORTANT Before using the robot, you must check if each joint's range is free of obstacles.



Before assembly the devices, make sure that each qbmove is in zero position (see section 5.2)

5 Software

This chapter explains how to install the software, connect the actuators and use them.

The qbmove GUI will help you to test and use the single qbmove; A Simulink package also can be used to control the qbMoves; The qbAPI software protocol lets you build a custom program to use the devices you have; Afterwards you will see where to find ROS libraries and how to use them to create your project.

5.1 Installing the drivers

Independently from the chosen solution between GUI, Simulink package or qbAPI tools, it is necessary to download and install the drivers from FTDI's site.

Instructions vary between operative systems, but it is only necessary to install the drivers under MAC OSX or Windows.

- Mac: It is necessary to install the <u>FTDI serial drivers</u> to make possible the device is seen as serial port (i.e.: /dev/tty.usbserial-128).
- Windows: It is necessary to install the <u>FTDI serial drivers</u> to make possible the device is seen as serial port. Before using anyone of the software utilities provided, is good use to check on which COM serial port the device has connected. The supported ones are between COM1 and COM9. To change the port number, go under <u>Control Panel ></u> <u>Hardware and Sound > Device Manager.</u>

Open the <u>Ports (COM & LPT)</u> drop down menu, right click on the COM port and select <u>Proprieties</u>.

Select the <u>Port Settings</u> tab and then click on <u>Advanced</u>. Once in Advanced menu select from the drop-down menu a COM number between COM1 and COM9 and click on <u>OK</u>.

The device should now be recognized properly.

5.2 qbmove GUI

The qbmove GUI is an application useful to quickly test a qbmove and eventually to diagnose troubles about the hardware or software.

5.2.1 Installation procedure

• Mac:

If not already installed, it is necessary to install the FTDI serial drivers to make possible the device is seen as a serial port.

Double click on the App and use it. There is also a .dmg package. You can open it and move the application to your dock or the Application folder.**Windows:** To use the GUI is only necessary to run the *"qbtools.exe"* file

• Windows:

If not already installed, it is necessary to install the FTDI serial drivers to make possible the device is seen as a serial port. If the serial port is not seen on the GUI main window, it is possible that the system has recognized the COM port with a high COM number. The supported serial ports are from COM1 to COM9. To change the port number, go under Control Panel > Hardware and Sound > Device Manager.

Open the Ports (COM & LPT) drop down menu, right click on the COM port and select Proprieties.

Select the Port Settings tab and then click on Advanced. Once in Advanced menu select from the drop down menu a COM number between COM1 and COM9 and click on OK. The device should now be recognized properly.

• Unix: Before using the GUI, it is mandatory to add user to the dialout group. To do so, you

must execute:

sudo adduser user name dialout

where <code>user_name</code> is the username under which the GUI is used. Once this command is executed, it is necessary to log out and back in, for the changes to take effect.

5.2.2 Main Layout

qbtools				-
	✓ Scan Ports	Connect Not connected	qb robotics	Get Info
Basic	ices IDs present:			
	Current ID: Undefine	ed		
New ID:	1 ~	Set		
Activate	Undefined			
Position and Stiffness Position(degree):	Inputs 0	Zero		
Stiffness(%):	0			
	Get Measurements			
Mot1	Mot2 Get Currents	Link		
Curr1 (mA)		Curr2 (mA)		

Figure 5-1: qbMove GUI main window

Application is structured in one tab only:

• **Basic:** Basic commands. ID setting, activation of the board, measurements and currents reading, position and current inputs.

To use the qbMove with this application you have to click on "Scan Ports". If the qbMove is properly connected, you'll see the serial port in the little window on the left. Once the serial port is seen you could connect the device clicking on the Connect button. This operation, if successful, enables all of the buttons and you should see a green "Connected" text near the Connect button. The qbmoves must be powered with 24V power supply. (Figure 5-2).



Figure 5-2: Serial Port connection

"Get Info" button prints useful information about qbMove and its control board, in the blank window on the right side of the application. e.g.: firmware version, parameters values and position measurements.



5.2.3 Basic Tab

Basic			
Devic	es IDs present:	~	
	Current ID: Undefine	ed	
New ID:	1 ~	Set	
Activate	Activate Undefined		
Position and Stiffness In	nputs		
Position(degree): 0		Zero	
Stiffness(%):	0		
Get Measurements			
Mot1	Mot2	Link	
	Get Currents		
Curr1 (mA)		Curr2 (mA)	

Figure 5-3: qbMove GUI Basic tab

- **Devices IDs present:** Once the device, or devices (if connected in a chain), are connected clicking on *"Connect"*, a list of their IDs is going to be showed here. Selecting the desired ID from drop-down menu will make the application connect to that device and is then possible to use that specific device only;
- New ID: Default ID of the board is 1. Clicking on drop-down menu will show a list of available IDs (1 to 127). Clicking on "Set" button will set ID equal to the New ID selected;
- Activation: To use qbMove you first have to activate qbMove's drivers. You can do this by clicking on "Activate". The string next to the button will show board' state of activation.
- Inputs: Depending on the control modality set, which by default is *"Position"*, it is possible to move the drivers of the qbMove using sliders or Motor 1/2 Input fields. The corresponding groupBox relative to its control modality is activated/deactivated when the control modality is set (in advanced tab);

• Measurements and Currents: Clicking on "Get Measurements" and "Get Currents" will show encoder measurements for the first one and current measurements, for the second one. Encoder measurements are shown in encoder ticks, Currents in mA;

IMPORTANT Remember to power the qbMove, or the chain, before using it.

5.3 Simulink package

5.3.1 Installation

5.3.1.1 Download

The first necessary steps to use the qbMove with Simulink, are:

- 1. Download the <u>gbmove simulink</u> repository
- 2. Download the <u>qbAPI</u> repository

Before compiling be sure to have a folder tree like the following, i.e. rename the folders accordingly:

your_working_directory:

- qbAPI
- qbmove_simulink

5.3.1.2 The C++ compiler

This step is optional if you have already set a C++ compiler on MATLAB or whether you can exploit the default one depending on the current MATLAB and OS versions.

If you run `mex -setup C++` and it executes without errors, you are just fine. Otherwise you might need to install a C++ compiler on your system first, usually `g++` on Linux, `MSVC++` on Windows and `Xcode` on macOS.

If you have any trouble during this installation requirement, have a look at the MATLAB documentation to find the proper way to install a C++ compiler on your system (<u>https://it.mathworks.com/help/matlab/ref/mex.html</u>).

5.3.1.3 Compile the library

If the previous steps are fine, navigate under `~/your_working_directory/qbmove_simulink` in the MATLAB tree explorer and then execute the install script (either by executing `install` from the MATLAB shell or by running the `install.m` MATLAB script).

