
Dexterous Hand Documentation

Shadow Robot Company

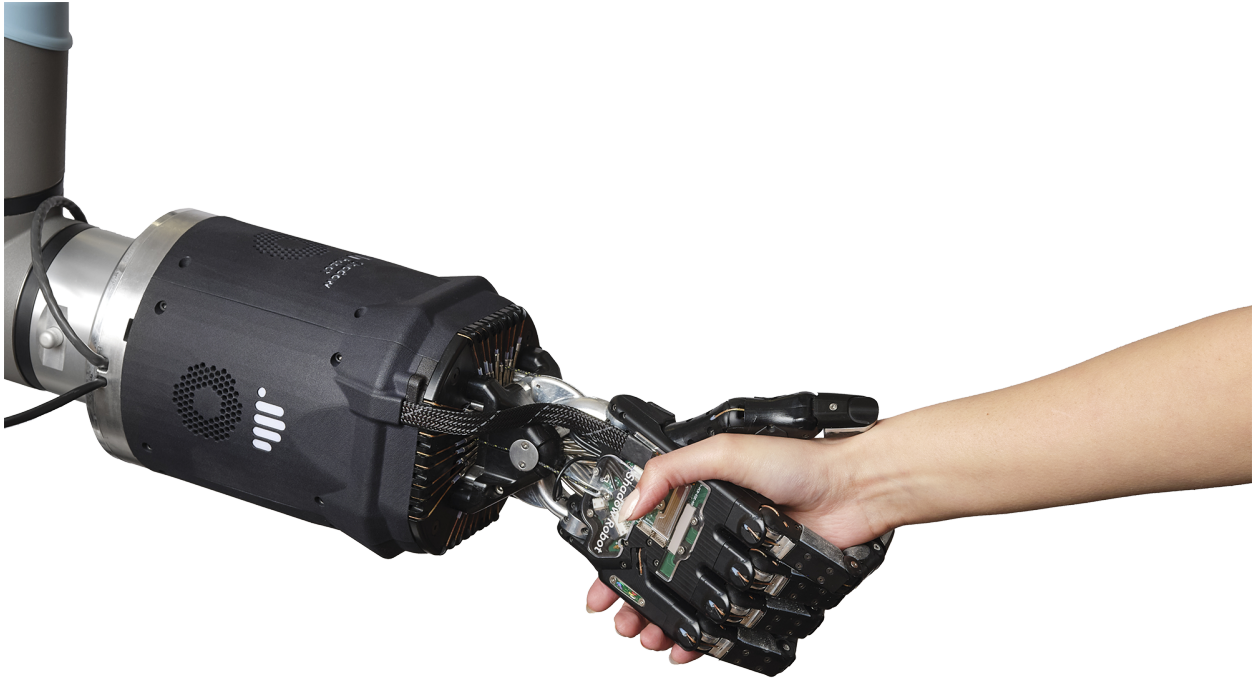
Jul 26, 2022

OVERVIEW

1	First time users	3
2	Hand peli case contents	5
3	Laptop box contents	7
4	Connecting Cables	9
4.1	Connection procedure	10
5	Mounting the hand to an arm	17
6	Launching the hands	19
6.1	Jiggling	19
6.2	Power off	19
7	Lights on the hand	21
8	Uploading log files to our server	23
9	Gazebo	25
9.1	Installing the software (sim)	25
9.2	Starting a robot simulation	26
10	Mujoco	31
10.1	Obtaining the Mujoco simulation	31
10.2	Running the Mujoco simulation	31
10.3	Re-Using your Mujoco container	32
11	Installing the software on a new PC	33
11.1	Hardware specifications	33
11.2	Get ROS Upload login credentials	33
11.3	Run the one-liner	34
12	Why do we use a NUC?	35
12.1	Testing the overruns	35
12.2	What will happen if a NUC is not used?	36
12.3	Running the hand without the NUC	36
13	ROS MASTER and connecting additional computers	37
14	rqt	45

15 Robot descriptions (URDF)	63
15.1 Shadow Hands	63
15.2 Shadow Hands mounted on UR arms	64
15.3 Usage	65
15.4 Autodetection parameters	65
16 Support and Teamviewer	85
16.1 Support Contact	85
16.2 Teamviewer	85
17 Abbreviations	89

This is the starting point for the Shadow Dexterous Hand Documentation



FIRST TIME USERS

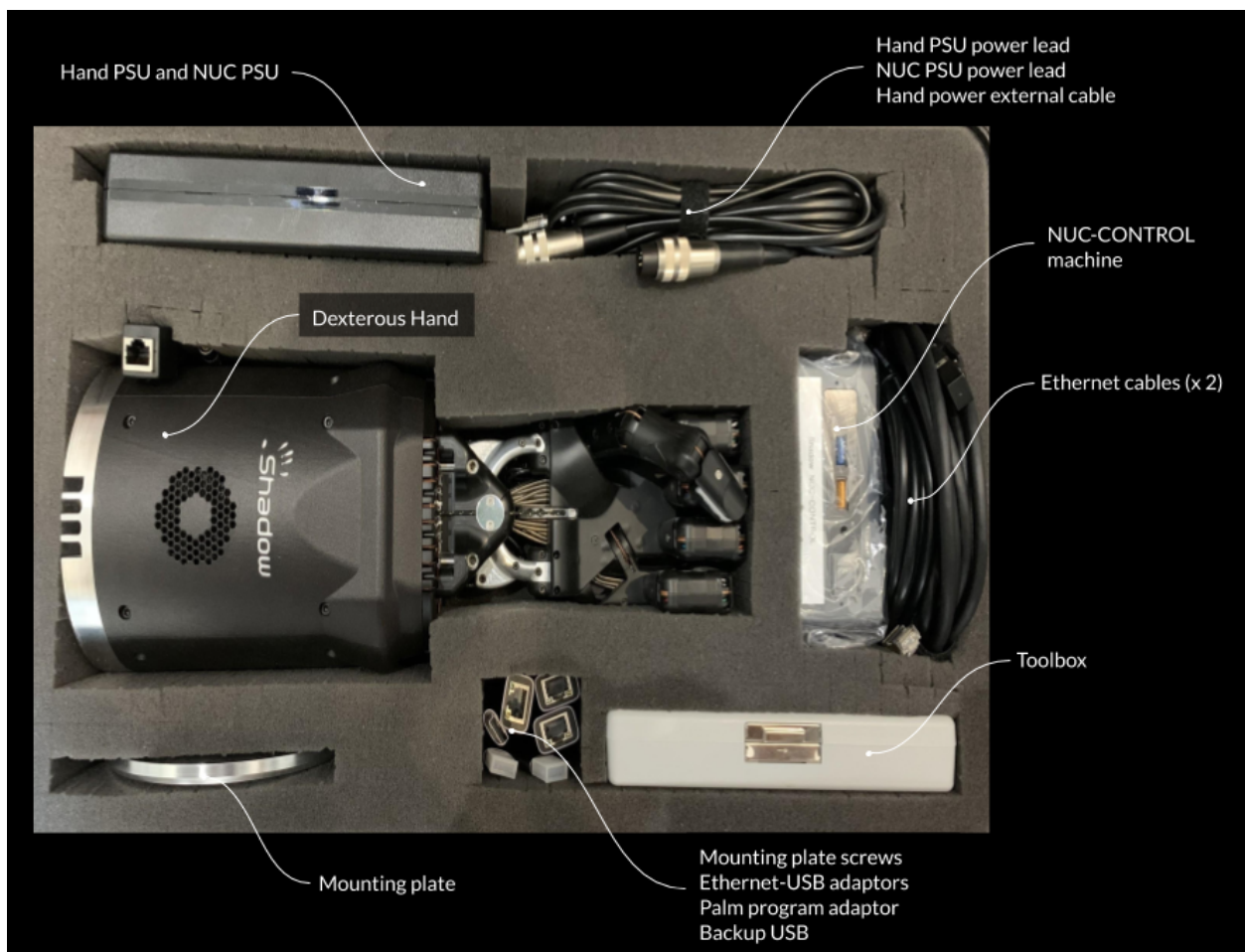
If you are unfamiliar with ROS and intend to use the ROS API, it is highly recommended that you read the [ROS Tutorials](#).

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.

HAND PELI CASE CONTENTS

When you receive your Dexterous Hand, this is what you will find in the peli case:



Equipment	Quantity
Dexterous Hand	1
Hand Power External Cable	1
Hand Power Supply	1
Hand Power Supply lead according to destination country	1
Mounting plate (to connect the hand to UR arm)	1
Mounting Plate screws for mounting plate	4
Mounting Plate screws for the hand	8
5m Ethernet cable	1
“NUC-CONTROL” USB-ethernet adapter	1 (only 1 per bimanual system)
“SERVER” USB-ethernet adapter	1 (only 1 per bimanual system)
“RIGHT/LEFT HAND” Labelled USB-ethernet adapter	1
NUC labelled SHADOW NUC-CONTROL for running hand’s driver and the arm	1 (only 1 per bimanual system)
NUC PSU	1 (only 1 per bimanual system)
NUC PSU power lead according to destination country	1 (only 1 per bimanual system)
Toolbox (contains hex drivers to robot maintenance)	1
Cut allen key (inside the Toolbox)	1
Allen key (inside the Toolbox)	1
Spooltool (inside the Toolbox)	1
Luggage locks for peli case	2
“Shadow Backup” USB w. Clonezilla images of NUC and server	1 (only 1 per bimanual system)
Hand Delivery Instructions	1 (only 1 per bimanual system)

LAPTOP BOX CONTENTS

This is an extra box, additional to the hand(s) peli case(s). There will be 1 per bimanual system.

Equipment	Quantity
Laptop-Server	1
Laptop PSU	1
5m Ethernet cable	1
Mouse	1
Power adapter for destination country	1

CONNECTING CABLES

- Connect the hands to the NUC-CONTROL. It is very important that the exact USB->Ethernet adapters are used.
 - The right hand should be connected to a USB->Ethernet adapter labelled: **HAND RIGHT**, which should be connected to one of the USB ports of the NUC-CONTROL (it does not matter which one).
 - The left hand should be connected to a USB->Ethernet adapter labelled: **HAND LEFT**, which should be connected to one of the USB ports of the NUC-CONTROL (it does not matter which one).
- Connect one USB->Ethernet adapter labelled **NUC-CONTROL** to another USB port on the NUC and the other USB->Ethernet labelled **SERVER** to any of the ports in your **SERVER Laptop** (provided by Shadow or a custom one).
- Connect the two adaptors together with an Ethernet cable.

You have been supplied with medium length Ethernet leads, but if you require a different length, you can simply use a standard commercial Ethernet Cat 5 (or better) cable, available from most computer parts suppliers. If you require internet connection in the laptop, connect an Ethernet cable providing external internet connection to the back of the laptop, to an Ethernet port labelled INTERNET.

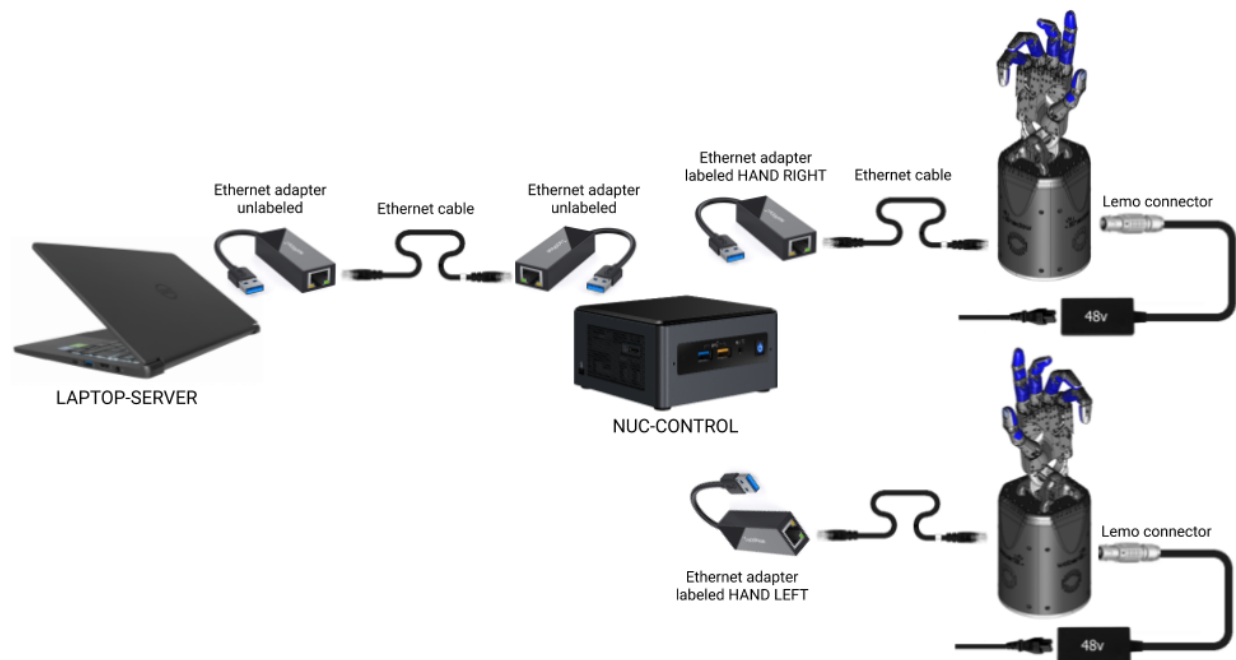


Fig. 1: Connections diagram

- Finally, connect the external power supply to the hands using the metal Lemo connector, making sure to line up the red dots. When power is applied to the hand, the fans will be heard immediately. If you require a longer or shorter cable, please contact us at support@shadowrobot.com.

4.1 Connection procedure

1. Connect the Ethernet between the NUC and the laptop using the instructions above
2. Power on the laptop
3. Connect an Ethernet cable providing external internet connection to the back of the laptop
4. Power on the NUC
5. Make sure the laptop has only 1 USB-Ethernet adapter connected to it.
6. In case of using another laptop than one provided, please follow the instructions below to install the software.
7. Power on the hand(s)
8. Connect the right hand to the USB-Ethernet adapter labelled “HAND RIGHT” which should be plugged in to the NUC, as explained above
9. Connect the left hand to the USB-Ethernet adapter labelled “HAND LEFT” which should be plugged in to the NUC, as explained above

Desktop icons for the hand

The following icons will be available on the server laptop desktop for launching and controlling the right, left and bimanual hands. Please note that in Ubuntu 20.04 these icons will be located inside a folder called ‘Shadow Launcher’ on the desktop. You might need to right Click with your mouse and select ‘Allow Launching’.

Main desktop icons

```
eval rst .. table:
```

```
:class: tight-table
```

Icon picture	Icon explanation	Icon text
<code>.. image:: ../img/log-icon.png</code>	Saves ROS logs from server and NUC, uploads them	Shadow ROS Logs Saver and Uploader
<code>:width: 100</code>	to Shadow servers and emails them to Shadow	
<code>.. image:: ../img/hand-e.png</code>	Launches the right hand (container, ROS core, NUC hardware control loop, server GUI)	Launch Shadow Right Hand
<code>:width: 100</code>		
<code>.. image:: ../img/hand-e-left.png</code>	Launches the left hand (container, ROS core, NUC hardware control loop, server GUI)	Launch Shadow Left Hand
<code>:width: 100</code>		

(continues on next page)

(continued from previous page)

↪	NUC hardware control loop, server GUI)		
+	-----+	-----+	
↪	.. image:: ../img/hand-e-bimanual.png	Launch Shadow Bimanual Hands	↪
↪		Launches the both hands (container, ROS core,	
↪	:width: 100		↪
↪		NUC hardware control loop, server GUI)	
+	-----+	-----+	
↪	.. image:: ../img/ROS_logo.png	Shadow NUC RQT	↪
↪		Once the hand has been launched, this icon	
↪	:width: 100		↪
↪		starts ROS RQT inside the NUC's docker container	
+	-----+	-----+	
↪	.. image:: ../img/documentation_icon.png	Dexterous Hand Documentation	↪
↪		Opens online documentation if internet connected	
↪	:width: 100		↪
↪		or offline documentation if no internet	
+	-----+	-----+	
↪	.. image:: ../img/close_icon.png	Shadow Close Everything	↪
↪		Cleanly stops ROS processes, closes containers,	
↪	:width: 100		↪
↪		and closes all Shadow related terminals	
+	-----+	-----+	
↪			

```
''' ## Shadow Demos folder
```

The following icons will be available in the Shadow Demos folder. They will only work once the hand has been launched.

```
'''eval_rst .. table:
```

:class: tight-table			
+	-----+	-----+	
↪	Icon picture	Icon text	↪
↪		Icon explanation	
+	=====+	=====+	
↪	.. image:: ../img/close-hand-icon-left.png	Close Left Hand	↪
↪		Once the hand has been launched, this icon will	
↪	:width: 100		↪
↪		close (pack) the left hand	
+	-----+	-----+	
↪	.. image:: ../img/close-hand-icon-right.png	Close Right Hand	↪
↪		Once the hand has been launched, this icon will	
↪	:width: 100		↪
↪		close (pack) the right hand	
+	-----+	-----+	

(continues on next page)

(continued from previous page)

↪	-----+-----		
	.. image:: ../img/close-hand-icon-bimanual.png	Close Bimanual Hands	↪
↪		Once bimanual hands have been launched, this icon will	
	:width: 100		↪
↪		close (pack) both hands	
↪	-----+-----		
	.. image:: ../img/demo-hand-icon-left.png	Biotac Demo Left Hand/Demo Left	↪
↪	Hand	Once the hand has been launched, this icon will	
	:width: 100		↪
↪		run various (tactile/keyboard-activated) left hand demos	
↪	-----+-----		
	.. image:: ../img/demo-hand-icon-right.png	Biotac Demo Right Hand/Demo Right	↪
↪	Hand	Once the hand has been launched, this icon will	
	:width: 100		↪
↪		run various (tactile/keyboard-activated) right hand demos	
↪	-----+-----		
	.. image:: ../img/demo-hand-icon-bimanual.png	Biotac Demo Bimanual Hands/Demo	↪
↪	Bimanual Hands	Once bimanual hands have been launched, this icon will	
	:width: 100		↪
↪		run various (tactile/keyboard-activated) bimanual hands demos	
↪	-----+-----		
	.. image:: ../img/open-hand-icon-left.png	Open Left Hand	↪
↪		Once the hand has been launched, this icon will	
	:width: 100		↪
↪		fully open the left hand	
↪	-----+-----		
	.. image:: ../img/open-hand-icon-right.png	Open Right Hand	↪
↪		Once the hand has been launched, this icon will	
	:width: 100		↪
↪		fully open the right hand	
↪	-----+-----		
	.. image:: ../img/open-hand-icon-bimanual.png	Open Bimanual Hands	↪
↪		Once bimanual hands have been launched, this icon will	
	:width: 100		↪
↪		fully open both hands	
↪	-----+-----		

...

