EtherCAT Dextrous Hand

- User Manual -

Shadow Software team - software@shadowrobot.com

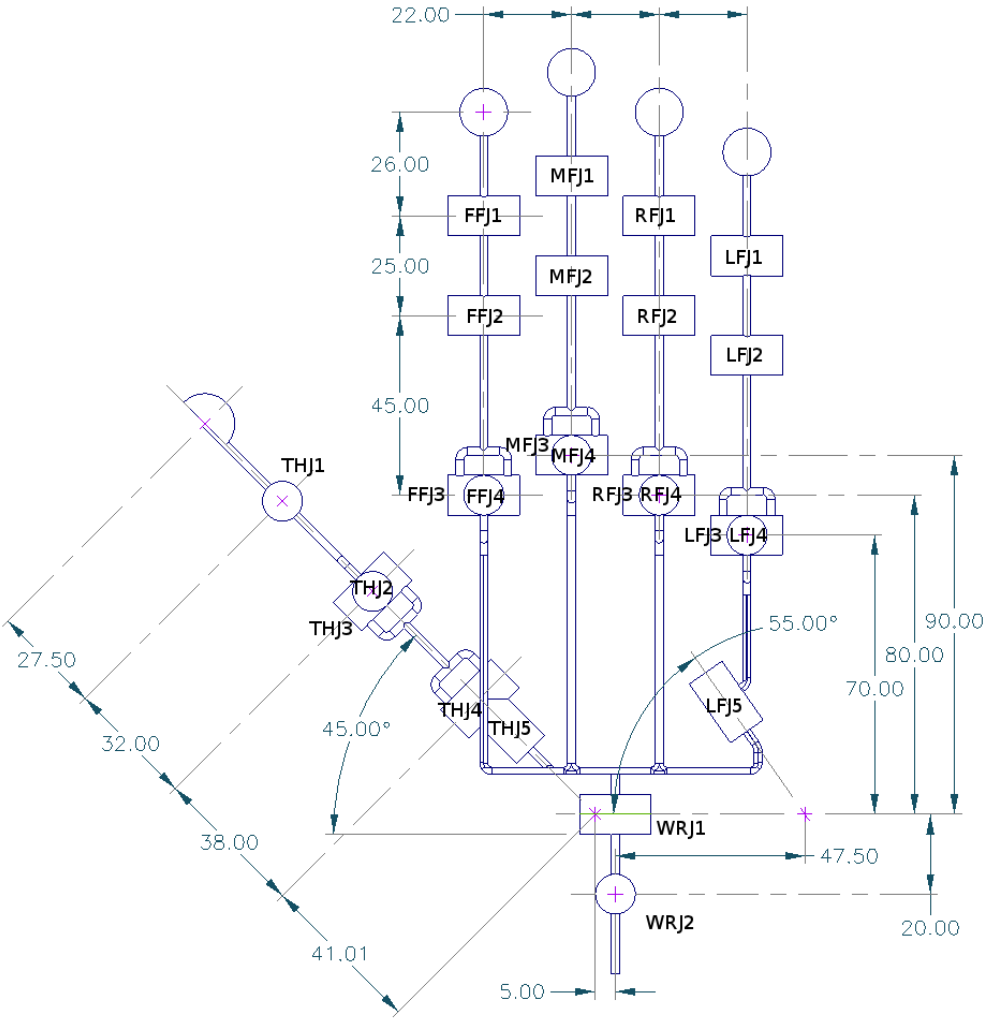
November 21, 2012

Contents

1 Overview

This user manual will get you started with using our EtherCAT hand or simulated robots. We'll guide you through installing the code in **section 2**, explain where to find things in **section 3** and show you some basic commands and code samples in **section 4**. This guide ends with some pointers to essential documentation you should read (**section 5**), and some troubleshooting tips (**section 6**).

You can find below a kinematics diagram of the Shadow Hand. The joint names are indicated on the diagram. As a convention, and because of the way the Hand is actuated, the joint 1 and 2 for the first finger, middle finger, ring finger and little finger are grouped in the joint 0: `FFJ0 = FFJ1 + FFJ2`, `MFJ0 = MFJ1 + MFJ2`, etc.



If you have any questions, don't hesitate to contact us at: `software@shadowrobot.com`.

2 Installing our Software

For a typical user, we really recommend installing our software directly from the apt repository (see section 2.1.1). This will ensure that you keep the latest released software on your computer.