The script installs the library accordingly to your current MATLAB version and update the MATLAB path. If no error is prompted out, the library is correctly installed and you can start using it.

5.3.2 Using the library

5.3.2.1 Create a new simulink model

- 1. Click on the "Simulink Library" icon or type "simulink" in the Matlab Command Window.
- 2. Create a new model using: "File -> New -> Model"

3. In the new model go to: "Simulation -> Model Configuration Parameters". Under "Solver" select as *Type* "Fixed-Step", as *Solver* "ode1 (Euler)" or "discrete (no continuous states)" and as *Fixed-step size* type the delta_t in seconds. (e.g. if you want to retrieve positions and set new inputs every 5 milliseconds, type 0.005). Click "OK".

Every qbmove needs at least 1 millisecond to read and set new positions, so the minimum step time allowed is 1 millisecond multiplied by the number of qbmoves in the chain.

4. You should be able to see the various blocks under libraries in Simulink Library Browser. From here drag and drop the desired blocks onto your model.

5.3.2.2 The blocks

• qbmove

This block is the interface between your computer and the real qbMove. By default, you will see 4 input ports and 4 output ports. This means that the block is set to both receive information from the sensors and send new position reference and stiffness to the qbMove. If you double-click to the block, the Function Block Parameters will open and you will be able to set the parameters of the block.

The block inputs have the following functions:

- Handle: Is used to connect the "QB Move Init" block which stores the serial port
- Pos.1: This input depends on the Input modality set. It could have 2 values: pos.1 or eq.pos. The first one is used when the "Prime Movers" Modality is chosen, the second one is used when one between "Equilibrium Position and Stiffness Preset" or "Equilibrium Position and Stiffness Preset Percentage" modality is chosen.
- Pos.2: As the *pos.1* input, this input can change depending on which input modality is chosen. It could have 3 values: *pos.2, s.preset* and *stiff 0-100%*. The first one is used when the *"Prime Movers"* modality is chosen, the second one is used when *"Equilibrium Position and Stiffness Preset"* modality is chosen and the third one is used when the *"Equilibrium Position and Stiffness Preset Percentage"* modality is chosen.
- Activation: This input is used when the "Activation on Startup" checkbox is unchecked. If there is this input, a different from zero signal must be routed there if the device needs to be activated, vice versa if needs to be deactivate. Also, works at runtime and the devices can be activated or deactivated at runtime. If multiple devices are used, this input must receive an array as wide as the number of devices connected.

The block outputs have the following functions: **Pos.1**: Gives the position measurements of the first motor of the device.

- Pos.2: Gives the position measurements of the second motor of the device.
- Pos.L: Gives the position measurement of the qbMove shaft.
- Error: Debug output. A terminator block can be connected to it.

For what concerns the block parameters, the explanation follows below:

Block Parameters: Qb Move ×				
qb move (mask) (link) This block provides a interface to the Ob Move.				
It needs an handle from Qb Move Init block.				
Options				
ID				
[1, 2, 3]				
Communication Directions Both		•		
Input Type Prime Movers Positions				
Activation on Start UP				
Watchdog Timer 0.0 500.0 0.0	Unit © Degrees ○ Radians ○ TICK			
OK Ca	ancel Help Apply			

Figure 5-5: qbMove Simulink Block parameters

ID: If it is connected only one device, the ID could be put without brackets, otherwise the syntax is like in the picture.

Communication direction: Is used to show which ports are shown on the block. If it is on "Both" the block will have the ports like on the above image. If it is on "Tx", it will have only the output ports. If it is on "Rx", it will have only the input ports.

Input Type: This is used to decide what kind of inputs the device will receive on the input ports.

>	handle	pos.1	>
>	pos.1	pos.2	>
>	pos.2	pos.L	>

Figure 5-4: qbMove simulink block

Qb Move

- "Equilibrium Position and Stiffness Preset" This modality works only with the qbmove. It drives the shaft position and its stiffness. Both the inputs are in the measurement unit defined by the field "Unit"
- Equilibrium Position and Stiffness Preset Percentage" This modality works only with the qbmove. It drives the shaft position and its stiffness. The position is in the measurement unit defined by the field "Unit" and the stiffness is in percentage, from the minimum to the maximum of the device.
- "Prime Movers Positions" the inputs will be directly the motor positions.



It could be dangerous to use this modality with the qbmove. The motors and the cable can be seriously damaged and the device could work no more. Use this modality at your own risk.

Activation on Startup: If this checkbox is set, the block will have only three inputs and the device, or the devices, will activate when the simulation is started. Otherwise an "Activation" input port will show and a 0/1 signal should be routed to that input port.

Watchdog Timer: Setting this value to a value different from 0, will set a parameter on the device that is going to deactivate that device if it doesn't receive inputs in the time defined by the Timer.

Unit: This parameter defines the measurement unit of the inputs and outputs of the device. The Tick value is referred ad the encoder ticks of the sensors mounted on the device motors.

qbmove Init

When a Simulink scheme is made, this block is mandatory because it stores the communication port of the device. The output must be connected to the *"handle"* input of the QB Move block.



Figure 5-6: qbMove Init block

The serial port name varies between operative systems:

- WINDOWS: Under windows the devices are Communication Ports COM#. To set the port name, double click on the block and insert in the line edit 'COM#' with the right number of the serial port. The communication baudrate should be put equal to 2000000.
- UNIX: On Unix the devices are usually seen as /dev/tty.USBx. To set the port name, double click on the block and insert in the line edit '/dev/tty.USBx'

with the right number of the serial port. The communication baudrate should be put equal to 2000000.

 MAC OSX: On OSX the devices are usually seen as /dev/tty.usbserial-XX. To set the port name, double click on the block and insert in the line edit '/dev/tty.usbserial-XX' with the right number of the serial port. The communication baudrate should be put equal to 2000000.

qbmove – Get current

This block is used to retrieve currents from the device, or the devices, connected.

Clicking on the block will open the Options panel. It is possible to set the IDs of the devices connected, in the same way of the QB Move block. The input port, in order to retrieve the currents, must be connected to the output of the QB Move Init block. The outputs are the currents of motors of the devices.



Figure 5-7: qbMove Get Current block

qbmove - Get Curr and Meas

This block works the same way of the QB Move block, but with also the benefit of the QB Move – Get Current block. The motor measurements are all returned at the same time. With this block, there is no need to use the QB Move and the QB Move – Get Current block, it is all done with this one. The parameters of the block are the same of the QB Move one, so refer to that one for any explanation on those.