Shadow Advanced Launchers folder

The following icons will be available in the Shadow Advanced Launchers folder.

- If an icon in Shadow Advanced Launchers starts with a number, it is meant to be run in numerical sequence after the lower-numbered icons.
- If an icon in Shadow Advanced Launchers doesn't start with a number, it can be run independently

The Launch Shadow Right/Left/Bimanual Hand(s) icon in the main desktop is equivalent to launching:

- 1 - Launch Server Container
- 2 - Launch Server ROSCORE
- 3 - Launch NUC Container and Right/Left/Bimanual Hands Hardware Control Loop
- 4 - Launch Server Right/Left/Bimanual GUI

However, with the Shadow Advanced Launcher icons, you can have more granular and customised control of launching different parts of the Shadow software.

```eval\_rst .. table:

```
:class: tight-table
```

| Icon picture                          | Icon explanation                                                                            | Icon text                                                         |
|---------------------------------------|---------------------------------------------------------------------------------------------|-------------------------------------------------------------------|
| .. image:: ../img/laptop.jpg          | Launches the server laptop's docker container.                                              | 1 - Launch Server Container                                       |
| :width: 100                           |                                                                                             |                                                                   |
| .. image:: ../img/ROS_logo.png        | Launches the ROSCORE inside the server laptop's docker container                            | 2 - Launch Server ROSCORE                                         |
| :width: 100                           |                                                                                             |                                                                   |
| .. image:: ../img/hand-e.png          | SSH'es to the NUC, starts its container, and launches the right hand realtime control loop  | 3 - Launch NUC Container and Right Hand Hardware Control Loop     |
| :width: 100                           |                                                                                             |                                                                   |
| .. image:: ../img/hand-e-left.png     | SSH'es to the NUC, starts its container, and launches the left hand realtime control loop   | 3 - Launch NUC Container and Left Hand Hardware Control Loop      |
| :width: 100                           |                                                                                             |                                                                   |
| .. image:: ../img/hand-e-bimanual.png | SSH'es to the NUC, starts its container, and launches the bimanual realtime control loop    | 3 - Launch NUC Container and Bimanual Hands Hardware Control Loop |
| :width: 100                           |                                                                                             |                                                                   |
| .. image:: ../img/hand-e.png          | Launches the right hand (connected to NUC) in zero force mode (fingers can be moved easily) | 3 - Zero Force Mode - Right Hand                                  |
| :width: 100                           |                                                                                             |                                                                   |

(continues on next page)

(continued from previous page)

|   |                                                  |                                    |   |
|---|--------------------------------------------------|------------------------------------|---|
| ↪ | .. image:: ../img/hand-e-left.png                | 3 - Zero Force Mode - Left Hand    | ↪ |
| ↪ | Launches the left hand (connected to NUC) in     |                                    | ↪ |
| ↪ | :width: 100                                      |                                    | ↪ |
| ↪ | zero force mode (fingers can be moved easily)    |                                    | ↪ |
| ↪ | .. image:: ../img/rviz.png                       | 4 - Launch Server Right Hand GUI   | ↪ |
| ↪ | Launches the GUI (Rviz) on server laptop for the |                                    | ↪ |
| ↪ | :width: 100                                      |                                    | ↪ |
| ↪ | right hand                                       |                                    | ↪ |
| ↪ | .. image:: ../img/rviz.png                       | 4 - Launch Server Left Hand GUI    | ↪ |
| ↪ | Launches the GUI (Rviz) on server laptop for the |                                    | ↪ |
| ↪ | :width: 100                                      |                                    | ↪ |
| ↪ | left hand                                        |                                    | ↪ |
| ↪ | .. image:: ../img/rviz.png                       | 4 - Launch Server Bimanuals GUI    | ↪ |
| ↪ | Launches the GUI (Rviz) on server laptop for the |                                    | ↪ |
| ↪ | :width: 100                                      |                                    | ↪ |
| ↪ | bimanual hands                                   |                                    | ↪ |
| ↪ | .. image:: ../img/hand-e.png                     | Launch Local Shadow Right Hand     | ↪ |
| ↪ | Launches the right hand (connected to server     |                                    | ↪ |
| ↪ | :width: 100                                      |                                    | ↪ |
| ↪ | laptop) using the same USB-ethernet adapter      |                                    | ↪ |
| ↪ | .. image:: ../img/hand-e-left.png                | Launch Local Shadow Left Hand      | ↪ |
| ↪ | Launches the left hand (connected to server      |                                    | ↪ |
| ↪ | :width: 100                                      |                                    | ↪ |
| ↪ | laptop) using the same USB-ethernet adapter      |                                    | ↪ |
| ↪ | .. image:: ../img/hand-e-bimanual.png            | Launch Local Shadow Bimanual Hands | ↪ |
| ↪ | Launches bimanual hands (connected to server     |                                    | ↪ |
| ↪ | :width: 100                                      |                                    | ↪ |
| ↪ | laptop) using the same USB-ethernet adapters     |                                    | ↪ |
| ↪ | .. image:: ../img/nuc.png                        | Launch NUC Container               | ↪ |
| ↪ | SSH'es to the NUC, starts NUC's container and    |                                    | ↪ |
| ↪ | :width: 100                                      |                                    | ↪ |
| ↪ | starts a terminal session inside it              |                                    | ↪ |
| ↪ | .. image:: ../img/hand-e.png                     | Local Zero Force Mode - Right Hand | ↪ |
| ↪ | Launches the right hand (connected to server) in |                                    | ↪ |

(continues on next page)

(continued from previous page)

|   |                                   |                                                 |                                   |   |
|---|-----------------------------------|-------------------------------------------------|-----------------------------------|---|
|   | :width: 100                       |                                                 |                                   |   |
| ↩ |                                   | zero force mode (fingers can be moved easily)   |                                   | ↪ |
| + | -----                             | +                                               | -----                             | + |
| ↩ | +                                 | +                                               | +                                 | + |
|   | .. image:: ../img/hand-e-left.png |                                                 | Local Zero Force Mode - Left Hand | ↪ |
| ↩ |                                   | Launches the left hand (connected to server) in |                                   |   |
|   | :width: 100                       |                                                 |                                   | ↪ |
| ↩ |                                   | zero force mode (fingers can be moved easily)   |                                   |   |
| + | -----                             | +                                               | -----                             | + |
| ↩ | +                                 | +                                               | +                                 | + |

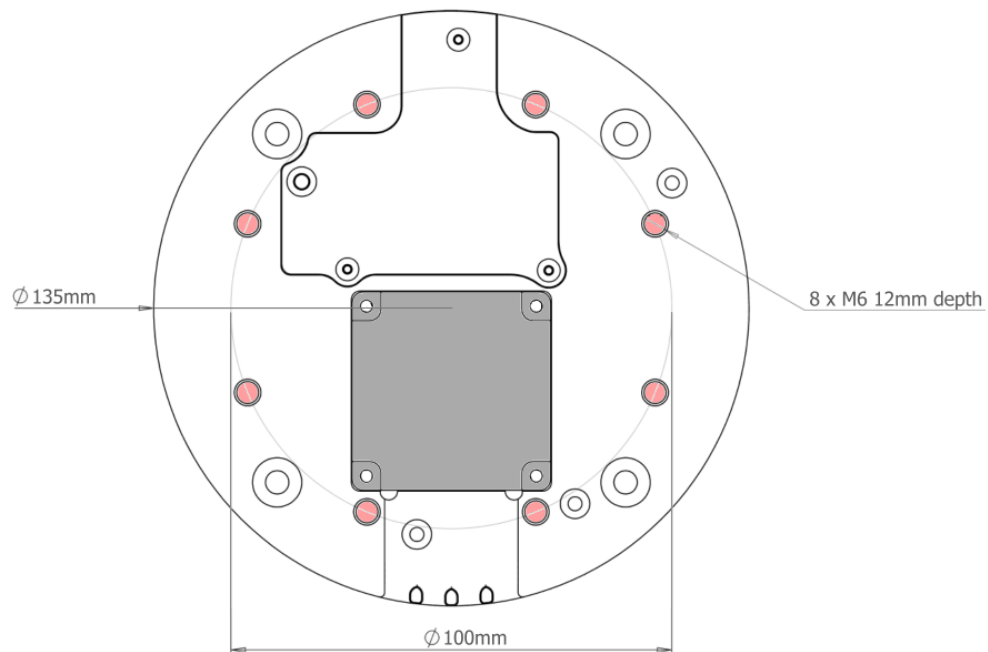
...





## MOUNTING THE HAND TO AN ARM

Shadow Robot supplies an elbow adaptor plate to adapt the Hand to most robot arms. 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



To mount the hand properly to an UR arm so that it is aligned with our xacros, you need to rotate it as shown in the picture below:

The hand's palm points in the direction of the TCP point of the arm.

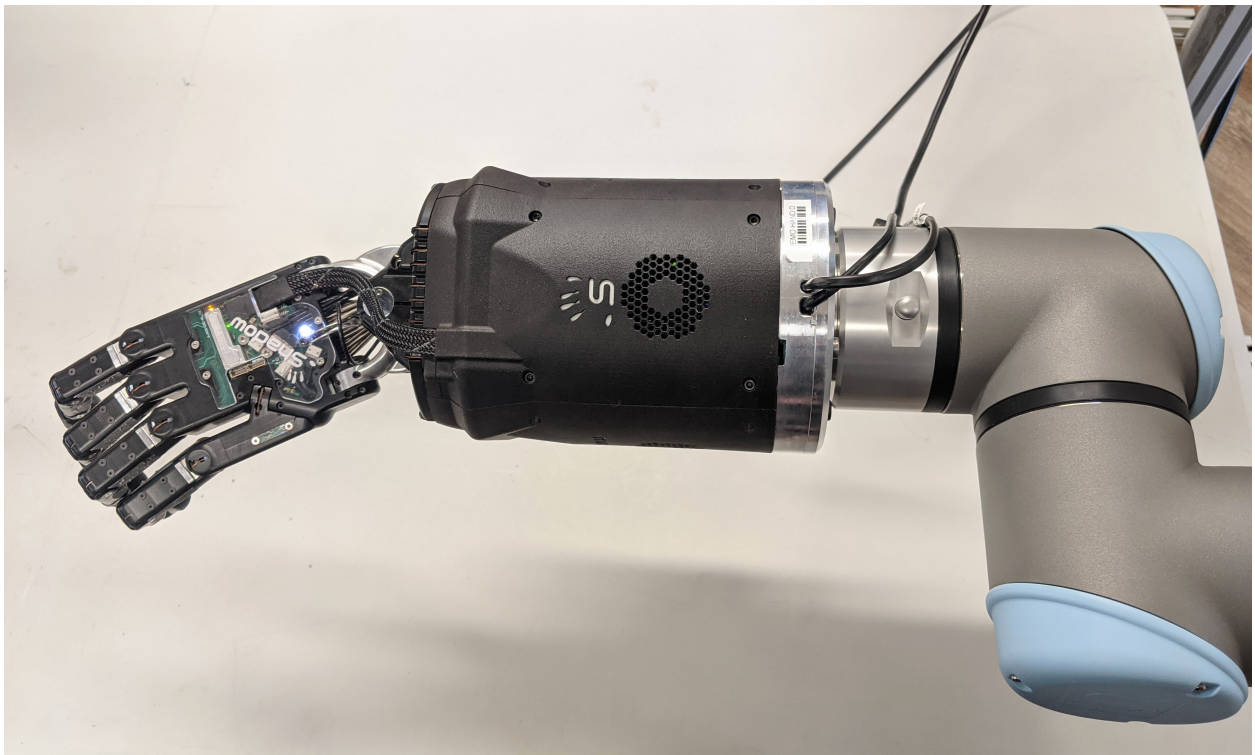


Fig. 1: Correct way to align the hand to the UR arms

## LAUNCHING THE HANDS

Depending on what you want to launch: click on Launch Shadow Right Hand or Launch Shadow Left Hand or Launch Shadow Bimanual Hands. The hand(s) should vibrate and Rviz opens.

**Warning:** If you want to launch the hand on the SERVER laptop without using the NUC-CONTROL (not recommended), plug in the hand ethernet adapter to the laptop and use the Shadow Advanced Launchers folder icon - Launch Local Shadow Right Hand, Launch Local Shadow Left Hand or Launch Local Shadow Bimanual Hands.

You can use the icons in “Shadow Demos” folder to close and open the hand(s) and run the standard demo(s), as well as save and upload ROS logs.

### 6.1 Jiggling

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.

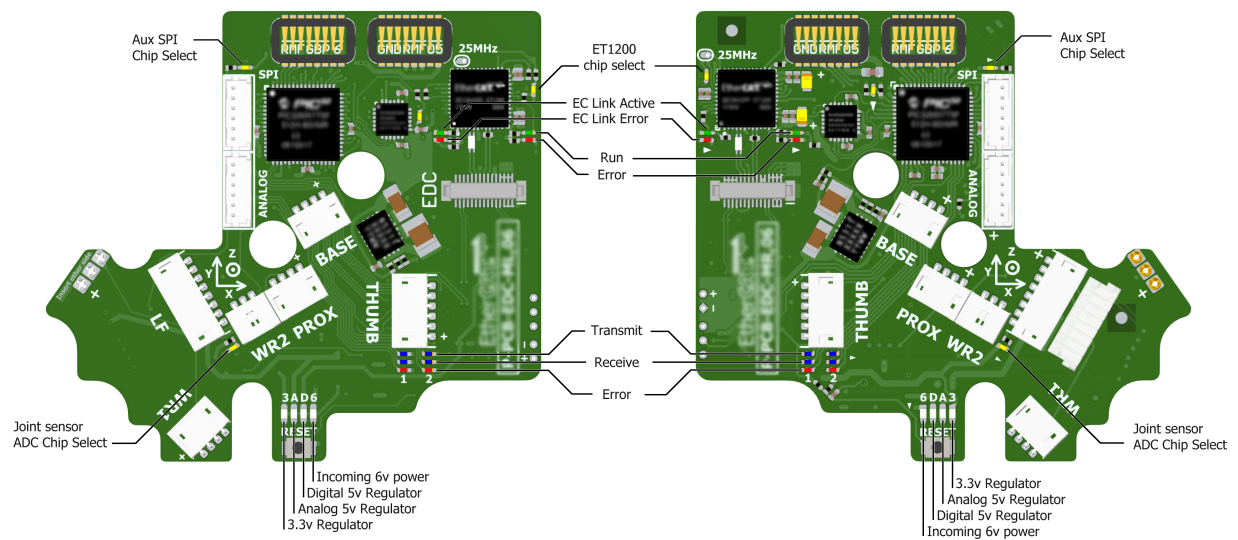
### 6.2 Power off

Power off the hand by disconnecting the power supply. When you want to shut down the NUC, press and hold the power button of the NUC for at least 3 seconds and then let go. Power off the Server with the power off button available in Ubuntu.



## LIGHTS ON THE HAND

Here is an annotation of the back of the hand's lights:



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

| Item                       | Color  | Activity         | Meaning                         |
|----------------------------|--------|------------------|---------------------------------|
| 6V Power (Power LED)       | White  | On               | Power good                      |
| A 5V regulator (Power LED) | White  | On               | Power good                      |
| D 5V regulator (Power LED) | White  | On               | Power good                      |
| 3.3V Regulator (Power LED) | White  | On               | Power good                      |
| EC Link Active             | Green  | On               | EtherCAT link established       |
| EC Link Error              | Red    | Off              | No EtherCAT link error          |
| Run                        | Green  | Off              | Hand is in Init state           |
| Error (Application Layer)  | Red    | On (during boot) | Verifying ET1200 EEPROM         |
| Error (Application Layer)  | Red    | Then off         | No EtherCAT packet error        |
| ET1200 chip select         | Yellow | On               | PIC32 communicating with ET1200 |

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

When the ROS driver is running you should see the following lights on the Palm:

| Light                        | Colour | Activity       | Meaning                                    |
|------------------------------|--------|----------------|--------------------------------------------|
| Run                          | Green  | On             | Hand is in Operational state               |
| Transmit (CAN1/2)            | Blue   | V.fast flicker | Demand values are being sent to the motors |
| Receive (CAN1/2)             | Blue   | V.fast flicker | Motors are sending sensor data             |
| Joint sensor ADC chip select | Yellow | On             | Sensors being sampled                      |

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

| Light                        | Colour | Activity | Meaning                            |
|------------------------------|--------|----------|------------------------------------|
| Run                          | Green  | Blinking | Hand is in Pre-Operational state   |
| Transmit (CAN1/2)            | Blue   | Off      | No messages transmitted on CAN 1/2 |
| Receive (CAN1/2)             | Blue   | Off      | No messages received on CAN 1/2    |
| Joint sensor ADC chip select | Yellow | Off      | Sensors not being sampled          |

## **UPLOADING LOG FILES TO OUR SERVER**

You will find a desktop icon named *Shadow ROS Logs Saver and Uploader* that is used to retrieve and copy all the available logs files from the active containers locally to 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, if you press yes it will close all active containers. After pressing "yes", you will have to enter a description of the logging event and will start copying 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.





## GAZEBO

[Gazebo](#) is our default simulator. Follow the instructions on the next section to install and run a simulation of our robot hands using Gazebo.

### 9.1 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 Noetic (Recommended):

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

- ROS Melodic:

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

- ROS Kinetic:

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

- ROS Indigo:

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

You can also add `reinstall=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 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=true` to use nvidia-docker.

More params and their explanation can be found [here](#).

## 9.2 Starting a robot simulation

First you need to start the hand container by either double clicking the icon 1 - Launch Server Container in the “Shadow Advanced Launchers” folder or running the following command:

```
$ docker start dexterous_hand_real_hw
```

### 9.2.1 Shadow Dexterous hands

- The hand will start automatically if you have run the one-liner with the argument `launch_hand=true`. To start it manually, simply run the following command in the container:

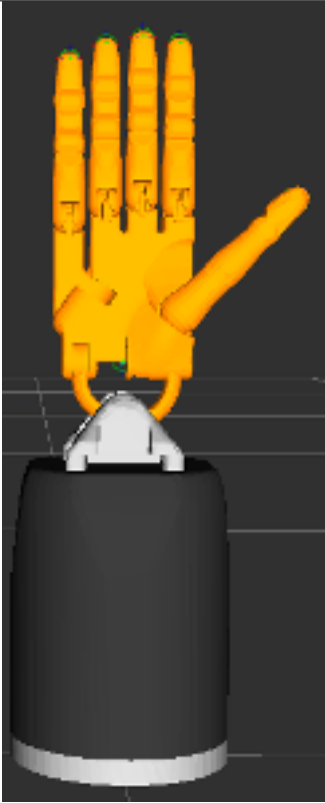


```
$ roslaunch sr_robot_launch srhand.launch
```

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:

|                                                                                     | Hand        | hand_type<br>parameter | Pa-<br>rameter | Left Hand Param-<br>eter |
|-------------------------------------------------------------------------------------|-------------|------------------------|----------------|--------------------------|
|   | Hand E      | hand_e                 |                | hand_id:=lh              |
|  | Hand E Lite | hand_lite              |                | Right hand only          |
|  |             |                        |                |                          |

To start the simulation, you can run:

```
$ roslaunch sr_robot_launch srhand.launch hand_type=hand_e
```

The `hand_type` 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 hand_type=hand_e_plus hand_id:=lh
```

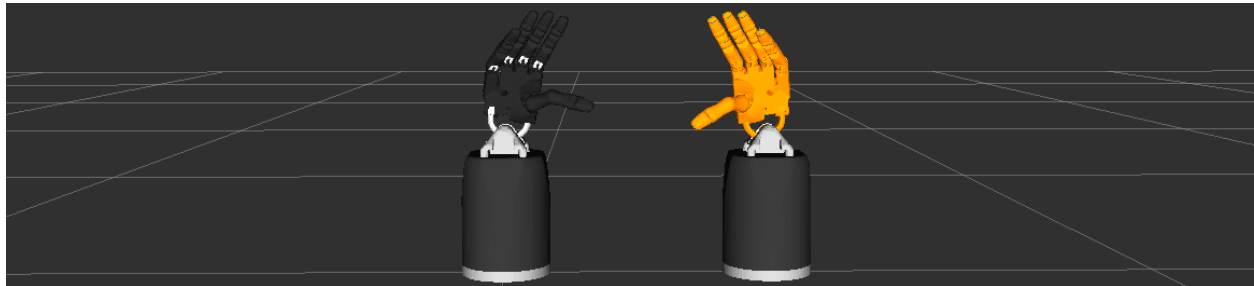
- Moveit will enable advanced behaviour (inverse kinematics, planning, collision detection, etc...), but if it is not needed, you can set `use_moveit:=false`

---

**Note:** 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 the `nvidia` 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.

---

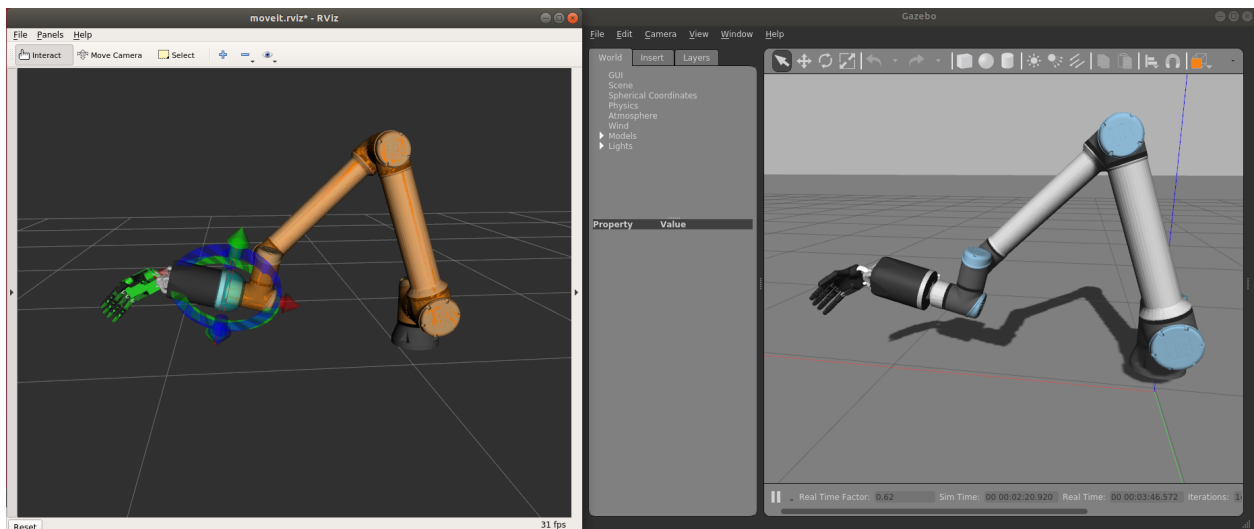
### 9.2.2 Bimanual hand system



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

```
$ roslaunch sr_robot_launch sr_bimanual.launch
```

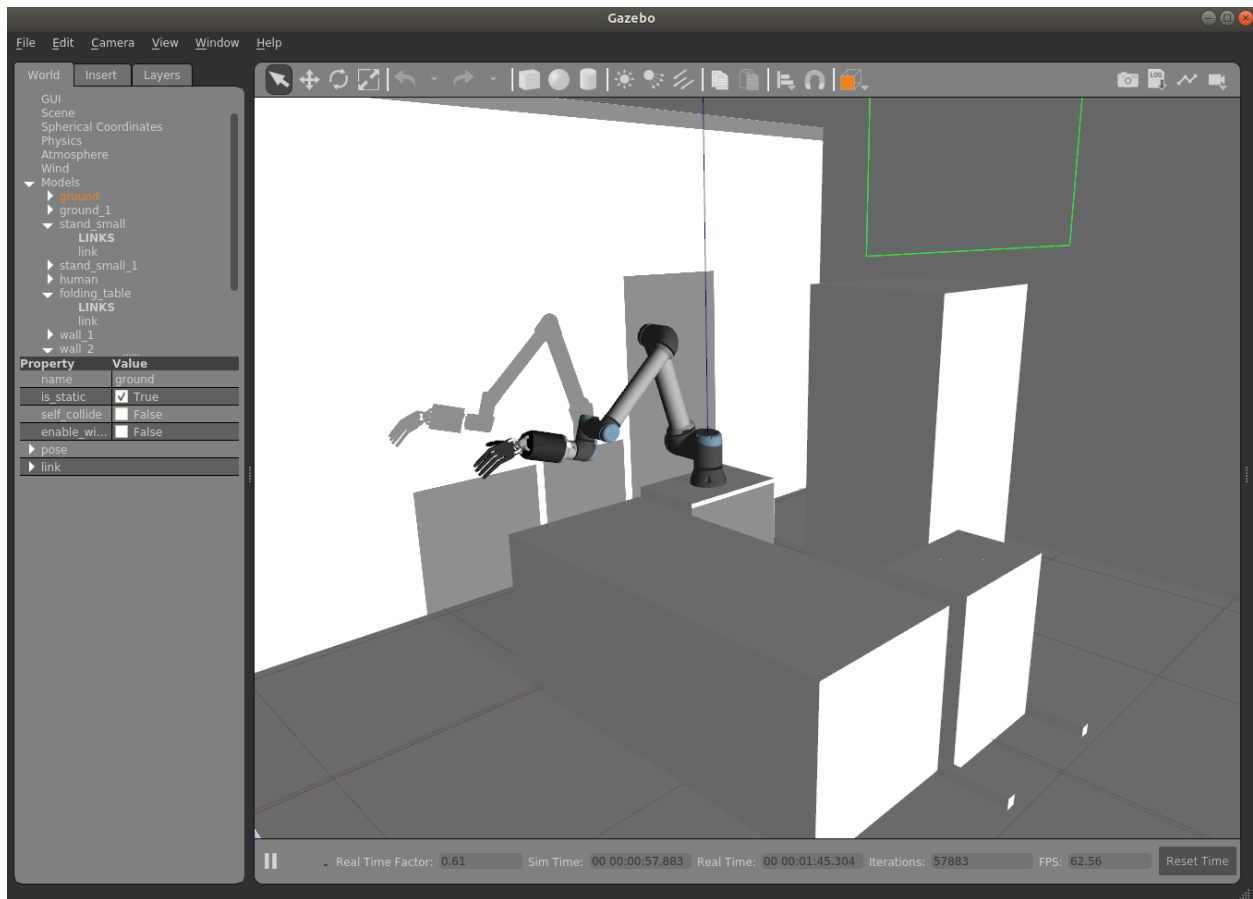
### 9.2.3 Unimanual arm and hand system



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

```
$ roslaunch sr_robot_launch sr_right_ur10arm_hand.launch
```

To add a scene, you can add `scene:=true` and you our default scene. You can also add your own scene adding a `scene_file` parameter.



Similarly, to start the simulation of a unimanual left system, you can run:

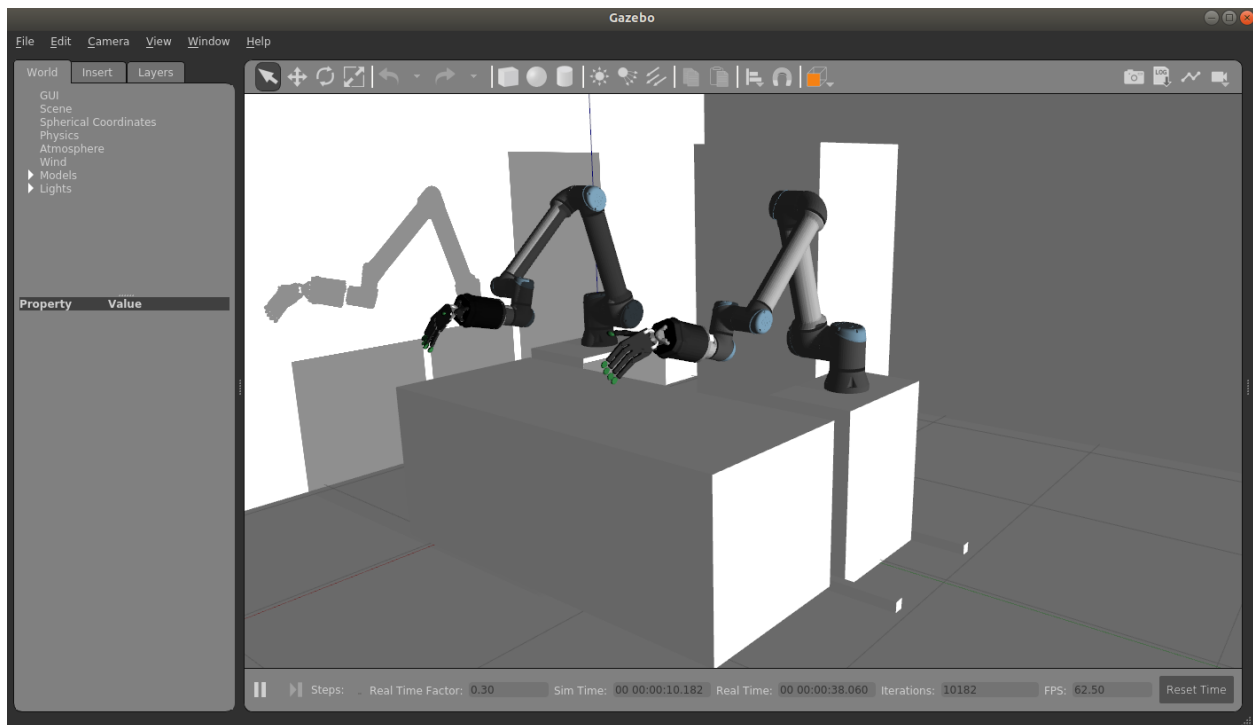
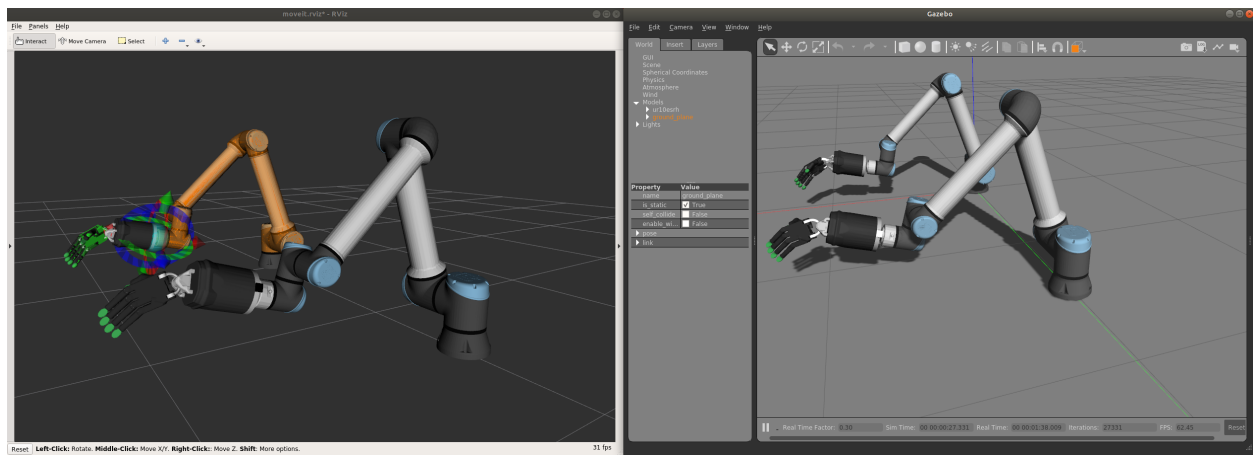
```
$ roslaunch sr_robot_launch sr_left_ur10arm_hand.launch
```

## 9.2.4 Bimanual arm and hand system

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

```
$ roslaunch sr_robot_launch sr_right_ur10arm_hand.launch
```

To add a scene, you can add `scene:=true` and you our default scene. You can also add your own scene adding a `scene_file` parameter.



## 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](#).

### 10.1 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.

Run the following command to pull the docker image:

```
$ docker pull shadowrobot/dexterous-hand:melodic-mujoco-v0.0.2
```

#### 10.1.1 Non-Nvidia GPU systems

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

```
$ docker run --name mujoco_container -it -e DISPLAY -e LOCAL_USER_ID=$(id -u) -e QT_X11_NO_MITSHM=1 -v /dev:/dev
```

#### 10.1.2 Nvidia GPU systems

If you have Nvidia GPU, use following command instead:

```
$ docker run -it --name mujoco_container --net=host --privileged -e DISPLAY -e QT_X11_NO_MITSHM=1 --gpus=all
```

### 10.2 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 of the hand by running the following in the docker container terminal:

```
$ roslaunch sr_robot_launch srhand_mujoco.launch
```

By default, this will launch a right Dexterous Hand Plus. You can also launch a left hand by appending *hand\_id:=lh*:

```
$ roslaunch sr_robot_launch srhand_mujoco.launch hand_id:=lh
```

You can also launch a non-Plus Dexterous Hand by appending *hand\_type:=hand\_e*:

```
$ roslaunch sr_robot_launch srhand_mujoco.launch hand_type:=hand_e
```

These arguments can be combined to launch a non-Plus left Dexterous Hand.