We assume you've already installed ROS Fuerte, as detailed on the ROS Wiki[1].

However if you plan to modify some parts of our code, you can use an overlay[2] to install some of the sources from our publicly available Launchpad repository on top of your existing installation.

2.1 Installing Routes

2.1.1 From ROS apt repository (recommended)

You can install pre-built packages for Ubuntu 12.04[3]. This is the easiest way of installing our software. You'll also receive updates automatically whenever you update your system.

Once you've installed ROS Fuerte, you can easily install our stacks:

```
> sudo apt-get install ros-fuerte-shadow-robot \
  ros-fuerte-shadow-robot-ethercat ros-fuerte-sr-visualization \
  ros-fuerte-sr-teleop
```

2.1.1.1 Install a sr_config overlay for the EtherCAT Hand If you have an EtherCAT Hand, you should create an overlay for the `sr_config` stack: this stack contains all the different configuration files which can be modified to reflect your specific system. You shouldn't use the one installed by default as they are defined for the simulated hand.

To create this overlay, just follow **section 2.1.2**.

2.1.2 From sources

If you need to modify one of the packages for your own use, you'll want to first install the released version from the apt repository as explained in **section 2.1.1**. This will ensure that you have all the code you need. Once you've identified in which stack the package you want to modify is, you can create an overlay to use the source version.

To create an overlay, we'll use the rosws[4] utility. In this example, we'll overlay the sr_config[5] stack, using trunk version of the code hosted on launchpad using bzr[6]:

[1] url: http://www.ros.org/wiki/fuerte/Installation/Ubuntu

[2] **Overlay:** use another version of a stack than the default installed one. The path to the package in the overlay will be prepended to the ROS_PACKAGE_PATH, so it will be used instead of the default installed version.

[3] url: http://releases.ubuntu.com/precise/

[4] url: http://ros.org/wiki/rosws

[5] url: http://launchpad.net/sr-config

[6] url: http://bazaar.canonical.com

`lp:sr-config`.

- First we need to install the rosws/rosinstall utilities if you haven't done so already (you've probably already installed this when following the ROS wiki instructions):

```
> sudo apt-get install python-rosinstall python-rosdep bzr
```

- We're now going to create a workspace (in the ~/`workspace` directory), based on your currently installed ROS system (we're assuming you're using ROS Fuerte), and add the stacks we want to overlay to the workspace. We'll then update the workspace to download the code into it.

```
> rosws init ~/workspace /opt/ros/fuerte
> source ~/workspace/setup.sh
> rosws set sr_config --bzr lp:sr-config
> rosws update
```

- To use the newly created environment, simply source the setup.sh created in the previous step (to have it sourced automatically when you open a new terminal, simply add it to your ~/`.bashrc` as show on the second line).

```
> source ~/workspace/setup.sh
> echo source ~/workspace/setup.sh >> ~/.bashrc
```

- To install the system dependencies, we'll use the rosdep[7] utility. This utility needs to be initialised the first time you use it:

```
> sudo rosdep init
> rosdep update
```

- You now have the up-to-date database of system dependencies on your system. You can install the dependencies for the stack you want to build:

```
> rosdep install sr_config
```

- Finally, you can build the whole stack in one go using the `rosmake` command:

```
> rosmake sr_config
```

2.1.2.1 Available rosinstall files

This step is only recommended for people who are going to be heavily modifying our stacks (or for those who are not using ubuntu, and thus can't use the pre-built packages).

If you want to install all our stacks in one go, you can use the `rosinstall` files available here[8].

Once you've downloaded the code using the `rosinstall` commands described previously, you can build all our stacks:

[7] url: http://ros.org/doc/api/rosdep2/html/overview.html

[8] url: http://bazaar.launchpad.net/~shadowrobot/sr-build-tools/trunk/view/head:/data/shadow_robot-fuerte.rosinstall

```
> rosdep install shadow_robot shadow_robot_ethercat\
  sr_contrib sr_visualization sr_teleop
> rosmake shadow_robot shadow_robot_ethercat\
  sr_contrib sr_visualization sr_teleop
```

2.2 Import our custom Environment Variables

As described in **section 3.1.1**, we're using a file to define some useful environment variables. To make sure the variables are imported in your setup, you need to source it in your ~/.bashrc, after sourcing the ros setup.sh. It should look like this:

~/.bashrc

```
#source the ros setup.sh
source /opt/ros/fuerte/setup.sh
#or the one created in your workspace
# if you used an overlay
#source ~/workspace/setup.sh

#now source our additional environment variables
source `rosstack find sr_config`/bashrc/env_variables.bashrc
```

3 Navigating our code

Our code is split into different stacks, each containing a set of packages:

- **shadow_robot:** contains simulation software, utility packages, ... This is the core stack.
- **shadow_robot_ethercat:** contains the code for running our etherCAT hand.
- **sr_contrib:** contains experimental high level packages. If you write an interesting package, we'd be happy to add it to this stack.
- **sr_visualization:** contains the GUI and visualisation tools.
- **sr_teleop:** contains drivers for different tools used to teleoperate our hand (cyberglove[9], ...)
- **sr_config:** contains the configuration files. If you have one of our Hands, you should have an overlay of this configuration to make sure you don't lose your configuration when updating your system (as described in **section 2.1.2**).

3.1 Configuration and settings

We moved all the configuration to the `sr_config` stack. The idea is to isolate the data that is specific to the user in a given stack, this way you can easily create an overlay, as described in **section 2**, to make sure your specific configuration is not overwritten by an update.

3.1.1 Specifying system parameters

The system parameters are specified in the file env_variables.bashrc[10]. To load them automatically in your environment, please go to **section 2.2**. This File is heavily commented and should be easy to modify to suit your installation (**the default options should be ok for most users**).[11]

env_variables.bashrc

```
#set to 1 to compile sr_hand and sr_tactiles with the gazebo interface
export GAZEBO=1

# set to 1 for one finger hands
export ONE_FINGER=0

# set to 1 for left hands
export LEFT_HAND=0

# set to 1 if you have a CAN hand
```

[9] url: http://www.cyberglovesystems.com

[10] url: http://bazaar.launchpad.net/~shadowrobot/sr-config/trunk/view/head:/bashrc/env_variables.bashrc

[11] If you bought an etherCAT hand, the system you'll receive from Shadow will already be configured properly for you. There are no variables stating that you have an etherCAT hand in this file, because the launch file for starting the simulator and the one used to run the hand are different, as you can see in this manual.

```
export REAL_HAND=0

# set to 1 if you have a CAN arm
export REAL_ARM=0

#set to 1 if you have a muscle arm or hand
export MUSCLE=0

#set to 1 if you want to build the compatibility interface
# for the etherCAT hand (makes it possible to run  your programs
# developed for the CAN Hand on the etherCAT hand)
# default is 1 as it just builds another node.
export ETHERCAT=1

#port in which the etherCAT hand is plugged in
# (more info in section 4.1.1)
export ETHERCAT_PORT=eth1

#set to 1 if you're using PWM control on the etherCAT hand motors by default
export PWM_CONTROL=0

#set to 1 if you want to have access to the internal firmware repository
#NOTE: for Shadow employees only for the time being
export INTERNAL_FIRMWARE=0

#set to 1 if you have ellipsoid fingertips (ATI nano sensors)
export ELLIPSOID=0

#set to 1 if you want to publish more debug information
# on the etherCAT hand.
export DEBUG=1
```

3.1.2 Specifying default controllers settings, and other useful settings

The controller settings (as well as the different mappings and polling rates for the etherCAT hand) can be found in the **sr_ethercat_hand_config** package. You can modify the controller values which will be loaded by default by modifying the files in this package. For example if you want to modify the mixed position velocity controllers, you can open **sr_edc_mixed_position_velocity_joint_controllers.yaml** (in **controls/host**). We recommend using the controller tuner GUI plugin (see section 4.3) for tuning and saving those parameters.

3.2 Important Packages

- **sr_hand:** Contains the code and the different launch files for the simulated Hand and the CAN hand.
- **sr_edc_launch:** Contains the main launch file for the EtherCAT Hand.
- **sr_example:** Contains some simple code examples.

4 Running our code

4.1 Starting the real hand

4.1.1 Setting up the Ethernet port

The first time you connect an EtherCAT Hand, you will need to check on which Ethernet port the hand is connected to. Usually, you will have 2 network ports in your computer, one for your network, the other one for the hand.

