## Contents

1. Contents ...................................................... 3
2. Abbreviations ................................................. 55
This is the starting point for the Shadow Dexterous Hand Documentation
1.1 Setting up the hand

1.1.1 First time users

If you are unfamiliar with ROS and intend to use the ROS API, it is highly recommended that you read the ROS Tutorials.

If you are unfamiliar with the terminal on Linux, you should look here.

Shadow software is deployed using Docker. Docker is a container framework where each container image is a lightweight, stand-alone, executable package that includes everything needed to run it. It is similar to a virtual machine but with much less overhead. Follow the instructions in the next section to get the latest Docker container of the hand driver and interface up and running.

1.1.2 Hardware specifications

In order to run our software and the ROS software stack you will need to meet some hardware requirements.

CPU: Intel i5 or above RAM: 4GB or above Hard Drive: Fast HDD or SSD (Laptop HDD are very slow) Graphics Card: Nvidia GPU (optional) LAN: A spare LAN port to connect the Hand (even with a USB to LAN adaptor) OS: Ubuntu 18.04, 16.04 Kinetic (Active development) or 14.04 Indigo for older releases.

The most important one is to have a fast HDD or an SSD.

1.1.3 Setting up a real hand
What's in the box?

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shadow Hand E2M3 or E2PT</td>
<td>Hand Unit</td>
</tr>
<tr>
<td>PC</td>
<td>Host PC control unit for the hand</td>
</tr>
<tr>
<td>PSU for Hand</td>
<td>48v for motor hand</td>
</tr>
<tr>
<td>Kettle Leads</td>
<td>To connect power supplies to mains</td>
</tr>
<tr>
<td>Power Cable</td>
<td>4-pin Large Lemo connector, already fitted to the hand</td>
</tr>
<tr>
<td>EtherCAT Extension Cable</td>
<td>50cm EtherCAT extension lead, already fitted to the Hand</td>
</tr>
<tr>
<td>Calibration Jigs</td>
<td>Bag containing calibration jigs for all joints</td>
</tr>
<tr>
<td>Toolbox</td>
<td>Contains hex drivers to perform required mainten ance</td>
</tr>
<tr>
<td>User Manual</td>
<td>This document</td>
</tr>
</tbody>
</table>

Connecting Cables

There are two ways to connect the EtherCAT and power cables to the hand.

External connections

If your hand already has cables fitted, then you can simply connect the EtherCAT and power connectors immediately.

EtherCAT: Connect the Ethernet cable to the hand’s Ethernet socket, and connect the other end to the PC’s second Ethernet port. If you have a Bi-manual system, connect the Left and Right hands correctly to the labelled ports. You have been supplied with a medium length Ethernet lead, but if you require a longer or shorter one, you can simply use a standard commercial Ethernet Cat 5 cable, available from most computer parts suppliers.

Power: Connect the external power supply to the hand using the metal Lemo connector, making sure to line up the red dots. If you require a longer or shorter cable, please contact the Shadow Robot Company.

Internal connections

If you are connecting the hand to a robot with internal cabling, then you may wish to use the internal connectors. Turn the hand over, and use the orange and green hex drivers to remove the connec-
tor cover. Connect the two cables to their relevant sockets. Now affix the hand to your robot arm.

Mounting the hand

Shadow Robot can supply an elbow adaptor plate to adapt the Hand to most other robot arms. However, if you wish to make your own fitting for the Hand:
Mounting

The Hand’s elbow plate contains eight screw holes which accept M6 bolts to a depth of 12mm. The holes are spaced equally from the centre on a circle with diameter 100mm. The overall diameter of the elbow plate is 135mm.

Powering up

You can power up the hand and PC in any order. You do not have to power up one before the other. When power is applied to the hand, the fans will be heard immediately.

Lights

On power up, the lights will be in the following state:

<table>
<thead>
<tr>
<th>Item</th>
<th>Color</th>
<th>Activity</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power LEDs</td>
<td>White</td>
<td>On</td>
<td>Power good</td>
</tr>
<tr>
<td>EC Link Active</td>
<td>Green</td>
<td>On</td>
<td>EtherCAT link established</td>
</tr>
<tr>
<td>EC Link Error</td>
<td>Red</td>
<td>Off</td>
<td>No EtherCAT link error</td>
</tr>
<tr>
<td>Run</td>
<td>Green</td>
<td>Off</td>
<td>Hand is in Init state</td>
</tr>
<tr>
<td>Application Layer Error</td>
<td>Red</td>
<td>On (during boot)</td>
<td>Verifying ET1200 EEPROM</td>
</tr>
<tr>
<td>Application Layer Error</td>
<td>Red</td>
<td>Then off</td>
<td>No EtherCAT packet error</td>
</tr>
<tr>
<td>ET1200 chip select</td>
<td>Yellow</td>
<td>On</td>
<td>PIC32 communicating with ET1200</td>
</tr>
</tbody>
</table>

Lights will also appear inside the base, indicating 5v, 6v and 24v (or 28v) supplies. These can only be seen by removing the covers.
Jiggling

This applies to the motor hand only. On reset, all of the strain gauges (torque sensors) in the motors need to be zeroed. This happens automatically. The motors are driven back and forth to try to relieve any tension on the tendons. Then both gauges are zeroed. You will therefore see all joints of the hand move slightly on power up or reset or power up.

Installing the software

On a new PC using the one-liner

We have created a one-liner that is able to install Docker, download the image and create a new container for you. It will also create two desktop icons, one to start the container and launch the hand and another one to save the log files locally. To use it, you first need to have a PC with Ubuntu installed on it (preferable version 16.04) then follow these steps:

• Check your hand interface ID:
  Before setting up the docker container, the EtherCAT interface ID for the hand needs to be discovered. In order to do so, after plugging the hand’s ethernet cable into your machine and powering it up, please run

  $ sudo dmesg

  command in the console. At the bottom, there will be information similar to the one below:


  In the above example, ‘enp0s25’ is the interface ID that is needed.

• Get ROS Upload login credentials
  If you want to upload technical logged data (ROS logs, backtraces, crash dumps etc.) to our server and notify the Shadow’s software team to investigate your bug then you need to enable logs uploading in the one-liner. In order to use this option you need to obtain a unique upload key by emailing sysadmin@shadowrobot.com. When you receive the key you can use it when running the one-liner installation tool. To enable the logs uploading you need to add the command line option use_aws=true to the one-liner. After executing the one-liner, it will prompt you to enter your upload key and press enter to continue. Please copy and paste your key from the email you received by Shadow Robot.

• Check your hand configuration branch:
  You should have the name of your sr_config hand branch which contains the specific configuration of your hand (calibration, controller tuning etc…). Usually it is something like this: shadowrobot_XXXXXX. Where XXXXXX are the 6 digits contained in the serial number of the hand labelled underneath the robot base.

  If you are unsure please contact us.

• Run the one-liner:
  The one-liner will install Docker, pull the image from Docker Hub, and create and run a container with the parameters specified. In order to use it, use the following command:

  Please replace [EtherCAT interface ID] with your Interface ID and [sr_config_branch] with your unique sr_config branch

  ROS Kinetic (Recommended):

Examples: For Interface ID ens0s25 and sr_config_branch shadow_12345

```
$ bash <(curl -Ls bit.ly/run-aurora) docker_deploy product=hand_e ethercat_
  →interface=ens0s25 config_branch=shadow_12345
```

Same as above but with ROS logs upload enabled

```
$ bash <(curl -Ls bit.ly/run-aurora) docker_deploy product=hand_e ethercat_
  →interface=ens0s25 config_branch=shadow_12345 use_aws=true
```

If you have an Nvidia graphics card, you can add nvidia_docker to set the nvidia-docker version. Use nvidia_docker=1 or nvidia_docker=2 for version 1.0 or 2.0 respectively.

ROS Indigo:

```
$ bash <(curl -Ls bit.ly/run-aurora) docker_deploy product=hand_e ethercat_
  →interface=[EtherCAT interface ID] config_branch=[sr_config_branch] tag=indigo-
  →release
```

Examples: For Interface ID ens0s25 and sr_config_branch shadow_12345

```
$ bash <(curl -Ls bit.ly/run-aurora) docker_deploy product=hand_e ethercat_
  →interface=ens0s25 config_branch=shadow_12345 tag=indigo-release
```

Same as above but with ROS logs upload enabled

```
$ bash <(curl -Ls bit.ly/run-aurora) docker_deploy product=hand_e ethercat_
  →interface=ens0s25 config_branch=shadow_12345 tag=indigo-release use_aws=true
```

You can also add reinstall=true true in case you want to reinstall the docker image and container. When it finishes it will show if it was successful or not and it will create five desktop icons on your desktop that you can double-click to launch the hand container, save the log files from the active containers to your desktop and perform various actions on the hand (open, close and demo). The icon that launches the hand looks like this:

![Hand_Launcher](desktop_icon)

And for saving the logs:

![ROS_Logs_Saver](log_icon)

**Using a PC that Shadow provided**

In this case, the previous steps would have been performed by the Shadow team before, then the only thing to do to start the Hand is to either double-click the desktop icon or to run the container using:
$ docker start dexterous_hand_real_hw

You can check the currently available containers using:

$ docker ps -a

The container will be ready when fingers move to the zero position.

