Shadow Robot

Dexterous Hand Series

User Manual

Date of issue: **January 23, 2024**

Release Version: **latest**

Overview

1

Preface

This is the starting point for the Shadow Dexterous Hand Documentation. The Shadow Dexterous Hand is an advanced humanoid robot hand system that provides 24 movements to reproduce as closely as possible the kinematics and dexterity of the human hand. It has been designed to provide comparable force output and movement precision to the human hand. Shadow Hand systems have been used for research in grasping, manipulation, neural control, brain computer interface, industrial quality control, and hazardous material handling.

The Shadow Dexterous Hand is a self-contained system - all actuation and sensing is built into the hand and forearm. All versions of the Hand use an EtherCAT bus (Ethernet for Control Automation

Technology), providing a 100Mbps Ethernet-based communications field-bus, and full integration into ROS (Robot Operating System).

The Hands use Shadow's electric "Smart Motor" actuation system and integrates force and position control electronics, motor drive electronics, motor, gearbox, force sensing and communications into a compact module, 20 of which are packed into the Hand base.

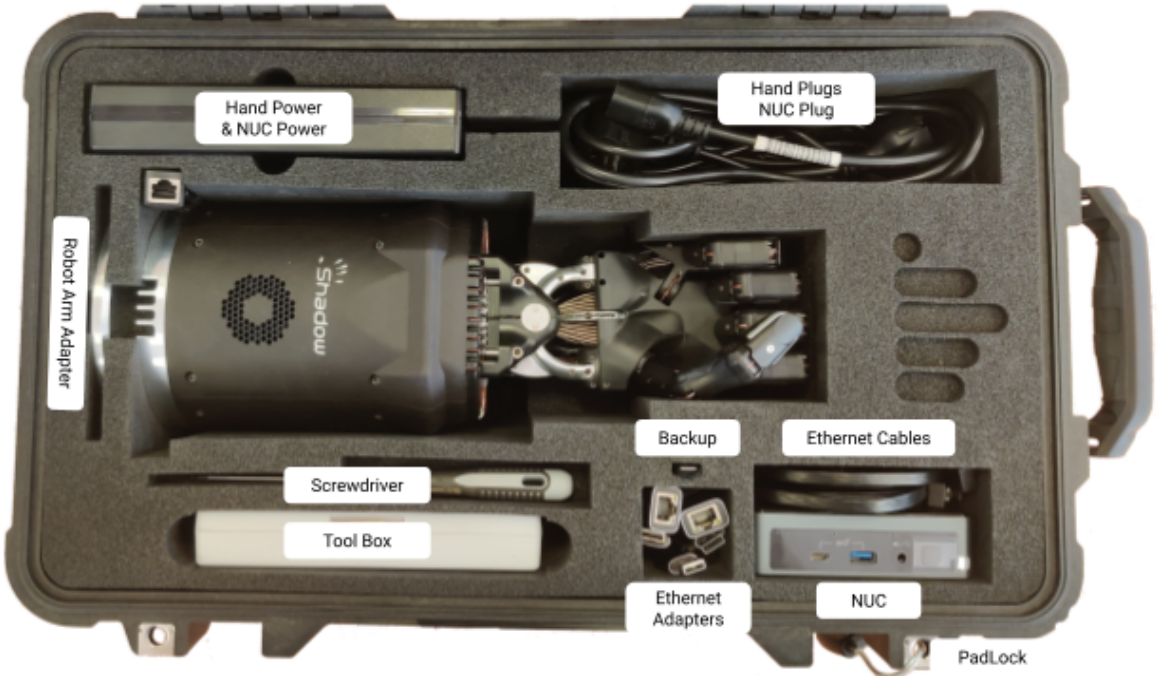
2

Overview

- *Hand Peli Case Contents*
- *Laptop Box Contents*
- *Abbreviations*

2.1. Hand Peli Case Contents

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



Equipment	Quantity	
	Unimanual	Bimanual
Dexterous Hand	1	2
Hand Power External Cable	1	2
Hand Power Supply	1	2
Hand Power Supply lead according to destination country	1	2
Mounting plate (to connect the hand to UR arm)	1	2
Mounting Plate screws for mounting plate	4	8
Mounting Plate screws for the hand	8	16
3m Ethernet cable	2	4
“NUC-CONTROL” USB-Ethernet adapter	1	1
“SERVER” USB-Ethernet adapter	1	1
“RIGHT/LEFT HAND” Labelled USB-Ethernet adapter	1	2
NUC labelled SHADOW NUC-CONTROL	1	1
NUC PSU	1	1
NUC PSU power lead according to destination country	1	1
Toolbox (contains hex drivers to robot maintenance)	1	2
Cut allen key (inside the Toolbox)	1	2
Allen key (inside the Toolbox)	1	2
Spooltool (inside the Toolbox)	1	2
Luggage locks for peli case	2	4
“Shadow Backup” USB w. Clonezilla images of NUC and server	1	1
Hand Delivery Instructions	1	1

2.2. 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
Power adapter for destination country	1

2.3. Abbreviations

Abbreviation	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.

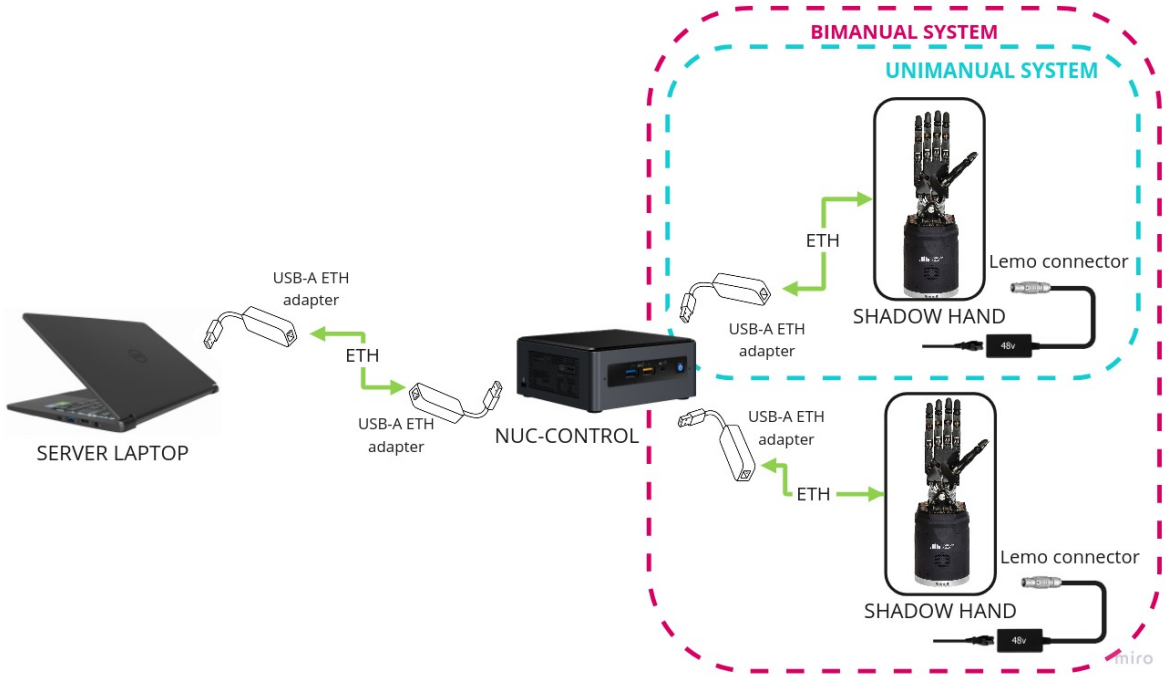
3

Setting up the hand

- *Connecting Cables*
- *Desktop Icons For The Hand*
- *Mounting The Hand To An Arm*
- *Launching The Hands*
- *Lights On The Hand*
- *Uploading Log Files To Our Server*

3.1. Connecting Cables

Note: 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.



- 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.
- 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.
- 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.

3.1.1. 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

3.2. Desktop Icons For The Hand

The following icons will be available on the server laptop desktop for launching and controlling the right, left and bi-manual hands. These icons will be located inside a folder named after your container name on the desktop. You might need to right Click with your mouse and select ‘Allow Launching’.

When running a bimanual system, due to all the combinations of icons that are possible we split the icons into sub-folders, to make it easier to select the right icons.

3.2.1. Main desktop icons

Icon text	Icon explanation
Shadow ROS Logs Saver and Uploader	Saves ROS logs from server and NUC, uploads them to Shadow servers and emails them to Shadow
Launch Shadow Right Hand	Launches the right hand (container, ROS core, NUC hardware control loop, server GUI)
Launch Shadow Left Hand	Launches the left hand (container, ROS core, NUC hardware control loop, server GUI)
Launch Shadow Bimanual Hands	Launches the both hands (container, ROS core, NUC hardware control loop, server GUI)
Shadow NUC RQT	Once the hand has been launched, this icon starts ROS RQT inside the NUC’s docker container
Dexterous Hand Documentation	Opens online documentation if internet connected or offline documentation if no internet
Shadow Close Everything	Cleanly stops ROS processes, closes containers, and closes all Shadow related terminals

3.2.2. Shadow Demos folder

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

Icon text	Icon explanation
Close Left Hand	Once the hand has been launched, this icon will close (pack) the left hand
Close Right Hand	Once the hand has been launched, this icon will close (pack) the right hand
Close Bimanual Hands	Once bimanual hands have been launched, this icon will close (pack) both hands
Biotac Demo Left Hand/Demo Left Hand	Once the hand has been launched, this icon will run various (tactile/keyboard-actived) left hand demos
Biotac Demo Right Hand/Demo Right Hand	Once the hand has been launched, this icon will run various (tactile/keyboard-actived) right hand demos
Biotac Demo Bimanual Hands/Demo Bimanual Hands	Once bimanual hands have been launched, this icon will run various (tactile/keyboard-actived) bimanual hands demos
Open Left Hand	Once the hand has been launched, this icon will fully open the left hand
Open Right Hand	Once the hand has been launched, this icon will fully open the right hand
Open Bimanual Hands	Once bimanual hands have been launched, this icon will fully open both hands

3.2.3. Simulation folder

The simulation folder gives you icons that automatically launch the hand in simulation mode on your local server.

Icon text	Icon explanation
Launch Shadow Bimanual Hands Simulation	Sets up a bimanual simulation robot with Rviz and Gazebo
Launch Shadow Left Hand Simulation	Sets up a left hand simulation robot with Rviz and Gazebo
Launch Shadow Right Hand Simulation	Sets up a right hand simulation robot with Rviz and Gazebo

3.2.4. 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/Bi-manual 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/Bi-manual Hands Hardware Control Loop
- 4 - Launch Server Right/Left/Bi-manual GUI

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

Unimanual Icons (Either left **or** right hand)

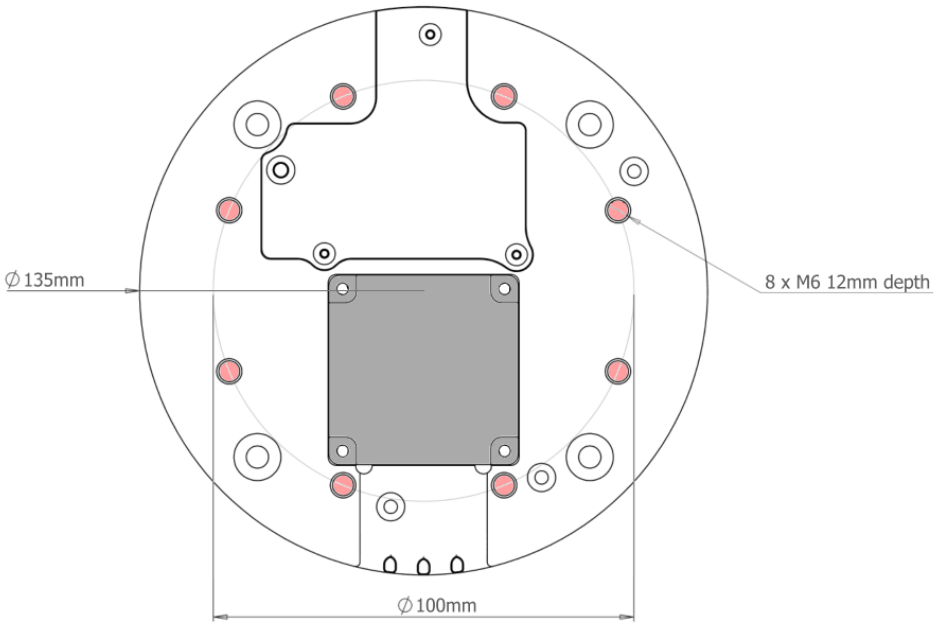
Icon text	Icon explanation
1 - Launch Server Container	Launches the server laptop's docker container
2 - Launch Server ROSCORE	Launches the ROSCORE inside the server laptop's docker container
3 - Launch NUC Container and Right/Left Hand Hardware Control Loop	SSH'es to the NUC, starts its container, and launches the right hand realtime control loop
3 - Zero Force Mode - Right/Left Hand	Launches the right hand (connected to NUC) in zero force mode (fingers can be moved easily)
4 - Launch Server Right/Left Hand GUI	Launches the GUI (Rviz) on server laptop for the right hand
Launch NUC Container	SSH'es to the NUC, starts NUC's container and starts a terminal session inside it
Local Launch/Launch Local Shadow Right/Left Hand	Launches the right hand (connected to server laptop) using the same USB-Ethernet adapter
Local Launch/Local Zero Force Mode - Right/Left Hand	Launches the right hand (connected to server) in zero force mode (fingers can be moved easily)

Bimanual Icons

Icon text	Icon explanation
Right Side/1 - Launch Server Container	Launches the server laptop's docker container
Right Side/2 - Launch Server ROSCORE	Launches the ROSCORE inside the server laptop's docker container
Right Side/3 - Launch NUC Container and Right Hand Hardware Control Loop	SSH'es to the NUC, starts its container, and launches the right hand realtime control loop
Right Side/3 - Zero Force Mode - Right Hand	Launches the right hand (connected to NUC) in zero force mode (fingers can be moved easily)
Right Side/4 - Launch Server Right Hand GUI	Launches the GUI (Rviz) on server laptop for the right hand
Left Side/1 - Launch Server Container	Launches the server laptop's docker container
Left Side/2 - Launch Server ROSCORE	Launches the ROSCORE inside the server laptop's docker container
Left Side/3 - Launch NUC Container and Left Hand Hardware Control Loop	SSH'es to the NUC, starts its container, and launches the left hand realtime control loop
Left Side/3 - Zero Force Mode - Left Hand	Launches the left hand (connected to NUC) in zero force mode (fingers can be moved easily)
Left Side/4 - Launch Server Left Hand GUI	Launches the GUI (Rviz) on server laptop for the
Bimanual/1 - Launch Server Container	Launches the server laptop's docker container
Bimanual/2 - Launch Server ROSCORE	Launches the ROSCORE inside the server laptop's docker container
Bimanual/3 - Launch NUC Container and Bimanual Hands Hardware Control Loop	SSH'es to the NUC, starts its container, and launches the bimanual realtime control loop
Bimanual/4 - Launch Server Bimanuals GUI	Launches the GUI (Rviz) on server laptop for the bimanual hands
Local Launch/Launch Local Shadow Right Hand	Launches the right hand (connected to server laptop) using the same USB-Ethernet adapter
Local Launch/Launch Local Shadow Left Hand	Launches the left hand (connected to server laptop) using the same USB-Ethernet adapter
Local Launch/Launch Local Shadow Bimanual Hands	Launches bimanual hands (connected to server laptop) using the same USB-Ethernet adapters
Local Launch/Local Zero Force Mode - Right Hand	Launches the right hand (connected to server) in zero force mode (fingers can be moved easily)
Local Launch/Local Zero Force Mode - Left Hand	Launches the left hand (connected to server) in zero force mode (fingers can be moved easily)
Launch NUC Container	SSH'es to the NUC, starts NUC's container and starts a terminal session inside it
Launch Server Container	Launches the server laptop's docker container

3.3. 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.