For arm plus hand simulation (ur10 + right Dexterous Hand Plus at the moment) run the following:

```
$ roslaunch sr_robot_launch sr_ur_arm_mujoco.launch
```

## 10.3 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
```

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.



## INSTALLING THE SOFTWARE ON A NEW PC

---

**Note:** By default, we will provide machines that already have all the software set up for you.

---

However, even though each delivery will consist of a NUC-CONTROL machine for Hand's driver (which we always recommend to use), the SERVER Laptop is optional. In case you want to set up a custom machine as a SERVER, please follow the instructions below. The values for each field can be found in the Hand Delivery Instructions provided with the hand.

### 11.1 Hardware specifications

In order to run our software and its dependencies you will need to meet some hardware requirements.

|               |                                                                                                            |
|---------------|------------------------------------------------------------------------------------------------------------|
| CPU           | Intel i5 or above (i7 recommended)                                                                         |
| RAM           | 4GB or above (16 GB recommended)                                                                           |
| Hard Drive    | Fast HDD or SSD as laptop HDD are very slow (500 GB SSD recommended)                                       |
| Graphics Card | Nvidia GPU (optional)                                                                                      |
| LAN           | A spare LAN port to connect the Hand (even with a USB to LAN adaptor)                                      |
| OS            | Ubuntu 20.04 Noetic (Active development), 18.04 Melodic, 16.04 Kinetic or 14.04 Indigo for older releases. |

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

We have created a one-liner that is able to install Docker, download the Docker image and create a new container for you. It will also create desktop icons, one to start the container, one to launch the hand driver on the control box and one to save the log files locally. To use it, you first need to have a PC with Ubuntu installed on it, then follow the next steps.

### 11.2 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. It can be found in the delivering instructions or by emailing [sysadmin@shadowrobot.com](mailto: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 `--read-secure customer_key` to the one-liner. After executing the one-liner, it will prompt you to enter your "Secure data input for customer\_key". Please copy and paste here your key.

## 11.3 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, follow these instructions:

1. Connect the ethernet between the NUC-CONTROL and the new PC using the instructions above
2. Power on the new PC
3. Connect an ethernet cable providing external internet connection to the back of the new PC
4. Power on the NUC-CONTROL
5. Install the hand software on the new PC by running the following on a terminal (Ctrl+Alt+T):

ROS Noetic (Recommended) for a Right Hand:

```
$ bash <(curl -Ls bit.ly/run-aurora) server_and_nuc_deploy --read-secure customer_key ethercat_right_han
```

ROS Noetic (Recommended) for a Left Hand:

```
$ bash <(curl -Ls bit.ly/run-aurora) server_and_nuc_deploy --read-secure customer_key ethercat_left_han
```

ROS Noetic (Recommended) for a Bimanual Hand System:

```
$ bash <(curl -Ls bit.ly/run-aurora) server_and_nuc_deploy --read-secure customer_key product=hand_e etl
```

ROS Melodic for a Right Hand:

```
$ bash <(curl -Ls bit.ly/run-aurora) server_and_nuc_deploy --read-secure customer_key ethercat_right_han
```

ROS Melodic for a Left Hand:

```
$ bash <(curl -Ls bit.ly/run-aurora) server_and_nuc_deploy --read-secure customer_key ethercat_left_han
```

ROS Melodic for a Bimanual Hand System:

```
$ bash <(curl -Ls bit.ly/run-aurora) server_and_nuc_deploy --read-secure customer_key product=hand_e etl
```

where `<ethercat_interface>`, `<config_branch>` and `<product>` are values that will be provided in the Hand Delivery Instructions by Shadow.

If you do not have an Nvidia graphics card, you can add `nvidia_docker=false`.

You can also change `reinstall=false` in case you do not want to reinstall the Docker image and container. When it finishes it will show if it was successful or not and will create 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).

More params and their explanation can be found [here](#).

**Warning:** If for whatever reason the installation does not proceed well or it takes too long, contact us at [support@shadowrobot.com](mailto:support@shadowrobot.com) with the error message. Also, try rerunning the installation script.

## WHY DO WE USE A NUC?

We have migrated the control loop to a separate NUC computer. The reason for that is that we have experienced high cycle overruns with certain setups. We have seen the issue especially in complex systems such as Teleoperation when we are running other heavy programs on the same laptop. If you run just the hand, it might be ok but if you run arm + hand + any other computationally expensive software such as vision, etc. you will start to see the overruns.

### 12.1 Testing the overruns

You can follow these instructions to test the overruns in your computer:

1. Install stress on the computer where the hand is running:

```
$ sudo apt install stress
```

2. Start the hand driver (using either the Launch Shadow Right/Left/Bimanual Hand(s) icon if the hand is connected to the NUC or Launch Local Shadow Right/Left/Bimanual Hand(s) (in Shadow Advanced Launchers folder)

3. Start this command on the computer host where the hand is running:

```
$ stress --cpu 8 --io 4 --vm 2 --timeout 200s --vm-bytes 128M
```

4. In the Docker container of the computer where the hand is running (either Server container if hand running on NUC or if hand is running on the laptop, right-click on the Terminator window and “Split Horizontally”, run this (rh is right hand, if using a left hand use lh)

```
$ rosrun sr_utilities_common overrun_experiments.py -ht hand_e -t 120 -id rh
```

5. Wait 2 minutes. You should see this:

```
$ Your data has been recorded to ./overruns_data.txt file.
```

```
$ Overrun average: <overrun_average> Drop average: <drop_average>
```

We normally want overrun\_average to be less than 0.05 and the drop\_average (dropped packets) to be less than 0.1. This would mean that over 120 seconds, there should be less than 6 overruns and less than 12 drops. You can also open the overruns\_data.txt file to see what the overruns and drops have been.

## 12.2 What will happen if a NUC is not used?

If we don't use a NUC, and even if we use a moderately powerful laptop (i7-8750H/9750H/10750H and 16GB RAM and NVIDIA 1650/1650 TI GPU), when the laptop is stressed (e.g. running any software on the laptop or opening a browser), we get an overrun average of about 11 and a drop average of about 0.45, which means that over 120 seconds we would have 1320 overruns and 54 drops. This would seriously degrade the real-time performance of the hand.

## 12.3 Running the hand without the NUC

Running the hand using the NUC is recommended but not mandatory. There are extra icons to start the hand without a NUC in the “Shadow Advanced Launchers” folder. In that folder, you can use the icon `Launch Local Shadow Right/Left/Bimanual Hand(s)` to run the hand without a NUC (hand has to be connected to server laptop using the same USB-ethernet adapter). Running the hand with an arm is not supported without a NUC. More information on the Advanced Launch icons can be found [here](#).

### ## Repositories

Our code is split into different repositories:

- `[sr_common]`([https://github.com/shadow-robot/sr\\_common](https://github.com/shadow-robot/sr_common)): This repository contains the bare minimum for communicating with the Shadow Hand from a remote computer (urdf models and messages).
- `[sr_core]`([https://github.com/shadow-robot/sr\\_core](https://github.com/shadow-robot/sr_core)): These are the core packages for the Shadow Robot hardware and simulation.
- `[sr_interface]`([https://github.com/shadow-robot/sr\\_interface](https://github.com/shadow-robot/sr_interface)): This repository contains the high level interface and its dependencies for interacting simply with our robots.
- `[sr_tools]`([https://github.com/shadow-robot/sr\\_tools](https://github.com/shadow-robot/sr_tools)): This repository contains more advanced tools that might be needed in specific use cases.
- `[sr_visualization]`([https://github.com/shadow-robot/sr\\_visualization](https://github.com/shadow-robot/sr_visualization)): This repository contains the various rqt\_gui plugins we developed.
- `[sr_hand_config]`([https://github.com/shadow-robot/sr\\_hand\\_config](https://github.com/shadow-robot/sr_hand_config)): This repository contains the customer specific configuration for the Shadow Robot Hand.

## ROS MASTER AND CONNECTING ADDITIONAL COMPUTERS

There are 3 computers involved:

Server laptop, hostname: `serverhostname` (replace it with the actual hostname in these instructions) (IP: 10.9.11.1)

NUC (IP: 10.9.11.2)

Third computer, hostname: `thirdcomputerhostname` (replace it with the actual hostname in these instructions)

The server laptop acts as the ROS MASTER. To connect additional computers with ROS to server laptop and NUC ROS network (to control and see data from the hand/arm), it is only necessary to have the server laptop and the additional non-Shadow computer with ROS on the same network.

To connect the Third computer to the same network as the server laptop and the NUC, follow these steps:

## I have a Shadow-provided router

Connect an ethernet cable from the Shadow-provided router to the Third computer. The Third computer will then get an IP in the same network as the Server laptop and the NUC (e.g. 10.9.11.5)

IPs will be (for example):

Server laptop (IP: 10.9.11.1)

NUC (IP: 10.9.11.2)

Third computer (IP: 10.9.11.5)

### On the server laptop local machine:

Make sure the server laptop hostname (`serverhostname`) is present exactly twice in the server laptop local machine `/etc/hosts`:

```
127.0.0.1 serverhostname
```

```
10.9.11.1 serverhostname
```

Also make sure the Third computer IP is present in this `/etc/hosts` file:

```
10.9.11.5 thirdcomputerhostname
```

### In the server laptop container

You can start a container terminal by clicking on 1 - Launch Server Container in Shadow Advanced Launchers), make sure the name server laptop hostname is also present exactly twice in the `/etc/hosts` of the container:

```
127.0.0.1 serverhostname
```

```
10.9.11.1 serverhostname
```

Also make sure the Third computer IP is present in this `/etc/hosts` file:

```
10.9.11.5 thirdcomputerhostname
```

### On the Third computer (and also in the Third computer Docker container if using Docker on the Third computer)

Make sure the third computer hostname (`thirdcomputerhostname`) is present exactly twice in the third computer `/etc/hosts`:

```
127.0.0.1 thirdcomputerhostname
```

```
10.9.11.5 thirdcomputerhostname
```

Also make sure the server laptop is present in this `/etc/hosts` file:

```
10.9.11.1 serverhostname
```

Run the following command in the Third computer (or inside its Docker container if using Docker):

```
`bash $ export ROS_MASTER_URI=http://serverhostname:11311 `
```

## I don't have a Shadow-provided router

Connect an ethernet cable from your home/office router to the Server laptop or connect the server laptop to your home/office wifi. The server laptop will acquire a second IP address (e.g. 192.168.1.12). The Third computer will be connected to your home/office network and will have a similar IP address (e.g. 192.168.1.11)

IPs will be (for example):

Server laptop (IP: 10.9.11.1 and 192.168.1.12)

NUC (IP: 10.9.11.2)

Third computer (IP: 192.168.1.11)

### On the server laptop local machine:

Make sure the server laptop hostname (`serverhostname`) is present exactly twice in the server laptop local machine `/etc/hosts`:

```
127.0.0.1 serverhostname
```

```
192.168.1.12 serverhostname
```

Also make sure the Third computer IP is present in this `/etc/hosts` file:

```
192.168.1.11 thirdcomputerhostname
```

### In the server laptop container

You can start a container terminal by clicking on 1 - Launch Server Container in Shadow Advanced Launchers), make sure the name server laptop hostname is also present exactly twice in the `/etc/hosts` of the container:

```
127.0.0.1 serverhostname
```

```
192.168.1.12 serverhostname
```

Also make sure the Third computer IP is present in this `/etc/hosts` file:

```
192.168.1.11 thirdcomputerhostname
```

### On the Third computer (and also in the Third computer Docker container if using Docker on the Third computer)

Make sure the third computer hostname (`thirdcomputerhostname`) is present exactly twice in the third computer `/etc/hosts`:

```
127.0.0.1 thirdcomputerhostname
```

```
192.168.1.11 thirdcomputerhostname
```

Also make sure the server laptop is present in this `/etc/hosts` file:

```
192.168.1.12 serverhostname
```

Run the following command in the Third computer (or inside its Docker container if using Docker):

```
`bash $ export ROS_MASTER_URI=http://serverhostname:11311 `
Testing
```

Start the hand using icons on the server laptop. Then, test if the additional computer can see the ROS topics and echo the contents:

```
`bash $ rostopic list `
`bash $ rostopic echo /joint_states `
```

Now the additional computer is fully connected ROS MASTER of the server laptop. See the Software Description > Software description of the Hand > Command line interface

# Software description of the Hand

## Robot Operating System (ROS)

Our hand works within the ROS framework.

“ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.” - ROS.org

You can find the fundamental ROS concepts explained [here](<http://wiki.ros.org/ROS/Concepts>) and a technical overview of the implementation of ROS [here](<http://wiki.ros.org/ROS/Technical%20Overview>).

## Accessing Data from the Hand

There are four main ways to access data from the hand: \* Graphical User Interface (defined in the section below) \* Command line interface (defined in the sections below) \* SrHandCommander (defined in the sections below) \* Using [rospy](<http://wiki.ros.org/rospy>) or [roscpp](<http://wiki.ros.org/roscpp>)

### Example: accessing joint state data

- Using the graphical user interface to view the joint state data in the Data Visualizer.
- Using the Command line interface to view the joint state data in the topic `/joint_state`
- Using SrHandCommander methods of: `* current_state = hand_commander.get_current_state()` `* joints_position = hand_commander.get_joints_position()` `* joints_velocity = hand_commander.get_joints_velocity()`
- Using [ROS Python subscriber]([https://github.com/shadow-robot/sr\\_interface/blob/noetic-devel/sr\\_example/scripts/sr\\_example/advanced/sr\\_subscriber\\_example.py](https://github.com/shadow-robot/sr_interface/blob/noetic-devel/sr_example/scripts/sr_example/advanced/sr_subscriber_example.py)) or [ROS CPP subscriber](<http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29>)

## 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: ``bash $ 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.

#### Starting the interface with namespaces Namespaces are very useful in ROS because they allow users to isolate elements of the network to prevent accidental errors as explained [here](<http://wiki.ros.org/Names>). In order to open the Graphical User Interface within a certain namespace, type: ``bash $ rosrn rqt_gui rqt_gui __ns:=<namespace> ``

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

**Plugins → Robot Tools → Diagnostic Viewer**

```
`eval_rst .. image:: ../img/robot_monitor.png`
```

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.

```
`eval_rst .. image:: ../img/monitor_single_motor.png`
```

The following table has some more information on what each of these fields means.

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

### Hand Tuning It is possible to adjust the settings for any of the Position or Force (Motor) controllers.

**Plugins → Shadow Robot → Advanced → Hand Tuning**



#### Position controller `eval\_rst .. image:: ../img/adjust\_position\_controllers.png`

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 `eval\_rst .. image:: ../img/adjust\_torque\_controllers.png`

- **“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 PWM 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  $\pm$ 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. 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.

`eval\_rst .. image:: ../img/bootloading\_new\_firmware.png`

- **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.

`eval\_rst .. 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 Robot Control Mode Use the *Change Robot Control Mode* plugin to load one of the 4 different types of controllers set by default. Simply click on a controller type, and it will call a service from the controller\_manager to unload the currently running controller if necessary, and load the one you've selected.

**Plugins → Shadow Robot → Change Robot Control Mode**

`eval\_rst .. image:: ../img/selecting\_different\_control\_mode\_1.png`

```
```eval_rst
```

Note: Please allow some time between control changes!

```
```
```

```
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 → Advanced → Motor Resetter**

```
`eval_rst .. image:: ../img/resetting_motor_microcontrollers.png`
```

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**

```
`eval_rst .. image:: ../img/joint_sliders.png`
```

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. The calibration has to be run on the NUC machine, therefore rqt has to be started from it. To do that, you can use a desktop icon prepared for this purpose (see the `Shadow NUC RQT` icon and explanation [here]([https://dexterous-hand.readthedocs.io/en/master/user\\_guide/1\\_2\\_10\\_icons\\_for\\_hand.html#main-desktop-icons](https://dexterous-hand.readthedocs.io/en/master/user_guide/1_2_10_icons_for_hand.html#main-desktop-icons)))

**Within rqt, go to:**

**Plugins → Shadow Robot → Advanced → Hand Calibration**

```
`eval_rst .. image:: ../img/calibrating_joint_sensors.png`
```

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](mailto: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**

```
`eval_rst .. image:: ../img/data_visualization_gui_1.png`
```

You also can launch it separately from rqt with an optional rosbag by running the following command: `sh roslaunch sr_data_visualization data_visualizer.launch rosbag_path:=<absolute_path>`

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 specific data to show (scaled) or you can choose “All” to see several graphs being displayed at the same time (unscaled).

The check buttons next to each graph name allow you to show the graphs you select in larger detail by checking the boxes of the graphs you want to see and clicking “Show Selected”. To return to the full graph view click “Reset”.