**Saving log files and uploading data to our server**

When running the one-liner, along with the icon that starts the Dexterous Hand, you will also notice a second icon named Save logs that is used to retrieve and copy all the available logs files from the active containers locally on your Desktop. This icon will create a folder that matches the active container’s name and the next level will include the date and timestamp it was executed. When it starts, it will prompt you if you want to continue, as by pressing yes it will close all active containers. After pressing “yes”, you will have to enter a description of the logging event and will start coping the bag files, logs and configuration files from the container and then exit. Otherwise, the window will close and no further action will happen. If you provided an upload key with the one-liner installation then the script will also upload your LOGS in compressed format to our server and notify the Shadow’s software team about the upload. This will allow the team to fully investigate your issue and provide support where needed.

**Starting the driver**

- **Shadow Hand Driver** Launch the driver for the Shadow Hand using the desktop icon ‘Shadow_Hand_Launcher’ if the one-liner was executed using the launch_hand=true argument or at a terminal (in the container), type:

  $ roslaunch sr_ethercat_hand_config sr_rhand.launch

- **Lights in the hand**: When the ROS driver is running you should see the following lights on the Palm:

<table>
<thead>
<tr>
<th>Light</th>
<th>Colour</th>
<th>Activity</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run</td>
<td>Green</td>
<td>On</td>
<td>Hand is in Operational state</td>
</tr>
<tr>
<td>CAN1/2 Transmit</td>
<td>Blue</td>
<td>V.fast flicker</td>
<td>Demand values are being sent to the motors</td>
</tr>
<tr>
<td>CAN1/2 Receive</td>
<td>Blue</td>
<td>V.fast flicker</td>
<td>Motors are sending sensor data</td>
</tr>
<tr>
<td>Joint sensor chip select</td>
<td>Yellow</td>
<td>On</td>
<td>Sensors being sampled</td>
</tr>
</tbody>
</table>

  After killing the driver, the lights will be in a new state:

<table>
<thead>
<tr>
<th>Light</th>
<th>Colour</th>
<th>Activity</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run</td>
<td>Green</td>
<td>Blinking</td>
<td>Hand is in Pre-Operational state</td>
</tr>
<tr>
<td>CAN1/2 Transmit</td>
<td>Blue</td>
<td>Off</td>
<td>No messages transmitted on CAN 1/2</td>
</tr>
<tr>
<td>CAN1/2 Receive</td>
<td>Blue</td>
<td>Off</td>
<td>No messages received on CAN 1/2</td>
</tr>
<tr>
<td>Joint sensor chip select</td>
<td>Yellow</td>
<td>Off</td>
<td>Sensors not being sampled</td>
</tr>
</tbody>
</table>

**1.1.4 Setting up a simulated hand**

**Gazebo**

Gazebo is our default simultator. So follow the intructions on the next section to install and run a simulation of our robot hands using Gazebo.
Installing the software (sim)

If you do not actually have a real hand but would like to use our hand in simulation, then please run the following command:

ROS Kinetic (Recommended):

```
$ bash <(curl -Ls bit.ly/run-aurora) docker_deploy product=hand_e sim_hand=true, launch_hand=true
```

ROS Indigo:

```
$ bash <(curl -Ls bit.ly/run-aurora) docker_deploy product=hand_e tag=indigo-release, sim_hand=true launch_hand=true
```

You can also add reinstall=true true in case you want to reinstall the docker image and container. When it finishes it will show:

```
Operation completed
```

and it will create two desktop icons on your desktop that you can double-click to launch the hand or save the log files from the active containers to your desktop.

If you have an Nvidia graphics card, you can add nvidia_docker to set the nvidia-docker version. Use nvidia_docker=1 or nvidia_docker=2 for version 1.0 or 2.0 respectively.

Starting a robot simulation

First you need to start the hand container by either double clicking the icon “Hand_Container” or running the following command:

```
$ docker start dexterous_hand_real_hw
```

Shadow Dexterous hands

- The hand will start automatically if you have ran the one-liner with the argument `launch_hand=true` or to start it manually, simply do (in the container):

```
$ roslaunch sr_robot_launch srhand.launch
```

This will launch the five finger hand (shadowhand_motor) by default.

- If you want to start the dexterous hand plus, you can add the `hand_type` like this:

```
$ roslaunch sr_robot_launch srhand.launch hand_type:=hand_e_plus
```

- If you want to launch another hand, these are the hands available:
To start the simulation, you can run:

```
$ roslaunch sr_robot_launch srhand.launch robot_description=`rospack find sr_description`/robots/shadowhand_motor.urdf.xacro
```

The `robot_description` param can be changed to start any of the available Shadow hands shown in the table.

- If it is a left hand, `hand_id:=lh` should be added. For example:

  ```
  $ roslaunch sr_robot_launch srhand.launch robot_description=`rospack find sr_description`/robots/shadowhand_left_motor.urdf.xacro hand_id:=lh
  ```

- Moveit will enable advanced behaviour (inverse kinematics, planning, collision detection, etc...), but if it is not needed, you can set `use_moveit:=false`

If when you launch the hand you see some errors related to LibGL, this is a good indication that you have an NVidia card and should add an ‘nv’ flag when running the installation one liner. Run the one liner again with the correct NVidia flags mentioned above and also `-r true` to reinstall the docker image and container.
Bimanual system

To start the simulation of a bimanual system, you can run:

```bash
$ roslaunch sr_robot_launch sr_bimanual.launch use_moveit:=true
```

Mujoco

Mujoco is a robot simulator that has now been adopted by a wide community of researchers and developers, specially for machine learning applications. We have developed the tools and the model of our dexterous hand to use Mujoco as an alternative to Gazebo. Mujoco is not free so follow the next instructions if you have already a Mujoco License.

Obtaining the mujoco simulation

The software is most easily obtained by downloading and running our docker images. Which image you should use depends on whether your host machine has an Nvidia GPU.

Non-Nvidia GPU systems

Run the following command to pull the docker image:

```bash
$ docker pull shadowrobot/dexterous-hand:kinetic-mujoco-release
```

Then use this to run the docker container for the first time:

```bash
$ docker run --name mujoco_container -it -e DISPLAY -e LOCAL_USER_ID=$(id -u) -e QT_X11_NO_MITSHM=1 -v /tmp/.X11-unix:/tmp/.X11-unix:rw --net=host --privileged -w /shadowrobot/dexterous-hand:kinetic-mujoco-release bash
```

Nvidia GPU systems

If you have Nvidia GPU, for steps 1 and 2, use following commands instead:

```bash
$ docker pull shadowrobot/dexterous-hand:kinetic-mujoco-release-nvidia
```

```bash
$ nvidia-docker run --name mujoco_container -it -e DISPLAY -e LOCAL_USER_ID=$(id -u) -e QT_X11_NO_MITSHM=1 -v /tmp/.X11-unix:/tmp/.X11-unix:rw --net=host --privileged -w /shadowrobot/dexterous-hand:kinetic-mujoco-release-nvidia bash
```

Note that you will need `nvidia-docker` (version 1) installed. Version 2 support is coming soon.
Running the Mujoco Simulation

Inside the container, put your Mujoco key in /home/user/mjpro150/bin/mjkey.txt.
The easiest way is to just open the file inside of the container using “vim” and paste the contents of the key there.
You could also use docker cp, on your host machine terminal:

```
$ docker cp <path to your mujoco key file> mujoco_container:/home/user/mjpro150/bin/mjkey.txt
```

You can then start the simulation by running the following in the docker container terminal:

```
roslaunch sr_robot_launch srhand_mujoco.launch
```

Re-Using your Mujoco Container

After stopping your container (in order to shut down your machine, for example), you can re-use the same container by running:

```
docker start mujoco_container && docker attach mujoco_container
```

This will start the container and connect you to the container terminal again. You can run the same roslaunch command as above to start the simulation again.

1.2 Software description of the Hand

1.2.1 Graphical User Interface

The majority of functionality is provided by the software Application Programmer Interface (API). However, a few simple functions are provided in the Graphical User Interface (GUI) to test the hand, validate that it is working correctly, and adjust some of its settings.

Starting the interface

You may open the Graphical User Interface to try out some functions of the hand. From the Docker terminal, type:

```
$ rqt
```

This interface contains a number of plugins for interacting with the EtherCAT hand. Most of them are available from the Plugins → Shadow Robot menu.