3.4. 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.

When shutting down the hand(s) after use, please make sure to use the *Shadow Close Everything* icon as this closes the software gracefully and also does a final calibration jiggle to ensure the strain gauges are zeroed.

3.4.1. Jiggling

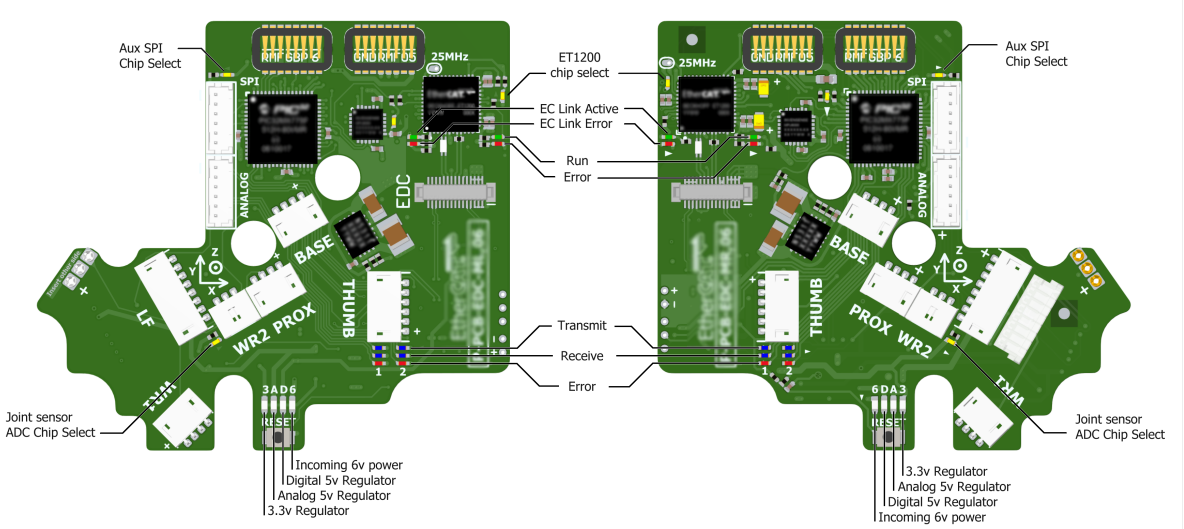
On reset, all of the strain gauges (torque sensors) in the motors need to be zeroed. This happens automatically. The motors are driven back and forth to try to relieve any tension on the tendons. Then both gauges are zeroed. You will therefore see all joints of the hand move slightly on power up or reset.

3.4.2. 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.

3.5. 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

3.6. 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.

4

Simulation

- *Gazebo*

4.1. 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.

4.1.1. Starting a robot simulation

The simulation of the system you have can be launched using the simulation icons found in a folder called “Simulation” in the “Shadow Advanced Launchers” folder. Alternatively you can use the below commands in a terminal:

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

Shadow Dexterous hands

- To start the simulation, you can run:

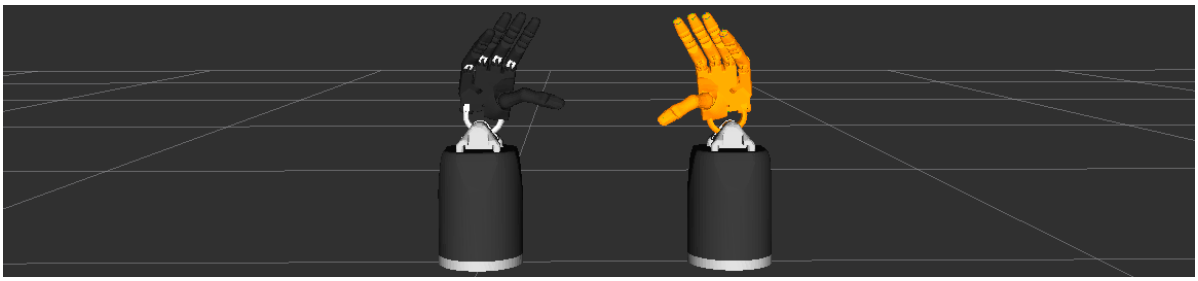
```
roslaunch sr_robot_launch srhand.launch sim:=true
```

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

```
roslaunch sr_robot_launch srhand.launch sim:=true side:=left
```

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

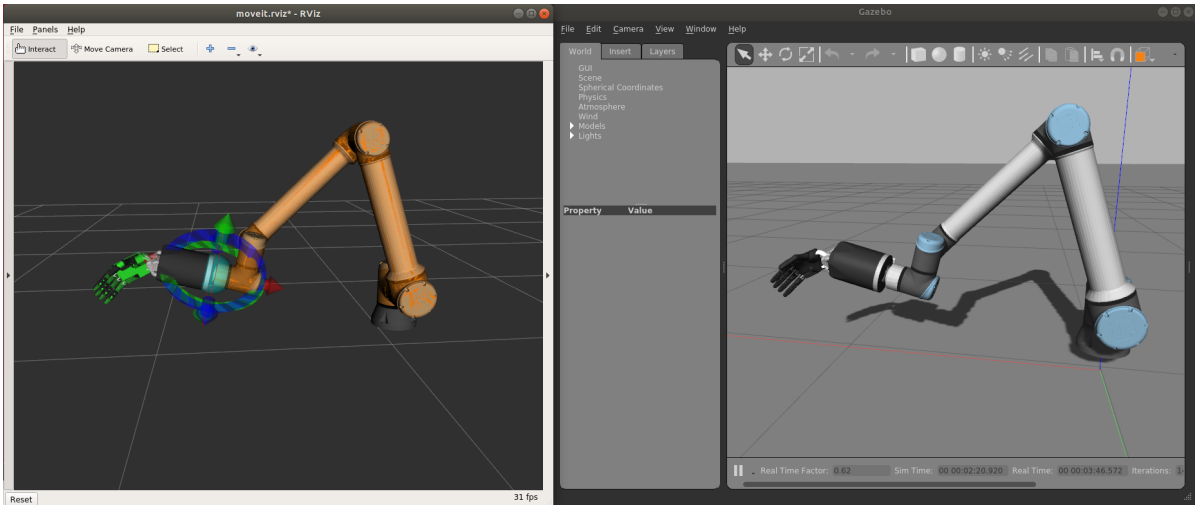
Bimanual hand system



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

```
roslaunch sr_robot_launch sr_bimanual.launch sim:=true
```

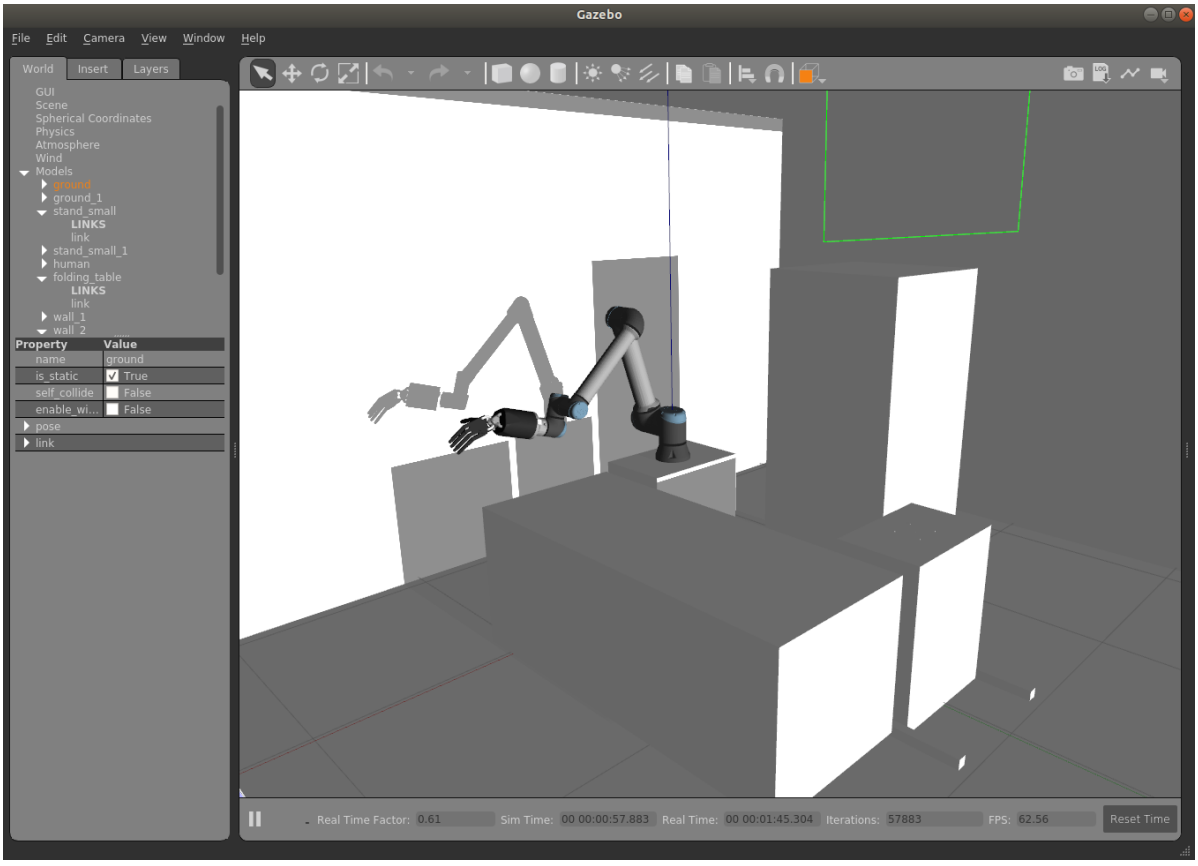
Unimanual arm and hand system



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

```
roslaunch sr_robot_launch sr_right_ur10arm_hand.launch sim:=true
```

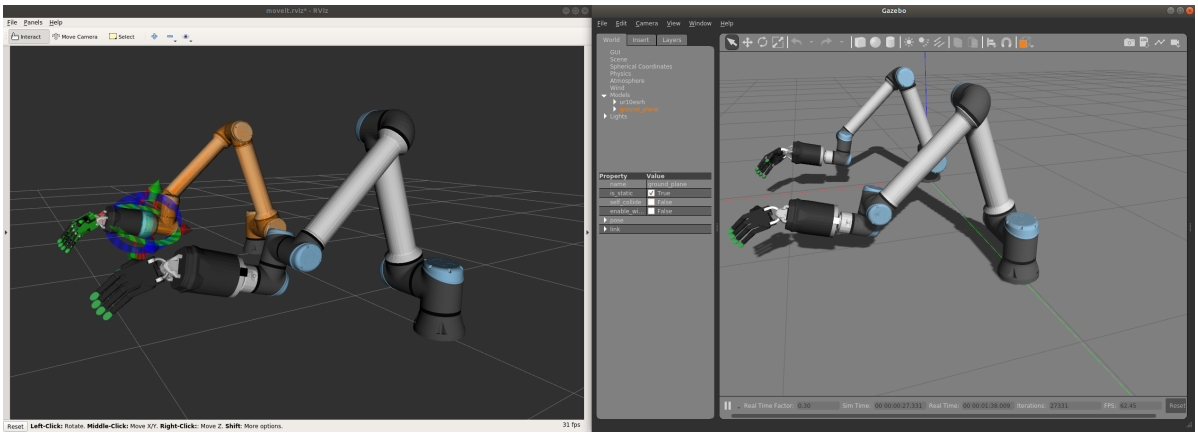
To add a scene, you can add scene:=true and you our default scene. You can also add your own scene adding a scene_file parameter.