QB Move - Get Curr and Meas

Figure 5-8: qbMove Get Current and Measurements block

QB Pacer

<u>This is a mandatory block</u>. Every simulation schematic needs to have one. Is used make the simulation go as the same velocity as the devices connected. As you can see in the following example, is always used with a Clock to see if the commands of the simulation are going at the same velocity of the device, or slower.



49 22 settembre 2020

5.3.2.3 Example

In the main folder, you will find an example called "qbmove_example.slx". This is a simple configuration which you can use to test your qbmove.



Figure 5-10: qbMove control Simulink example

This example is structured for one device only, with ID equal to 65. As you can see in the above image on the left there are two constant inputs with a slider gain each, which can be tweaked while the simulation is running and you can see the device will move. In this case the input modality is chose to be *"Equilibrium Position and Stiffness Preset"* so, the example, as it is, will work only on a qbMove.

On the right there are 3 displays where you can read the values in the measurement unit chosen of the 3 sensors (depending on the device used, the value of the outputs changes. Refer to previous section for better explanation).

The error output is used to debug communication. It's not needed to make the simulation work, so you can leave a terminator block connected.

As you can see, the QB Pacer block is used to confront time with the clock of the simulation. The two times, the clock and the real time, are confronted in a scope. There should be two lines in it. If the simulation step is set correctly the two lines should be overlapped. If the two lines diverge, a bigger step size must be set.

In this configuration, the step size is set to 2 milliseconds and the computer should be able to run it in real time. This means that every 2 milliseconds you send a new reference position to the qbMove and the current position is read. Furthermore, a current reading is done and you can see the milliampere absorbed by each of the two motors.

Please remember that in a multi-device configuration (e.g. for the Delta Kit see 4.5 ID configuration), each qbrobotics device connected to your system must have a unique ID.

Indeed, if any device shares the same ID with another one on the same chain (all the qbmove devices have ID equals 1 when they are shipped from qbrobotics) communication issues will lead to unpredictable behaviors, i.e. no data sent or received can be trusted.

5.3.3 Delta robot example

When using the ROS packages with a Delta Kit, it is possible to exploit the specific aids provided by qbrobotics to help in a quicker kinematic structure control.

You can inspect the example by opening the Simulink model:

"/your_working_directory/qbmove_simulink/examples/delta/qbdelta_r2018.slx" which is valid from MATLAB r2018a on, while `qbdelta_r2017.slx` refers to previous MATLAB version.

5.3.3.1 Inverse kinematics and control inputs

A specific Delta Kit controller has not been implemented since each single qbmove controller can be exploited to control the whole system. Anyway, controlling a Delta kinematic structure only w.r.t. the joint space is not the best option for the end user.

For this reason, we have created the direct and inverse kinematics algorithms which allow a more user-friendly approach. Without digging too much into the implementation details, which is publicly available under *"/your_working_directory/qbmove_simulink/examples/delta/"*, the major improvements are the followings:

- By exploiting the direct and inverse kinematics of the Delta kinematic structure, the user can control the end-effector position w.r.t. the cartesian space (cf. the reference system details in 4.5 ID configuration).
- The 3D positioning of the end-effector can be more or less compliant to the surrounding environment by adjusting the stiffness of the three upper joints. This leads respectively to a less or more precise pursuing of the control reference, but the compliance can be exploited to achieve many tasks that could not have been done otherwise.

• The end-effector qbmove can be treated as a two-finger gripper with the advantage of exploiting the positioning and compliance control proper of qbmove devices. Indeed, the grip can be more or less strong w.r.t. the applied stiffness.

Moreover, following the example in the script "delta_waypoints.m", it is possible to send predefined waypoints in an infinite loop fashion, which can be used to create a simple static demo application. Later, when you are more confident with the whole system, you can create your own dynamic controller based on a similar approach.

5.3.3.2 ROS-style URDF model

In addition, in a way similar to the ROS rviz visualizer, it is possible to visualize the Delta Kit structure by exploiting the ROS Toolbox and the Robotics System Toolbox.

The URDF model is not exactly the same as the one provided in the ROS packages and indeed it has been simplified to properly work on MATLAB (you can inspect it by opening *"qbdelta.urdf"*, together with the *"delta-meshes"* directory).

Lastly, the "delta_show.m" script enable MATLAB to plot and visualize the URDF model and give a remote feedback on what is happening in the real system.

IMPORTANT It is worth noticing that this is just a state visualizer and not a simulator.

5.4 qbAPI software protocol

5.4.1 Compiler Installation

Before using the qbAPI on your system, if not already installed it is good use to install a c++ compiler. Below here are listed all the necessary instructions for each operative system.

- **Unix:** You should have both *gcc/g++* and *make* already installed.
- Mac OSX: Download XCode from Mac App Store, this will install gcc/g++ and make utility.
- Windows: Download MinGW and install it. Open MinGW Installation Manager, from the left panel select basic setup, then from the right panel select mingw32-base and mingw32-gcc-g++, then click on <u>Installation -> Apply Changes</u>. This will install the gcc/g++ compiler.

It is necessary to provide Windows the path to the executable binaries. To do so, go on <u>Control Panel>System and Security>System</u> and click on <u>Advanced</u> <u>System Settings</u>. Then select <u>Environment Variables</u> and look for <u>path</u>, select it and click edit.

In Windows 10 click on New and add a new row with the value <u>C:\MinGW\bin;</u>

 In previous Windows versions go to the end of Variable Value field add a ";" separator and append the path to binary folder for gcc, which is <u>C:\MinGW\bin;</u>

Download the make utility from here. Follow the installation instructions. When the installation procedure has ended you will need to add the binary path to the Environment Variables. To do that follow the previous steps. The binary folder for make utility is in C:\Program Files (x86)\GnuWin32\bin.

NOTE: if you have the command shell (CMD) already opened when performing the installation, you probably will need to close and re-open to use the tools.

5.4.2 Integrating the functions

In order to integrate the functions within your code, it is necessary to include the *commands.h* and *qbmove_communications.h* libraries and compile *qbmove_communications.cpp* with your code.

Then is only necessary to call the functions when it is needed.

All the functions have an explanation, in form of comments, in "*qbmove_communications.h*". Following here there will be a brief explanation of the functions and what are used for.

5.4.3 Basic Functions

In most cases these will be all the necessary functions that you will need to use the qbmove within your system.

5.4.3.1 RS485ListPorts

This function is used to retrieve the name of all serial ports connected to the computer. If no serial ports are found, a '0' value is returned. Otherwise the number of serial ports connected to the computer is returned.

The port name is used to create a file descriptor associated to that name. Arguments:

char list_of_ports[10][255] - List of ports connected

Example:

5.4.3.2 openRS485

This function, once a list of serial ports is defined, and a series of file descriptors are created, is used to open the serial port and allow communication with the device connected.