Robot Monitor

We can check that everything on the robot is working correctly using the Diagnostic Viewer.

Plugins → Robot Tools → Diagnostic Viewer
Errors
✓ /Other : OK
✓ /Other/EtherCAT Master : Motors halted
✓ /Other/Realtime Control Loop : OK

Warnings

All
✓ (E: 2, W: 0) Other : OK
✓ (E: 25, W: 0) Shadow Hand : OK
✓ Devices : OK
  ✓ EtherCAT Device #01 (Product SIX) : OK
  ✓ SRBridge : 00 : OK
✓ (E: 25, W: 0) Fingers : OK
  ✓ (E: 3, W: 0) First Finger : OK
  ✓ (E: 4, W: 0) Little Finger : OK
  ✓ (E: 2, W: 0) Middle Finger : OK
  ✓ (E: 3, W: 0) Ring Finger : OK
  ✓ (E: 5, W: 0) Thumb : OK
  ✓ (E: 2, W: 0) Wrist : OK
  ⚠️ Tactiles : No items found, expected 5

Last message received 0 seconds ago
This brings up a dialog box containing a tree of all parts of the robot. All parts should be marked with a green tick.

You can examine one motor in detail by double-clicking on it. This brings up the Motor Monitor dialog. This window can be used to check the status of a motor, or debug any problems.
**Full name:** /Shadow Hand/Fingers/Middle Finger/SRDMotor MFJ4  
**Component:** SRDMotor MFJ4  
**Hardware ID:** 6-21  
**Level:** OK  
**Message:** OK  

**Motor ID:** 1  
**Motor ID in message:** 0  
**Strain Gauge Left:** 1  
**Strain Gauge Right:** 4  
**Executed Effort:** 0.000000  
**Motor Flags:** None  
**Measured Current:** 0.000000  
**Measured Voltage:** 105.144531  
**Temperature:** 35.722656  
**Number of CAN messages received:** 4630  
**Number of CAN messages transmitted:** 4630  
**Force control Pterm:** 0  
**Force control Iterm:** 0  
**Force control Dterm:** 0  
**Force control F:** 1900  
**Force control P:** 4800  
**Force control I:** 320  
**Force control D:** 18000  
**Force control Imax:** 1700  
**Force control Deadband:** 5  
**Force control Sign:** +  
**Last Measured Effort:** -3.000000  
**Last Commanded Effort:** 0.000000  
**Encoder Position:** -0.676130  
**Firmware svn revision (server / pic / modified):** 1837 / 1815 / False  
**Tests:** 0
The following table has some more information on what each of these fields mean.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Name</td>
<td></td>
</tr>
<tr>
<td>Component</td>
<td></td>
</tr>
<tr>
<td>Hardware ID</td>
<td></td>
</tr>
<tr>
<td>Level</td>
<td></td>
</tr>
<tr>
<td>Message</td>
<td>Any error or status messages</td>
</tr>
<tr>
<td>Motor ID</td>
<td>This is the motor number. Range [0..19]</td>
</tr>
<tr>
<td>Motor ID in message</td>
<td>For debugging only</td>
</tr>
<tr>
<td>Strain Gauge Left / Right</td>
<td>These are the ADC readings from the two gauges</td>
</tr>
<tr>
<td>Executed Effort</td>
<td></td>
</tr>
<tr>
<td>Motor Flags</td>
<td>See motor flags table below</td>
</tr>
<tr>
<td>Measured current</td>
<td>Current flowing through the motor (Amps)</td>
</tr>
<tr>
<td>Measured Voltage</td>
<td>The motor power supply voltage. Not the voltage at the motor</td>
</tr>
<tr>
<td>Temperature</td>
<td>The temperature measured near the motor. The actual motor winding tempera-</td>
</tr>
<tr>
<td></td>
<td>ture will be higher than this. (°C)</td>
</tr>
<tr>
<td>Number of CAN messages</td>
<td>Received messages should be twice the transmitted messages</td>
</tr>
<tr>
<td>Force control P, I, D terms</td>
<td>These are the PID terms from inside the motor’s torque controller. They may be</td>
</tr>
<tr>
<td></td>
<td>useful for debugging if plotted.</td>
</tr>
<tr>
<td>Force control F, P, I, D, Imax, Deadband, Sign</td>
<td>These are the FPID gain settings used by the motor’s torque controller. They can be changed using the controller tuner.</td>
</tr>
<tr>
<td>Last Measured Effort</td>
<td>Difference between the two gauge readings (Torque)</td>
</tr>
<tr>
<td>Last Commanded Effort</td>
<td>Torque requested by the host-side control algorithms</td>
</tr>
<tr>
<td>Encoder Position</td>
<td>The angle of the joint in radians (ROS always calls this Encoder position, even if the robot uses Hall effect sensors)</td>
</tr>
<tr>
<td>Firmware svn revision</td>
<td>xxxx: The latest version of the firmware available at build time</td>
</tr>
<tr>
<td></td>
<td>xxxx: The version of the firmware in the motor MCU</td>
</tr>
<tr>
<td></td>
<td>False: There are no un-checked-in modifications to this firmware. This should never be true.</td>
</tr>
</tbody>
</table>

**Controller tuner**

It is possible to adjust the settings for any of the Position or Force (Motor) controllers. Plugins → Shadow Robot → Basic → Controller Tuner
Here you can select a finger, thumb or wrist joints, and adjust the different position control parameters. Click Set Selected to send the new values to the motors and make them take effect.

- **“P”, “I” & “D” terms**: Gain parameters of the position PID controller. By default, Shadow tunes the parameters using P or PD combinations. The user can add “I” gains in the control if they consider it necessary.

- **Max_force**: This puts a limit on the output (PWM) value that will be sent from the host to the motor by the position controller. It can be useful when setting up a controller for the first time to limit the motor power to a safe level.

- **Position_Deadband**: The error is considered to be zero if it is within \( \pm \)deadband. This value should be set as a little more than the noise on the sensor. The units of deadband are the same as the value being controlled. So, the deadband for a position controller is in radians.
Force controller

- **“P”, “I” & “D” terms**: Gain parameters of the torque PID controller. By default, Shadow tunes the parameters using just P gain for the torque control.

- **Max_PWM**: This puts a limit on the final PMW value that will be sent to the motor by the torque controller. It can be useful when setting up a controller for the first time to limit the motor power to a safe level.

- **Deadband**: The error is considered to be zero if it is within ±deadband. This value should be set as a little more than the noise on the sensor. The units of deadband are the same as the value being controlled. The deadband for a torque controller is in the units of the strain gauges.

- **Torque_Limit**: This value is used to limit the PWM at the end of the control loop. The control algorithm reduces the final PWM that goes to the motor making sure that the force in the strain gauge doesn’t overcome this limit value.

Click **Save** to save your settings.

Bootloader

The firmware in the motors MCUs can be updated from the PC, without opening up the motor base. This can be done from the GUI. Shadow will send you a new HEX if there is an update. **Plugins → Shadow Robot → Advanced → Motor Bootloader**

You will see a window listing each motor board, along with its current firmware SVN revision number.
• **Select Bootloader Hex File:** Next, tell the plugin which firmware to use. The file you should choose here is the one sent by Shadow.

• **Select your motors:** Now you may choose which motors to program. Either select one or more motors using the tick boxes, or click the Select All or Deselect All button.

• **Program Motors:** Now you can click the Bootload Motors button. The process is fairly slow, and takes about a 30 second per motor.

**Danger:** The change of file should be previously confirmed with us to ensure that is compatible with your hardware. **A wrong motor firmware update can crash the system of the robot.**

**Change controllers**

Use the *Change Controllers* plugin to load one of the three different types of controllers set by default. Simply click on a controller type, and it will call a service from the pr2_controller_manager to unload the currently running controller if necessary, and load the one you’ve selected. **Plugins → Shadow Robot → Change Controllers**
Apart from the three standard controls, you can set the parameters for different control strategies (host – motor) from this plugin. **Plugins → Shadow Robot → Advanced → Advanced Controls**

This plugin will not work if you have selected the trajectory controller.
**Danger:** Currently, the only fully supported types are position - pwm control (position control), and effort - torque control (teach mode control). **SELECTING OTHER TYPES MAY CAUSE UNPREDICTABLE RESULTS AND DAMAGE THE HARDWARE.**

**Motor Resetter**

If for some reason you need to reset the firmware on a motor, you can either press the reset button on the PCB itself (which requires removal of the base covers), or use this plugin. **Plugins → Shadow Robot → Basic → Motor Resetter**

Tick the motors you wish to reset, and click **Reset Motors**. You should see the corresponding joints jiggle as the motors auto-zero the strain gauges.