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

```
roslaunch sr_robot_launch sr_left_ur10arm_hand.launch
```

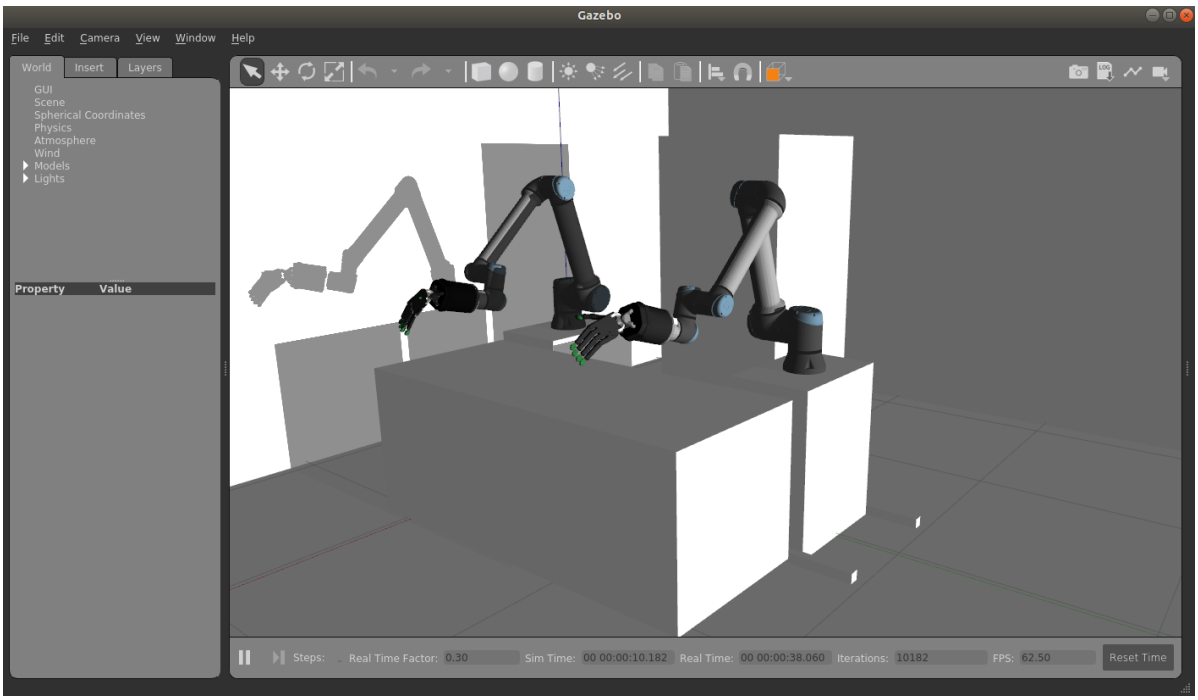
Bimanual arm and hand system



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

```
roslaunch sr_robot_launch sr_bimanual_ur10arms_hands.launch external_control_loop:=false sim:=true␣
↪arm_speed_scale:=0.7 scene:=true
```

To add a scene, you can add scene:=true and you will see our default scene.



You can also add your own scene adding a scene_file parameter.

4.1.2. Installing the simulator in a different computer

Follow these instructions if do not have a real hand but would like to use our hand in simulation or you want to install only the simulator on a different computer.

- ROS Noetic:

```
bash <(curl -Ls https://raw.githubusercontent.com/shadow-robot/aurora/master/bin/run-ansible.sh)␣
↪docker_deploy --branch master tag=noetic-v1.0.29 product=hand_e nvidia_docker=true␣
↪reinstall=true sim_icon=true container_name=dexterous_hand_simulated
```

Additional parameter	Values	Description
product	hand_e, hand_lite, hand_extra_lite	Describes the shadow hand product you want to install.
reinstall	true, false	Flag to know if the docker container should be fully reinstalled.
nvidia_docker	true, false	Define if nvidia-container-toolkit is used. Use with nvidia GPU.
launch_hand	true, false	Specify if hand driver should start when double clicking desktop icon
sim_hand	true, false	If true the icon's will autolaunch hand in simulation mode.
hand_side	right, left	Specify if the hand is right or left (ignored if bimanual=true)
bimanual	true, false	Specify if both hands are used or not.

You can tell if the installation via the one-liner was successful based on it returnining:

```
Operation completed
```

The one-liner will then create a dekstop icon that you can open and use to launch the container. If you did not have the parameter `launch_hand=true` in your one-liner then you can use the commands shown at the top of this page to launch the simulated hand.

More params and their explanation can be found here.

5

Software description

- *First Time Users*
- *Controlling The Hand*
- *Accessing Data From The Hand*
- *Graphical User Interface*
- *Command Line Interface*
- *Repositories*
- *The Robot Commander*
- *Saving States*
- *Hand Autodetection*
- *Robot Descriptions (URDF)*
- *Fingertips*
- *Firmware*

5.1. First Time Users

Our systems work 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

If you are unfamiliar with ROS and intend to use the ROS API, you can find the fundamental ROS concepts explained here and a technical overview of the implementation of ROS here. It is highly recommended that you also check the ROS Tutorials.

If you are unfamiliar with Linux and its 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. You can find more information here.

5.2. Controlling The Hand

5.2.1. Control Modes

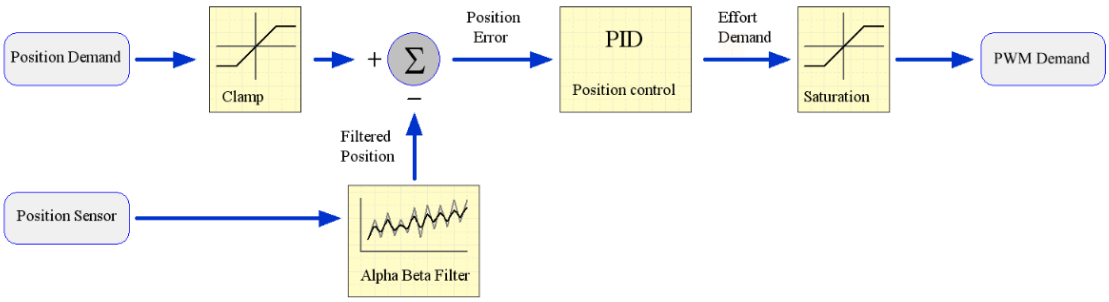
Effort and Torque

ROS uses the concept of effort as something that actuators provide. The word effort is used, rather than torque, because it can be applied to any type of actuator (rotary, linear, pressure, etc.), whereas torque only applies to rotary actuators. Since all motors on the Shadow hand are rotary, we use the words effort and torque interchangeably.

Controller options

The host supports two types of control for the Shadow Hand: torque (effort) control or position control.

Teach mode: No control is implemented on the host. The Effort demand is sent to the motor which implements it using a 5kHz control loop. See *Firmware* for details of the Effort control algorithm.



Position: This uses a PID position controller. The output of the host side PID controller is sent to the motor as a PWM demand. No effort controller is used for position control.

Trajectory: This controller allows the user to define a joint space trajectory, that is a series of waypoints consisting of joint positions. Each waypoint has an associated time. The trajectory controller uses quintic spline interpolation to produce a position target every 1ms, so that the position control loop for each joint runs at 1KHz. This allows the user to define a smooth trajectory and control the speed of the joint.

5.2.2. 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.

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.

5.2.3. Deeper settings

Editing PID settings

The motor controller PID settings are stored in YAML files. You can find the files in the following folder in the subfolder of your specific hand:

```
$ roscd sr_hand_config
```

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:

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

The complete list of motor sensors appears in the file, along with a number

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.

5.3. Accessing Data From The Hand

There are four main ways to access data from the hand:

- *Graphical User Interface*
- *Command Line Interface*
- *The Robot Commander*
- Using rospy or 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 or ROS CPP subscriber

5.3.1. 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

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 can be used:

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

To find information about the rosbag *joint_state_bag.bag*:

```
rosbag info joint_state_bag.bag
```

To play back this ROS Bag:

```
rosbag play joint_state_bag.bag
```

The rosbag command-line has many options of how to record and playback various topics that are published, these can be found here.

5.3.2. 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.

Coping FROM the container TO the file system:

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

Copying FROM the file system TO the container:

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

Some of the *[OPTIONS]* include:

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

5.4. 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.

5.4.1. Starting the interface

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

```
$ rqt
```

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

Starting the interface with namespaces

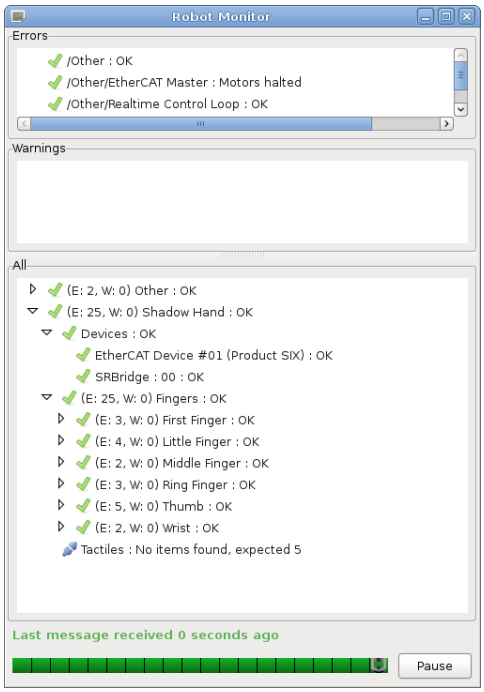
Namespaces are very useful in ROS because they allow users to isolate elements of the network to prevent accidental errors as explained here. In order to open the Graphical User Interface within a certain namespace, type:

```
$ rosrun rqt_gui rqt_gui ___ns:=<namespace>
```

5.4.2. Robot Monitor

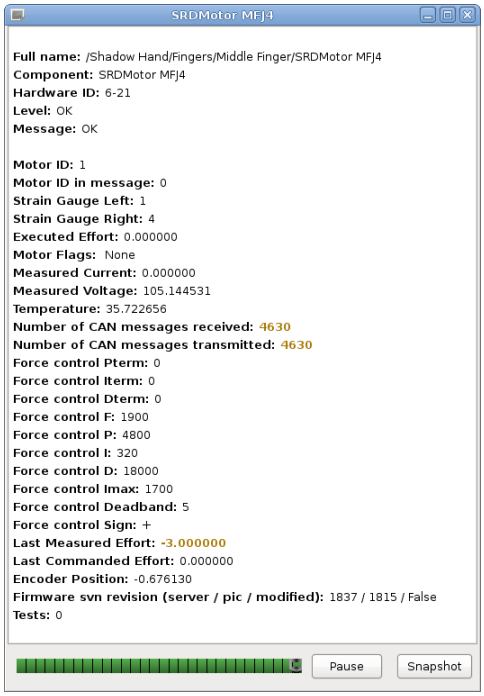
We can check that everything on the robot is working correctly using the Diagnostic Viewer.

Plugins » Robot Tools » Diagnostic Viewer



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.



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

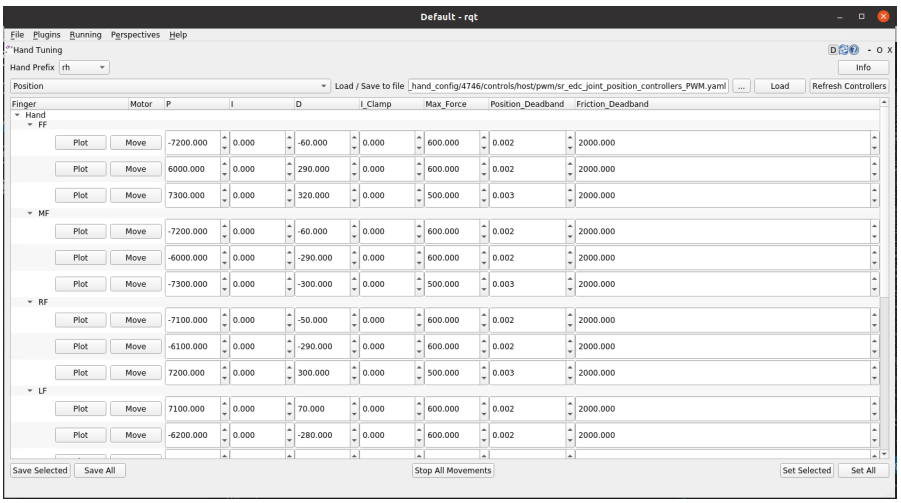
Item	Description
Full Name	
Component	
Hardware ID	
Level	
Message	Any error or status messages
Motor ID	This is the motor number. Range [0..19]
Motor ID in message	For debugging only
Strain Gauge Left / Right	These are the ADC readings from the two gauges
Executed Effort	
Motor Flags	See motor flags 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. (ºC)
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.

5.4.3. Hand Tuning

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

Plugins » Shadow Robot » Advanced » Hand Tuning

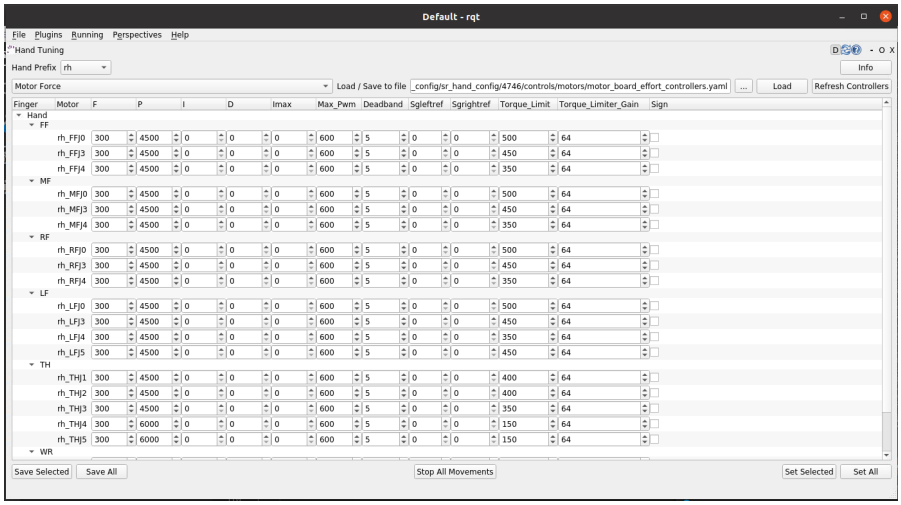
Position controller



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 ±deadband. This value should be set as a little more than the noise on the sensor. The units of deadband are the same as the value being controlled. So, the deadband for a position controller is in radians.

Force controller



- **"P", "I" & "D" terms:** Gain parameters of the torque PID controller. By default, Shadow tunes the paramenters 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 ±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.

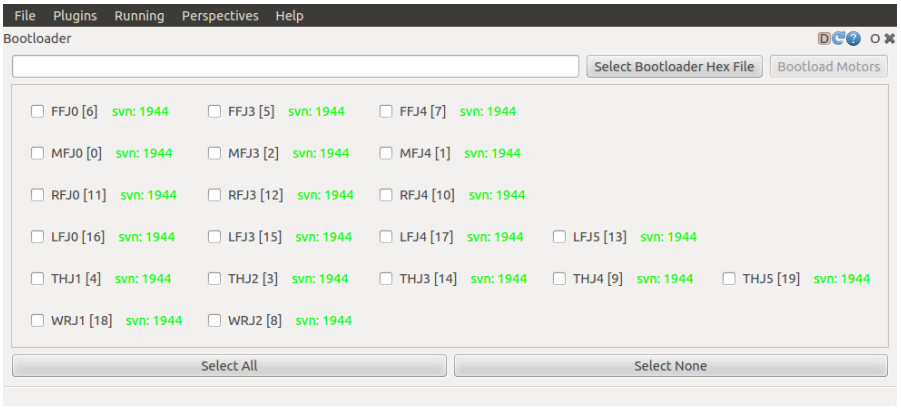
Click `Save` to save your settings.

5.4.4. Bootloader

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

Plugins » Shadow Robot » Advanced » Motor Bootloader

You will see a window listing each motor board, along with its current firmware SVN revision number.