Arguments:

- comm_settings *comm_settings_t Structure containing info about communication settings
- const char *port_s String of the serial port path
- int BAUD_RATE Default baud rate of the communication. By default, is equal to 2MBaud

Example:

5.4.3.3 closeRS485

This function is used to close communication between the computer and the device. Is necessary to use it before a program is terminated.

Arguments:

comm_settings *comm_settings_t - Structure containing info about communication settings

Example:

```
comm_settings comm_settings_t;
openRS485(&comm_settings_t,"/dev/tty.usbserial-128");
//Doing stuff
closeRS485(&comm settings t);
```

5.4.3.4 commActivate

This function activates the motor drives (or the motor drive if only one present) of the device. It is necessary to activate the motor drives before using the device, otherwise the device will not work.

Arguments:

- comm_settings *comm_settings_t Structure containing info about communication settings
- int id The device's ID number
- char activate The activation value. 0x03 activates the board, 0x00 deactivates it

```
comm_settings comm_settings_t;
int device_id = 65;
openRS485(&comm_settings_t,"/dev/tty.usbserial-128");
commActivate(&comm_settings_t, device_id, TRUE);
closeRS485(&comm_settings_t);
```

5.4.3.5 commGetActivate

This function is used to retrieve the activation status of the board. It is mostly used after the commActivate function to see if the board was correctly activated or deactivated. It is needed an inactive program time if used right after the <u>commActivate</u>.

Arguments:

- comm_settings *comm_settings_t Structure containing info about communication settings
- int id The device's ID number
- char *activate Device activation status

Example:

5.4.3.6 commSetInputs

This function is used to give the device reference inputs to motors. The inputs are counted in encoder ticks. <u>Be aware that the inputs depend on the device used</u>. The abmove needs two input values. <u>Be careful to use this function with the abmove. The device can be severely damaged if the function is not used properly</u>. With that device is recommended to use the <u>commSetPosStiff</u> rather than the <u>commSetInputs</u>.

Arguments:

- comm_settings *comm_settings_t Structure containing info about communication settings
- int id The device's ID number
- short int inputs [] The array used to store the inputs to be sent to the devices

```
comm_settings comm_settings_t;
int device_id = 65;
short int inputs[2];
openRS485(&comm_settings_t,"/dev/tty.usbserial-128");
inputs[0] = 1000; inputs[1] = -1000;
commSetInputs(&comm_settings_t, device_id, inputs);
closeRS485(&comm_settings_t);
```

5.4.3.7 commSetPosStiff

This function is used to give the device reference inputs to motors. Is used only with a qbMove device. The two inputs are the output shaft position (in degrees) and the stiffness preset of the shaft. The stiffness preset goes from 0 to 30 degrees.

Arguments:

- comm_settings *comm_settings_t Structure containing info about communication settings
- int id The device's ID number
- short int inputs [] The array used to store the inputs to be sent to the devices

Example:

```
comm_settings comm_settings_t;
int device_id = 65;
short int inputs[2];
openRS485(&comm_settings_t,"/dev/tty.usbserial-128");
inputs[0] = 100; //Degrees
inputs[1] = 30; //stiffness preset
commSetPosStiff(&comm_settings_t, device_id, inputs);
closeRS485(&comm_settings_t);
```

5.4.3.8 commGetInputs

This function is used to retrieve the inputs given to the device.

Arguments:

- comm_settings *comm_settings_t Structure containing info about communication settings
- int id The device's ID number
- short int inputs[] The array used to store the input values to be taken from the devices

comm_settings int	<pre>comm_settings_t; device id = 65;</pre>
short int	inputs[2];
openRS485(&comm_set	<pre>tings_t,"/dev/tty.usbserial-128");</pre>
<pre>if(!commGetInputs(&c</pre>	<pre>comm_settings_t, DEVICE_ID, inputs)) : %d\t%d\n",inputs[0], inputs[1]);</pre>
else	
puts("Couldn't i	retrieve device inputs.");
closeRS485(&comm_set	tings t):

5.4.3.9 commGetMeasurements

This function is used to retrieve the encoder measurements from the device. The measurements returned are in encoder ticks. If you are using the qbMove, the constant to convert from ticks to degrees is 360/32768.

Arguments:

- comm_settings *comm_settings_t Structure containing info about communication settings
- int id The device's ID number
- short int measurements[] The array used to store the encoder measurements of the device connected

Example:

```
comm settings
                 comm settings t;
                   device id = 65;
int.
short int
                   measurements[3];
openRS485(&comm settings t, "/dev/tty.usbserial-128");
if (!commGetMeasurements (&comm settings t,
                                                        DEVICE ID,
measurements)) {
     printf("Measurements:\t");
     for(int i = 0; i < 3; i++)
         printf("%d\t", measurements[i]);
     printf("\n");
}
else
    puts("Couldn't retrieve measurements.");
closeRS485(&comm settings t);
```

5.4.3.10 commGetCurrents

This function is used to retrieve the motor currents from the device. **Arguments**:

- comm_settings *comm_settings_t Structure containing info about communication settings
- int id The device's ID number
- short int currents[] The array used to store the motor currents of the device connected

```
comm_settings comm_settings_t;
int device_id = 65;
short int currents[2];
openRS485(&comm_settings_t,"/dev/tty.usbserial-128");
if(!commGetCurrents(&comm_settings_t, device_id, currents))
```

```
printf("Measurements: %d\t%d\n",currents[0], currents[1]);
else
   puts("Couldn't retrieve currents.");
closeRS485(&comm settings t);
```

5.4.3.11 commGetCurrAndMeas

This function is used to retrieve both motor currents and encoder measurements from the device with only one command.

Arguments:

- comm_settings *comm_settings_t Structure containing info about communication settings
- int id The device's ID number
- short int *values The array used to store motor currents and encoder measurements of the device connected

Example:

```
comm_settings comm_settings_t;
int device_id = 65;
short int values[5];
openRS485(&comm_settings_t, "/dev/tty.usbserial-128");
if(!commGetCurrAndMeas(&comm_settings_t, device_id, currents)){
    printf("Currents: %d\t%d\t%d\n",values[0], values[1]);
    printf("Measurements:
%d\t%d\t%d\n",values[2],values[3],values[4]);
    }
else    puts("Couldn't retrieve currents and measurements.");
closeRS485(&comm_settings_t);
```

5.4.4 Code Examples

5.4.4.1 Theoretical Example

The purpose of this example is to show how the functions are supposed to be used, in which order and with what arguments.



This is not a complete program that use the device with all its capabilities but it is only an example to see what is the correct order of functions and what functions should be used to integrate the device within your system.



This example may not work properly; We suggest to not use it as it is in a complete program.



This example and the functions used in it, may work only for a qbmove device.