**Joint Sliders**

A simple interface has been provided to control the position of each joint using a slider. **Plugins → Shadow Robot → Joint Sliders**
A window with twenty sliders will appear. Moving any slider will cause the corresponding joint on the hand to move. You have to start the hand in either position control or teach mode. If the control is changed, reload the plugin to make sure that the sliders correspond to the control that is running at this moment.

**Hand Calibration**

This plugin is used internally by Shadow to calibrate the raw data from the position sensors. Plugins → Shadow Robot → Basic → Shadow Hand Calibration
It’s very unlikely that the sensors moved inside of the hand, BUT, if you find misalignments with the model and you require a re-calibration, contact Shadow Robot Company here: support@shadowrobot.com.

**Data Visualizer**

A GUI is provided to show all the data available for the Dexterous Hand. Plugins → Shadow Robot → Dexterous Hand Data Visualizer
You also can launch it separately from rqt by running the following command:

```bash
roslaunch sr_data_visualization data_visualizer.launch
```

In each tab, you can find information about:

- Joint states (position, effort, velocity)
- Control loops (setpoint, input, dinput/dt, output, error)
- Motor stats (Strain Gauge Left, Strain Gauge Right, Measured PWM, Measured Current, Measured Voltage, Measured Effort, Temperature, Unfiltered position, Unfiltered force, Last Commanded Effort, Encoder Position)
- Palm extras (Accelerometer, Gyro-meter, Analog inputs)
- Tactile sensor data (Pressure AC 0, Pressure AC 1, Pressure DC, Temperature AC, Temperature DC)
- Tactile sensor visualizer

The radio buttons let you choose a specific data to show or you can choose “All” to see several graphs being displayed at the same time.

### 1.2.2 Command line interface

All functions of the hand are available from the command line.

In the following sections, **Hand** refers to the shadow dexterous hand and **Host** refers to the host computer which is controlling the hand. Assume that all the topics are read only unless specified otherwise.

**Using rostopic**

To check how to interact with ROS topics, see: [http://wiki.ros.org/rostopic](http://wiki.ros.org/rostopic)

The following rqt_graph shows the flow of topics between nodes whilst the hand is running.
Here is a list of the available topics:

- **Calibration (Real hand only)**

  These topics are used during the Hand startup routine to make sure that the Hand is calibrated:

  ```
  /cal_sh_rh_*/calibrated
  /calibrated
  ```

  An empty message is published to the `/cal_sh_rh_*/calibrated` topics for each joint when they are calibrated. The `/calibrate_sr_edc` node subscribes to these topics and when all of them have had an empty message published to them, it publishes True to the `/calibrated` topic. Before empty messages have been received by all the joints it publishes False to the `/calibrated` topic.

- **Diagnostics (Real hand only)**

  ```
  /diagnostics
  /diagnostics_agg
  /diagnostics_toplevel_state
  ```

  These topics update at 2 Hz with information on each joint’s Temperature, Current, Measured effort and Command effort, as well as information about the EtherCat devices and firmware version.

- **Joint states**

  ```
  /joint_states
  ```

  This topic is read-only and updates at 100 Hz with the name, position, velocity and effort values of all joints in a Hand.

  **Example topic message:**

  ```
  name: [rh_FFJ1, rh_FFJ2, rh_FFJ3, rh_FFJ4, rh_LFJ1, rh_LFJ2, rh_LFJ3, rh_LFJ4, rh_LFJ5, rh_MFJ1, rh_MFJ2, rh_MFJ3, rh_MFJ4, rh_RFJ1, rh_RFJ2, rh_RFJ3, rh_RFJ4, rh_THJ1, rh_THJ2, rh_THJ3, rh_THJ4, rh_THJ5, rh_WRJ1, rh_WRJ2]
  position: [1.279751244673038, 1.7231505348398373, 1.2957917583498741, -0.00406710173435502, 0.054689233814909366, 1.253488840949725, 1.5395435039130654, -0.02170017906073821, 0.1489674305718295, 1.08814400717011, 1.638917596069165, 1.4315445985097324, 0.00989364236002074, 1.2257618075487349, 0.21905854304869088, -0.048455186771971595, -0.003280332337213066]
  ```
velocity: [-7.48433985952662e-06, -7.48433985952662e-06, 0.0023735860019749185, 0.00062181267775619, -0.0005871136552505063, -0.0005871136552505063, 0.0020967687295392933, 0.0004985252400775274, -9.485516545601461e-06, -9.485516545601461e-06, -0.0007068752456452666, -0.0012475428276090576, 0.0008426052935621657, 0.0008426052935621657, 0.00017469681801687975, -4.900148416020751e-05, 6.1559570145597239e-05, -1.3660655058510802, -1.3660655058510802, -1.957775048877130092884, -1.504877130092884, -0.3374653182042338, -1.507775048877130092884, -8.476660697182016, -8.476660697182016, -3.3867013328219056, -2.3404145772688683, -0.7688013735971971, 11.02319645071454, 0.8482082620071664, 0.08818910881575533, 1.127772119947565, -2.2344970991165316, -3.5544023107705667]

effort: [-1.3660655058510802, -1.3660655058510802, -1.957775048877130092884, -1.504877130092884, -0.3374653182042338, -1.507775048877130092884, -8.476660697182016, -8.476660697182016, -3.3867013328219056, -2.3404145772688683, -0.7688013735971971, 11.02319645071454, 0.8482082620071664, 0.08818910881575533, 1.127772119947565, -2.2344970991165316, -3.5544023107705667]

- etherCAT (Real hand only)
  
  /rh/debug_etherCAT_data

This topic is published by the driver and updates at 800 Hz with data from the Hand as it is received over EtherCAT, which is useful for debugging.

- **sensors** are the position sensors in the joints, which are included in every packet.
- **tactile** is the data from the tactile sensors, which are included in every packet.
- Data is received in two alternative packets for the motor torques, each holds data for half of the 20 motors. If **which_motors** is 0 then the data is for the first 10 motors. If 1, the data is for the second 10 motors.

- **motor_data_packet_torque** is the raw difference between the strain gauge in tension and the strain gauge in compression for each motor.
- **motor_data_type** is used to specify the data in **motor_data_packet_misc**. This data has been requested from the host. What value corresponds to which data is defined [here](#).
- **which_motor_data_arrived** is a bitmap, 20x1 dimensional array for the 20 motors, which shows which motors data has been received from. For example 349525 = 01010101010101010101.
- **which_motor_data_had_errors** is a bitmap for the motors which have errors.
- The tactile sensors attached to the Hand are selected during startup, their corresponding values are [here](#).
- **tactile_data_type** is used to specify the data in tactile, similar to **motor_data_type** and **motor_data_packet_misc**. In the Example topic message below the PST fingertip sensors are used, its value is referenced [here](#).
- **tactile_data_valid** is a bitmap for the 5 sensors that is 1 when there are no errors.
- **idle_time_us** is the time margin once the Hand has completed its processing and is ready for communication on the EtherCAT bus.

Note: More data is transmitted from the tactile sensors than is published to the etherCAT topic by default.

Example /rh/debug_etherCAT_data topic message:

```shell
header:
  seq: 176798
```

(continues on next page)
• Palm Extras

```
/rh/palm_extras
```

This topic updates at 84 Hz with data from additional devices plugged into the palm.

Example topic message:

```
layout:
  dim:
    - label: "accelerometer"
      size: 3
      stride: 0
    - label: "gyrometer"
      size: 3
```
The first six values are readings from an IMU set in the hand. The IMU is an add-on feature so some hands might not have this data available.

- **Tactile (Only for a real hand with tactile sensors)**

```
/rh/tactile
```

This topic is published by the driver at 100 Hz with data from tactile sensors.