- **Select Bootloader Hex File:** Next, tell the plugin which firmware to use. The file you should choose here is the one sent by Shadow.
- **Select your motors:** Now you may choose which motors to program. Either select one or more motors using the tick boxes, or click the `Select All` or `Deselect All` button.
- **Program Motors:** Now you can click the `Bootload Motors` button. The process is fairly slow, and takes about a 30 second per motor.

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

5.4.5. 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.

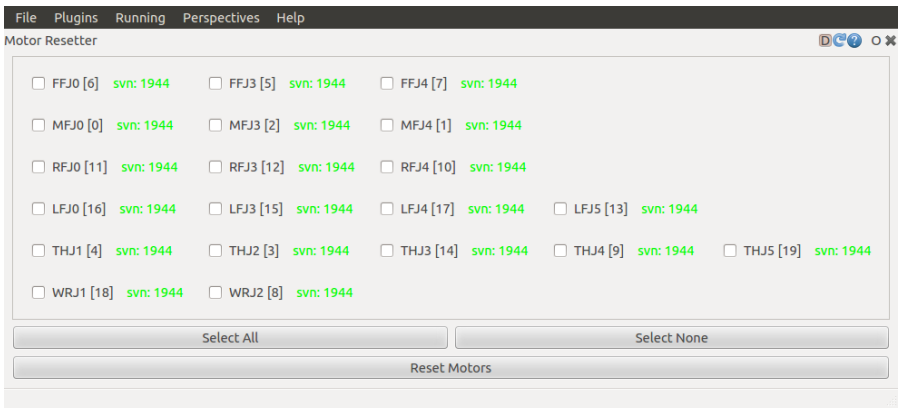
Plugins » Shadow Robot » Change Robot Control Mode

Note: Please allow some time between control changes!

5.4.6. 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

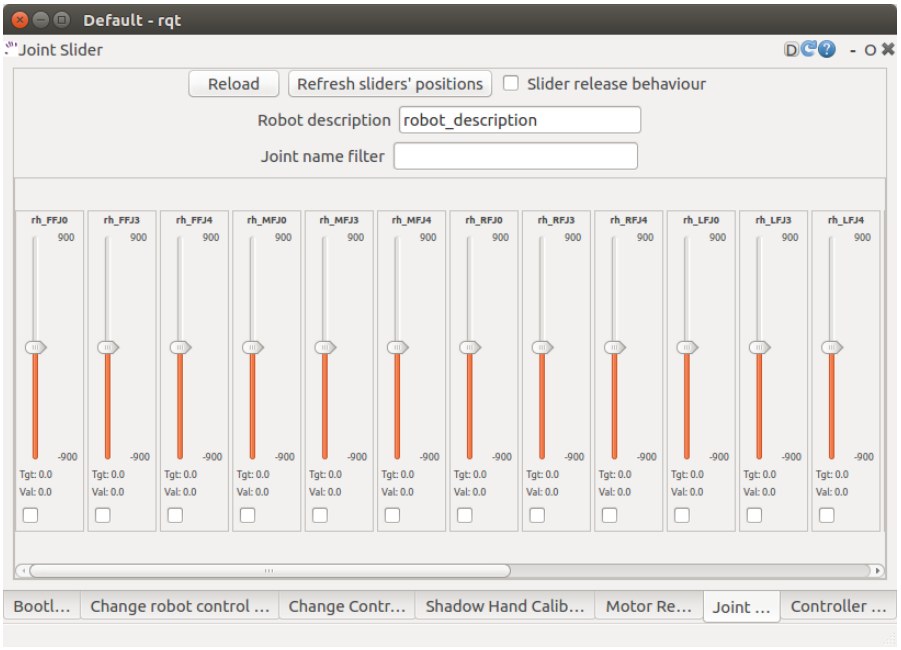


Tick the motors you wish to reset, and click `Reset Motors`. You should see the corresponding joints jiggle as the motors auto-zero the strain gauges.

5.4.7. Joint Sliders

A simple interface has been provided to control the position of each joint using a slider.

Plugins » Shadow Robot » Joint Sliders



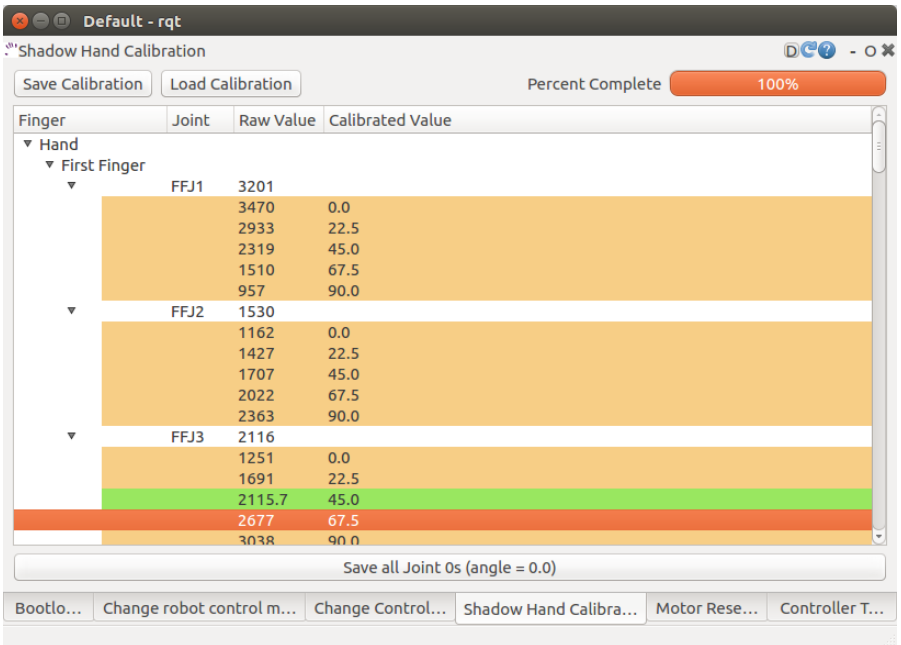
A window with twenty sliders will appear. Moving any slider will cause the corresponding joint on the hand to move. You have to start the hand in either position control or teach mode. If the control is changed, reload the plugin to make sure that the sliders correspond to the control that is running at this moment.

5.4.8. 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)

Within rqt, go to:

Plugins » Shadow Robot » Advanced » Hand Calibration

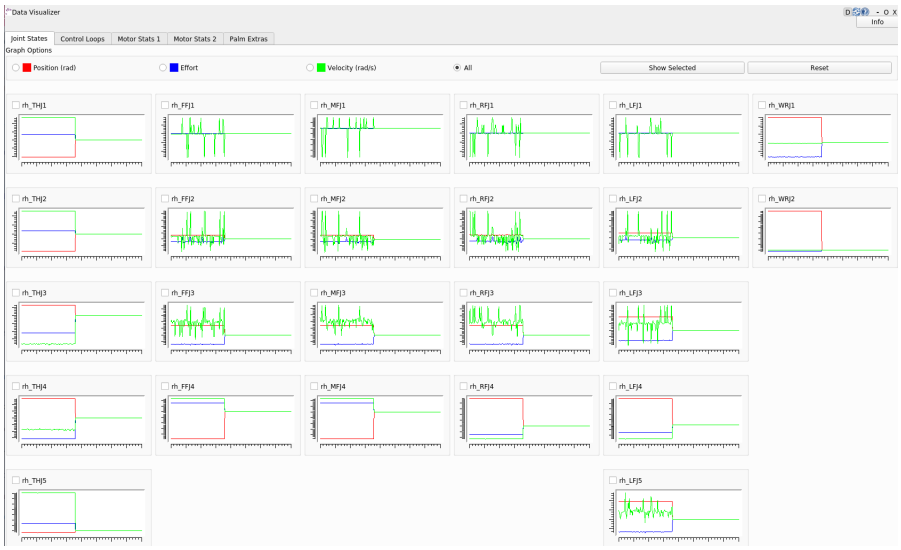


It's very unlikely that the sensors moved inside the hand, BUT, if you find misalignments with the model and you require a re-calibration, contact Shadow Robot Company here.

5.4.9. Data Visualizer

A GUI is provided to show all the data available for the Dexterous Hand.

Plugins » Shadow Robot » Dexterous Hand Data Visualizer



You also can launch it separately from rqt with an optional rosbag by running the following command:

```
$ 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

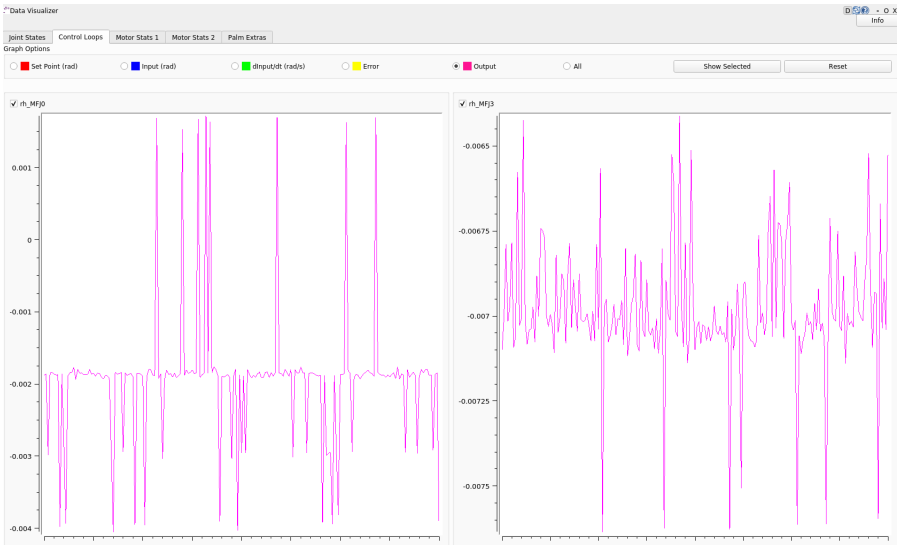
The radio buttons let you choose specific data to show (scaled) or you can choose "All" to see several graphs being displayed at the same time (unscaled).

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.

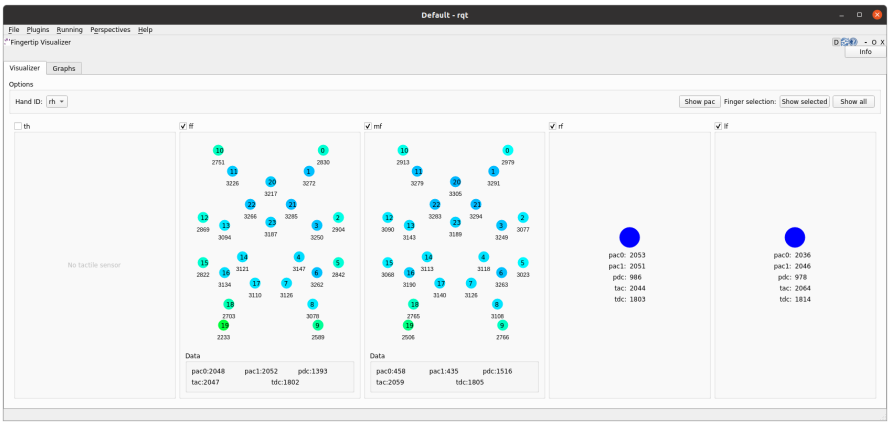
Note: The more graphs that are on show on the data visualizer will be slower and can be unreadable. To be able to see a full scaled view of a specific data type, toggle the correct radio button and check the graphs you want to see clearer.



5.4.10. Fingertip visualization

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

Plugins » Shadow Robot » Fingertip Visualization



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:

$ rqt

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 ROS-bag.

5.5. Command Line Interface

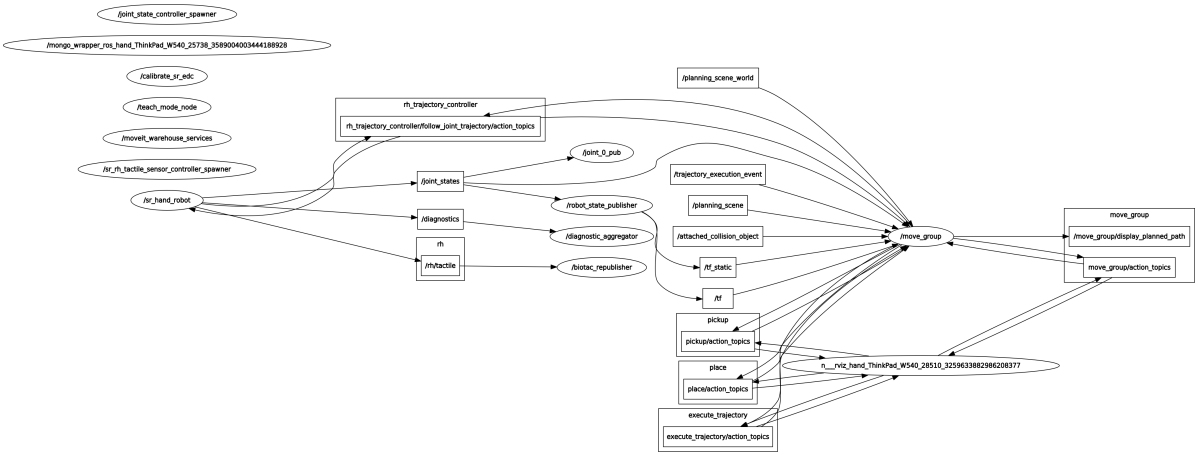
All functions of the hand are available from the command line.

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.

5.5.1. Using rostopic

To check how to interact with ROS topics, see this link.

The following rqt_graph shows the flow of topics between nodes whilst the hand is running.



Here is a list of the available topics:

Calibration (Real hand only)

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

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

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

Diagnostics (Real hand only)

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

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

Joint states

```
/joint_states
```

This topic is read-only and updates at 125 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]
velocity: [-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, 0.00123701167977189, -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 1000 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.

- *which_motor_data_arrived* is a bitmap, 20x1 demensional array for the 20 motors, which shows which motors data has been received from. For example 349525 = 01010101010101010101.
- *which_motor_data_had_errors* is a bitmap for the motors which have errors.
- The tactile sensors attached to the Hand are selected during startup, their corresponding values are here.
- *tactile_data_type* is used to specify the data in tactile, similar to motor_data_type and motor_data_packet_misc. In the Example topic message below the PST fingertip sensors are used, its value is refered here.
- *tactile_data_valid* is a bitmap for the 5 sensors that is 1 when there are no errors.
- *idle_time_us* is the time margin once the Hand has completed its processing and is ready to communicate on the EtherCAT bus.

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

Example */rh/debug_etherCAT_data* topic message:

```
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:

```
tactiles:
-
pac0: 2048
pac1: 2054
pdc: 2533
tac: 2029
tdc: 2556
electrodes: [2622, 3155, 2525, 3062, 2992, 2511, 3083, 137, 2623, 2552, 2928, 3249, 2705, 3037, 3020, 2405,␣
↪3049, 948, 2458, 2592, 3276, 3237, 3244, 3119]
-
pac0: 0
pac1: 0
pdc: -9784
```

(continues on next page)

(continued from previous page)

```
tac: 32518
tdc: 0
electrodes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
-
pac0: 0
pac1: 0
pdc: -9784
tac: 32518
tdc: 0
electrodes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
-
pac0: 0
pac1: 0
pdc: -9784
tac: 32518
tdc: 0
electrodes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
-
pac0: 0
pac1: 0
pdc: -9784
tac: 32518
tdc: 0
electrodes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

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

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

Finger Trajectory Controller

- Command

  ```
  /rh_trajectory_controller/command
  ```

  This topic can be published to and is the set position for the fingertrajectory 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]
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]
velocities: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
↪0]
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 the 20 finger joints.

Example topic message:

```
header:
seq: 29135
stamp:
  secs: 583
  nsecs: 274000000
frame_id: ''
joint_names:
  - 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
desired:
  positions: [0.00011967184218224583, 0.0005548183242297389, 0.00041894754950187046, -0.
```

(continues on next page)

(continued from previous page)

```
↪10062701434085283, 8.783502460599571e-05, 0.0004641775977824551, 0.0006856740495884452, 0.
↪1096183605452164, 0.004913053249636874, 0.00010239955504565482, 0.00047931794332974065, 0.
↪00013111648840283263, -0.10127973823850217, 8.996037101196772e-05, 0.0004488761418596776, 0.
↪0001876294034980873, 0.1001350676276958, -0.035240921470978015, -0.31784852833458305, 0.
↪0007460299926848393, 0.04911052560971607, -0.0002772503508325329]
  velocities: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,␣
↪0.0]
  accelerations: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,␣
↪0.0, 0.0]
  effort: []
  time_from_start:
    secs: 0
    nsecs:         0
actual:
  positions: [7.10895251927468e-05, 0.0006767078325600195, 0.0004175216727215769, -0.
↪10062678076768616, -0.00027645955245603204, 0.0005525155619778843, 0.0006916863497421488, 0.
↪10961843159979079, 0.004913188891993769, 4.852612279648838e-05, 0.0005853580394772351, 0.
↪00013062204099512087, -0.10127949778047185, -3.317031927352332e-06, 0.0005469772724389088, 0.
↪00018736835619481695, 0.10013483724669392, -0.03524090059726159, -0.31784850071453175, 0.
↪0007460361368888506, 0.04911112453716715, -0.0002772527343921638]
  velocities: [0.08902813995150381, -0.001099212107393761, 0.006855683840805191, 0.
↪030914854686922167, -0.2850721338474761, 0.009299471978875831, 0.0092727557285639, -0.
↪034998219897310806, 0.006294029941129844, -0.004965571363457199, 0.0018022999079858118, 0.
↪009204137459833733, 0.03093467522055917, -0.007240949470287823, -0.00034343975205673206, 0.
↪008327083950443404, -0.02997268098080695, 0.07474706451076522, 0.6818870795543708, -0.
↪003969150639010634, -0.013970489880483824, 0.02504911604093861]
  accelerations: []
  effort: []
  time_from_start:
    secs: 582
    nsecs: 721000000
error:
  positions: [4.85823169893429e-05, -0.0001218895083301419, 1.425876780203339e-06, -2.
↪3357316658589866e-07, 0.00036429457706210755, -8.833796419560613e-05, -6.012300153734884e-06, -
↪7.105457422440509e-08, -1.3564235690211035e-07, 5.3873432249051945e-05, -0.0001060400961474528,
↪ 4.944474079060512e-07, -2.404580303760895e-07, 9.327740293940678e-05, -9.810113057939773e-05, 2.
↪6104730332932036e-07, 2.3038100183114807e-07, -2.087371653303194e-08, -2.762005113510213e-08, -6.
↪144203812397109e-09, -5.989274511897236e-07, 2.3835595719390312e-09]
  velocities: [-0.08902813995150381, 0.001099212107393761, -0.006855683840805191, -0.
↪030914854686922167, 0.2850721338474761, -0.009299471978875831, -0.0092727557285639, 0.
↪034998219897310806, -0.006294029941129844, 0.004965571363457199, -0.0018022999079858118, -0.
↪009204137459833733, -0.03093467522055917, 0.007240949470287823, 0.00034343975205673206, -0.
↪008327083950443404, 0.02997268098080695, -0.07474706451076522, -0.6818870795543708, 0.
↪003969150639010634, 0.013970489880483824, -0.02504911604093861]
  accelerations: []
  effort: []
  time_from_start:
    secs: -583
    nsecs: 279000000
```

- 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
```

Wrist Trajectory Controller

- Command

```
/rh_wr_trajectory_controller/command
```

This topic can be published to and is the set position for the wrist 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_WRJ1, rh_WRJ2]
points:
-
positions: [-0.03490658503988659, 0.0]
velocities: [0.0, 0.0]
accelerations: []
effort: []
time_from_start:
secs: 0
nsecs: 5000000
```

- State

```
/rh_wr_trajectory_controller/state
```

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

Example topic message:

```
header:
seq: 23029
stamp:
  secs: 461
  nsecs: 154000000
frame_id: ''
joint_names:
  - rh_WRJ1
  - rh_WRJ2
desired:
  positions: [0.12041453184118814, 0.17462944274957784]
  velocities: [0.0, 0.0]
  accelerations: [0.0, 0.0]
  effort: []
  time_from_start:
    secs: 0
    nsecs:         0
actual:
  positions: [0.12041412527154005, 0.17453432025078097]
  velocities: [-0.006723029810312021, 8.480557225661457e-05]
  accelerations: []
  effort: []
  time_from_start:
```

(continues on next page)

(continued from previous page)

```
      secs: 460
      nsecs: 601000000
error:
  positions: [4.065696481703185e-07, 9.51224987968402e-05]
  velocities: [0.006723029810312021, -8.480557225661457e-05]
  accelerations: []
  effort: []
  time_from_start:
    secs: -461
    nsecs: 399000000
```

- 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_wr_trajectory_controller/follow_joint_trajectory/feedback
/rh_wr_trajectory_controller/follow_joint_trajectory/goal
/rh_wr_trajectory_controller/follow_joint_trajectory/result
/rh_wr_trajectory_controller/follow_joint_trajectory/status
```

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

```
/rh_wr_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

  $ 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. *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 tfs are fixed and the dynamic tfs update at 100 Hz. They can be published to, as well as read from. For further information on ROS tfs see the ROS wiki.

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

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.

```
/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
```

5.5.2. Using rosservice

To reset individual motors, E.G. RFJ4:

```
$ rosservice call /sr_hand_robot/lh/reset_motor_RFJ4
```

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

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

5.6. Repositories

Our code is split into different repositories:

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

5.7. 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: base class.
- SrHandCommander: hand management class.
- SrArmCommander: arm management class.

5.7.1. SrRobotCommander

Overview

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

Examples of usage can be found here.

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

Basic terminology

A robot is described using an srdf file which contains the semantic description that is not available in the urdf. It describes a robot as a collection of **groups** that are representations of different sets of joints 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

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

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

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

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

```
rospy.init_node("robot_commander_examples", anonymous=True)

hand_commander = SrHandCommander(name="right_hand")
```

(continues on next page)

(continued from previous page)

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

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

Plan/move to a predefined group state

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

Parameters:

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

Example

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

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

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

```
trajectory = [
  {
    'name': 'open',
    'interpolate_time': 3.0
  },
  {
    'name': 'pack',
    'interpolate_time': 3.0,
    'pause_time': 2
  },
  {
    'name': 'open',
    'interpolate_time': 3.0
  },
  {
    'name': 'pack',
    'interpolate_time': 3.0
  }
]

hand_commander.run_named_trajectory(trajectory)

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

Check if a plan is valid and execute it

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

Example

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

hand_commander = SrHandCommander()

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

Stop the robot

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

Example

```
hand_commander.send_stop_trajectory_unsafe()
```

5.7.2. SrHandCommander

Overview

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

Setup

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

```
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

```
# 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

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

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

```
## 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)
```

5.7.3. SrArmCommander

The SrArmCommander inherits all methods from the 'robot commander](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

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

To return the reference frame for planning in cartesian space:

```
reference_frame = arm_commander.get_pose_reference_frame()
```

Plan/move to a position target

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

```
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

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:

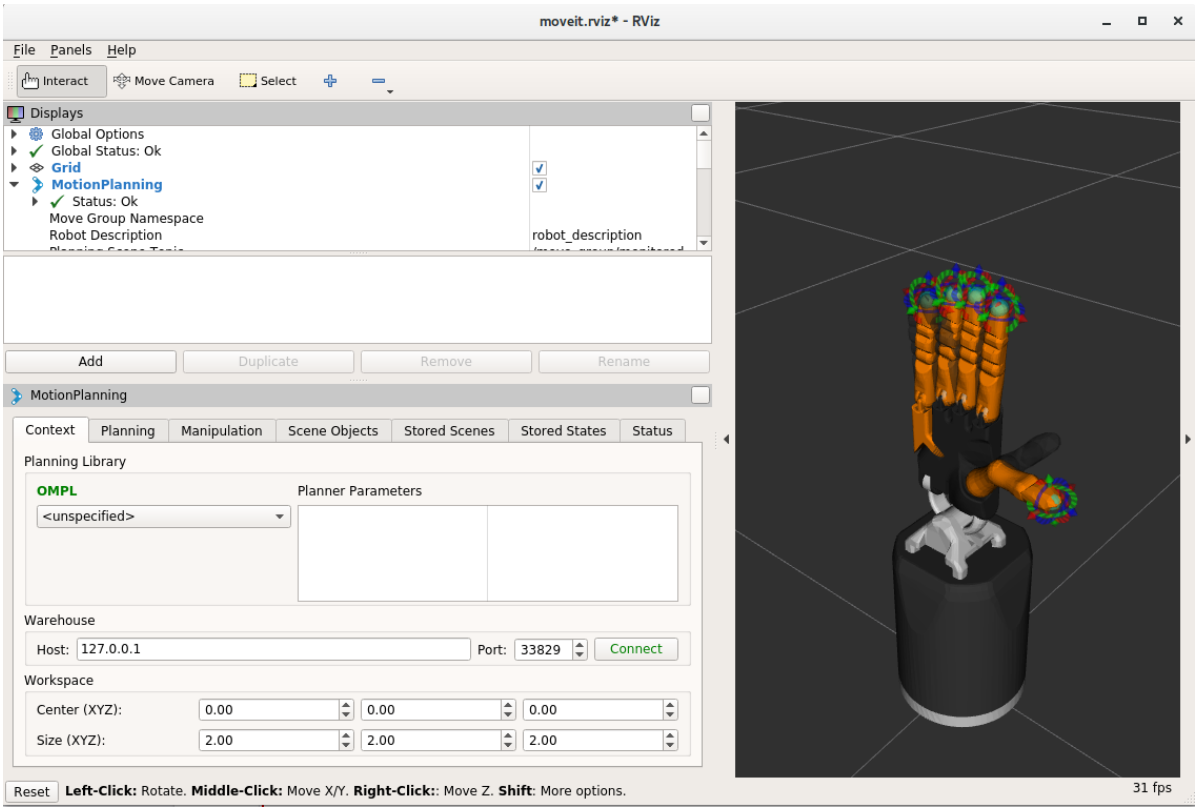
- *pose* desired pose of end-effector: a Pose message, a PoseStamped message or a list of 6 floats: [x, y, z, rot_x, rot_y, rot_z] or a list of 7 floats [x, y, z, qx, qy, qz, qw]
- *end_effector_link* name of the end effector link (default value is empty string)
- *wait* indicates if the method should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

Example

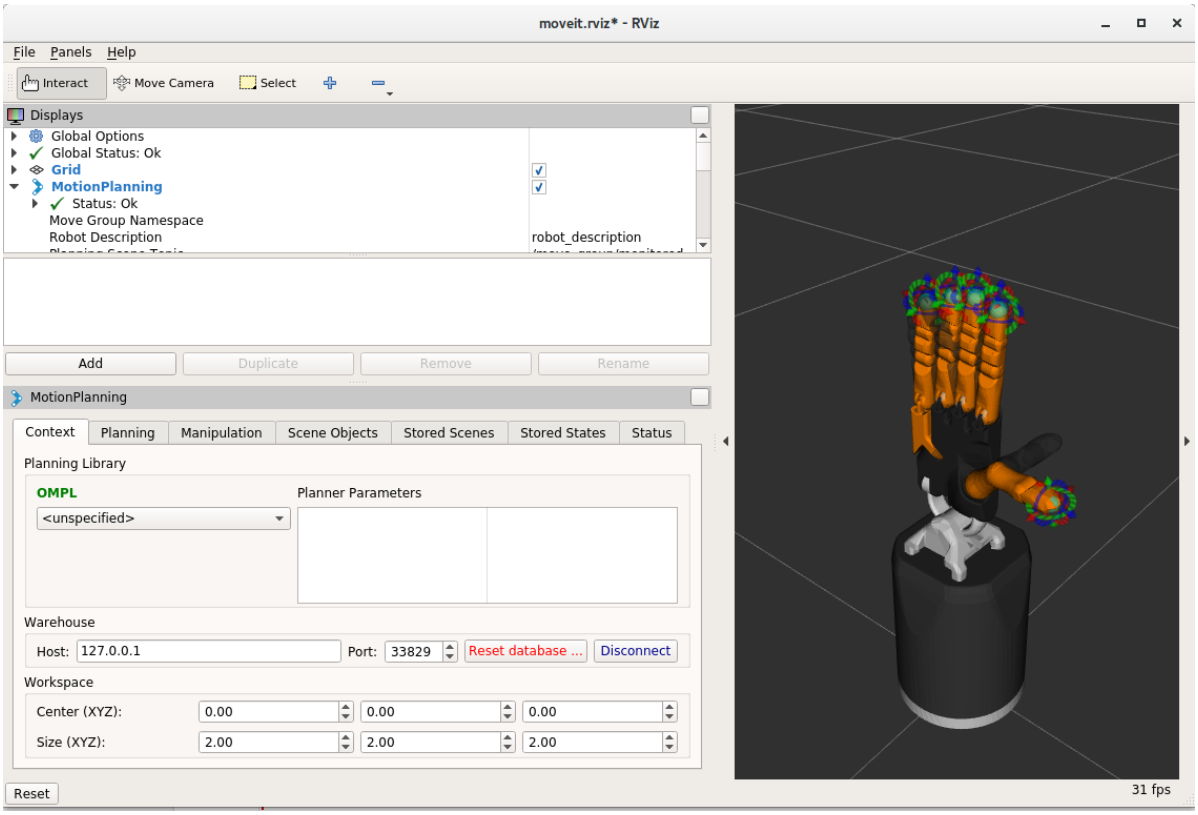
```
rospy.init_node("robot_commander_examples", anonymous=True)
arm_commander = SrArmCommander(name="right_arm", set_ground=True)