An easy way to see which port the hand is using is to unplug the hand network cable from your computer and then plug it back in. You can now type `dmesg` and you should see a line saying which port has just been plugged in, for example, in the lines below, you see that `eth1` is used by the hand:

```
[16661.901920] e1000e: eth1 NIC Link is Down
[16666.460518] e1000e: eth1 NIC Link is Up 1000 Mbps Full Duplex
```

Now that we know which Ethernet port is used by the hand (`eth1` in our example), we can edit the interface in the system parameters, as shown in **section 3.1.1**:

- Make sure you've created your overlay for the `sr_config` stack as described in **section 2.1.1.1**.
- Let's open the configuration file:

```
> roscd sr_config
> cd bashrc
> gedit env_variables.bashrc
```

- Set the `ETHERCAT_PORT` to the correct value (it is `eth1` by default)
- Save and restart your terminal to make sure the new options have been taken into account.

4.1.2 Starting the driver

You can now start the driver. To be able to run the driver, you need to elevate the permissions to root access:

```
#elevate permissions
> sudo -s
> roslaunch sr_edc_launch sr_edc.launch
```

4.2 Starting the simulated robot

To start the simulated robot you can simply run:

```
> roslaunch sr_hand gazebo_hand.launch
```

Or if you want a simulated Shadow Arm as well as our hand:

```
> roslaunch sr_hand gazebo_arm_and_hand.launch
```

4.3 Starting the GUI

To start the GUI, run:

```
> rosrun rqt_gui rqt_gui
```

We're using the rqt_gui[12] for driving the hand. We implemented some plugins to help you control the hand. They are all in the Shadow Robot item in the plugins menu.

4.4 Checking it works

4.4.1 Reading data out of the Hand

First look at the data coming out of the hand.

```
> rostopic echo /joint_states
---
header:
  seq: 7235
  stamp:
    secs: 73
    nsecs: 785000000
  frame_id:
name: [WRJ2, WRJ1, FFJ4, FFJ3, FFJ1, FFJ2, ...]
position: [-0.007345193851005405, 0.0007282010249589632, ...]
velocity: [0.0002164649710794793, 0.0015999764385997265, ...]
effort: [0.0, 0.0, 0.0, 0.0, ...]
```

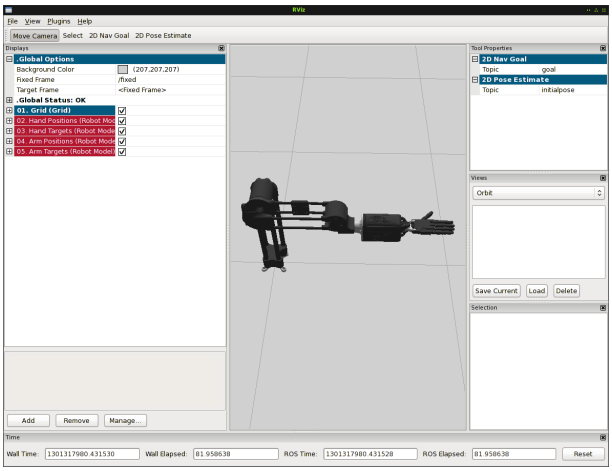
The data is published on the `/joint_states` topic (or `/gazebo/joint_states` for the simulated hand). You will see the incoming data streaming in your console. You have access to the joint names as well as the current positions (in radians), velocity (in radians per second) and effort (an arbitrary value, proportional to the effort applied at the motor-end of the tendon.) for each joint as shown below (stop with `ctrl+c`):

You should also be able to visualise the current state of the robot using rviz[13].

To start rviz run the command below. You'll need to select a base link and add a Robot Model from the "add" button in rviz.

```
> rosrun rviz rviz
```

[12] url: http://www.ros.org/wiki/rqt_gui
[13] url: http://www.ros.org/wiki/rviz



4.4.2 Reading the tactile sensors

Depending on the configuration of your hand, you may have different types of tactile sensors. Your sensor type is automatically detected by the driver which then publishes the correct information on the `/tactile` topic. The following command will output the data coming from the sensors on the screen.

```
> rostopic echo /tactile
```

- The fields from the Shadow PSTs are:

  int16[] pressure: an array of integers containing the current pressure value for the sensors.

  int16[] temperature: an array of integers containing the current temperature for the sensors.