This plugin supports a connected hand or a recorded ROS bag. Currently only 1 hand at a time is supported - in case of two hands connected, the plugin will populate its plots for the first detected hand.

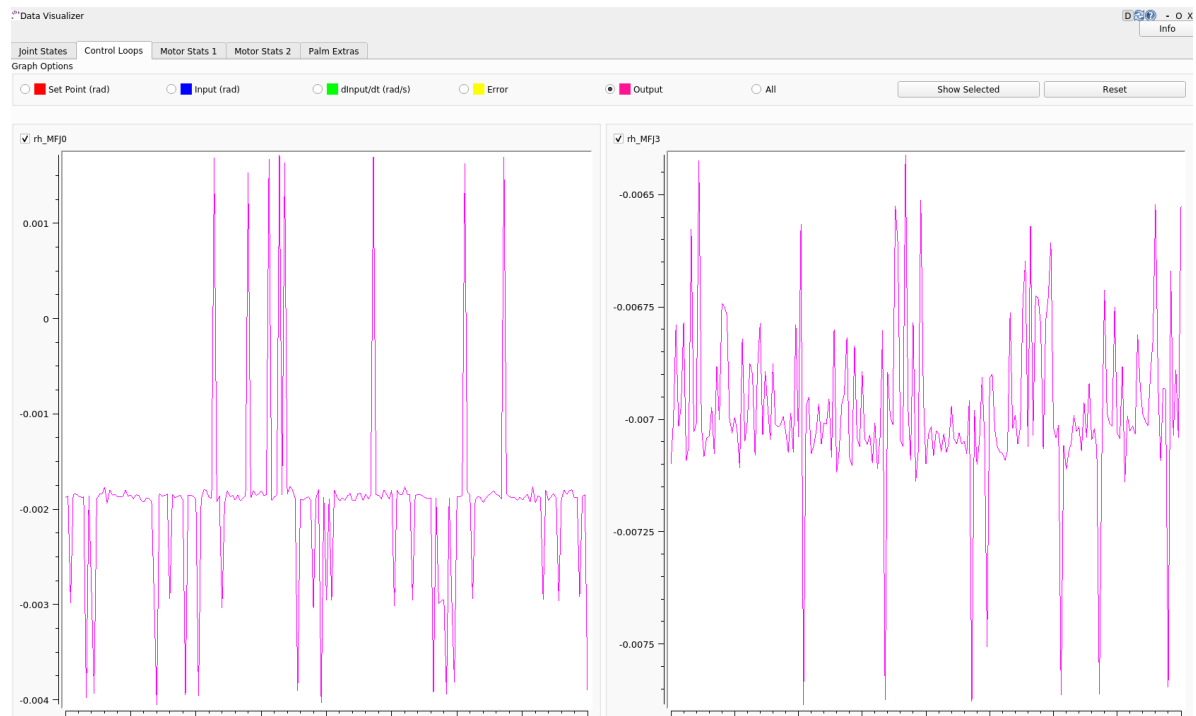
This plugin supports a connected hand or a recorded ROS bag. Currently only 1 hand at a time is supported - in case of two hands connected, the plugin will populate its plots for the first detected hand.

```
eval_rst
```

**Note:** The more graphs that are on show on the data visualizer will be slower and can be unreadable. To be able to see a full scaled view of a specific data type, toggle the correct radio button and check the graphs you want to see clearer.

```
eval_rst
```

```
eval_rst
```

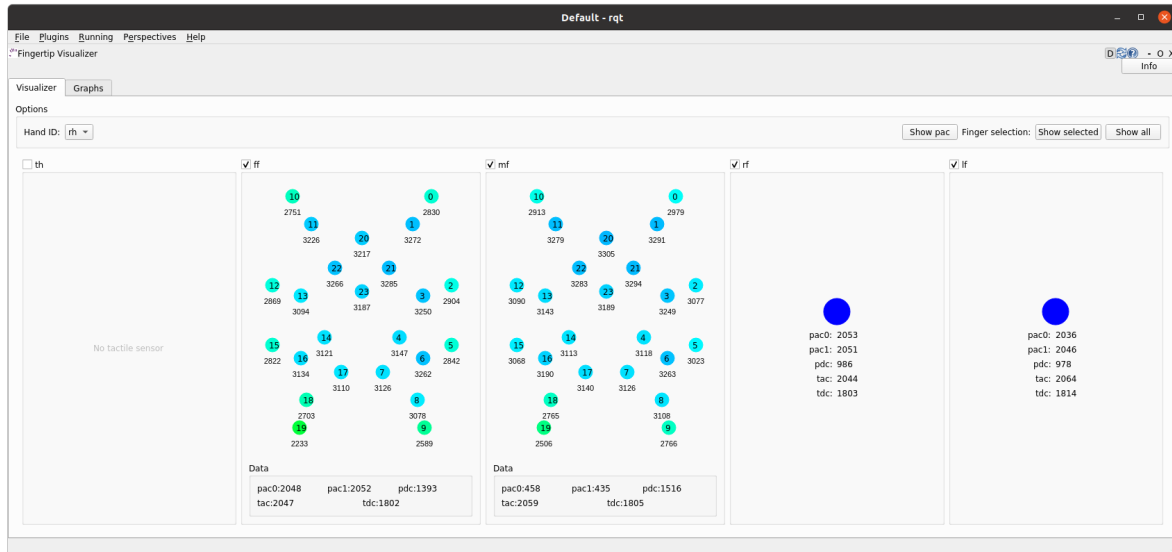


```
eval_rst
```

### Fingertip visualization This is a package to graphically display data coming from the tactile sensors of the Dexterous Hand.

Plugins → Shadow Robot → Fingertip Visualization

```
eval_rst
```



\*\*\*

There are 2 available tabs: - **Visualizer** - **Graphs**

As a user you can select which hands and corresponding sensors you would like to inspect by selecting the **HandID**. Selecting a specific finger will enable or disable the refreshing. You have also the possibility to present only selected fingers by pressing **Show selected** or bring back all of the fingers to the tab by pressing **Show all**.

The **Visualizer** tab represents the data in the form of tactile points changing their colours based on the value coming from the sensors. In the case of a Dexterous Hand equipped with Biotacs as tactile sensors, there is also a button which will allow you to switch the visual representation mode of the tactile points between **electrodes** or **pac** values coming from the sensor.

The **Graphs** tab represents the data in the form of plots for all of the data coming from the sensors. Ticking the corresponding checkbox for the datatype will either add or remove the plot from the graph of the finger.

## How to use it

The gui can be started via roslaunch with an optional robag. The rosbag will be played with the -l option (infinite loop):

```
` roslaunch sr_fingertip_visualization tactile_visualizer.launch rosbag_path:=<absolute_path>`
` or as an rqt plugin:
```

and go to Plugins -> Shadow Robot -> Fingertip Visualizer

This plugin supports presenting the data coming in real time from the Dexterous Hand or from a ROSbag.

## 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>>.

The following rqt\_graph shows the flow of topics between nodes whilst the hand is running.

```
'''eval_rst .. image:: ../img/ethercat_sr_rhand.png

 target
 ../_images/ethercat_sr_rhand.png

'''
```

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, 1.8331224739256338, 1.2888368284819698, -0.13269012433948385,
0.14435534682895756, 0.6980816915624072, 0.18782898954368935, 1.124295322901818,
0.21905854304869088, -0.048455186771971595, -0.0032803323337213066] veloc-
ity: [-7.484333985952662e-06, -7.484333985952662e-06, 0.0023735860019749185,
0.00062181267775619, -0.0005871136552505063, -0.0005871136552505063,
0.0020967687295392933, 0.0001739028157522596, 0.0004985252400775274, -
9.485516545601461e-06, -9.485516545601461e-06, -0.0007068752456452666, -
0.0012475428276090576, 0.0008426052935621657, 0.0008426052935621657,
0.001237001167977189, -0.0026444893567459573, 0.0025260047430310925, -
0.0003217106977882921, 6.159570145597239e-05, -0.0023454723015513593,
0.0009436399232442155, 0.00017469681801687975, -4.900148416020751e-05] effort: [-
1.3660655058510802, -1.3660655058510802, -2.030169817308198, -1.9577332816789155, 0.0,
0.0, -17.29928766980003, -1.5006516553524243, -1.8579749510438912, -1.504877130092884,
-1.504877130092884, -0.3374653182042338, -1.6492254479379729, -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. Which value corresponds to which data is defined [here.]([https://github.com/shadow-robot/hand-firmware/blob/ff95fa8fc50a372c37f5fedcc5b916f4d5c4afe2/PIC32/nodes/0220\\_palm\\_edc/0220\\_palm\\_edc\\_ethercat\\_protocol.h#L88](https://github.com/shadow-robot/hand-firmware/blob/ff95fa8fc50a372c37f5fedcc5b916f4d5c4afe2/PIC32/nodes/0220_palm_edc/0220_palm_edc_ethercat_protocol.h#L88))
- *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 = 010101010101010101.
- *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.]([https://github.com/shadow-robot/hand-firmware/blob/ff95fa8fc50a372c37f5fedcc5b916f4d5c4afe2/PIC32/nodes/common/tactile\\_edc\\_ethercat\\_protocol.h#L74](https://github.com/shadow-robot/hand-firmware/blob/ff95fa8fc50a372c37f5fedcc5b916f4d5c4afe2/PIC32/nodes/common/tactile_edc_ethercat_protocol.h#L74))
- *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 referred [here.](<https://github.com/shadow-robot/hand-firmware/blob/>)

ff95fa8fc50a372c37f5fedcc5b916f4d5c4afe2/PIC32/nodes/common/tactile\_edc\_ethercat\_protocol.h#L93)

- *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 to communicate on the EtherCAT bus.

`eval\_rst .. 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:

**header:**

seq: 176798 stamp:

secs: 1528812878 nsecs: 323410491

frame\_id: ''

sensors: [1303, 1574, 3205, 1780, 1382, 1523, 3164, 1938, 904, 1332, 2977, 1706, 1730, 1434, 3060, 1853, 1955, 1814, 2132, 2294, 2496, 4029, 1668, 2931, 1768, 1377, 26, 27, 28, 29, 30, 31, 0, 19, 8, 9, 0] motor\_data\_type:

data: 3

which\_motors: 0 which\_motor\_data\_arrived: 349525 which\_motor\_data\_had\_errors: 0 motor\_data\_packet\_torque: [15, -31, -4, 3, 0, 0, -207, -3, -55, -3] motor\_data\_packet\_misc: [-105, -47, 0, -39, 0, 0, 120, 0, 79, 0] tactile\_data\_type: 0 tactile\_data\_valid: 31 tactile: [407, 429, 416, 398, 389] idle\_time\_us: 430 — header:

seq: 176799 stamp:

secs: 1528812878 nsecs: 324399217

frame\_id: ''

sensors: [1303, 1574, 3205, 1780, 1382, 1523, 3164, 1938, 904, 1332, 2977, 1706, 1731, 1434, 3060, 1853, 1955, 1814, 2131, 2294, 2496, 4030, 1669, 2931, 1768, 1376, 26, 27, 28, 29, 30, 31, 19, 10, 0, 0, 0] motor\_data\_type:

data: 4

which\_motors: 1 which\_motor\_data\_arrived: 699050 which\_motor\_data\_had\_errors: 0 motor\_data\_packet\_torque: [-29, -3, 1, -35, -1, -22, -18, 35, 4, 5] motor\_data\_packet\_misc: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0] tactile\_data\_type: 0 tactile\_data\_valid: 0 tactile: [407, 429, 416, 398, 389] idle\_time\_us: 394

- 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 stride: 0
- label: “analog\_inputs” size: 4 stride: 0

data\_offset: 0

```
data: [26.0, 27.0, 28.0, 29.0, 30.0, 31.0, 4.0, 5.0, 0.0, 8.0]
```

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:

**header:**

seq: 126618 stamp:

secs: 1528813967 nsecs: 440903704

frame\_id: “rh\_distal”

pressure: [405, 428, 422, 401, 384] temperature: [1224, 1198, 1225, 1242, 1266]

Example topic message when using BioTac fingertip sensors:

[illegible]

- BioTac (Only for a real hand with Biotac tactile sensors)

These topics are read-only and updated 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 biotacs, refer to their documentation: <[https://www.syntouchinc.com/wp-content/uploads/2016/12/BioTac\\_SP\\_Product\\_Manual.pdf](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]

- Trajectory Controller - Command

/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:

```
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,
0.8552113334772214, -0.17453292519943295, 1.0297442586766545,
```



```

1.4311699866353502, 1.413716694115407, 0.007182575752410699,
0.9773843811168246, 1.5707963267948966, 1.2566370614359172, -
0.12217304763960307, 0.4014257279586958, 1.2566370614359172,
1.5184364492350666, 0.017453292519943295, 0.13962634015954636,
0.12217304763960307, 0.6632251157578453, 0.17453292519943295,
1.117010721276371, -0.7504915783575618, -0.03490658503988659, 0.0] velocities:
[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 updates at 50 Hz from the trajectory controller with the positions and velocities of all 24 joints.

Example topic message:

```

positions: [0.0029928404547430176, 0.0007821521859359137, 0.004102784627362688,
-0.001230489872427576, 0.002876479952986344, 0.0006426181816490129,
0.006354919224207833, 0.00213663812281073, 0.003279618063753098,
0.0020929781564538175, 0.0063066586043154516, 0.0038023568140372888, -
0.002289758750686488, -1.1040675065743244e-05, 0.008137524637908733, -
2.1288137004304986e-05, 0.0009348013388894572, -0.003295237358051928,
0.039981480504079236, -0.0035961821430152696, 0.0032603043080507987,
2.9988784142176428e-05, -0.00029934074598525484, -8.999634459527783e-05] veloc-
ities: [-0.0008510441551395189, -0.0008510441551395189, 0.00016883698712266695,
0.00034715798956923955, -0.00017869100331692196, -0.00017869100331692196,
-0.001275520583476054, -0.0004885423191519772, 0.00012555078906251334,
0.00012555078906251334, 0.0028653614401722843, -0.0008023399951605057,
0.0011760287859774613, 0.0011760287859774613, -0.0005423468659163991, -
0.00017066612487367117, 0.0003102610817406156, -0.001127052578802167, -
0.001465708865391472, -0.00028520412005307133, -0.00029795158858164227,
0.0002596403670543647, -5.819600689424957e-05, -0.0002980347643777659]

```

#### – 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 Robot Control Mode rqt plugin.

Example of running `roscat` bash

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

```

```

Type: std\_msgs/Float64 Publishers:

/rqt\_gui\_py\_node\_23644 (<http://shadow-bravo:38385/>) Subscribers:

/sr\_hand\_robot (<http://shadow-bravo:45091/>)

/rostopic\_15687\_1526406188893 (<http://shadow-bravo:36637/>)

/record (<http://shadow-bravo:35575/>)

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 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: 0.0d: 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]([https://github.com/shadow-robot/sr-config/blob/kinetic-devel/sr\\_ethercat\\_hand\\_config/controls/host/rh/sr\\_edc\\_joint\\_position\\_controllers\\_PWM.yaml#L9](https://github.com/shadow-robot/sr-config/blob/kinetic-devel/sr_ethercat_hand_config/controls/host/rh/sr_edc_joint_position_controllers_PWM.yaml#L9)). *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

/sh\_rh\_\*\_position\_controller/pid/parameter\_descriptions /sh\_rh\_\*\_position\_controller/pid/parameter\_updates

These topics are read-only and contain parameters used for tuning the position controllers. They should not be published directly, but can be accessed through rqt\_reconfigure.

- TF

/tf /tf\_static

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

- Mechanism Statistics

/mechanism\_statistics

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

```
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/>>

- Collisions

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

```
/attached_collision_object /collision_object
```

- Trajectory Execution

Live information regarding the current trajectory execution.

```
/execute_trajectory/cancel /execute_trajectory/feedback /execute_trajectory/goal /execute_trajectory/result /execute_trajectory/status
```

- 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)>

```
/rviz_*/motionplanning_planning_scene_monitor/parameter_descriptions
/rviz_*/motionplanning_planning_scene_monitor/parameter_updates
/rviz_moveit_motion_planning_display/robot_interaction_interactive_marker_topic/feedback
/rviz_moveit_motion_planning_display/robot_interaction_interactive_marker_topic/update
/rviz_moveit_motion_planning_display/robot_interaction_interactive_marker_topic/update_full
```

### Using rosservice To reset individual motors, E.G. FFJ3:

```
***bash
$ rosservice call /realtime_loop/reset_motor_FFJ3

```

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

```
***bash
$ rosservice call /realtime_loop/xxxxxx

```

## Writing controllers Rather than use the ROS topics to access sensor data, you will need to write a plugin for the 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 Controller Manager:

<[http://wiki.ros.org/ros\\_control](http://wiki.ros.org/ros_control)>

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.