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)
```

5.8. Saving States

To save a state you must first be connected to the warehouse. After launching the hand, click the green **Connect** button in the 'Context' tab of rviz.

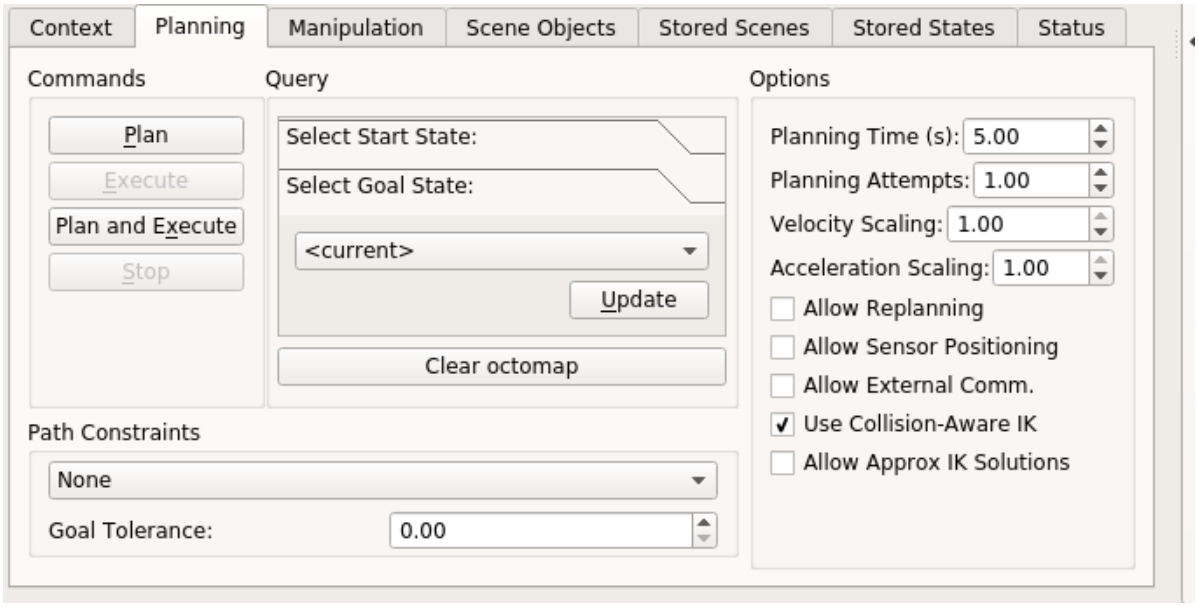


If you have connected successfully you should see two new buttons, **Reset database** and **Disconnect**, as can be seen in the following picture:

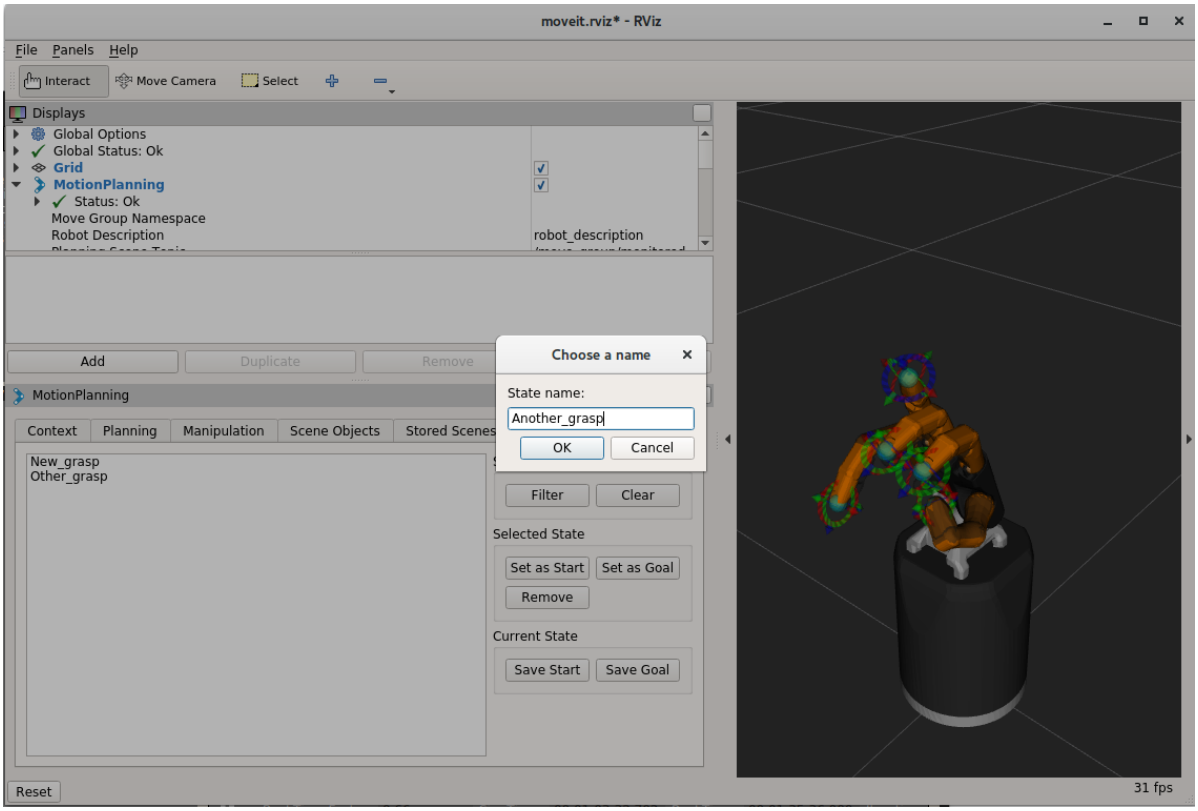


Next, go to the 'Stored States' tab in 'Motion Planning'. Here you have full control over the saved states in the warehouse. You can then follow these steps:

- move the hand to the grasp position
- Go to the 'Planning' tab and in the 'Select Goal State' select 'current' and click **update**.



- Finally, go to the 'Stored States' tab and click the button **Save Goal** under the 'Current State' group. A prompt will appear to ask you to name the state. Once named, you can plan to and from this state.



5.9. Hand Autodetection

This feature (**new in Noetic**) 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 and consists of two scripts.

5.9.1. Installation

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-noetic-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 into your workspace.

5.9.2. sr_hand_detector_node

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!
```

5.9.3. sr_hand_autodetect

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.

5.10. 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.

5.10.1. Shadow Hands

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

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

5.10.2. Shadow Hands mounted on UR arms

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

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

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

5.10.3. 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:=th,ff,mf,rf,lf tip_sensors:=ff=bt_2p,lf=bt_sp,mf=pst,rf=pst,th=bt_sp mid_sensors:=none␣
↪prox_sensors:=none palm_sensor:=none sim:=true
```

- Xacro command:

```
xacro <xacro file> side:=right hand_type:=hand_g hand_version:=G1M5 fingers:=th,ff,mf,rf,lf tip_
↪sensors:=ff=bt_2p,lf=bt_sp,mf=pst,rf=pst,th=bt_sp mid_sensors:=none prox_sensors:=none␣
↪palm_sensor:=none
```

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

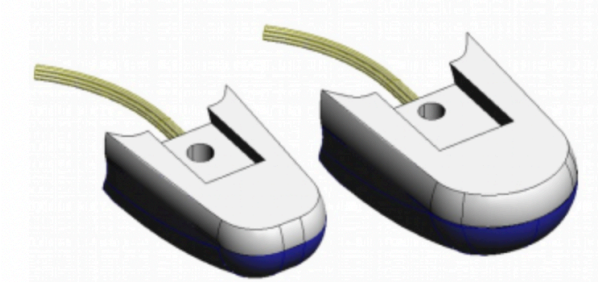
5.10.4. Autodetection parameters

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 xacros. 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.

5.11. Fingertips

5.11.1. PST Sensor

These are simple sensors, fitted as standard, which measure the air pressure within a bubble at the finger tip. When the finger tip presses on an object, the pressure in the bubble increases. The sensor incorporates an automatic drift and temperature compensation algorithm (essentially a high pass filter with an extremely low cut off frequency).



Topics

PST sensor data will be published on the following topics:

/rh/tactile

Example topic message when using PST sensors:

```
header:
-
seq: 6306
stamp: .
secs: 1660831064
nsecs: 585176249
frame_id: "rh_distal"
pressure: [ 22560, 256, 22560, 22560, 22560 ]
temperature: [ 32635, 637, 32635, 32635, 32635 ]
-
```

5.11.2. BioTacs

The BioTacSP® is a biologically inspired tactile sensor from SynTouch LLC. It consists of a rigid core surrounded by an elastic skin filled with a fluid to give a compliance similar to the human fingertip. The BioTac is capable of detecting the full range of sensory information that human fingers can detect: forces, microvibrations, and thermal gradients. The skin is an easily replaced, low-cost, moulded elastomeric sleeve.

Sensor	Update rate
Pressure AC signal	2000Hz
Pressure DC signal	90Hz
Temperature AC & DC	90Hz
19 Normal force sensors	90Hz each

Topics

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

```
/rh/tactile
```

Example topic message when using BioTac fingertip sensors:

```
tactiles:
-
pac0: 2048
pac1: 2054
pdc: 2533
tac: 2029
tdc: 2556
electrodes: [2622, 3155, 2525, 3062, 2992, 2511, 3083, 137, 2623, 2552, 2928, 3249, 2705, 3037, 3020,␣
↪2405, 3049, 948, 2458, 2592, 3276, 3237, 3244, 3119]
-
pac0: 0
pac1: 0
pdc: -9784
tac: 32518
tdc: 0
electrodes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
-
pac0: 0
pac1: 0
pdc: -9784
tac: 32518
tdc: 0
electrodes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
-
pac0: 0
pac1: 0
pdc: -9784
tac: 32518
tdc: 0
electrodes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
-
pac0: 0
pac1: 0
pdc: -9784
tac: 32518
tdc: 0
electrodes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

- The following topics are specific for each sensor 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.

```
/rh/biotac_
```

Example */rh/biotac_*** topic message:

```
pac0: 2056
pac1: 2043
pdc: 2543
tac: 2020
tdc: 2454
```

(continues on next page)

(continued from previous page)

```
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]
```

5.11.3. 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.

For more information on setup and getting started with the optoforce sensors, look here.

Topics

Optoforce sensor data will be published on the following topics:

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

5.12. Firmware

5.12.1. Palm firmware

The palm firmware is responsible for the following:

- Managing the ET1200 EtherCAT ASIC and EtherCAT state
- Receiving command data from the ET1200
- Transmitting the contents of the command packet to the motors
- Receiving data from the motors
- Sampling data from the joint sensors
- Sampling data from the Tactile sensors
- Loading status data into the ET1200

5.12.2. Command data for motor hand

Command data is sent from the host, and received by the palm. It consists of the following:

Item	Size	Description
EDC_Command	32 bits	Used for switching the palm into test mode
Motor data type request	32 bits	Which sensor data should the motors return?
Even or Odd motors?	16 bits	Which motors should return data?
Type of motor demand	32 bits	Are we demanding torque or PWM? Can also be used to send config values to the motors.
Motor demand data	16 bits x 20	All 20 torque or PWM demands. May also contain config data for the motors.
Tactile sensor data type request	32 bits	Which type of sensor data should the tactile sensors return?

5.12.3. Status data for motor hand

Status data is sent from the palm, and received by the host. It consists of the following:

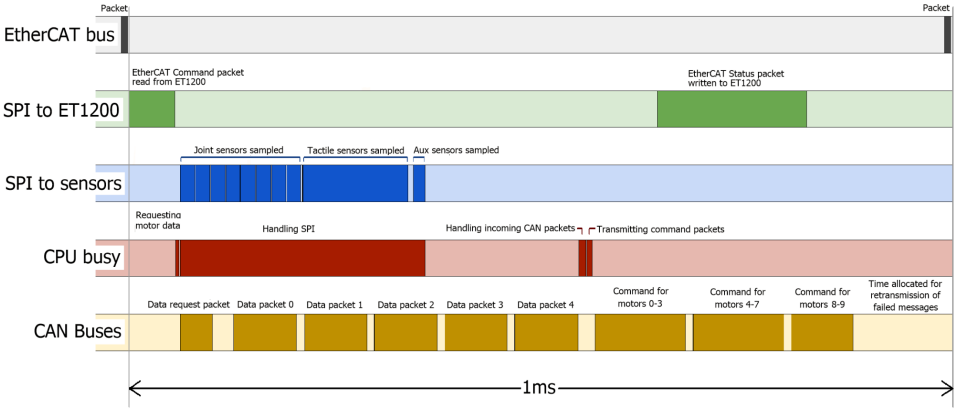
Item	Size	Description
EDC_Command	32 bits	Copy of the same value from command data
Joint sensor/IMU data	16 bits x 37	All of the joint sensors, the Auxiliary Analog channels, and the IMU sensors.
Motor data type	32 bits	Copy of the same value from command data
Even or Odd motors ?	16 bits	Copy of the same value from command data
Which motor data arrived	32 bits	Flags indicate which CAN messages were seen
Which motor data had errors	32 bits	Flags indicate that the wrong type of data was sent by this motor.
Motor data	16 bits x2 x10	Torque + one other sensor from 10 motors.
Tactile sensor data type	32 bits	Copy of the same value from command data
Which tactile data is valid ?	16 bits	Flags indicate which tactile data is valid.
Tactile sensor data	16 bits x8 x5	

5.12.4. Time frame

The Palm firmware has a considerable amount of work to complete in the 1 millisecond time frame:

- Detect the incoming EtherCAT packet
- Download the command data from the ET1200
- Request sensor data from the motors
- Sample all of the joint sensors
- Request data from the tactile sensors
- Receive sensor data from the motors
- Transmit demand data to the motors
- Upload status data into the ET1200

In this diagram, we can see a breakdown of the time frame:



SPI to ET1200: All of the data must be written to the ET1200, before the next EtherCAT packet arrives. If it does not, then the packet's status data will be filled with zeros.