- The fields from the Biotac sensors are:

  int16 pac0: first pressure AC reading (this is used to detect pressure changes, behaves as a differential)

  int16 pac1: second pressure AC reading

  int16 pdc: pressure DC reading (this is proportional to the pressure exerted on the finger)

  int16 tac: temperature AC reading (this is used to detect temperature changes, behaves as a differential)

  int16 tdc: temperature DC reading (this is proportional to the temperature off the sensor)

  int16[19] electrodes: the values for the electrodes of this tactile sensor (gives you information on which zone is being touched on the finger)

4.4.3 Sending Commands to the Hand

You can easily control the different joints of the Hand: each of the joint is controlled by a controller. Each of those controllers has a `/command` and a `/state` topic. The `/state`

topic prints useful debug information regarding the controllers, while the `/command` topic is used to send new targets to the controllers.

To identify the topic on which you want to send the target, you can use the `rostopic list` command. Let's say we want to send a target to `FFJ3`:

```
> rostopic list | grep ffj3
/sh_ffj3_mixed_position_velocity_controller/command
/sh_ffj3_mixed_position_velocity_controller/state
```

This means that we're going to publish our target on the topic:
`/sh_ffj3_mixed_position_velocity_controller/command`.
To do this, we can simply use the `rostopic pub` command. For example, to send a target of 1 rad to `FFJ3` at 10Hz:

```
> rostopic pub /sh_ffj3_mixed_position_velocity_controller/command \
  std_msgs/Float64 -r 10 1.0
```

To see a code example you can have a look at the package `sr_ethercat_example`.

4.4.4 Getting information on the current state of the controllers

As you can see above, each controller publishes its own `/state` topic. This topic contains lots of useful information regarding the current state of the controller. If you subscribe to it, you'll be able to have access to the current target, position, error, derivative, etc...

Different informations are published depending on the type of controller that is loaded:

- The mixed position velocity controller (a position control loop running on top of a velocity control loop) information is the following:
  - **header**: contains a timestamp
  - **set_point**: the position target
  - **process_value**: the current joint position
  - **process_value_dot**: the current joint velocity
  - **commanded_velocity**: the velocity demand generated by the position control loop
  - **error**: the different between the position target and the current position
  - **time_step**: the timestep between two control cycles
  - **command**: the torque command sent to the motor
  - **measured_effort**: the torque measurement from the motor
  - **friction_compensation**: value applied from the friction compensation map (not used yet)
  - **position_p**: the Kp for the position PID loop.
  - **position_i**: the Ki for the position PID loop.
  - **position_d**: the Kd for the position PID loop.

  - **position_i_clamp**: the value used for clamping the I term for the position PID loop.
  - **velocity_p**: the Kp for the velocity PID loop.
  - **velocity_i**: the Ki for the velocity PID loop.
  - **velocity_d**: the Kd for the velocity PID loop.
  - **velocity_i_clamp**: the value used for clamping the I term for the velocity PID loop.

- For the position controller:
  - **header**: contains a timestamp
  - **set_point**: the position target
  - **process_value**: the current joint position
  - **process_value_dot**: the current joint velocity
  - **error**: the different between the position target and the current position
  - **time_step**: the timestep between two control cycles
  - **command**: the torque command sent to the motor
  - **p**: the Kp for the position PID loop.
  - **i**: the Ki for the position PID loop.
  - **d**: the Kd for the position PID loop.
  - **i_clamp**: the value used for clamping the I term for the position PID loop.

4.4.5 Checking the diagnostics

The etherCAT Hand regularly publishes diagnostics about the state of the hardware. You can view them using the robot_monitor[14] utility. This way you can monitor the different state of the motors, the sensors, etc...

[14] url: http://ros.org/wiki/robot_monitor

5 Where to go next

The ROS wiki[15] is probably the best place to start learning about ROS. We strongly recommend going through the excellent ROS Start Guide[16], especially the set of tutorials[17].

You should also look at some of our tutorials for the EtherCAT Hand[18].

We'd be delighted to get your contributions (patches, nice demos, etc...). Don't forget our software is open-source, so if you want to contribute, you should probably check this wiki page[19] out.