Code:

```
int device id = 1;
                         // By default the Device ID is 1. Check
                          your device
                        // use the GUI to know what ID has yours
comm settings comm settings t;
const int deg to tick = 32768/360;
int num ports = 0;
char ports[10][255];
char port[255];
short int pos, stiff;
char aux char;
// Firstly all connected ports must be recognized
num ports = RS485listPorts(ports);
strcpy(port, ports[0]); //We select the first one, just as example
// Open the port
openRS485(&comm settings t, port s);
if (comm settings t.file handle == INVALID HANDLE VALUE)
{
    puts("Couldn't connect to the serial port.");
   return 0;
}
usleep(500000);
puts("Port correctly opened\n");
// Activate motors
commActivate(&comm settings t, device id, 1);
pos = 0;
puts ("Insert 'w' and press enter to move the shaft to the positive
limit, 's' to move the shaft to the negative limit, 'x' to make
the shaft rigid, 'a' to move to the negative limit with rigid
shaft, 'd' to move to the positive limit with rigid shaft, 'q' to
quit the program");
scanf("%c", &aux char);
```

```
switch(aux char) {
    case w': //Send positive input
      pos = 180 * deg to ticks;
      inputs[0] = pos;
      inputs[1] = 0;
      commSetPosStiff(&comm settings t, device id, inputs);
    break;
    case 's': //Send negative input
     pos = -180 * deg to ticks;
      inputs[0] = pos;
      inputs[1] = 0;
      commSetPosStiff(&comm settings t, device id, inputs);
    break;
    case `x': //Send stiff input
      stiff = 3000;
      inputs[0] = 0;
      inputs[1] = stiff;
      commSetPosStiff(&comm settings t, device id, inputs);
    break;
    case 'a': //Send negative stiff input
      pos = -180 * deg to ticks;
      stiff = 3000;
      inputs[0] = pos;
      inputs[1] = stiff;
      commSetPosStiff(&comm settings t, device id, inputs);
    break;
    case 'd': //Send positive stiff input
      pos = 180 * deg to ticks;
      stiff = 3000;
      inputs[0] = pos;
      inputs[1] = stiff;
      commSetPosStiff(&comm settings t, device id, inputs);
    break;
    case 'q': // Close the program
      commActivate(&comm settings t, device id, 0); // Deactivate
motors
      closeRS485(&comm settings t); // Close serial port
    break;
    default:
    break;
```

5.5 ROS

5.5.1 Installation

IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT IMPORTANT

5.5.1.1 Requirements

If you have never set it up, you probably need to add your linux user to the dialout group to grant right access to the serial port resources. To do so, just open a terminal and execute the following command:

sudo gpasswd -a <user name> dialout

where you need to replace the $<user_name>$ with your current linux username. Then — don't forget to — logout or reboot.

5.5.1.2 Ubuntu Packages

If you prefer to leave your catkin workspace as it is, you can simply install all the ROS packages from the Ubuntu official repositories:

```
sudo apt update
sudo apt install ros-kinetic-qb-move ros-kinetic-qb-chain
```

Then, you need to integrate the qbrobotics' \rightarrow classes in your own code (cf. 1.1.1.1 for an overview of the C++ classes).

5.5.1.3 Sources

Another common apporach to install' ROS' packages is to clone their sources in the Catkin Workspace and then recompile everything, which is pretty straightforward for a ROS user. The following steps show how to do it even for ROS beginners:

 Clone both the qb_device, qb_move and qb_chain packages to your Catkin Workspace, e.g. ~/catkin_ws:

```
cd ~/catkin_ws/src
git clone https://bitbucket.org/qbrobotics/qbdevice-ros.git
git clone https://bitbucket.org/qbrobotics/qbmove-ros.git
git clone https://bitbucket.org/qbrobotics/qbchain-ros.git
```

IMPORTANT

It is worth noticing that the Catkin Workspace is expected to be already initialized. If you have never used ROS on your current machine, you should follow this guide first: http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment

2. Compile the packages using catkin:

```
cd ~/catkin_ws
catkin make
```

 If you were not familiar with ROS you should be happy now: everything is done! Nonetheless, if you encounter some troubles during the compilation, feel free to ask for support on <u>our Bitbucket</u>.

IMPORTANTDepending on your ROS installation, you may need some extra packages to
properly compile the code. Please, be sure that you have already installed
at least ros-kinetic-ros-controllers, ros-kinetic-transmission-interface, ros-
kinetic-joint-limits-interface, ros-kinetic-combined-robot-hw, and their
dependencies (e.g. use sudo apt install <ros-pkg>).

5.5.1.4 Device setup

Connect a *qbmove* to your system is basically a matter of plugging in a USB cable. Nonetheless, **read carefully** this manual to understand all the requirements and advices about either single-device or chained configurations, e.g. the Delta kinematic structure of this manual.

5.5.2 Usage

As shown in the following picture there are two distinct configurations to control several *abrobotics* devices connected to the system:

- The first (and recommended) groups all the Hardware Interfaces together (thanks to the <u>combined robot hw</u>) and exploits them as a unique robot system. We have called it *"synchronous"* just to point out that every sequence of reads and writes is always done in the same predefined order.
- The second mode threats every device as an independent Hardware Interface with its dedicated ROS Node which executes the control loop independently w.r.t. the rest of the system, i.e. "asynchronously".



Figure 5-11: Synchronous vs asynchronous control mode (either qb SoftHand or qbmove devices)

Mixed configurations can be also achieved through a proper setup. In such a case we can think of synchronous sub-systems which execute asynchronously w.r.t. each other.

Note that in a single-device system the synchronous mode is a nonsense.

In both cases there is always one central Node which manages the shared resources for the serial communication (e.g. one or many USB ports) and which provides several ROS services to whom wants to interact with the connected devices. This Node is called *Communication Handler* and it is usually started in a separate terminal.



To properly set qbmove IDs for the Delta Kit, see the section **Errore. L'origine riferimento non è stata trovata.**

5.5.2.1 Details

To understand what is hiding under the hood, have a look at the C++ classes overview which sums up all the main concepts of our ROS packages:

packages overview



Figure 5-12: Package overview with C++ class details

5.5.2.2 Communication Handler

The Communication Handler Node has no parameters to be set, therefore it is always launched like this:

```
roslaunch qb device driver communication handler.launch
```

On start, it scans the serial communication resources connected to your system and shows a list of the devices it has found. By default, it never scans again for new devices, apart from asking it explicitly during the initialization of a control Node.



If the Communication Handler find fewer (or even more) devices than expected, check the device IDs to be sure that there are no devices that are sharing the same ID.

This check can be easily achieved by connecting one device at a time and starting the Communication Handler each time.