SPI to Sensors: The SPI bandwidth is really the limiting factor in the time frame. Data cannot be written back to the ET1200 until it has been collected by the MCU.

CPU Busy: We can see that the CPU is busy for most of the time, communicating with the ET1200, sampling sensors etc.

CAN buses: The CAN buses are close to maximum utilization. A little time is left during each frame to allow for re-transmission attempts. The time frame begins with a request-for-data message from the palm. The motors drivers respond immediately with their data. As soon as all 10 messages have been received, the palm sends out the demand values to all motor drivers.

5.12.5. Tactile sensors

The palm firmware supports different types of tactile sensor. The type of sensor is automatically detected, and the correct protocol is used between the hand and the sensor. The host PC is also informed of the sensor type so that it can interpret the data correctly. If more than one type of sensor is connected, then it is not possible to communicate with any of them, and no tactile sensor information will be available. The host will be informed of the conflict.

5.12.6. Motor Firmware

The motor firmware is responsible for the following:

- Ensuring the safety of the motor
- Actively controlling the force applied to the tendons by the motor
- Returning sensor data to the host

5.12.7. Safety

The motor will be halted under the following circumstances:

- The measured temperature of the motor exceeds 60oC
- The A3950 H-bridge reports a fault
- The CRC for the FPID configuration is bad
- No demand values are seen for 20ms

5.12.8. Sensors

Every motor returns two 16-bit sensor values every 2ms. The first sensor value is usually Torque, and the second is requested by the host. Therefore it is possible for the host driver to modify the transmission rates of the sensors on the fly. By default, the rates are set in the file sr_robot_lib/config/motor_data_polling.yaml , and can be changed by the customer. The customer may also wish to modify the driver to have real-time control over the rates.

5.12.9. Demands

Two types of demand may be sent to the motors, depending on the type of control / debugging desired.
PWM demand: This is used for basic position control, and is used by default on a new hand. The PWM demand value is sent straight to the motor, unless there is a safety cutout.

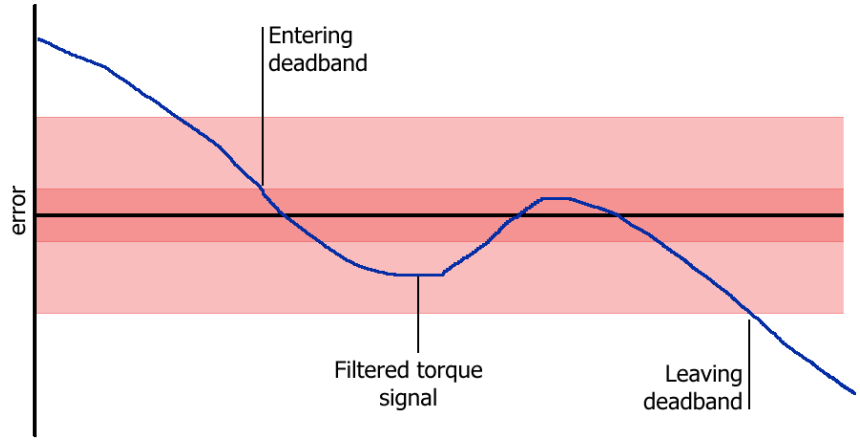
Torque demand: This is an alternative method of control. The motor MCU will use its FPID algorithm to maintain the demanded torque at the tendons.

5.12.10. Control

The motor firmware implements an FPID algorithm, running at 5kHz. FPID is a Feed-forward, Proportional, Integral, Derivative algorithm, where a proportion of the torque demand is fed forward to the

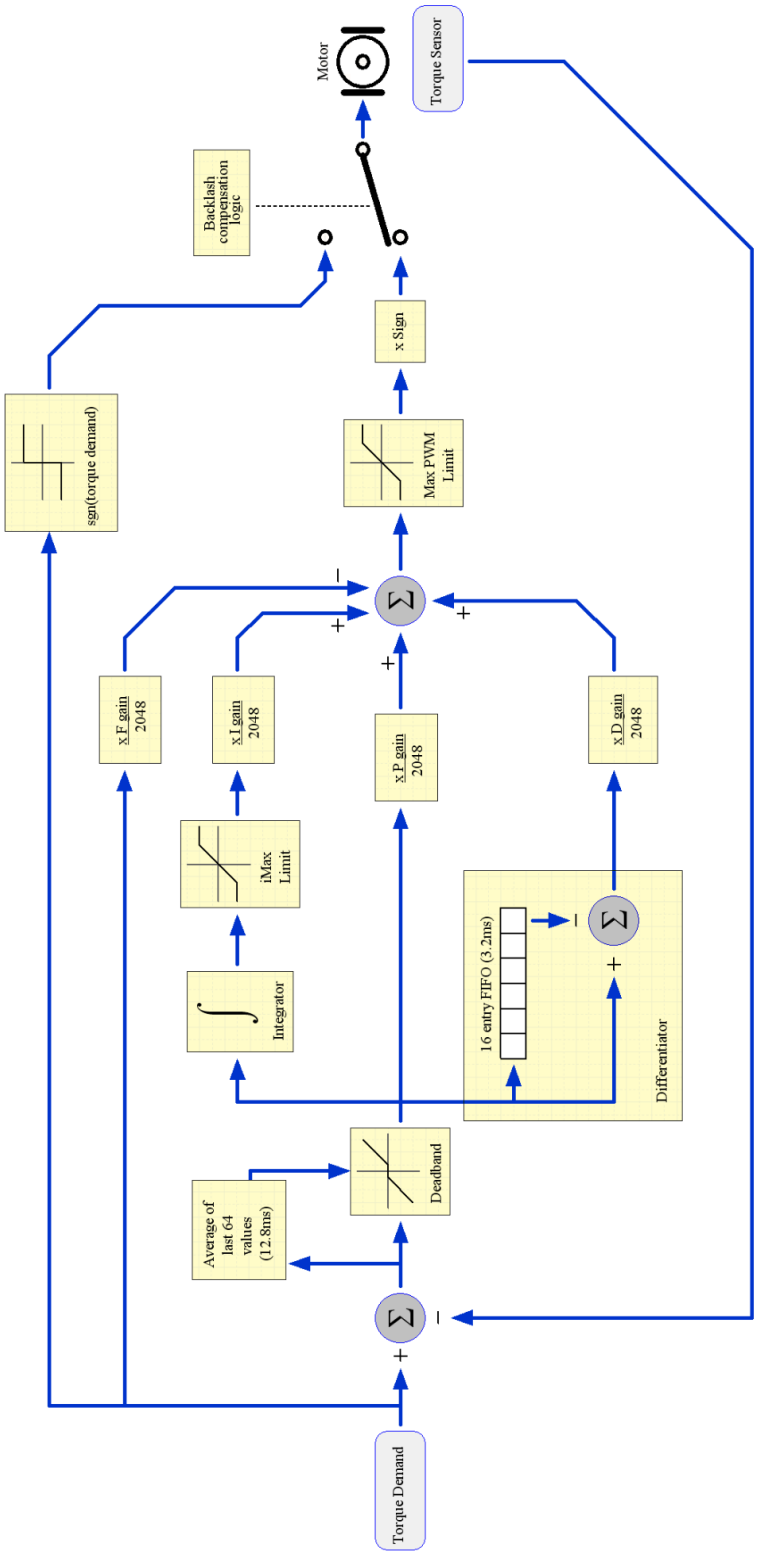
output. The algorithm supports a number of other features to ensure the safety of the motor, stability of the control and speed of response. See next page for a flow diagram of the control algorithm.

Deadband: When the torque is sufficiently close to its target position, ideally we would like the motor to stop, drawing no power, and preventing oscillation. This is achieved with the deadband. This deadband algorithm uses the average of the last 64 torque readings (equivalent to 12.8ms) to decide whether or not the torque target has been reached. It also includes hysteresis to prevent chattering when close to the deadband.



Derivative: The derivative is implemented using a 16-entry FIFO (equivalent to 3.2ms). The derivative is the difference between the first and last entries in the FIFO.

Backlash Compensation: Due to the mechanical nature of the hand, there must be some slack in the tendons. When the motor changes direction, there will be a short time period while the spool winds in the slack. This is known as backlash, and is a known problem in machine control. Therefore, in order to improve the response time of the controller, the motor is driven at full power when the torque demand changes sign. This takes up the slack as fast as possible. Normal control is resumed as soon as tension is felt on tendon.


Motor
Torque Sensor
Backlash compensation logic
x Sign
sgn(torque demand)
Max PWM Limit
x F gain 2048
x I gain 2048
x P gain 2048
x D gain 2048
iMax Limit
Integrator
16 entry FIFO (3.2ms)
Differentiator
Deadband
Average of last 64 values (12.8ms)
Torque Demand

6

Mechanical description

- *Dimensions*
- *Kinematics*
- *Finger*
- *Thumb*
- *Wrist*
- *Ranges*
- *Position Sensors*
- *Motor Unit*
- *Motor Layout*

6.1. Dimensions

The Hand has been designed to be similar to a typical male hand, however the fingers are all the same length, although the knuckles are staggered to give comparable fingertip locations to the human hand.

6.2. Kinematics

6.3. Finger

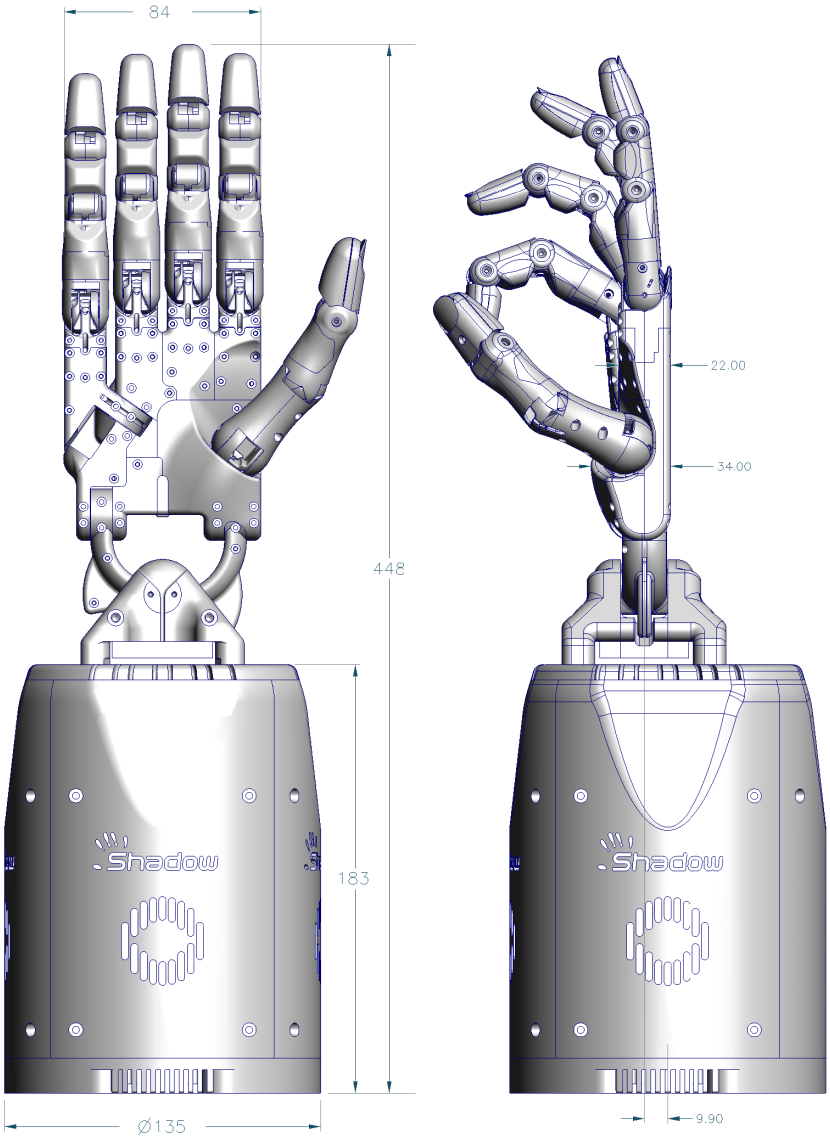
The four fingers are named according to the UK convention: First, Middle, Ring, Little.

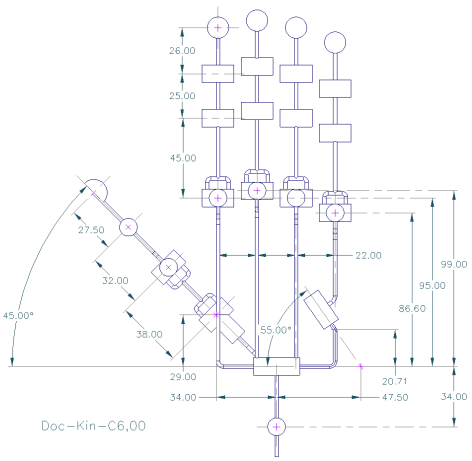
6.3.1. Naming and angle conventions

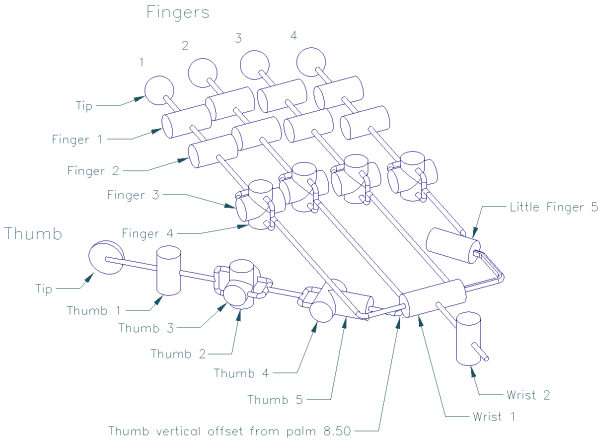
The four finger joints are the distal (finger tip), middle, proximal (nearest the palm), and the adduction/abduction joint (sideways movement) which is coplanar with the proximal joint. Joints are numbered from 1, starting at the distal end. Arrows on the diagram show the direction of positive rotation. Lines show the axis of rotation. For joint 4s, spreading the fingers is negative rotation. i.e. for FF and MF, anti-clockwise is positive, and for RF and LF, clockwise is positive.