[15] url: http://ros.org/wiki
[16] url: http://ros.org/wiki/ROS/StartGuide
[17] url: http://ros.org/wiki/ROS/Tutorials
[18] url: http://ros.org/wiki/shadow_robot_etherCAT/Tutorials
[19] url: http://ros.org/wiki/shadow_robot#Contributing

6 What to do if it doesn't work

6.1 Check your ROS setup

You can use very simple examples to check different things:

- Check you can access our stacks (if this command doesn't get you to the shadow_robot folder, make sure you've installed the code properly and sourced the generated setup.sh as described in **section 2**):

```
> roscd shadow_robot
```

- Check you can publish and subscribe to a simple topic. You'll need to start each of the following commands in a separate terminal. The first command is used to start the `rosmaster`. The second one will publish some data on the `test` topic while the third command subscribes to the topic and print the output on the screen. If this test works, you should see the incoming data streaming in your third terminal.

```
#in terminal 1, start the rosmaster
# we start it in a separate xterm window
# as you need to keep it running during
# the whole experience
> xterm -e roscore &
#in terminal 2 publish data at 5Hz on /test
> rostopic pub /test std_msgs/Float64 -r 5 1.0
#in terminal 3, subscribe to the /test topic
# you should see the streaming data
# stop it with Ctrl+c
> rostopic echo /test
data: 1.0
---
data: 1.0
---
data: 1.0
```

- If all hope is lost, run the roswtf[20] utility.

6.2 Use the ROS community

A good thing to remember is that you're probably not the first one having run into this problem. You should check the ROS community website: answers.ros.org[21] and post your question if you can't find a relevant answer.

6.3 Contact us

Don't hesitate to contact us: `software@shadowrobot.com`.

[20] url: http://ros.org/wiki/roswtf
[21] url: http://answers.ros.org

7 Changelog

7.1 shadow_robot

- **20/08/2012 - release 1.1.0**: New organisation: the urdf and descriptions are regrouped in the new sr_description stack. Numerous bug fixes.
- **20/08/2012 - release 1.0.1**: Resetting the controllers when restarting them (zeroing motor strain gauges)
- **09/08/2012 - release 1.0.0**: Better mixed position velocity controllers, Hand with Biotacs model added, Reorganised the code in different stacks. Removed some packages from the shadow_robot stack to put them in either sr_contrib, sr_config, sr_teleop, sr_visualisation, sr_demo stacks. We're keeping only the core packages in this stack for easier release / maintenance.

7.2 shadow_robot_ethercat

- **20/08/2012 - release 1.1.0**: Using sr_description and loaders in sr_edc.launch. Publishing extra messages from palm (analog inputs) on /palm_extras. Numerous bug fixes.
- **20/08/2012 - release 1.0.1**: Backlash compensation and asynchronous reset
- **09/08/2012 - release 1.0.0**: Using lowpass filter instead of alpha-beta filter for better control, added support for three finger hands, Control inputs from the host (backlash compensation on/off, sgl increase, jiggle, write eeprom, ...), Switching from PWM demand to TORQUE demand on the fly. Added a service to nullify the demand sent to the motor easily. This way the user can easily stop the control of the hand. Rewrote bootloader

7.3 sr_contrib

- **17/02/2012 - rev. 328**: reorganising the code. Created sr_contrib stack to contain some high level packages which may be a bit less stable than the core ones found in shadow_robot.

7.4 sr_visualization

- **19/11/2012 - release 1.1.0**: Bug fixes and minor improvements.
- **20/08/2012 - release 1.0.1**: Small bug fixes.
- **09/08/2012 - release 1.0.0**: First version of the gui using rqt_gui, different plugins are available for controlling our hand and arm, tuning them, etc...

7.5 sr_teleop

- **19/11/2012 - release 1.1.0**: Using sr_config to store the configuration files.
- **10/08/2012 - release 1.0.0**: Created sr_teleop stack to contain packages used to teleoperate our robots, contains the cyberglove package.

7.6 sr_config

- **19/11/2012 - release 1.1.0**: Added config files for PWM control, cyberglove default calibration and mapping.
- **10/08/2012 - release 1.0.0**: Created the sr_config stack to store the user specific config. The values by default should work for most users (simulated right hand with all the fingers). Users should overlay sr_config if they want to modify the config files. Useful for easy release.

7.7 sr_visualization

- **19/11/2012 - release 1.1.0**: Bug fixes and minor improvements.
- **20/08/2012 - release 1.0.1**: Small bug fixes.
- **09/08/2012 - release 1.0.0**: First version of the gui using rqt_gui, different plugins are available for controlling our hand and arm, tuning them, etc...