This is a simple example when starting the Communication Handler with two *qbrobotics*[®] devices connected on two distinct USB ports:

```
[ INFO] [1524044523.511369300]: [CommunicationHandler] handles [
/dev/ttyUSB1].
[ INFO] [1524044524.426984697]: [CommunicationHandler] handles [
/dev/ttyUSB0].
[ INFO] [1524044525.218613760]: [CommunicationHandler] has found
```

```
[2] devices connected:
[ INFO] [1524044525.218696997]:
[1] connected through [/dev/ttyUSB0]
[ INFO] [1524044525.218736612]:
[2] connected through [/dev/ttyUSB1]
```

- device - device

When the Communication Handler is on, it provides all the Services required to interact with the connected devices: e.g. *get info or measurements, activate or deactivate motors, set commands,* and even more... A detailed description of the services can be found in the <u>gb device driver</u> package wiki.

5.5.2.3 Control

As shown before, the control Node exploits the <u>ros_control</u> Controller Manager which loads and runs the device controllers. Each controller provides an Action Server that, together with the Hardware Interface structure, allows the user to send commands to the relative device and get its measurements.

From an API point of view, it is implemented an Action Client which matches the relative trajectory controller and provides a method to send Goals, i.e. command references, directly to the given device. Additionally, the Action Client is subscribed to a Topic (*_controller/command) that can be used to send reference commands from outside the code, e.g. asynchronously from the command line, or from a higher-level control Node, e.g. as a result of a planning algorithm.

It is recommended not to mix these two control modes: choose either to control the device directly from the code by extending our API or through this command Topic.

Regardless the control mode chosen for the given application, and apart form a customization of the API, the following launch file templates can be used respectively to control several devices or a single one:





5.5.2.4 Control Modes

For the sake of simplicity, we are going to cover all the control modes for a single *qbmove*, but it is just a matter of putting things together and set the launch file parameters properly to control several devices together (<u>gb chain control</u> is dedicated for such a scope).

All the control modes are initialized in the same manner but with distinct command line arguments. The default single-device control Node which brings everything up and simply waits for commands on the above-mentioned Action topic is the following:

```
roslaunch qb_move_control control.launch standalone:=true activa
te_on_initialization:=true device_id:=<actual_device_id>
```

The arguments explained

- activate_on_initialization [false]: Activates the motors at startup (the device will not move since the first command reference is received).
- device_id [1]: Each device has its own ID, you need to set the one of the actual device connect to your system.
- standalone [false]: Starts the Communication Handler together with the control Node. If you set this to false (or remove it since the default value is false), you need to launch the Communication Handler in a separate terminal.

It is worth noting that the activation of the motors can be postponed to improved safety if you are not aware of the state of the system at startup. To do so just set activate_on_initialization:=false (or remove it since the default value is

false) and make a call to the Communication Handler <code>activate_motors</code> Service, when your system is ready, e.g. as follows:

```
rosservice call /communication_handler/activate_motors {"id: <ac
tual_device_id>, max_repeats: 0"}
```

Additional arguments

- control_duration [0.01]: The duration of the control loop expressed in seconds.
- get_currents [true]: Choose whether or not to retrieve current measurements from the device.
- get_positions [true]: Choose whether or not to retrieve position measurements from the device.
- get_distinct_packages [false]: Choose whether or not to retrieve current and position measurements from the device in two distinct packages.
- max_repeats [3]: The maximum number of consecutive repetitions to mark
 retrieved data as corrupted.
- set_commands [true]: Choose whether or not to send command positions to the device.
- set_commands_async [false]: Choose whether or not to send commands
 without waiting for ack.
- use_rviz [true]: Choose whether or not to use rviz. If enabled you should see a virtual *qbmove* on screen performing a similar behavior, i.e. moving the shaft and both the actuators accordingly.

When the ROS Node is started by using the above `roslaunch` command it is possible to send a simple position and stiffness command through the given command Topic, as follow:

```
rostopic pub -1 /qbmovel/control/qbmovel position and preset tra
jectory controller/command trajectory msgs/JointTrajectory "head
er:
 seq: 0
 stamp:
    secs: 0
   nsecs: 0
 frame id: ''
joint names: [qbmovel shaft joint, qbmovel stiffness preset virt
ual joint]
points:
- positions: [0.5,0.5]
 velocities: [0,0]
 accelerations: [0,0]
  effort: [0,0]
  time from start: {secs: 1, nsecs: 0}"
```

The followings are particular control modes which are enabled with few parameters, but the concepts of this paragraph hold for all of them.

5.5.2.5 GUI Control

This control mode is the simpler and the one suggested to test that everything is working as expected. You are able to move the *qbmove* shaft position and its stiffness interactively, but nothing more than this.

You will probably need this only the very first times and for debugging.

To start this mode just add use_controller_gui:=true to the general roslaunch command (be sure that the opposite use waypoints is not used).

After a while a GUI should appear to screen with two empty dropdown menus, a red enable button below them, and a *speed scaling* slider at the bottom.

- 1. Select the Controller Manager namespace from the left menu, e.g. /<robot_namespace>/control/controller_manager (where <robot_namespace> is an additional argument of the launch file needed with several devices). This enables the right menu which provides all the controllers available for the connected device.
- 2. Select the *qbmove* controller from the second dropdown menu and enable it through the circular button.
- 3. Two slider will appear in the GUI: the first controls the shaft position (which ranges respectively within the shaft position limits expressed in radians), while the second sets the stiffness preset, which ranges from 0 (lowest stiffness) to 1 (highest stiffness). You can also vary the speed through the bottom *speed scaling* slider if you like a faster/slower motion. No other timing constraints can be set in this mode.

Int_joint_trajectory_controller	JointTrajectoryController - rqt
Joint trajectory controller	D ? - 0
controller manager ns c	controller
/qbmove1/control/controller_manag ‡	qbmove1_position_and_preset_traje 拿
C	
joints	
qbmove1_shaft_joint	0,42
qbmove1_stiffness_preset_virtual_joint	0,12
speed scaling	
	50%

Figure 5-14: Example of GUI to control the *qbmove* (0.42 rad for the shaft position and 12% stiffness value)
5.5.2.6 Waypoint control

This control mode is a bit more structured and useful than the previous: it allows to set a fixed trajectory of any number of position waypoints (with timing constraints) and set the robot to cycle infinitely on it (because of the loop it is recommended to set the first and last waypoint in a similar configuration to avoid unwanted sudden changes).

To start this mode just add use_waypoints:=true to the general roslaunch command (be sure that the opposite use_controller_gui is not used). You won't see any control interface in this case but the *qbmove* should start moving according to the given trajectory, parsed from a yaml file located at

<robot_package>_control/config/<robot_name>_waypoints.yaml

where robot name and robot package are two additional launch file arguments.