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

```
***bash
$ 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 that you can edit. To edit the file:

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

The complete list of motor sensors appears in the file, along with a number `eval_rst =====`  
`===== Number Meaning ===== -2 Read`  
 once when the driver is launched -1 Read as fast as possible

0 Do not use zero

>0 Read period in seconds `=====`

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.

## How to control the hand - the robot commander

The robot commander provides a high level interface to easily control the different robots supported by Shadow Robot. It encapsulates functionality provided by different ROS packages, especially the `moveit_commander`, providing access via a simplified interface.

There are three classes available: \* `[SrRobotCommander](https://github.com/shadow-robot/sr_interface/blob/noetic-devel/sr_robot_commander/src/sr_robot_commander/sr_robot_commander.py)`: base class. Documentation can be found in the following [link](https://dexterous-hand.readthedocs.io/en/latest/user\_guide/2\_software\_description.html#srrobotcommander). \* `[SrHandCommander](https://github.com/shadow-robot/sr_interface/blob/noetic-devel/sr_robot_commander/src/sr_robot_commander/sr_hand_commander.py)`: hand management class. Documentation can be found in the following [link](https://dexterous-hand.readthedocs.io/en/latest/user\_guide/2\_software\_description.html#srhandcommander). \* `[SrArmCommander](https://github.com/shadow-robot/sr_interface/blob/noetic-devel/sr_robot_commander/src/sr_robot_commander/sr_arm_commander.py)`: hand management class

### SrRobotCommander

#### Overview

`eval_rst`

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 descriptions of the most relevant functions of the hand commander.

#### Basic terminology `eval_rst` 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 that 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 `eval_rst`

Import the hand commander along with basic rospy libraries:

```
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:

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

```` ##### Getting basic information ````eval\_rst We can get the name of the robot, group or planning reference frame:

```
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:

```
# 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:

```
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:

```
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 ````eval\_rst You can change the reference frame to get pose information:

```
hand_commander.set_pose_reference_frame("palm")
```

You can also activate or deactivate the teach mode for the robot:

```
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 ````eval\_rst Using the methods `plan_to_joint_value_target`, `move_to_joint_value_target` or `move_to_joint_value_target_unsafe`, a set of the 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 joints' 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`  
````eval_rst`

```

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': 0.0,
                 '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 ```eval_rst`

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 ```eval_rst`

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

```

<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"/>

```

(continues on next page)

(continued from previous page)

```

<joint name="rh_LFJ3" value="1.5707"/>
<joint name="rh_LFJ4" value="0"/>
<joint name="rh_LFJ5" value="0"/>
<joint name="rh_WRJ1" value="0"/>
<joint name="rh_WRJ2" value="0"/>
</group_state>

```

Here is how to move to it:

```

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 ``eval_rst 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** specifies a dictionary of waypoints with the following elements:

- name: the name of the waypoint
- interpolate_time: time to move from last waypoint
- pause_time: time to wait at this waypoint

Example ``eval_rst

```

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_

```

(continues on next page)

(continued from previous page)

```
↳ can use the unsafe method:
hand_commander.run_named_trajectory_unsafe(trajectory)
```

```
``` ##### Check if a plan is valid and execute it ```eval_rst
```

Use the method `check_plan_is_valid` and `execute` to check if the current plan contains a valid trajectory and execute it. This only has meaning if called after a planning function has been attempted. ``` ##### Example ```eval\_rst

```
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 ```eval_rst Use the method ``send_stop_trajectory_unsafe to send a tra-
jectory with the current joint state to stop the robot at its current position. ``` ##### Example ```eval_rst
```

```
hand_commander.send_stop_trajectory_unsafe()
```

```
```
```

```
### SrHandCommander
```

```
##### Overview ```eval_rst
```

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 ```eval_rst

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

```
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)
```

The constructor for the `SrHandCommander` takes a name parameter that should match the group name of the robot to be used. Also it takes the hand prefix, parameters and serial number that can be retrieved using the `HandFinder`.

```
``` ##### Example ```eval_rst
```

```
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)
```

(continues on next page)



(continued from previous page)

```
Alternatively you can launch the hand directly
hand_commander = SrHandCommander(name = "right_hand", prefix = "rh")
```

```
``` ##### Getting information ```eval_rst
```

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

```
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`.

```
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 ```eval_rst
```

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 ```eval_rst
```

```
## 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)
```

```
```
```

```
SrArmCommander
```

The `SrArmCommander` inherits all methods from the [robot commander]([https://dexterous-hand.readthedocs.io/en/latest/user\\_guide/2\\_software\\_description.html#srrobotcommander](https://dexterous-hand.readthedocs.io/en/latest/user_guide/2_software_description.html#srrobotcommander)) and provides commands specific to the arm. It allows movement to a certain position in cartesian space, to a configuration in joint space or move using a trajectory.

##### Setup ```eval\_rst Import the arm commander along with basic rospy libraries and the arm finder:

```
import rospy
from sr_robot_commander.sr_arm_commander import SrArmCommander
from sr_utilities.arm_finder import ArmFinder
```

The constructors for `SrArmCommander` take a name parameter that should match the group name of the robot to be used and has the option to add ground to the scene.

```
arm_commander = SrArmCommander(name="right_arm", set_ground=True)
```

Use the `ArmFinder` to get the parameters (such as prefix) and joint names of the arm currently running on the system:

```

arm_finder = ArmFinder()

To get the prefix or mapping of the arm joints. Mapping is the same as prefix but
↳ without underscore.
arm_finder.get_arm_parameters().joint_prefix.values()
arm_finder.get_arm_parameters().mapping.values()

To get the arm joints
arm_finder.get_arm_joints()

```

```

"""

```

##### Getting basic information `eval_rst` To return the reference frame for planning in cartesian space:

```

reference_frame = arm_commander.get_pose_reference_frame()

```

```

"""

```

##### Plan/move to a position target `eval_rst` Using the method `move_to_position_target`, the end effector of the arm can be moved to a certain point in space represented by (x, y, z) coordinates. The orientation of the end effector can take any value.

Parameters:

- `xyz` desired position of end-effector
- `end_effector_link` name of the end effector link (default value is empty string)
- `wait` indicates if the method should wait for the movement to end or not (default value is True)

```

""" ##### Example """eval_rst

```

```

rospy.init_node("robot_commander_examples", anonymous=True)
arm_commander = SrArmCommander(name="right_arm", set_ground=True)

new_position = [0.25527, 0.36682, 0.5426]

To only plan
arm_commander.plan_to_position_target(new_position)

To plan and move
arm_commander.move_to_position_target(new_position)

```

```

"""

```

##### Plan/move to a pose target `eval_rst` Using the method `move_to_pose_target` allows the end effector of the arm to be moved to a certain pose (position and orientation) in the space represented by (x, y, z, rot\_x, rot\_y, rot\_z).

Parameters:

- `pose` desired pose of end-effector: a Pose message, a PoseStamped message or a list of 6 floats: [x, y, z, rot\_x, rot\_y, rot\_z] or a list of 7 floats [x, y, z, qx, qy, qz, qw]
- `end_effector_link` name of the end effector link (default value is empty string)
- `wait` indicates if the method should wait for the movement to end or not (default value is True)

```

""" ##### Example """eval_rst

```

```

rospy.init_node("robot_commander_examples", anonymous=True)
arm_commander = SrArmCommander(name="right_arm", set_ground=True)

new_pose = [0.5, 0.3, 1.2, 0, 1.57, 0]

To only plan
arm_commander.plan_to_pose_target(new_pose)

To plan and move
arm_commander.move_to_pose_target(new_pose)

```

```

'''

```

## 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.

```

`eval_rst .. image:: ../img/rviz_warehouse_connect.png`

```

If you have connected successfully you should see two new buttons, **Reset database** and **Disconnect**, as can be seen in the following picture:

```

`eval_rst .. image:: ../img/rviz_warehouse_connected.png`

```

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 to the 'Planning' tab and in the 'Select Goal State' select 'current' and click **update**.

```

`eval_rst .. image:: ../img/rviz_select_goal_state.png`

```

- 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.

```

`eval_rst .. image:: ../img/save_state.png`

```

### ## Recording ROS Bags

A rosbag or bag is a file format in ROS for storing ROS message data. These bags are often created by subscribing to one or more ROS topics, and storing the received message data in an efficient file structure.

The different ways to record and playback ROS bags can be found [here](<http://wiki.ros.org/rosbag>)

### Example: Recording and playing a ROS Bag of joint states To record a ROS Bag of the /joint\_states topic for 1 minute and name it *joint\_state\_bag.bag*. The [command-line tool](<http://wiki.ros.org/rosbag/Commandline>) can be used:

```

'''eval_rst

```

```

rosbag record --duration=1m joint_state_bag.bag /joint_states

```

```

'''

```

To find information about the rosbag *joint\_state\_bag.bag*:

```

'''eval_rst

```

```

rosbag info joint_state_bag.bag

```

```

'''

```

To play back this ROS Bag:

```

'''eval_rst

```

```
roslaunch joint_state_bag
```

\*\*\*

The roslaunch command-line has many different options of how to record and playback various topics that are published, these can be found [here](<http://wiki.ros.org/roslaunch/CommandLine>).

## Copying data out of the dexterous hand container

`docker cp` is a way to copy files/folders between a container and the local filesystem. An extended description can be found [here](<https://docs.docker.com/engine/reference/commandline/cp/>).

Copying FROM the container TO the file system:

\*\*\*eval\_rst

```
docker cp [OPTIONS] CONTAINER:SRC_PATH DEST_PATH
```

\*\*\*

Copying FROM the file system TO the container:

\*\*\*eval\_rst

```
docker cp [OPTIONS] DEST_PATH CONTAINER:SRC_PATH
```

\*\*\*

Some of the *[OPTIONS]* include:

\*\*\*eval\_rst

| Name, shorthand   | Description                                 |
|-------------------|---------------------------------------------|
| -archive , -a     | Archive mode (copy all uid/gid information) |
| -follow-link , -L | Always follow symbol link in SRC_PATH       |

\*\*\*

## Hand autodetection (**new in Noetic**)

This feature allows users to detect Shadow Hands without knowing the ethernet interface or the hand serial and run launchfiles without needing to provide detailed information about the hands. It is implemented in the `[sr_hand_detector package]`([https://github.com/shadow-robot/sr\\_hand\\_detector](https://github.com/shadow-robot/sr_hand_detector)) and consists of two scripts.

### Installation \*\*\*eval\_rst

In all Shadow's docker images the feature will be available out of the box, however, for custom setups, you might need to install it manually. Recommended way is just to use debian installation:

```
sudo apt update && sudo apt install ros-<rostdistro>-sr-hand-detector
```

If for some reason a manual installation is required, you can follow steps below:

1. Clone the repository to your ROS workspace
2. Compile the code
3. Copy both executables of the `sr_hand_detector` package (found in `<your_workspace>/devel/lib/sr_hand_detector`) to `/usr/local/bin`.
4. Give one of the executables capability to access ethernet devices:

```
sudo setcap cap_net_raw+ep sr_hand_detector_node
```

```
```
```

Finally, if you want to use the autodetection feature with our launchfiles, you need to clone [sr_hand_config package](https://github.com/shadow-robot/sr_hand_config) into your workspace.

sr_hand_detector_node ```eval_rst The script is purely for hand detection. Usage:

```
sr_hand_detector_node
```

Example output:

```
Detected hand on port: enx000ec653b31a
Hand's serial number: 634
```

Apart from the console output, all detected hand ethernet port names together with corresponding hand serial numbers will be set inside of the /tmp/sr_hand_detector.yaml file.

If there are no hands detected on any of the ports, a warning will be shown:

```
No hand detected on any of the ports!
```

```
``` ### sr_hand_autodetect ```eval_rst
```

This script is a launchfile wrapper, and allows users to run Shadow Robot launch files without providing information like hand serial, ethercat port or hand side. Example usage:

```
sr_hand_autodetect roslaunch sr_robot_launch srhand.launch sim:=false
```

which will effectively run:

```
roslaunch sr_robot_launch srhand.launch sim:=false eth_port:=<eth_port> hand_serial:=
<hand_serial> side:=<hand_side> hand_type:=<hand_type> mapping_path:=<mapping_path>
```

```
```
```

When using the wrapper, all the necessary information is extracted from the [sr_hand_config package](https://github.com/shadow-robot/sr_hand_config).

ROBOT DESCRIPTIONS (URDF)

We currently have modular xacro files for our robots including hands and arms setups, allowing the robots to start in various configurations. They can be found in our [sr_description](#) and [sr_interface](#) packages.

15.1 Shadow Hands

15.1.1 Unimanual

The main xacro file to use is [sr_hand.urdf.xacro](#) when you are using only one of our hands.

The following arguments are available:

- **side** - defines the side of the hand. Allowed options: `right/left`
- **hand_type** - defines the type of the hand. Allowed options: `hand_e/hand_g/hand_c`
- **hand_version** - defines version for particular type of hand.
- **fingers** - defines which fingers does the hand have, can be `all` or a string in a format of `th,ff,mf,rf,lf`

Current allowed configurations are the following:

| | Dexterous Hand | Dexterous Hand Lite | Dexterous Hand Extra Lite | Muscle_hand (deprecated) |
|--------------|----------------|---------------------|---------------------------|--------------------------|
| hand_type | hand_e | hand_g | hand_g | hand_c |
| hand_version | E3M5, E2M3 | G1M5 | G1M5 | C6M2 |
| fingers | all | all | all | all |
| | th,ff,mf,rf,lf | th,ff,mf,rf | th,ff,mf | th,ff,mf,rf,lf |

There are also arguments that define where and which sensors are located on the hand. It allows placement of sensors on tip, mid and proximal parts of the fingers as well as the palm. Argument names: `tip_sensors`, `mid_sensors`, `prox_sensors`, `palm_sensor`. Currently, only sensors at the fingertips are available. There are three fingertip sensor types: `pst`/`bt_sp`/`bt_2p`.

| | PST | Syntouch Biotacs | |
|-------------|-----|------------------|-------|
| | | 2p | sp |
| tip_sensors | pst | bt_2p | bt_sp |

15.1.2 Bimanual

If you have a setup with two robot hands, this is the xacro to use: [sr_hand_bimanual.urdf.xacro](#)

The following arguments are available (similar to the hand-only scenario but with the side prefix to specify every configuration):

- `right_hand_type`
- `right_hand_version`
- `right_fingers`
- `right_tip_sensors`
- `right_mid_sensors`
- `right_prox_sensors`
- `right_palm_sensor`
- `left_hand_type`
- `left_hand_version`
- `left_fingers`
- `left_tip_sensors`
- `left_mid_sensors`
- `left_prox_sensors`
- `left_palm_sensor`

15.2 Shadow Hands mounted on UR arms

The main xacros for Universal Robot Arms and Shadow hand systems are:

15.2.1 Unimanual

- [srhand_ur.urdf.xacro](#)

Additional parameters:

- `robot_model` - defines which robot model is used. Allowed options: `ur10/ur10e/ur5/ur5e`
- `initial_z` - defines how high above the ground the robot is spawned

15.2.2 Bimanual

- Bimanual arms: [bimanual_ur.urdf.xacro](#)
- Bimanual arms and hands; [bimanual_srhand_ur.urdf.xacro](#)

Additional parameters:

- `robot_model` - defines which robot model is used. Allowed options: `ur10/ur10e/ur5/ur5e`
- `arm_1_z` - defines how high above the ground the right robot arm is spawned
- `arm_2_z` - defines how high above the ground the left robot arm is spawned

- `arm_x_separation` - x separation of the left arm with respect to the right arm
- `arm_y_separation` - y separation of the left arm with respect to the right arm

15.3 Usage

For usage example, refer to the xacro files themselves or the `unimanual` and `bimanual` launchfiles that use them. When used with Shadow Hands all the hand parameters are automatically set for you with the autodetection. However, if you are running in simulation or just want to omit the autodetection and set them manually, you can pass the args directly to the launchfile or xacro command. The following are examples on how to use them.

- Launch file:

```
$ roslaunch sr_robot_launch srhand.launch side:=right hand_type:=hand_g hand_version:=G1M5 fingers:=
```

- Xacro command:

```
$ xacro <xacro file> side:=right hand_type:=hand_g hand_version:=G1M5 fingers:=th,ff,mf,rf,lf tip_s
```

As far as SRDF's are concerned, all necessary ones are autogenerated from `robot_description` ros parameters spawned to the parameter server.

15.4 Autodetection parameters

For each of the hands, there is a `general_info.yaml` file that contains information about the hand and will be used to pass correct arguments to the launchfiles, and further to the xacos. When hand is being autodetected, the script will look into that file, extract all necessary arguments and provide them to the launchfile as a command suffix. All of the “general info” files can be found in `sr_hand_config` repository, inside hand serial folder corresponding to each particular hand.

Using Peripherals

Cyberglove

```
`eval_rst .. warning:: The Cyberglove is not supported in our latest releases neither
in Melodic nor Noetic ROS versions of our docker images. Documentation here is only for
reference but we don't provide support for this anymore. `
```

Introduction The Cyberglove is a data glove designed for use with Virtual Reality that Shadow has integrated with our Dexterous Hand to provide a 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.

```
`eval_rst .. image:: ../img/psu1.png `
```

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.

```
`eval_rst .. image:: ../img/power_switch.png `
```

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](https://dexterous-hand.readthedocs.io/en/master/user_guide/1_setting_up_the_hand.html#on-a-new-pc-using-the-one-liner) for instructions on launching a docker image.

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

```
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.