Example topic message when using PST fingertip sensors:

```
thrue: 126618
  th: 1528813967
  frame_id: "rh_distal"
  pressure: [405, 428, 422, 401, 384]
  temperature: [1224, 1198, 1225, 1242, 1266]
```

Example topic message when using BioTac fingertip sensors:

```
tactiles:
  pac0: 2048
  pac1: 2054
  pdc: 2533
  tac: 2029
  tdc: 2556
  electrodes: [2622, 3155, 2525, 3062, 2992, 2511, 3049, 2928,]
  ...
  pac0: 0
  pac1: 0
  pdc: -9784
  tac: 32518
  tdc: 0
  electrodes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,](continues on next page)
BioTac (Only for a real hand with Biotac tactile sensors)

These topics are read-only and update at 100 Hz with data from the biotac sensors, which comprises their pressure, temperature and electrode resistance. This topic is published from the /biotac_republisher node which receives this data from the driver via the /rh/tactile topic. For further information about the biotacts, refer to their documentation: https://www.syntouchinc.com/wp-content/uploads/2016/12/BioTac_SP_Product_Manual.pdf

Example /rh/biotac_** topic message:

```plaintext
pac0: 2056
pac1: 2043
pdc: 2543
tac: 2020
tdc: 2454
electrodes: [2512, 3062, 2404, 2960, 2902, 2382, 2984, 138, 2532, 2422, 2809, 3167, 2579, 2950, 2928, 2269, 2966, 981, 2374, 2532, 3199, 3152, 3155, 3033]
```

Trajectory Controller

- Command

```plaintext
/rh_trajectory_controller/command
```

This topic can be published to and is the set position for the trajectory controller. It comprises an array of all the joints set positions and is used for commanding the robot. For example the rqt joint sliders publish to it.

Example topic message:

```plaintext
joint_names: [rh_FFJ1, rh_FFJ2, rh_FFJ3, rh_FFJ4, rh_MFJ1, rh_MFJ2, rh_MFJ3, rh_MFJ4, rh_RFJ1, rh_RFJ2, rh_RFJ3, rh_RFJ4, rh_LFJ1, rh_LFJ2, rh_LFJ3, rh_LFJ4, rh_LFJ5, rh_THJ1, rh_THJ2, rh_THJ3, rh_THJ4, rh_THJ5, rh_WRJ1, rh_WRJ2]
points:
positions: [0.24434609527920614, 0.8203047484373349, 0.8552199347332949, 1.0297442586766545, 1.4311699866353502, 1.413716694115407, 0.007182575752410699, 0.9773843911618246, 1.]
```

(continues on next page)
velocities: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

accelerations: []

effort: []

time_from_start:
secs: 0
nsecs: 5000000

State

/rh_trajectory_controller/state

This topic is read-only and update at 50 Hz from the trajectory controller with the positions and velocities of all 24 joints.

Example topic message:

positions: [0.0029928404547430176, 0.0007821521859359137, 0.0041027846795928363, -0.001230489872427576, 0.002876479952986344, 0.0006426181816490129, 0.00635491922407833, 0.00213663812281073, 0.0032796180654738465, 0.0020929781564538175, 0.0063066586043154516, 0.0008137524637908733, -2.1288137004304986e-05, 0.0009348013388894572, 0.003295237358051928, 0.039881480505079236, -0.0035961821430152696, 0.003603043080507987, 2.998784142176428e-05, -0.002993407459852484, -8.9996446532783e-05] velocities: [-0.0008510441551395189, -0.0008510441551395189, 0.00016883698712266695, 0.00034715798956923955, -0.00017869100331692196, -0.0001275520583476054, -0.0004885423191519772, 0.00012555078906251334, 0.00012555078906251334, 0.0028653614401722843, -0.0008023399951605057, 0.0011760287859774613, 0.0011760287859774613, -0.0005423468659163991, -0.00017066612487367117, 0.0003102610817406156, -0.00112705278802167, -0.001465588685391472, -0.00028520412005307133, -0.00029795158858164227, 0.0002596403670543647, -5.819600689424957e-05, -0.000298034764377659]

follow_joint_trajectory

These topics provide information about positions, velocities and accelerations of joints whilst executing a trajectory from the current pose to the goal pose:

/rh_trajectory_controller/follow_joint_trajectory/feedback
/rh_trajectory_controller/follow_joint_trajectory/goal
/rh_trajectory_controller/follow_joint_trajectory/result
/rh_trajectory_controller/follow_joint_trajectory/status

The following topic is used to stop a currently executing trajectory:

/rh_trajectory_controller/follow_joint_trajectory/cancel

• Position Controller

  – Command

  /sh_rh_*_position_controller/command

These topics can be published to and are the set position of each joint in radians. The topics are subscribed to by the driver (/sr_hand_robot node). This topic is used to communicate the set position with the rqt
Joint Sliders plugin, when using position control. The Hand can be set to position control using the Change Controllers rqt plugin.

Example of running

```
$ rostopic info /sh_rh_ffj0_position_controller/command
```

Type: std_msgs/Float64
Publishers:

/rqt_gui_py_node_23644 (http://shadow-bravo:38385/)

Subscribers:

/sh_rh_ffj0_position_controller/state

Example topic message:

```
data: 0.628318530718
```

– State

```
/sh_rh_*_position_controller/state
```

These topics are published at 87 Hz by the driver (/sr_hand_robot node). They contain messages of type control_msgs/JointControllerState, which contain the parameters used for the each joints position controller.

Example topic message:

```
set_point: 1.1113358647
process_value: 1.11095072243
process_value_dot: 0.000426142920695
error: 0.0
time_step: 0.001
command: 0.0
p: -3800.0
i: 0.0
d: 0.0
i_clamp: 0.0
antiwindup: False
```

– Force

```
/sh_rh_*_position_controller/max_force_factor
```

The /sh_rh_*_position_controller/max_force_factor topic can be published to and scales down the maximum output command of the joints position controller. The output command is interpreted by the driver (/sr_hand_robot node) as PWM if the driver is in PWM mode, or as tendon force if it is in Torque mode. The maximum force is controlled by the parameter “max_force” that is specified in this yaml file. max_force_factor has a value between [0.0, 1.0] and controls the percentage of the max_force that will be effectively considered.

This parameter doesn’t exist in the grasp controller.

• PID parameters
These topics are read-only and contain parameters used for tuning the position controllers. They should not be published to directly, but can be accessed through rqt_reconfigure.

- TF

These topics store information on the active transforms in the ROS environment and holds their position and orientation in relation to their parents. Static tfs are fixed and the dynamic tfs update at 100 Hz. They can be published to, as well as read from. For further information on ROS tfs see the ROS wiki: [http://wiki.ros.org/tf](http://wiki.ros.org/tf)

- Mechanism Statistics

This topic is read-only and updates at 1 Hz with the attributes of each joint, for example:

```plaintext
position: 0.715602037549
velocity: 0.0
measured_effort: -11.088
commanded_effort: -10.799974692
is_calibrated: False
violated_limits: False
odometer: 0.0
min_position: 0.715218542352
max_position: 0.715985532746
max_abs_velocity: 0.0363159179688
max_abs_effort: 15.84
```

- MoveIt! Topics

In Position control the MoveIt topics are used for trajectory planning. They are described in their documentation here: [https://moveit.ros.org/documentation/](https://moveit.ros.org/documentation/)

- Collisions

These are used for object collision avoidance if it is active.

- Trajectory Execution

Live information regarding the current trajectory execution.

- RViz Topics

These topics are used to interface with RViz. Documentation for this can be found here: [http://wiki.ros.org/rviz#User_Documentation](http://wiki.ros.org/rviz#User_Documentation)
Using rosservice

To reset individual motors, E.G. FFJ3:

```
$ rosservice call /realtime_loop/reset_motor_FFJ3
```

To change control modes, E.G. teach mode:

```
$ rosservice call /realtime_loop/xxxxxx
```

1.2.3 Writing controllers

Rather than use the ROS topics to access sensor data, you will need to write a plugin for the PR2 Controller Manager. This will give you access to the sensor data at the full 1kHz rate, and allow you to create your own control algorithms for the hand. Please see this page for more information about the PR2 Controller Manager: http://ros.org/wiki/pr2_controller_manager

The Controller Manager is the node that talks to the hardware via EtherCAT and provides a facility for hosting plugins. The position controllers you have already used are examples of this. Note that the Controller Manager can host any number of running controllers but one should be loaded at a time for a given joint so they don’t fight for control.

1.2.4 Deeper settings

Editing PID settings

The motor controller PID settings are stored in a YAML files. You can find the files in the next folder:

```
$ roscd sr_ethercat_hand_config/controls/
```

Changing motor data update rates

Each motor can return two sensor readings every 2ms. The first is always the measured torque. The second is requested by the host. This allows the host to decide on the sensor update rate of each sensor. Currently, the rates cannot be adjusted at run-time, and are specified in a file which you can edit. To edit the file:

```
$ roscd sr_robot_lib/config
$ gedit motor_data_polling.yaml
```

The complete list of motor sensors appears in the file, along with a number
<table>
<thead>
<tr>
<th>Number</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>Read once when the driver is launched</td>
</tr>
<tr>
<td>-1</td>
<td>Read as fast as possible</td>
</tr>
<tr>
<td>0</td>
<td>Do not use zero</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Read period in seconds</td>
</tr>
</tbody>
</table>

Sensors set to -1 will be read in turn, unless it’s time to read another sensor. Usually 5 sensors are set to -1, meaning that they are sampled at 100Hz.

### 1.2.5 Repositories

Our code is split into different repositories:

- **sr_common**: This repository contains the bare minimum for communicating with the Shadow Hand from a remote computer (urdf models and messages).
- **sr_core**: These are the core packages for the Shadow Robot hardware and simulation.
- **sr_interface**: This repository contains the high level interface and its dependencies for interacting simply with our robots.
- **sr_tools**: This repository contains more advanced tools that might be needed in specific use cases.
- **sr_visualization**: This repository contains the various rqt_gui plugins we developed.
- **sr_config**: This repository contains the customer specific configuration for the Shadow Robot Hand.

### 1.2.6 Robot commander

The robot commander provides a high level interface to easily control the different robots supported by Shadow Robot. It encapsulate the functionality provided by different ROS packages, specially the moveit_commander, enabling their access throughout a more simplified interface.

There are two classes available:

- **SrRobotCommander**: base class
- **SrHandCommander**: hand management class

**SrRobotCommander**

**Overview**

The main purpose of the robot commander is to provide a base class to the hand commander. The RobotCommander should not be used directly unless necessary. Use the **SrHandCommander** instead.

Examples of usage can be found [here](#).

In the following sections, you can find decriptions of the most relevant functions of the hand commander.

**Basic terminology**

A robot is described using an **srdf** file which contains the semantic description that is not available in the **urdf**. It describes a robot as a collection of **groups** that are representations of different sets of joints which are useful for
planning. Each group can have its **end-effector** and **group states** specified. Group states are a specific set of joint values predefined for a group with a given name, for example `close_hand` or `open_hand`.

As the robot commander is a high level wrapper of the `moveit_commander`, its constructor takes the name of one of the robot groups for which the planning will be performed.

### Setup

Import the hand commander along with basic rospy libraries:

```python
import rospy
from sr_robot_commander.sr_hand_commander import SrHandCommander
```

The constructor for the `SrHandCommander` takes a name parameter that should match the group name of the robot to be used.

As well as creating an instance of the `SrHandCommander` class, we must also initialise our ros node:

```python
rospy.init_node("sr_hand_commander_example", anonymous=True)
hand_commander = SrHandCommander("right_hand")
```

### Getting basic information

We can get the name of the robot, group or planning reference frame:

```python
print "Robot name: ", hand_commander.get_robot_name()
print "Group name: ", hand_commander.get_group_name()
print "Planning frame: ", hand_commander.get_planning_frame()
```

Get the list of names of the predefined group states from the srdf and warehouse for the current group:

```python
# Refresh them first if they have recently changed
hand_commander.refresh_named_targets()
print "Named targets: ", hand_commander.get_named_targets()
```

Get the joints position and velocity:

```python
joints_position = hand_commander.get_joints_position()
joints_velocity = hand_commander.get_joints_velocity()
print("Hand joint positions\n" + str(joints_position) + "\n")
print("Hand joint velocities\n" + str(joints_velocity) + "\n")
```

Get the current joint state of the group being used:

```python
current_state = hand_commander.get_current_state()
# To get the current state while enforcing that each joint is within its limits
current_state = hand_commander.get_current_state_bounded()
```

### Setting functions

You can change the reference frame to get pose information:
You can also activate or deactivate the teach mode for the robot:

```python
# Activation: stops the trajectory controllers for the robot, and sets it to teach mode.
hand_commander.set_teach_mode(True)

# Deactivation: stops the teach mode and starts trajectory controllers for the robot.
# Currently this method blocks for a few seconds when called on a hand, while the hand parameters are reloaded.
hand_commander.set_teach_mode(False)
```

### Plan/move to a joint-space goal

Using the methods `plan_to_joint_value_target`, `move_to_joint_value_target` or `move_to_joint_value_target_unsafe`, a set of joint values can be given for the specified group to create a plan and send it for execution.

**Parameters:**

- `joint_states` is a dictionary with joint name and value. It can contain joint values of which need to be changed.
- `wait` indicates if the method should wait for the movement to end or not (default value is True)
- `angle_degrees` should be set to true if the input angles are in degrees (default value is False)

**IMPORTANT:** Bear in mind that the names of the joints are different for the right and left hand.

**Example**

```python
rospy.init_node("robot_commander_examples", anonymous=True)
hand_commander = SrHandCommander(name="right_hand")
joints_states = {
    'rh_FFJ1': 90, 'rh_FFJ2': 90, 'rh_FFJ3': 90, 'rh_FFJ4': 0.0,
    'rh_MFJ1': 90, 'rh_MFJ2': 90, 'rh_MFJ3': 90, 'rh_MFJ4': 0.0,
    'rh_RFJ1': 90, 'rh_RFJ2': 90, 'rh_RFJ3': 90, 'rh_RFJ4': 0.0,
    'rh_LFJ1': 90, 'rh_LFJ2': 90, 'rh_LFJ3': 90, 'rh_LFJ4': 0.0, 'rh_LFJ5': 0.0,
    'rh_THJ1': 40, 'rh_THJ2': 35, 'rh_THJ3': 0.0, 'rh_THJ4': 65, 'rh_THJ5': 15,
    'rh_WRJ1': 0.0, 'rh_WRJ2': 0.0}
hand_commander.move_to_joint_value_target(joints_states, wait=False, angle_degrees=True)
```

In this example, joint states for a hand are sent to the `HandCommander`, the method is prompted by the `wait=False` argument to not wait for the movement to finish executing before moving on to the next command and the `angle_degrees=True` argument tells the method that the input angles are in degrees, so require a conversion to radians.
**Plan/move to a predefined group state**

Using the methods `plan_to_named_target` or `move_to_named_target` will allow to plan or move the group to a predefined pose. This pose can be defined in the srdf or saved as a group state in the moveit warehouse.

Parameters:

- `name` is the unique identifier of the target pose
- `wait` indicates if the method should wait for the movement to end or not (default value is True)

**Example**

`pack` is a predefined pose defined in the SRDF file for the `right_hand` group:

```xml
<group_state group="right_hand" name="pack">
    <joint name="rh_THJ1" value="0.52"/>
    <joint name="rh_THJ2" value="0.61"/>
    <joint name="rh_THJ3" value="0.00"/>
    <joint name="rh_THJ4" value="1.20"/>
    <joint name="rh_THJ5" value="0.17"/>
    <joint name="rh_FFJ1" value="1.5707"/>
    <joint name="rh_FFJ2" value="1.5707"/>
    <joint name="rh_FFJ3" value="1.5707"/>
    <joint name="rh_FFJ4" value="0"/>
    <joint name="rh_MFJ1" value="1.5707"/>
    <joint name="rh_MFJ2" value="1.5707"/>
    <joint name="rh_MFJ3" value="1.5707"/>
    <joint name="rh_MFJ4" value="0"/>
    <joint name="rh_RFJ1" value="1.5707"/>
    <joint name="rh_RFJ2" value="1.5707"/>
    <joint name="rh_RFJ3" value="1.5707"/>
    <joint name="rh_RFJ4" value="0"/>
    <joint name="rh_LFJ1" value="1.5707"/>
    <joint name="rh_LFJ2" value="1.5707"/>
    <joint name="rh_LFJ3" value="1.5707"/>
    <joint name="rh_LFJ4" value="0"/>
    <joint name="rh_WRJ1" value="0"/>
    <joint name="rh_WRJ2" value="0"/>
</group_state>
```

Here is how to move to it:

```python
rospy.init_node("robot_commander_examples", anonymous=True)
hand_commander = SrHandCommander(name="right_hand")

# Only plan
hand_commander.plan_to_named_target("pack")

# Plan and execute
hand_commander.move_to_named_target("pack")
```
Move through a trajectory of predefined group states

Using the method `run_named_trajectory`, it is possible to specify a trajectory composed of a set of names of previously defined group states (either from SRDF or from warehouse), plan and move to follow it.

Parameters:

- `trajectory` specify a dictionary of waypoints with the following elements:
  - `name`: the name of the way point
  - `interpolate_time`: time to move from last waypoint
  - `pause_time`: time to wait at this waypoint

Example

```python
trajectory = [
  {
    'name': 'open',
    'interpolate_time': 3.0
  },
  {
    'name': 'pack',
    'interpolate_time': 3.0,
    'pause_time': 2
  },
  {
    'name': 'open',
    'interpolate_time': 3.0
  },
  {
    'name': 'pack',
    'interpolate_time': 3.0
  }
]
hand_commander.run_named_trajectory(trajectory)
```

# If you want to send the trajectory to the controller without using the planner, you can use the unsafe method:
hand_commander.run_named_trajectory_unsafe(trajectory)

Check if a plan is valid and execute it

Use the method `check_plan_is_valid` and `execute` to check if the current plan contains a valid trajectory and execute it. Only has meaning if called after a planning function has been attempted.

Example

```python
import rospy
from sr_robot_commander.sr_hand_commander import SrHandCommander
rospy.init_node("robot_commander_examples", anonymous=True)
```
hand_commander = SrHandCommander()

hand_commander.plan_to_named_target("open")
if hand_commander.check_plan_is_valid():
    hand_commander.execute()

Stop the robot

Use the method `send_stop_trajectoryunsafe` to send a trajectory with the current joint state to stop the robot at its current position.

Example