Customization

You can modify the waypoint trajectory to replicate the behavior you want: either change the <robot_package>_control/config/<robot_name>_waypoints.yaml or add another custom application-specific file in the config directory. In the second case, you need to set the argument robot_name properly when launching the command from the terminal. The waypoint configuration is as follows:

```
# Waypoints describe the desired motion trajectory:
# - time [s]: can be either a single value or an interval for w
hich joint positions hold
# - joint positions:
      - shaft position [radians] and stiffness preset [0,1];
#
# - joint velocities: optional, use it only if you want a nonze
ro values
     - shaft position [radians/s] and stiffness preset [0,1]/s;
# - joint accelerations: optional, use it only if you want a no
nzero values
      - shaft position [radians/s^2] and stiffness preset [0,1]/
#
s^2;
#
# It is worth noting that, if specified, joint positions, joint
velocities and joint accelerations must be of size two.
waypoints:
  time: [1.0]
  joint positions:
     <device name>: [0.0, 0.0]
   time: [2.75, 3.25]
   joint positions:
     <device name>: [1.57, 0.0]
```

```
time: [4.0]
joint_positions:
    <device_name>: [0.5, 0.0]
joint_velocities:
    <device_name>: [-0.5, 0.0]
-
    ...
```

5.5.2.7 API control

If you need a complex (i.e. real) control application, e.g. the *qbmove* is mounted on a robot which uses computer vision aid to grasp objects, the previous two control modes don't really help much. What we provide for real applications is the full ROS libraries to manage and control the *qbmove*.

You must dig into the <u>gb_move</u> package documentation and find what better suits for your needs, e.g. extend the <u>gbDeviceControl</u> class provided, or even redesign some of its parts by following an approach similar to ours.

IMPORTANT Our recommendation is to use as much as possible our resources, classes and macros to help you while developing your application. Don't reinvent the wheel!

At last, if you come up with a something useful for the whole community, it will be amazing if you propose your improvement with a Pull Request in the package of interest on <u>our Bitbucket</u>.

5.5.3 Delta robot example

When using the ROS packages with a Delta Kit, it is possible to exploit the specific aids provided by qbrobotics to help in a quicker kinematic structure control.

5.5.3.1 Launch file and URDF model

The Delta Kit launch file is built in a very similar way to the one reported in Figure 5-13, and you can also inspect it by opening *"~/catkin_ws/src/qbchain-ros/qb_chain_control/launch/qbdelta_control.launch"*.

This launch expects four qbmoves with IDs from 1 to 4 (cf. Delta Kit IDs configuration for details) linked to the specific Delta Kit URDF, and equip the whole system with a dedicated ROS controller.

To launch the Delta Kit, you just need to execute the following command from a terminal:

```
roslaunch qb_chain_control qbdelta_control.launch standalone:=tr
ue activate on initialization:=true use rviz:=true
```

And you can also enable the GUI or Waypoint control modes as usual (by passing either `use_controller_gui:=true` or `use_waypoints:=true` as additional arguments).

IMPORTANT It is worth noticing that this is just a state visualizer and not a simulator.

5.5.3.2 Dedicated controller

A specific Delta Kit controller could have not been implemented since each single qbmove controller can be exploited to control the whole system. Anyway, controlling a Delta kinematic structure only w.r.t. the joint space is not the best option for the end user.

For this reason, we have created the `qb_chain_controllers/DeltaKinematicController` (derived by the official `controller_interface::Controller<hardware_interface::PositionJointInterface>`) which is loaded automatically during startup and which allows a more user-friendly approach. Without digging to much into the C++ class implementation, which is publicly available under `~/catkin_ws/src/qbchain-ros/qb_chain_controllers/`, the major improvements are the followings:

- By exploiting the direct and inverse kinematics of the Delta kinematic structure, the user can control the end-effector position w.r.t. the cartesian space (cf. the reference system details in 4.5 ID configuration).
- The 3D positioning of the end-effector can be more or less compliant to the surrounding environment by adjusting the stiffness of the three upper joints. This leads respectively to a less or more precise pursuing of the control reference, but the compliance can be exploited to achieve many tasks that could not have been done otherwise.
- The end-effector qbmove can be treated as a two-finger gripper with the advantage of exploting the positioning and compliance control proper of qbmove devices. Indeed, the grip can be more or less strong w.r.t. the applied stiffness.

These differences are notable also in the Waypoint control mode which configuration is as follows:

```
# Waypoints describe the desired motion trajectory:
# - time [s]: it is mandatory and can be either a single value o
r an interval for which other values hold;
# - end_effector [m]: a three-element list of [x, y, z] coordin
ates for the delta end-effector position;
# - joint_stiffness [0,1]: a three-element list of stiffness va
lues, one for each of the three upper motors;
# - gripper: a two-element list containing the shaft position [
rad] and the stiffness value [0,1] of the gripper.
#
# Whenever these are not present in the Parameter Server, it is a
ssigned the last available value.
```

```
waypoints:
    time: [1.0]
    joint_stiffness: [0.75, 0.75, 0.75]
    end_effector: [0.0, 0.0, 0.15]
    gripper: [0.0, 0.5]
    time: [3.0, 4.0]
    # stiffness is not changed in this case
    end_effector: [0.0, 0.07, 0.15]
    gripper: [0.0, 0.5]
    time: [5.0]
    joint_stiffness: [0.5, 0.5, 0.5]
    end_effector: [0.0, 0.07, 0.15]
    gripper: [1.0, 0.5]
    ...
```

5.5.3.3 Interactive markers

In addition to the new controller, the Delta Kit URDF is equipped with Interactive Markers (http://wiki.ros.org/rviz/Tutorials/Interactive%20Markers%3A%20Getting%20Started) that enable a manual motion of the end-effector directly from rviz.

Also, the stiffness of the joints can be adjusted manually, together with the gripper position and stiffness.

This feature can be useful especially for beginners to have some practice and become more confident with this kinematic structure and its operative space. Moreover, a simple test when inventing a demo application can be done more quickly in this way.

5.5.4 ROS packages overview

You can find all the needed packages here:

- qb_device: <u>http://wiki.ros.org/qb_device</u>
- qb_move: <u>http://wiki.ros.org/qb_move</u>
- qb_chain: http://wiki.ros.org/qb_chain

This page was intentionally left blank

6 Troubleshooting

This chapter explains how to solve problems that you may encounter in the process of building a robot, programming a robot file, or operating a robot platform.



6.1 Qbmove output shaft doesn't move smoothly

[Cause]

In some cases, qbmove's output shaft may get stiff and won't rotate smoothly when you try to move it with your hands. This is not product failure but a situation caused by the tight arrangement of the internal gears.