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

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:

```
`eval_rst .. image:: ../img/glove_versions.jpeg`
```

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 *version:=2* and the newer one works with *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.

```
#### Calibration GUI `eval_rst .. image:: ../img/calibration_gui.png`
```

- 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 `eval_rst .. image:: ../img/tweak_gui.png`
```

- 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:

```
`eval_rst +-----+-----+-----+-----+-----+-----+-----+ |
Button | \+ \+ | \+ \+ | \+ | \- \- \- | \- \- | \- | +-----+-----+-----+
| Adjustment To Value | +0.1 | +0.01 | +0.001 | -0.1 | -0.01 | -0.001 |
+-----+-----+-----+-----+-----+-----+`
```

- 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.

```
## 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 will be set up automatically. Example of the oneliner is illustrated below:

```
bash $ bash <(curl -Ls http://bit.ly/launch-sh) -i shadowrobot/dexterous-hand:kinetic-release
-n dexterous-hand -sn Hand_Launcher -e [EtherCAT interface ID] -b [sr_config_branch] -o
true `
```

```
`eval rst .. Note::Please remember to replace [EtherCAT interface ID] with your Interface ID and [sr_config_branch] with your unique sr_config branch`
```

For more information on setup and getting started with the optoforce sensors, [look here](<https://github.com/shadow-robot/optoforce/tree/indigo-devel/optoforce>).

Topics

Optoforce sensor data will be published on the following topics:

/rh/optoforce_**

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 biotacts, refer to [their documentation](https://www.syntouchinc.com/wp-content/uploads/2016/12/BioTac_SP_Product_Manual.pdf).

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:

[illegible]

- 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 biotacs, 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]

Hand Frequently Asked Questions

A list of common issues and how to resolve them.

Hardware

Q: How do I know when the Hand is powered on?

A: Lights will come on on the back of the hand and the fans will be audible.

Q: The Hand will not power on, why is this?

A: Check all connections from the power supply to the Hand. Check that the plug socket is turned on and the mains lead is plugged in. If all the connections are OK and the Hand still won't power on, contact us by sending an email to support@shadowrobot.com.

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

A: Check: * Your ethernet cable connection * The link light on your computer's ethernet port is in a fixed state, not flashing * You're using the right interface - instructions to check this are [here](https://shadow-robot-company-dexterous-hand-and-arm.readthedocs-hosted.com/en/latest/user_guide/1_4_Installing_the_software.html#check-your-hand-interface-id).

Q: Can the Hand be controlled wirelessly?

A: The hand is currently not wireless, as it needs both power and data cables attached.

Q: How do the tactile sensors work and how do they communicate?

A: There are two types of tactile sensing for the fingertips: * **PST**: these are simple sensors, fitted as standard, which measure the air pressure within a bubble at the fingertip. When the fingertips press an object, the sensor detects the change in pressure. The sensor incorporates an automatic drift and temperature compensation algorithm (essentially a high pass filter with an extremely low cut off frequency). * **Biotacs**: please refer to their website for more information: <https://syntouchinc.com/>

Both tactile sensors are attached by a cable to the Finger Proximal Boards.

Q: Can I fix the Hand myself?

A: We provide a set of tools that allow you to re-tension the tendons and complete some other tasks on the Hand yourself, so it won't have to be sent back to Shadow. Where possible, one of our support staff will organise a video call with you to talk you through how to fix any problems you may have. For more complex diagnostic issues, the Hand will need to be sent back to Shadow.

Software

Q: The Hand doesn't react after startup of the container

A: Check that: * The interface ID is set correctly * You have set the correct branch for sr_config If it is still not responding, power cycle the Hand and try again.

Q: Why is the control loop now running on the NUC? Can I run the system without the NUC if I prefer?

A: The NUC was introduced to avoid cycle skips for certain setups. This is more common in our more complex systems when we are running other heavy programs on the same laptop. If you are running just the hand locally you may not experience this issue, but you will start to see overruns if running other computationally expensive software. We strongly advise that the Hand is run using the NUC to ensure best performance.

However, using the NUC is not mandatory and no changes have been made to the firmware. We have provided a set of icons that are in a Desktop folder called "Shadow Advanced Launchers". One of these icons, called "Launch Local Shadow Hand", will allow you to launch and run the hand connected locally to your laptop. More information on the Advanced Launch icons can be found [here](https://dexterous-hand.readthedocs.io/en/master/user_guide/1_2_10_icons_for_hand.html#shadow-advanced-launchers).

Q: How does the NUC fit in with secondary development for the Hand?

A: The NUC only handles low level communication with the Hand as it is only running the driver. If you would like to do your own development with the Hand, continue doing so on the laptop and your scripts will be able to control the Hand. If you would like to change something on the NUC side, the icon called "Launch NUC Container" (within the Shadow Advanced Launchers desktop folder) will give you quick access to the NUC and provide a terminal with full access to the driver.

Q: Why do you use Docker?

A: Using Docker helps us with customer support and development as we are able to replicate our customers' exact software environment if problems arise. If you would like to start the container within Docker without starting the whole system, this can be done using the "1 - Launch Server Container" and "Launch NUC Container" icons within the Shadow Advanced Launchers desktop folder. More information on the Advanced Launch

icons can be found [here](https://dexterous-hand.readthedocs.io/en/master/user_guide/1_2_10_icons_for_hand.html#shadow-advanced-launchers).

Q: Can I install Shadow Software using my own method?

A: We officially only offer support for installing the software using Docker and our Aurora script to avoid migration issues and other difficulties. The `rosinstall` file that we use to create the docker image can be found [here](https://github.com/shadow-robot/sr-build-tools/blob/master/data/shadow_robot-melodic.rosinstall).

If you have any questions about the installation one-liner, please contact us by sending an email to support@shadowrobot.com.

Q: Sometimes when I plan a trajectory using MoveIt! in Rviz, the Hand doesn't move.

A: This is often due to the sensors getting confused about their start state. Simply try trajectory planning again, or power cycle the Hand.

Q: I am having trouble connecting to the NUC, or am receiving errors to do with the DHCP server.

A: This is a known error that has been resolved in our latest software releases. In order to integrate these changes, please run the latest Aurora command following your Delivery Instructions. If you have any questions, please contact us at support@shadowrobot.com.

Common Error Messages

Error: EC slave 1 not in init state

Power cycle the Hand. This error will not affect the performance of the system.

WARNING: disk usage in log directory [...] is over 1GB.

A: This is just explaining that the logs are taking up a lot of disk space. Use the 'rosclean' command to clear these if you like.

Other Questions

Q: I would like to organise a second training session for some of my team that weren't able to make it to the first one. Is this possible?

A: We will likely be able to give you a second introduction to the system or organise a meeting to answer any further questions you may have. Please contact alex@shadowrobot.com to organise a time that works for you.

Q: I have bought multiple Hands from Shadow in the past, but the orders have included different servers and Ubuntu distributions. Why is this?

A: We sometimes change hardware suppliers if they are not meeting our lead time or spec requirements. We ensure that all of the servers and NUCs we supply are of high enough spec to work well with our software. We update the Ubuntu and ROS distributions we use to make use of the most up to date software available to us, and maintain compatibility.

Changelog

ROS Noetic

Version 1.0.18 (Aurora 2.1.5) - current noetic-release * Fixing demo behaviour when tactile sensors are installed

Version 1.0.17 * Update repository with `sr_hand_config` * Fix handling of active rosbags * Remove default hand serial parameter * Fixing bug config file pid parameters being erased when saving selected * Rounding up values for joint slider

Version 1.0.16 * Fixed linter errors * Improve realtime publisher fast pid divisor * Fix joint position/velocity filter * Fix broken rosbags * Increase wait for joints_states message timeout on TeachMode * Fixing arm only launch * Added a System Health Node * Removed incorrect error message

Version 1.0.15 * Improve realtime publisher fast pid divisor * Fix j0 pos vel filter * Added getter for hand trajectories * Fix broken rosbags * Supporting workspaces without UR components * Changed default version values

when launching hand in simulation * Fixing teachmode for different hand types * Fixing arm only launch * Increasing timeout time in teach mode node * Removed incorrect error message * Fixing access modifiers * Fixed linter errors

Version 1.0.14 (Aurora 2.1.4) - previous noetic-release * No changes (release testing image)

Version 1.0.13 * Adding more time to sleep to reload params * Added getter for hand trajectories * Do not require ur_description unless it is needed * Changed default version values when launching hand in simulation * Fixing teachmode for different hand types * Fixing access modifiers

Version 1.0.12 * Add aws manager test * Update shadowhands_prefix.srdf.xacro * Adding “first finger point” to named hand states * Update arm and hand examples

Version 1.0.11 * Update demo

Version 1.0.10 * Fix calibration loader * Fixing demo for left hand * Fixed linter errors * Adding bimanual support to data visualizer

Version 1.0.9 * No changes (release testing image)

Version 1.0.8 * No changes (release testing image)

Version 1.0.7 * Updated AWS Manager to allow for subfolders * Fix error with decoding git commands in ws_diff * removed unused imports from sr_ur_arm_calibration_loader.py * Removing sr_config repo * Fixing shebang and file saving in Hand Health Report * fix building error: This package requires sr_visualization_icons to build, and this patch fixes it * Solve bug Unfiltered position and force traces not shown * Update warnings in RQT * Adding serial number to FingertipVisualizer plugin

Version 1.0.6 * Removed roswrapper from launch files using Autodetect * Fix missing use namespace EigenCompiling packages for the *ros-o* initiative * Fixed mistake in file change_controllers.py * Delete sr_teleop_polhemus_documentation_server.py * Removed ros files for sr_teleop_polhemus_documentation

Version 1.0.5 * Changing default vaules of fingertip sensors srhand.launch

Version 1.0.4 * Xacros refactored * Remove obsolete scoped_ptr * Switching to new xacros * Fixing bugs in launch-files * Adding return to plan executions * Removing box from arm without hand and bimanual system without hands * Deleted sr_box_ur10_moveit_config folder * Refactor robot commander test * Removing sr_hand_dep * Removing deprecated field from general_info * Fix phantom hand * Removed old launch file with box and replaced with the new one from sr_interface * Support ImageMagick 6 and 7 * Hand side fix error

Version 1.0.3 * Migrating to dae and adding materials * Fixing the color of wrist mesh * Switching to new xacros * Update arm related arguments in sr_robot_launch * Adding a way of exiting the demo * Edit tactile threshold * Showing allowed options for general info template * Re-write data visualizer

Version 1.0.0 (Aurora 2.0.0) - previous noetic-release * Integrate UR driver from upstream * Refactoring sr_description: adapted test and added more parameters validation * Create trajectory command publisher utility class * Migrate controls and calibrations * Fixing wrist controller spawning and updating/cleaning up controller spawner script and docs. * Add voice feedback to voice controller * Listen to topics to detect speaker/microphone changes * Replace PyDub library with a direct call to ffmpeg * Adding republish tf new place * Integrate UR driver from upstream * Updating tf republisher * Adding collision scene for filling line * Add hybrid controller argument to more launch files * Removing external control option for sim * Removing sr_config references * Fix robot_commander test in AWS * Make wrist trajectory controller it's own entity * Integrate ur driver from upstream * Fixing scene spawning * Xacro package changed, now needs a function call to setup file stack for error reporting * Fixing controllers for hand lite * Fixing movegroup controller problem * Fix planning errors * Fixing wrist controller spawning * Fixing wrist controller spawning. * Fix __kinematics * Loading analyzers from new place * Migrate controls * Migrate calibrations * Loading rates from a new place * Deprecating sr config * Migrate controls * Migrate analyzers * Migrate calibrations * Migrate rates * Fixed the calibration for both lph and rph. * Integrating auto-detection * Fixing errors when changing controllers and resetting joint sliders

Version 0.0.18 * Update rviz_motor.launch * Fixed Relative path * Add hybrid controller configuration files * Load hybrid controller configuration * Remove redundant aws manager * Removing hand detector * Move sr_world_generator from common_resources to sr_tools * Add world & scene for XPrize competition * Fixed

aws_manager * Enhancing cond delay tool * Prepare the piezo driver to work with multiple dev-kits * simple executable ros wrapper * fixing the tests * Integrated autodetection * Add hybrid controller argument to more launch files * Removing robot description * Adding configs for clients in noetic * Move sr_world_generator from common_resources to sr_tools * Added missing resource and uis install for sr_data_visualization * Removing muscle rqt plugins * Added missing resource and uis install for sr_data_visualization * Removing grasp controller from plugins

Version 0.0.17 (Aurora 1.1.8) - previous noetic-release

- Update tactile_receiver.py
- Move conditional delayed rostool to src and add launch prefix for launching nodes
- Load hand trajectory controller for hand in sim use case
- Adding trajectory controllers for bimanual
- B revert wrist in arm controller move group fix

Version 0.0.16

- Robot commander fix

Version 0.0.15

- Adding new xacro for a hand extra lite with only two fingers mf and th
- Limiting sim speeds to 1.0, now that CPUs are fast enough.
- Fixed linter error in hpp file
- Dixed linter errors in hpp files

Version 0.0.12

- Update simple_transmission.hpp
- Revert “SRC-4962 Move controller switching to CPP (#647)”

Version 0.0.11

- Fixing SrRobotCommander

Version 0.0.10

- Adding hybrid file
- F#src 6473 handle 0 in git revision
- SRC-6470 Release noetic dexterous hand image
- SRC-4962 Add changes from teach_mode_node
- SRC-6063 Don't busy wait for params
- Changing to correct launchfile
- Adding prefix to ur10e yamls
- F#src 6509 optimise arm unlock noetic
- F#src 6509 optimise arm unlock
- SRC-4962 Use helper class from common_resources
- F#src 6477 sr ur arm unlock test noetic
- SRC-4962 Move controller switching to CPP
- initial commit for mock ur dashboard server

- Adding arm servo noetic
- SRC-6177 Fix little finger error reporting
- Integrating hybrid controller
- fixing noetic
- SRC-6470 Release noetic dexterous hand image
- Fixing bootloader path with casting to string

Version 0.0.9

- F#src 6509 optimise arm unlock noetic
- F#src 6509 optimise arm unlock
- Fixing bootlo* ader path with casting to string

Version 0.0.8

- F#src 6473 ha* ndle 0 in git revision
- SRC-6470 Rele* ase noetic dexterous hand image
- Adding prefix to ur10e yamls

Version 0.0.7

- SRC-6470 Rele* ase noetic dexterous hand image

Version 0.0.6

- Fixed deprecated .mesh
- F#98 modular * xacros
- SRC-6467 Intr* oduce git_revision field in GenericTactileData
- Update demo_r* .py
- Src 6413 create a collision model for the rack
- add only stan* s
- B fixing watchdog test
- F fixing speech control
- SRC-6470 Release noetic dexterous hand image
- SRC-6301 Implement reading of MST sensors
- Update package.xml

Version 0.0.5

- fix pedal bug
- B pedal restart fix

ROS Melodic

Version 0.0.62 (current melodic-release)

- Improving saving utility for Noetic
- Fixing yaml load
- Adding respawn

- Fixed calibration loader
- Automatic calibration loader not working in URSIM
- Adding missing arguments
- SRC-6043 Remove unused 'rename' arguments
- Adding kill node script
- SRC-5239: Adding speech control
- SRC-6183 Add __init__.py file
- SRC-6183 Various improvements for speech control
- Fixing yaml load
- arms braking
- fix home
- removing the required flags
- Fix_an_arm_and_hand_xacro
- Adding x and y separations to launch and xacros
- changing jiggle fraction default value
- Update sr_ur_arm_unlock
- fix syntax error
- Automatic calibration loader not working in URSIM
- Publish underactuation error
- Fixing srdf generation and saving of file
- Fixing yaml load
- improving hand and arm rostest
- Commenting trac_ik and replacing it to kdl until it is available in Noetic
- updating unimanual y separation
- Fix pedal reset for protective stop
- Add new driver for teleop pedal
- Update 90-VEC-USB-Footpedal.rules

Version 0.0.61

- Fix pedal reset for protective stop

Version 0.0.60

- Improving saving utility for Noetic
- Fixing yaml load
- Adding missing arguments
- Remove unused 'rename' arguments
- Adding kill node script
- Adding speech control

- Add `__init__.py` file
- Various improvements for speech control
- Fixing yaml load
- Publish underactuation error
- Fixing srdf generation and saving of file
- Fixing yaml load
- improving hand and arm rotest
- Commenting `trac_ik` and replacing it to `kdl` until it is available in Noetic

Version 0.0.58

- Changing paramiko version to 2.7.2
- Adding respawn
- Merging kinetic-devel back to melodic
- Fixed calibration loader
- Fixed arm and hand xacro
- Automatic calibration loader not working in URSIM
- Fixing orientation for left arms
- Fixing xacro
- Hand and arm test
- Arms braking
- Fix home
- Removing the required flags
- Updating unimanual y separation
- Adding X and Y separations to launch and xacros
- Changing jiggle fraction default value
- Update `sr_ur_arm_unlock`
- Fix syntax error
- Fix data visualization bug
- Add new driver for teleop pedal
- Update 90-VEC-USB-Footpedal.rules

Version 0.0.57 (previous melodic-release)