```python
hand_commander.send_stop_trajectoryunsafe()
```

SrHandCommander

Overview

The SrHandCommander inherits all methods from the `robot commander` and provides commands specific to the hand. It allows the state of the tactile sensors and joints effort to be read, and the maximum force to be set.

Setup

Import the hand commander along with basic rospy libraries and the hand finder:

```python
import rospy
from sr_robot_commander.sr_hand_commander import SrHandCommander
from sr_utilities.hand_finder import HandFinder

rospy.init_node("hand_finder_example", anonymous=True)

hand_commander = SrHandCommander(name = "rh_first_finger",
hand_parameters=hand_parameters,
hand_serial=hand_serial)
```

Example

```python
# Using the HandFinder
hand_finder = HandFinder()
hand_parameters = hand_finder.get_hand_parameters()
hand_serial = hand_parameters.mapping.keys()[0]

# If name is not provided, it will set "right_hand" or "left_hand" by default,
# depending on the hand.
hand_commander = SrHandCommander(name = "rh_first_finger",
hand_parameters=hand_parameters,
hand_serial=hand_serial)
```
Alternatively you can launch the hand directly

```python
hand_commander = SrHandCommander(name = "right_hand", prefix = "rh")
```

**Getting information**

Use the `get_joints_effort` method to get a dictionary with efforts of the group joints.

```python
hand_joints_effort = hand_commander.get_joints_effort()
print("Hand joints effort \n " + str(hand_joints_effort) + "\n")
```

Use the `get_tactile_type` to get a string indicating the type of tactile sensors present (e.g. PST, biotac, UBI0) or `get_tactile_state` to get an object containing tactile data. The structure of the data is different for every `tactile_type`.

```python
tactile_type = hand_commander.get_tactile_type()
tactile_state = hand_commander.get_tactile_state()
print("Hand tactile type\n" + tactile_type + "\n")
print("Hand tactile state\n" + str(tactile_state) + "\n")
```

**Set the maximum force**

Use the method `set_max_force` to set the maximum force for a hand joint.

Parameters:
- `joint_name` name of the joint.
- `value` maximum force value

**Example**

```python
## The limits in the current implementation of the firmware are from 200 to 1000 (measured in custom units)
hand_commander.set_max_force("rh_FFJ3", 600)
```

### 1.2.7 Saving States

To save a state you must first be connected to the warehouse. After launching the hand, click the green Connect button in the ‘Context’ tab of rviz.
If you have connected successfully you should see two new buttons, **Reset database** and **Disconnect**, as can be seen in the following picture:
Next, go to the ‘Stored States’ tab in ‘Motion Planning’. Here you have full control over the saved states in the warehouse. You can then follow these steps:

- move the hand to the grasp position
- Go the ‘Planning’ tab and in the ‘Select Goal State’ select ‘current’ and click **update**.
- Finally, go to the ‘Stored States’ tab and click the button **Save Goal** under the ‘Current State’ group. A prompt
will appear to ask you to name the state. Once named, you can plan to and from this state.

1.3 Using Peripherals

1.3.1 Cyberglove

Introduction

The Cyberglove is a data glove designed for use with Virtual Reality that Shadow has integrated with our Dexterous Hand to provide the most intuitive available method for control directly by a human operator. The gloves have 22 sensors which correspond approximately to the degrees of freedom of the robot. With practice, good control of the Shadow Hand is possible. The following is a guide for getting up and running with the glove.

Getting Started

Connecting

There are two variants of the cyberglove. Both use an RS-232 serial interface with a D-sub type connector and connect to a PC via a USB Serial adaptor. To start, connect the serial adaptor to the PC and the control box to the adaptor.

The two gloves are powered in slightly different ways. The glove with the large black control box has a large DC power supply which plugs into the control box. The power supply takes mains power via a kettle lead.
The other glove has a small PSU that plugs directly into the mains and connects to the small injector in the middle of the serial lead.
The smaller control box has a button disguised as the power LED. Push and hold it for a second or two. It will glow green once it’s powered on.

### Running in Docker

To run the glove from a docker container, you must connect and power up the glove before starting the container. This ensures that the USB serial device is available from inside the container.

The usual docker image to use is shadowrobot/dexterous-hand:kinetic. Please see here for instructions on launching a docker image.

### Launching

The usual method to launch the glove is with the following command:

```bash
roslaunch sr_cyberglove_config cyberglove.launch
```

This will only be present on systems which have been delivered with a glove. This in turn simply calls `cyberglove_trajectory/launch/cyberglove.launch` with setup specific options. The latter launch file can be used directly if desired.

### Launch Options

The launch file in `cyberglove_trajectory` has the following arguments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serial_port</td>
<td>/dev/ttyUSB0</td>
<td>The device name of the USB serial adaptor on the host PC</td>
</tr>
<tr>
<td>joint_prefix</td>
<td>rh</td>
<td>Prepended to the glove namespace, (e.g. rh_cyberglove). Necessary for bimanual systems</td>
</tr>
<tr>
<td>calibration</td>
<td>sr_cyberglove_config/calibrations/right_cyberglove.yaml</td>
<td>Path to yaml file containing per user glove calibration.</td>
</tr>
<tr>
<td>mapping</td>
<td>sr_cyberglove_config/mappings/GloveToHandMappings_generic</td>
<td>Path to glove → robot joint mapping matrix. Usually no need to modify this file.</td>
</tr>
<tr>
<td>version</td>
<td>2</td>
<td>Glove protocol version. 2 for the older glove at Shadow, 3 for the newer one (See below)</td>
</tr>
<tr>
<td>protocol</td>
<td>8bit</td>
<td>Sets 8 or 16 bit mode for protocol version 3. 8 bit is correct for both of Shadow’s gloves.</td>
</tr>
<tr>
<td>filter</td>
<td>true</td>
<td>Filter data internally before publishing.</td>
</tr>
<tr>
<td>trajectory_tx_delay</td>
<td>0.1</td>
<td>Offset in second to set the trajectory time stamp. It must be grater than the time it takes for the trajectory goal msg to reach the trajectory controller.</td>
</tr>
<tr>
<td>trajectory_delay</td>
<td>0.02</td>
<td>Delay from the beginning of the trajectory. I.e. the time_from_start of the single trajectory point.</td>
</tr>
</tbody>
</table>

Normally the default options are fine for either of Shadow’s gloves. The main exception is the glove version, as the two gloves require slightly different serial protocols. See below for further explanation.
Protocol choice for cybergloves

Shadow has two gloves of slightly different versions:

The older glove with the big black control box on the right is a Cyberglove 2. The newer one with the neater control box on the left is (probably) an early Cyberglove 3, although there is still some controversy over this fact. In any case, the old glove works with \texttt{version:=2} and the newer one works with \texttt{version:=3}.

Calibrating

To modify the glove calibration, there are two RQT plugins:

1. Glove Calibrator: User executes a sequence of gestures which are used to generate a new calibration file.
2. Glove Calibration Tweaker: Individual calibration points can be modified manually to adjust/improve an existing calibration.
• With a glove connected and started, run the calibration GUI.

• Using the pictures on the right as a guide, execute the sequence of hand positions.
  – Place your hand in the position shown in the picture.
  – Press ‘Calibrate’
  – Repeat for all positions.

• Save the new calibration. (N.B. The calibration will not be loaded until it is saved)
Tweaking GUI

- The sensors of the glove are enumerated. Each sensor has a picture to its right to show its location on the glove.
- The raw sensor value and its calibrated output in degrees are displayed on the left of each sensor’s display.
- Each sensor’s display is divided into its calibration points.
- The calibrated value and its position with respect to the calibration points is visualised for each sensor via the blue bar at the top of each sensor’s display.
- Each calibration point can be manually adjusted using the 6 buttons, with the buttons having the following effects:
Dexterous Hand Documentation

<table>
<thead>
<tr>
<th>Button</th>
<th>+++</th>
<th>++</th>
<th>+</th>
<th>-</th>
<th>-</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adjustment To Value</td>
<td>+0.1</td>
<td>+0.01</td>
<td>+0.001</td>
<td>-0.1</td>
<td>-0.01</td>
<td>-0.001</td>
</tr>
</tbody>
</table>

- Using a virtual hand for reference, the user should adjust the calibrations, one sensor at a time, to improve correlation between user and robot hands.
- Once satisfied with changes, the calibration can be saved using the save button.
- An existing calibration can be loaded using the load button.
- The most recently loaded calibration (or the one present when the GUI was started) can be reloaded using the reload button.

**Topics/Service**

**Topics**

- `/rh_cyberglove/raw/joint_state`
  Contains raw values, in raw ADC values, scaled 0.0->1.0
- `/rh_cyberglove/calibrated/joint_state`
  Contains sensor values, calibrated in radians.
- `/rh_trajectory_controller/follow_joint_trajectory/goal`
  Goal trajectory, published directly to trajectory controller.

**Service**

- `/rh_cyberglove/reload_calibration`
  Empty service called to instruct driver to reload glove calibration from parameter server (N.B Doesn’t reload calibration from disk)

**Synchronising Between Multiple Machines**

If the glove node runs on a different machine from the trajectory controller, both machines will need to be synchronised. chrony (sudo apt-get install chrony) has been used successfully to achieve that.

The argument trajectory_tx_delay should be increased slightly to account for the extra transmission time from the glove driver to the trajectory controller.

**1.3.2 Optoforce**

If the hand has optoforce sensors installed, it is recommended to use the one liner to install the docker container using the “-o true” option. Doing this, everything is going to be set up automatically. Example of the oneliner is illustrated below:

```bash
--release -n dexterous-hand -sn Hand_Launcher -e [EtherCAT interface ID] -b [sr-_n
--config_branch] -o true
```

(continues on next page)
Topics

Optoforce sensor data will be published on the following topics:

```
/rh/optoforce_**
```

### 1.3.3 BioTac

These topics are read-only and update at 100 Hz with data from the biotac sensors, which comprises their pressure, temperature and electrode resistance. For further information about the biotacs, refer to their documentation.

Topics

```
/rh/tactile
```

This topic is published by the driver at 100 Hz with data from tactile sensors.

Example topic message when using BioTac fingertip sensors:

```
tactiles:
- pac0: 2048
  pac1: 2054
  pdc: 2533
  tac: 2029
  tdc: 2556
electrodes: [2622, 3155, 2525, 3062, 2992, 2511, 3083, 137, 2623, 2552, 2928, 3249, 2705, 3037, 3020, 2405, 3049, 948, 2458, 2592, 2376, 237, 3244, 3119]
- pac0: 0
  pac1: 0
  pdc: -9784
  tac: 32518
  tdc: 0
electrodes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
- pac0: 0
  pac1: 0
  pdc: -9784
  tac: 32518
  tdc: 0
electrodes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
- pac0: 0
  pac1: 0
  pdc: -9784
  tac: 32518
  tdc: 0
electrodes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

(continues on next page)
- pac0: 0
- pac1: 0
- pdc: -9784
- tac: 32518
- tdc: 0
- electrodes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

• BioTac

These topics are read-only and update at 100 Hz with data from the biotac sensors, which comprises their pressure, temperature and electrode resistance. This topic is published from the /biotac_republisher node which receives this data from the driver via the /rh/tactile topic. For further information about the biotacts, refer to their documentation: https://www.syntouchinc.com/wp-content/uploads/2016/12/BioTac_SP_Product_Manual.pdf

Example /rh/biotac_** topic message:

- pac0: 2056
- pac1: 2043
- pdc: 2543
- tac: 2020
- tdc: 2454
- electrodes: [2512, 3062, 2404, 2960, 2902, 2382, 2984, 138, 2532, 2422, 2809, 3167, 2579, 2950, 2928, 2269, 2966, 981, 2374, 2532, 3199, 3152, 3155, 3033]

1.4 Frequently Asked Questions

A list of common issues and how to resolve them.

1.4.1 Hardware Issues

How do I know when the Hand is powered on?

• There are lights ...
  • The fans should be audible

The Hand is not powered.

• Check all connections from the power supply to the Hand

All of the connections are OK. The Hand still won't power on.

• Check that the plug socket is turned on and the mains lead is plugged in

The Hand STILL won’t power on.

• Get in touch with someone from Shadow: support@shadowrobot.com

The Hand is powered but I cannot connect to my PC.

• Check the ethernet cable connections
  • Check the link light on your computer's ethernet port (fixed state, not flashing)
  • Check you're using the right interface
1.4.2 Software Issues

Docker

The Hand doesn’t react after startup of the container

- Make sure that interface ID is set correctly
- Make sure you have set the correct branch for sr_config
- Power cycle the Hand and try again

1.5 Changelog

1.5.1 ROS Kinetic

Version 1.0.31

Features:
- Docker image now built in AWS

Version 1.0.26

Features:
- Added a feature that Docker Image release process checks for pre-existing Docker tags in Dockerhub

Version 1.0.25 (current kinetic-release)

Features:
- Updated launch files
- Added bimanual control
- General bugfixes

Version 1.0.24

Features:
- Fixing a few bugs with the Data Visualizer
- Hand E Data Visualizer GUI

Version 1.0.21

Features:
- System logging was added
Version 1.0.15

Features:

- Moveit warehouse branch was changed to our fork to work well. Official moveit warehouse was crashing

Version 1.0.12

Features:

- Moved CyberGlove configuration to its own repository. Using the CyberGlove requires the -cg Docker One-liner flag and correct CyberGlove branch to be specified
- If the hand is launched under simulation, use_sim_time is automatically set to true
- Added script to test real-time performance (control loop overruns and signal drops) of the computer running the hand and to specify how many seconds to run for
- Improved ROS save logs functionality by including debug symbols
- Improved ROS save logs functionality by deleting logs over 1 GB (to avoid the computer from filling up)
- Improved ROS save logs functionality (and the upload to AWS) to giving the user the option to decline uploading anything to AWS
- Added CyberGlobe calibration and tweaking plugins to rqt

Version 1.0.9

Features:

- The Docker container launches in a few seconds

Version 1.0.7

Features:

- Ability to easily upload ROS Logs to Amazon Web Services (AWS) and email them to Shadow Robot Company automatically
- PyQtGraph used for plotting back-end in rqt

Version 1.0.5

Features:

- Release of hand E software (kinetic-v1.0.5) and firmware (firmware release 3), using the new firmware release mechanism (GitHub)
- Ability to save ROS logs by clicking on an icon on the desktop

Version 1.0.2

- Initial version
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC</td>
<td>Analogue to Digital Converter. A chip which reads an analogue voltage signal.</td>
</tr>
<tr>
<td>ASIC</td>
<td>Application Specific Integrated Circuit.</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit. A computer processor which runs software.</td>
</tr>
<tr>
<td>CRC</td>
<td>Cyclic Redundancy Check. An algorithm (or output of) for checking the correctness of incoming data.</td>
</tr>
<tr>
<td>DAC</td>
<td>Digital to Analogue Converter. A chip which produces an analogue voltage signal.</td>
</tr>
<tr>
<td>EC</td>
<td>EtherCAT</td>
</tr>
<tr>
<td>EEPROM</td>
<td>Electrically Erasable Programmable Read Only Memory.</td>
</tr>
<tr>
<td>FPID</td>
<td>Feed-forward Proportional Integral Derivative controller. An algorithm used to control something on a robot.</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface.</td>
</tr>
<tr>
<td>I/O</td>
<td>Input / Output</td>
</tr>
<tr>
<td>ICD3</td>
<td>In Circuit Debugger 3. A tool used to change the firmware on a PIC microcontroller.</td>
</tr>
<tr>
<td>IMU</td>
<td>Inertial Measurement Unit. Sensors used to measure acceleration and rotation.</td>
</tr>
<tr>
<td>LED</td>
<td>Light Emitting Diode. A small coloured light.</td>
</tr>
<tr>
<td>LVDS</td>
<td>Low Voltage Differential Signal. One of two possible electrical physical layers supported by EtherCAT.</td>
</tr>
<tr>
<td>mA</td>
<td>milliamps. A unit of electrical current.</td>
</tr>
<tr>
<td>MCU</td>
<td>Micro Controller Unit. A small, usually embedded, CPU.</td>
</tr>
<tr>
<td>MIPS</td>
<td>No longer an abbreviation; a word in its own right. A brand of CPU / MCU.</td>
</tr>
<tr>
<td>Nm</td>
<td>Newton Meters. A unit of torque.</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer.</td>
</tr>
<tr>
<td>PCB</td>
<td>Printed Circuit Board.</td>
</tr>
<tr>
<td>PID</td>
<td>Proportional Integral Derivative controller. An algorithm used to control something on a robot.</td>
</tr>
<tr>
<td>PST</td>
<td>Pressure Sensor Tactile. A simple tactile sensor offered as standard.</td>
</tr>
<tr>
<td>PWM</td>
<td>Pulse Width Modulation. Digital method used to emulate an analog signal.</td>
</tr>
<tr>
<td>ROS</td>
<td>Robot Operating System, created by Willow Garage.</td>
</tr>
<tr>
<td>SPI</td>
<td>Serial Peripheral Interface, allowing an MCU to communicate with an ADC.</td>
</tr>
</tbody>
</table>