Tight gear arrangement is designed for precise motion control. So, you can see the robot moves smoothly without a problem when the assembly is finished and power is supplied to each of qbmove actuators

[Troubleshooting]

Solution 01

Turn the output shaft in the opposite direction first and try again to turn in the desired direction. If you can't provide enough force, use a provided flange to rotate the output shaft. Much greater torque can be leveraged and it's easier to rotate the shaft.

Solution 02

Connect qbmove to the PC and run "qbmove GUI". Make sure that the sliders of position and stiffness are in zero position, then activate the qbmove.

Once activated, you can move the device by moving the sliders. To reset the position to zero, click on "Zero" button. In this way, shaft will go to zero position and minimum stiffness configuration. You can then use the sliders to verify the correct behavior of the qbmove.

6.2 One or more qbmoves don't activate

[Cause]

The angular positions of actuator's internal shafts are too far from zero position.

[Troubleshooting]

Solution 01

Turn the output shaft back near the zero position. If you can't provide enough force, use a provided flange to rotate the output shaft or move directly the assembled robot.

Solution 02

Connect qbmove to the PC and run "qbmove GUI". Then click the button "Get Measurements". If one of the positions is higher than 13000 ticks, follow "Solution 1". Then you can try to activate the device.

6.3 Blue LED is off when the robot is powered

[Cause]

The blue LED is damaged or the actuator doesn't receive the electrical supply, directly from the power supply or from a previous actuator of the power chain.

[Troubleshooting]

Solution 01

Connect qbmove to your PC and run "qbmove GUI" (page 39), make sure that the sliders of position and stiffness are in zero position, then connect the power supply and active the qbmove. If the device works properly and the blue LED is off, it means that the LED is damaged but you can continue using the device.

If the blue LED is off and the device doesn't move, change the daisy chain port.

6.4 White LED is off when you use the robot

[Cause]

The white LED is damaged or the actuator doesn't receive the electrical supply from a previous actuator of the power chain or from the USB cable.

[Troubleshooting]

Solution 01

Connect qbmove to your pc using the USB cable provided and run "qbmove GUI". If LED is still off and the device doesn't communicate, replace the USB cable.

If the qbmove communicate and the LED is off, the LED is damaged and you can continue using it.

Solution 02

If the interested qbmove is a part of power chain, replace the ERNI cable. If the problem persists, follow the "Solution 1".

6.5 Clicking on "Scan Ports" on GUI results in no port shown

[Cause]

The COM port number is too high or there is a connection problem

[Troubleshooting]

Solution 01

Under Windows, try changing the COM port number by going under <u>Control Panel > Hardware</u> and <u>Sound > Device Manager</u>.

Open the <u>Ports (COM & LPT)</u> drop down menu, right click on the COM port and select <u>Proprieties</u>. Select the <u>Port Settings</u> tab and then click on <u>Advanced</u>. Once in Advanced menu select from the drop-down menu a COM number between COM1 and COM9 and click on <u>OK</u>.

Solution 02

Check if the one or more led is on, if it is. Try changing USB post on your computer. If it is not, change the USB cable or reboot your pc.

7 Commissioning and Maintenance

This chapter explains how to solve problems that you may encounter in the process of building a robot, programming a robot file, or operating a robot platform.



7.1 Commissioning



Hazards due to hot surfaces

Depending on the load and ambient temperature, the motor can overheat. Allow the motor to cool down after operation.



Risk of injury caused by protruding, rotating or moving parts of the driven mechanical units



Damage to the motor and/or Speed Controller because of incorrectly set control parameters

Before commissioning, check and if necessary adjust the configured parameters.



The drive performing unplanned movements during commissioning cannot be ruled out

Make sure that, even if the drive starts to move unintentionally, no danger can result for personnel or machinery

7.2 Maintenance and warranty

Products of the company qbrobotics s.r.l. are produced using the most modern production methods and are subject of strict quality inspections. All sales and deliveries are performed exclusively on the basis of our General Conditions of Sale and Delivery which can be viewed on the qbrobotics home page <u>www.qbrobotics.com</u>.

The warranty will not be valid in case of tampering with the device, or with the software.

8 Appendix

8.1 VSA papers

A decoupled Impedance observer for a Variable Stiffness Robot

http://www.centropiaggio.unipi.it/publications/decoupled-impedance-observer-variablestiffness-robot.html

A real time robust observer for an agonist antagonist variable stiffness actuator

http://www.centropiaggio.unipi.it/publications/real-time-robust-observer-agonist-antagonistvariable-stiffness-actuator.html

A Real-time Parametric Stiffness Observer for VSA devices

http://www.centropiaggio.unipi.it/publications/real-time-parametric-stiffness-observer-vsadevices.html

A Stiffness Estimator for Agonistic–Antagonistic Variable-Stiffness-Actuator Devices

http://www.centropiaggio.unipi.it/publications/stiffness-estimatoragonistic%E2%80%93antagonistic-variable-stiffness-actuator-devices.html

Variable Stiffness Control for Oscillation Damping

http://www.centropiaggio.unipi.it/publications/variable-stiffness-control-oscillationdamping.html

Variable Stiffness Actuators: the user's point of view

http://www.centropiaggio.unipi.it/publications/variable-stiffness-actuators-user%E2%80%99spoint-view.html

Controlling Soft Robots: Balancing Feedback and Feedforward Elements

http://www.centropiaggio.unipi.it/publications/controlling-soft-robots-balancing-feedbackand-feedforward-elements.html

8.2 qbmove papers

VSA - CubeBot. A modular variable stiffness platform for multi degrees of freedom systems

http://www.centropiaggio.unipi.it/publications/vsa-cubebot-modular-variable-stiffness-platform-multi-degrees-freedom-systems.html

Towards variable impedance assembly: the VSA peg-in-hole

http://www.centropiaggio.unipi.it/publications/towards-variable-impedance-assembly-vsa-peg-hole.html

Passive impedance control of a Qboid multi-DOF VSA-CubeBot manipulator

http://www.centropiaggio.unipi.it/publications/passive-impedance-control-qboid-multi-dofvsa-cubebot-manipulator.html

Open Source VSA-CubeBots for Rapid Soft Robot Prototyping

http://www.centropiaggio.unipi.it/publications/open-source-vsa-cubebots-rapid-soft-robotprototyping.html

Robust Optimization of System Compliance for Physical Interaction in Uncertain Scenarios

http://www.centropiaggio.unipi.it/publications/robust-optimization-system-compliance-physical-interaction-uncertain-scenarios.html

9 Record of documented revisions

Revision	Date	Remarks
2.9	09/04/18	First version
2.10	08/08/18	Updated ROS section
2.11	23/10/18	Updated ROS section
2.12	16/04/19	Updated Electrical connections section
2.20	22/09/2020	Update paragraph 6.3 and 6.5