- Merging kinetic-devel back to melodic
- Fixing orientation for left arms
- Fixing xacro for `sr_multi_description/urdf/right_srhand_lite_ur10e.urdf.xacro`
- Adding hand and arm tests in robot launch
- Fix data visualization plugin bug

Version 0.0.56

- Add wait for robot description in sr_robot_launch/launch/sr_ur_arm_box.launch
- Plotjuggler v3

Version 0.0.55

- Update calibration GUI

Version 0.0.54

- Fetch arm ips from param server
- fixing set_named_target method in robot commander

Version 0.0.53

- Fix for hand finder overwriting urdf joints with all joints
- Add default to launch arg list
- Delete pull_request_template.md
- Adding wait to watchdog
- Fixing home angle arg in sr_robot_launch files
- Updating worlds and scenes to bimanual
- Adding the planning group two_hands
- Updating state saver for more options

Version 0.0.52

- Delete pull_request_template.md
- Fix for hand finder overwriting urdf joints with all joints
- Add default to launch arg list in conditional delay

Version 0.0.51

- Update sr_bimanual_ur10arms_hands.launch
- Adding start state to stored states
- Update planner to BiTRRT
- Modify parameter to load robot description at this level only if requested

Version 0.0.50

- Demohand a with ur10e updated

Version 0.0.49

- Adding hybrid controller arbitrary frame
- Removing exclude wrist from controller spawner
- Removing include_wrist_in_arm_controller param
- Adding planning quality to examples
- Adding scripts and documentation for in-docker leap motion running
- Bimanual demohands a d changes
- wrist mimic rostop
- Fix left arm scene

- add sr_robot_msg dependency

Version 0.0.48

- Created bimanual xacro for hand lites biotacs

Version 0.0.47

- Fixed hybrid controller installation and controller spawner
- Tests for the scene

Version 0.0.46

- Added hybrid controller
- Added a xacro for shadow hand left lite with biotacs
- Fixed install of ros_heartbeat
- Updated aurora instructions to specify ethercat_right_hand rather than ethercat_interface
- Fixed conditional roslaunch (added extra conditions)
- Adding and updating hand ROS tests
- New scene and world for MS lab
- add cpp wait for param
- updating open hand demo for smoother opening

Version 0.0.45

- Added stand to simulation
- Updated README
- adding additional check

Version 0.0.44 (previous melodic-release)

- Created /run/user/1000 folder inside the docker container (to fix rqt graphics issue)

Version 0.0.43

- Local hw interface and fixed do switch with centre of gravity

Version 0.0.42

- Updated README.md

Version 0.0.41

- Fixed and added files to make the ur5e with box work and generify the launch file
- Added metapackage

Version 0.0.40

- Updated sr_system.launch
- Added full hand ur5e support
- Added ur5e normal hand configs

Version 0.0.39

- Shadow glove GUI updated and moved

Version 0.0.38

Features:

- Updated calibration GUI

Version 0.0.37

Features:

- Tone down UR10e tuning so the arm behaves more smoothly

Version 0.0.35

Features:

- Fix hand control parameter error in setting the payload for UR arm

Version 0.0.34

Features:

- Update motor effort file for left hand
- Add relay node with tcp_nodelay param
- Hand + UR arm: allow setting cog and payload
- Use Shadow's fork of universal robot reposeritor
- Fix biotac visualizer for bimanual
- change yaw roll, adjust formulas after real hw testing
- Fix sensor manager file

Version 0.0.33

Features:

- Changing expected delimiter from newline to '_' in arm firmware checker
- Adding x and y separation for left bimanual arm config

Version 0.0.32

Features:

- Set arm IP defaults to new values (10.8.1.1 and 10.8.2.1) and also added a comment about aurora using sed to replace these IPs
- Changed hand mapping path default to v4
- fix for arm in safety violation mode
- second try at adding ur10 config, minimal changes
- Fixing controller spawning bug in which WRJ1+2 would not work when wrist was included in arm trajectory control
- Fixing controller spawning bug in which WRJ1+2 would not work when wr
- Updating calibration gui

Version 0.0.31

Features:

- Fixed bug in Dexterity Test that stopped hand moving to the correct poses.
- Fixed bug in the Bimanual launch files to load correct planning groups.

- Mujoco ur hand
- Fix ur box
- Fixing bug wherein conditional delay script would count found parameter
- Adding gui for shadow glove calibration
- Moving hand meshes to a more standard path to make gzweb work
- parsing hand sides
- remove user choice, add conditional delay
- arm calibration loader 2
- Adding wrapper script for autodetecting shadow hands

Version 0.0.30

Features:

- Fixed bug in RQT Data Visualiser that stopped other plugins from plotting

Version 0.0.29

Features:

- Config and xacro for hand lite ur10e
- Fixed bug with ur_arm_release
- Fixed conditional delay bug in sr_interface

Version 0.0.28

Features:

- now correctly handles exception
- config and xacro for hand lite ur10e
- Adding support for ur5e and hand lite
- fixing error message

Version 0.0.27

Features:

- adding hand mapping v4 files
- enable ft sensor on ur e robots
- adding la_ur10e_with_box xacro
- fixed sr_hardware control loop bug
- Adding scene and world for ms garage
- Update sr_ur10arm_box.launch
- adding mapping v4
- Fixing args being limited to group scope
- Restoring **arm_** and hand_ctrl control loop arguments to the previous f
- Adding mock triple pedal
- Fixing intermittent bug in controller spawning

- Updating real time TF republisher for more flexibility
- adding ur10e with box yaml files

Version 0.0.26

Features:

- Updated controller spawner
- Replaced delay roslaunch with conditional roslaunch

Version 0.0.24

Features:

- Fixed an issue where the config files did not contain a robot_config_file parameter, preventing launch
- Fixed an issue where robot_description was not found for the NUC setup
- Fixed an issue preventing the effort controllers to launch

Version 0.0.20

Features:

- Fixed an issue where the hand Demo did not recognise Demo Hand D had biotacs

Version 0.0.17

Features:

- Fixed a hand serial issue with launching bimanual hands locally without a NUC

Version 0.0.16

Features:

- Fixed an issue in Rviz displaying left and right hands in the same location without separation when NUC with external control loop is being used

Version 0.0.15

Features:

- Fixed an issue in Gazebo9 not displaying the forearms of the hands properly
- Fixed an issue in Rviz displaying left and right hands in the same location without separation

Version 0.0.14

Features:

- Enabling the bimanual hands only system (no arms) to be run on NUC with external control loop

Version 0.0.13

Features:

- Fixed deprecation errors for melodic
- Added bimanual with no hands to sr_robot_launch

ROS Kinetic

Version 1.0.53 (current kinetic-release)

Features:

- Fixed an issue with Moveit trajectory planning in the Bimanual setup

Version 1.0.52

Features:

- Fixed a hand serial issue with launching bimanual hands locally without a NUC
- Fixed an issue with launching left or right hand locally without a NUC for ROS Kinetic

Version 1.0.51

Features:

- Fixed an issue in Rviz displaying left and right hands in the same location without separation when NUC with external control loop is being used

Version 1.0.50

Features:

- Fixed a bug causing incorrect launch of unimanual left hand in NUC external control loop for ROS kinetic only

Version 1.0.49

Features:

- Fixed an issue in Rviz displaying left and right hands in the same location without separation

Version 1.0.48

Features:

- Enabling the bimanual hands only system (no arms) to be run on NUC with external control loop

Version 1.0.45 (current kinetic-release)

Features:

- Allows Hand control from the NUC
- UR firmware check on docker startup
- New thumb calibration
- Launch files updated

Version 1.0.38

Features:

- Supports using an external control loop (in a NUC) to launch: hand only, arm only, hand+arm
- If an arm is connected, there is an automatic arm firmware compatibility check
- Automatic compatibility check of the Docker Image and hand firmwares

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

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
- Ability to save ROS logs by clicking on an icon on the desktop

Version 1.0.2

- Initial version

SUPPORT AND TEAMVIEWER

16.1 Support Contact

If you have any questions about your Shadow product or require assistance, please contact us by sending an email to support@shadowrobot.com.

16.2 Teamviewer

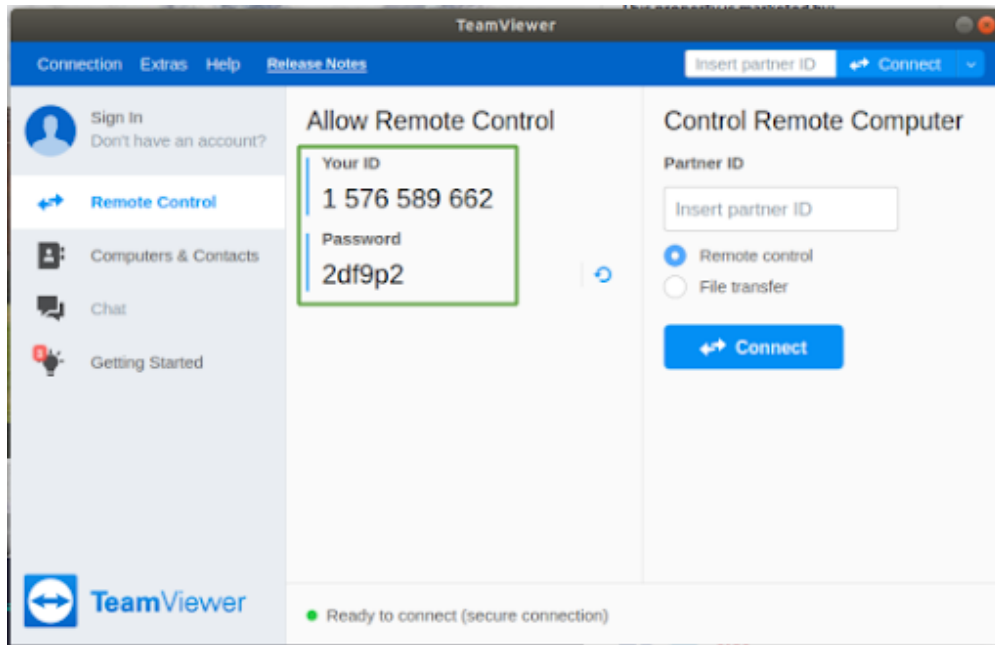
Teamviewer is software that allows for remote control and desktop sharing between computers. It can be a useful tool for debugging so that Shadow can see exactly what is happening on the server and help to solve any issues. We now install it on servers supplied by Shadow, but it is easily downloaded here:

<https://www.teamviewer.com/en/>

1. Open Teamviewer by clicking this icon on the taskbar.



2. A window will open which displays your Remote Control ID and Password. These credentials will change each time you use Teamviewer so make sure to share the most up to date information.



3. Once you share these details with a member of our support team, they will have access to control your computer remotely. Please note you must have a working internet connection for Teamviewer to work. They will be able to see your desktop and any programs you have running, and have control over the mouse. You **will not lose control** and can interrupt the session at any time by closing the Teamviewer application.

Shadow Backup USB stick

You can only follow these steps if you have been provided with a Shadow Backup USB stick. It has a 500 MB Clonezilla partition and a several GB for Clonezilla disk images. These can be used to restore any Shadow-provided laptop(s) and/or NUC(s) to their exact configuration (it's a full disk image restore) after the delivery was tested in Shadow's offices but before it was shipped to the customer.

Doing this means losing all the data on the laptop(s) and/or NUC(s) since the original shipment of the equipment from Shadow. Please back up your data to an external device before restoring any Shadow backups

If you are restoring a laptop backup, you only need the Shadow backup USB stick.

If you are restoring a NUC backup, you will need additional equipment (not provided by Shadow): A monitor with HDMI connector, an HDMI cable and a USB keyboard.

The steps for restoring a NUC with Shadow Backup USB stick are the same as for laptop, but you have to choose a different image to restore from the relevant list in the Clonezilla process.

You don't have to restore both NUC and laptop. You can choose to restore just 1 of them.

Clonezilla backup restore steps (here device means either laptop or NUC)

1. Connect the Shadow USB stick to the device you want to restore
2. Power on the device while tapping F7 and F8 on the keyboard. A blue dialog box will appear asking you which disk to boot from (if this doesn't work, try tapping F2/F12 and accessing the boot menu/boot override via BIOS): select the Shadow backup USB 500 MB Clonezilla partition (it might show up as a SanDisk device, or a similar brand matching the model of the USB stick)
3. A Clonezilla window will appear

```
`eval_rst .. image:: ../img/clonezilla_1.png`
```
4. Select Clonezilla live (Default settings, VGA 1024x768)

5. In the next menu, select English for the language: (just press Enter)

```
`eval_rst .. image:: ../img/clonezilla_2.png `
```

6. In the next screen, the default keyboard layout is US keyboard, and it's fine for our purposes here, so just press Enter

```
`eval_rst .. image:: ../img/clonezilla_3.png `
```

7. Select Start Clonezilla

```
`eval_rst .. image:: ../img/clonezilla_4.png `
```

8. Choose device-image

```
`eval_rst .. image:: ../img/clonezilla_5.png `
```

9. Choose local device:

```
`eval_rst .. image:: ../img/clonezilla_6.png `
```

10. Press enter when you see this screen:

```
`eval_rst .. image:: ../img/clonezilla_7.png `
```

11. Press Ctrl-C when you see this screen:

```
`eval_rst .. image:: ../img/clonezilla_8.png `
```

12. You should see a disk menu like this (not the exact image), but you need to select your Shadow Backup USB stick large partition with several GB (where the disk images are) (it might show up as a SanDisk device, or a similar brand matching the model of the USB stick, but with several GB of space)

```
`eval_rst .. image:: ../img/clonezilla_9.png `
```

13. In the folder selection screen below you should see 2 folders for the clonezilla images for NUC and laptop, don't select them, just select Done

```
`eval_rst .. image:: ../img/clonezilla_10.png `
```

14. Choose Beginner mode

```
`eval_rst .. image:: ../img/clonezilla_11.png `
```

15. Choose restoredisk option

```
`eval_rst .. image:: ../img/clonezilla_12.png `
```

16. Now is the time to select whether you want to restore a NUC image or a laptop image. Depending on which device you have connected the Shadow backup USB stick to, select either the NUC image (may be labelled with your customer name and NUC or NUC-CONTROL and 256GB) or the laptop image (may be labelled with your customer name and LAPTOP or SERVER and 500 GB)

```
`eval_rst .. image:: ../img/clonezilla_13.png `
```

17. Just press enter on the destination disk:

```
`eval_rst .. image:: ../img/clonezilla_14.png `
```

18. Select "No, skip checking the image before restoring"

```
`eval_rst .. image:: ../img/clonezilla_15.png `
```

19. Select -p choose

```
`eval_rst .. image:: ../img/clonezilla_16.png `
```

20. Press enter:

```
`eval_rst .. image:: ../img/clonezilla_17.png `
```

21. There will be 2 prompts and you have to press y and press Enter to each of them

```
`eval_rst .. image:: ../img/clonezilla_18.png `
```

22. Clonezilla is now restoring the device image:

```
`eval_rst .. image:: ../img/clonezilla_19.png `
```

23. It should take about 20 minutes or less. When Clonezilla is done, press Enter:

```
`eval_rst .. image:: ../img/clonezilla_20.png `
```

24. Choose poweroff

```
`eval_rst .. image:: ../img/clonezilla_21.png `
```

25. Unplug the Shadow Backup USB stick from the device

26. Power on the device. The device is now restored.

ABBREVIATIONS

| Abbrevia-
tion | Meaning |
|-------------------|--|
| ADC | Analogue to Digital Converter. A chip which reads an analogue voltage signal. |
| ASIC | Application Specific Integrated Circuit. |
| CAN | Controller Area Network. A 1Mb peer-peer network for vehicles and robots. |
| CPU | Central Processing Unit. A computer processor which runs software. |
| CRC | Cyclic Redundancy Check. An algorithm (or output of) for checking the correctness of incoming data. |
| DAC | Digital to Analogue Converter. A chip which produces an analogue voltage signal. |
| EC | EtherCAT |
| EEPROM | Electrically Erasable Programmable Read Only Memory. |
| FPID | Feed-forward Proportional Integral Derivative controller. An algorithm used to control something on a robot. |
| GUI | Graphical User Interface. |
| I/O | Input / Output |
| ICD3 | In Circuit Debugger 3. A tool used to change the firmware on a PIC microcontroller. |
| IMU | Intertial Measurement Unit. Sensors used to measure acceleration and rotation. |
| LED | Light Emitting Diode. A small coloured light. |
| LVDS | Low Voltage Differential Signal. One of two possible electrical physical layers supported by EtherCAT. |
| mA | milliamps. A unit of electrical current. |
| MCU | Micro Controller Unit. A small, usually embedded, CPU. |
| MIPS | No longer an abbreviation; a word in its own right. A brand of CPU / MCU. |
| Nm | Newton Meters. A unit of torque. |
| PC | Personal Computer. |
| PCB | Printed Circuit Board. |
| PID | Proportional Integral Derivative controller. An algorithm used to control something on a robot. |
| PST | Pressure Sensor Tactile. A simple tactile sensor offered as standard. |
| PWM | Pulse Width Modulation. Digital method used to emulate an analog signal. |
| ROS | Robot Operating System, created by Willow Garage. |
| SPI | Serial Peripheral Interface, allowing an MCU to communicate with an ADC. |