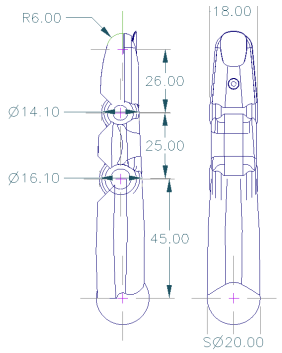
6.3.2. Loopback tendons and J0 coupling

In order to reduce the number of actuators in the forearm, joints 1 and 2 of the fingers are coupled together such that:


84
448
183
Ø135
22.00
34.00
9.90
Shadow
Shadow

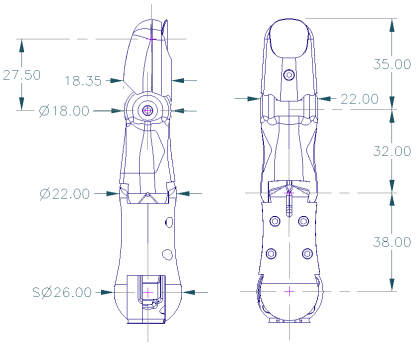




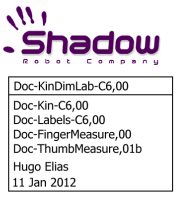


Finger

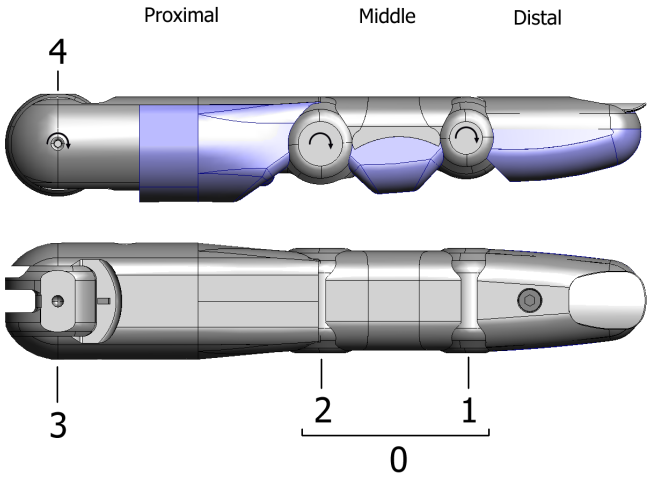
Positive rotations
are anti-clockwise

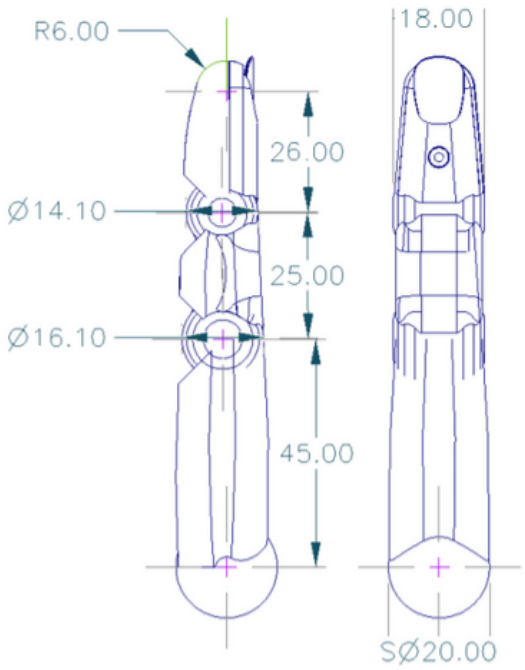


Thumb



Doc-KinDimLab-C6,00
Doc-Kin-C6,00 Doc-Labels-C6,00 Doc-FingerMeasure,00 Doc-ThumbMeasure,01b
Hugo Elias 11 Jan 2012



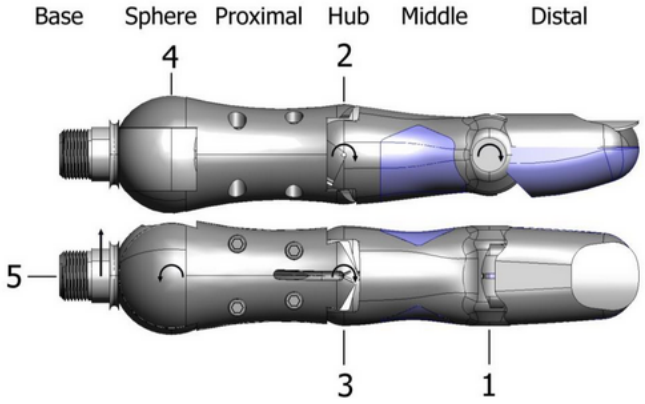


$$\text{joint1 angle} <= \text{joint2 angle}$$

Any flexion of joint 1, beyond the angle of joint 2, forces joint 2 to flex to maintain the constraint. Joints 1 and 2 are together connected to one motor. For the purposes of control, they are considered to be a single joint, 0 (zero).

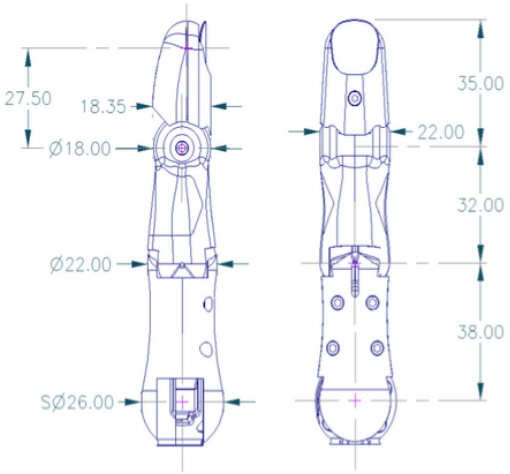
6.4. Thumb

There is one thumb which is mechanically different to the fingers.



6.4.1. Naming and angle conventions

The thumb has five degrees of freedom, and is stronger than the fingers, partly due to its larger diameter pulleys, and partly due to the larger motors actuating joints 4 and 5. All five joints of the thumb are independently actuated, and there is no intentional coupling between joints.

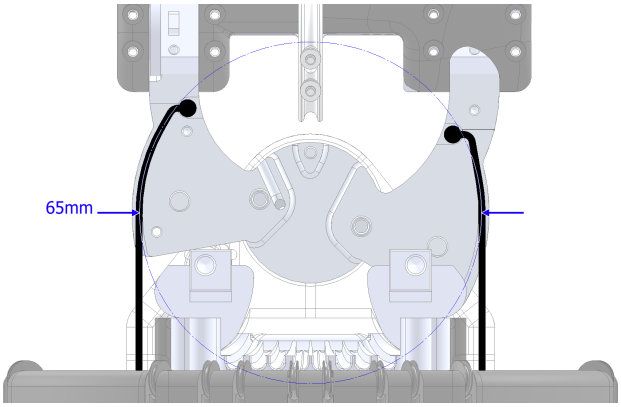


6.5. Wrist

The wrist has two degrees of freedom, Yaw and Pitch.

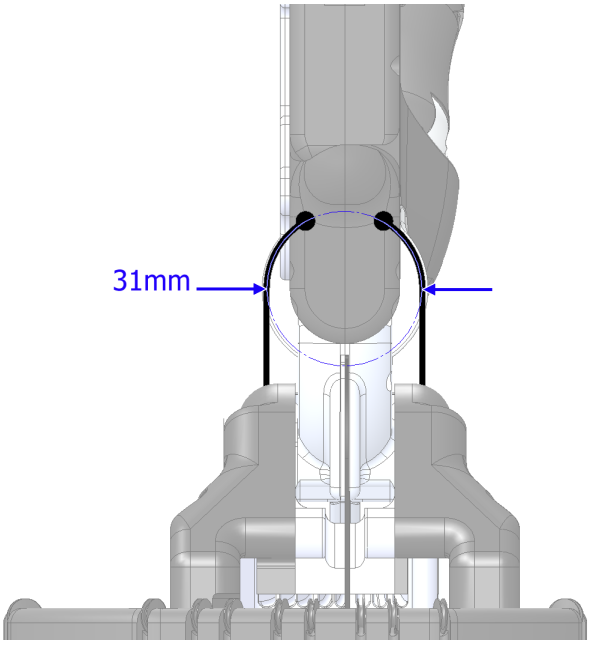
6.5.1. Yaw

The Yaw rotation is more proximal, and has a range of -28º (towards little finger) to 10º (towards thumb), and a pulley diameter of 65mm.



6.5.2. Pitch

The Pitch rotation is more distal, and has a range of -40º (extension) to 28º (flexion), and a pulley diameter of 31mm.



6.6. Ranges

Joint(s)	Min deg	Max deg	Min rad	Max rad	Notes
FF1, MF1, RF1, LF1	0	90	0	1.571	Coupled
FF2, MF2, RF2, LF2	0	90	0	1.571	
FF3, MF3, RF3, LF3	-15	90	-0.262	1.571	
FF4, MF4, RF4, LF4	-20	20	-0.349	0.349	
LF5	0	45	0	0.785	
TH1	-15	90	-0.262	1.571	
TH2	-40	40	-0.698	0.698	
TH3	-12	12	-0.209	0.209	
TH4	0	70	0	1.222	
TH5	-60	60	-1.047	1.047	
WR1	-40	28	-0.698	0.489	
WR2	-28	10	-0.489	0.174	

6.7. Position Sensors

The angle of each joint in the finger is measured by a Hall effect sensor moving through the magnetic field of either a diametrically polarised ring magnet, or an axially polarised button magnet. An ADC in the proximal phalange samples the two sensors for joints 1, 2 and 3, while an ADC in the palm samples the sensor for joint 4.

The angle of each joint in the thumb is measured by a Hall effect sensor moving through the magnetic field of either a diametrically polarised ring magnet, or an axially polarised button magnet. An ADC in the middle phalange samples the distal two sensors, while an ADC in the palm samples the rest.

A pair of Hall effect sensors are used to measure Joint 5. They are placed 90° apart, and the sampled values from each sensor are added together at the host to improve the signal to noise ratio.

6.8. Motor Unit

In a motor hand, every degree of freedom is driven from the array of twenty motors mounted on the forearm frame. Each motor drives two tendons to give a pull/pull control.

Force sensing is integrated into the tendons at the motors, and is used to provide compliant movements. Each pair of tendons couples a motor to a joint.

Each motor is managed by the Hand PC and completely controlled by the on-board electronics.

6.8.1. Small vs Large

Two types of motors are used in this hand design.

Small Motors: Sixteen small motors are used for all finger joints, and most thumb joints.

Part	Value	Units
Motor	Maxon 118608	
Gear	Maxon 352367	
Motor power	3	W
Gear ratio	131:1	
Max continuous safe tendon load	65	N
Tendon Load at full power	110	N
Max continuous current	217	mA

Large Motors: Four large motors are used for the two wrist joints, and thumb joints 4 and 5.

Part	Value	Units
Motor	Maxon 110151	
Gear	Maxon 143988	
Motor power	6	W
Gear ratio	128:1	
Max continuous safe tendon load	190	N
Tendon Load at full power	190	N
Max continuous current	350	mA

6.8.2. Tensioner

Each motor unit (except for the wrist) includes a tensioner unit. These exist purely to maintain a little tension on each tendon at all times, so that the tendons wind tidily onto the motor spool. The tension applied by the tensioner is very slight, and does nothing to compensate for backlash.

6.8.3. Torque measurement

The tendons exit the motor spool, and take a 90º turn around a metal bar. This bar is held at each end by a load cell, which is set up to measure the vertical component of any load exerted on the bar. The torque is calculated as the difference between the two readings.

If the tendons are not correctly adjusted, the tensioners may not be able to maintain tension on the spool. In this case, the tendons can become tangled around the spool.

6.8.4. Slack adjustment and tensioning.

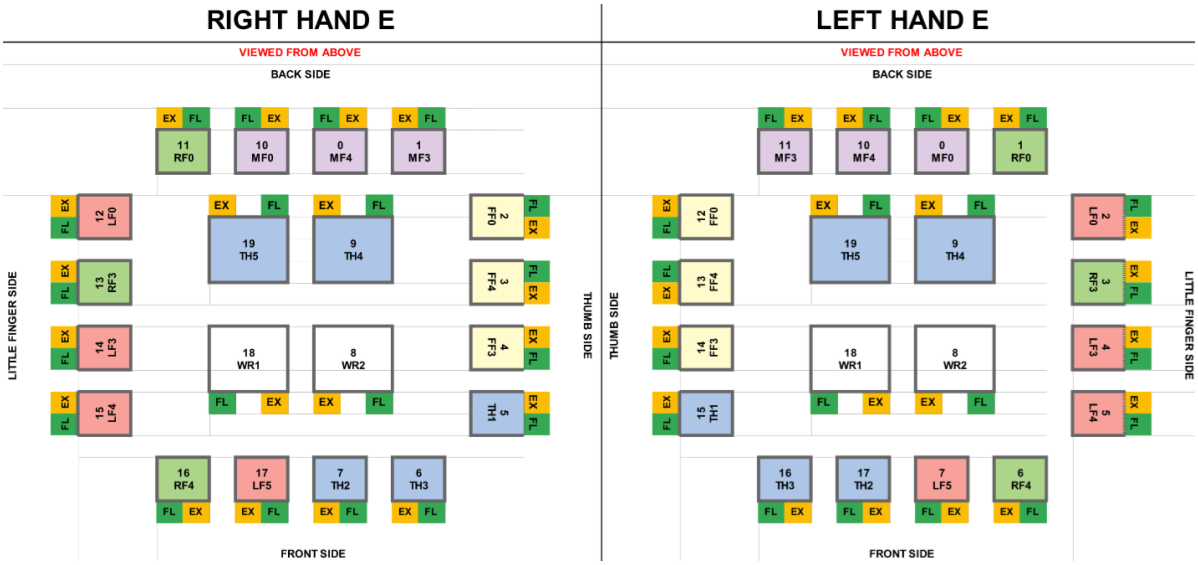
Refer to the section: **Maintaining the hand**.

The spool is split into two halves which can rotate relative to each other, but are normally held by a bolt fixedly with respect to each other. To take up any tendon slack, simply loosen the bolt, rotate the top half of the spool, and re-tighten the bolt.

Only one tendon is tight at any one time, while the other tendon is fairly loose. However, the loose tendon is kept under very slight tension by the tensioner to help it wind around the spool tidily. The wrist motors do not have tensioners.

6.9. Motor Layout

This is the layout of the motors in the base, as seen from the hand end.

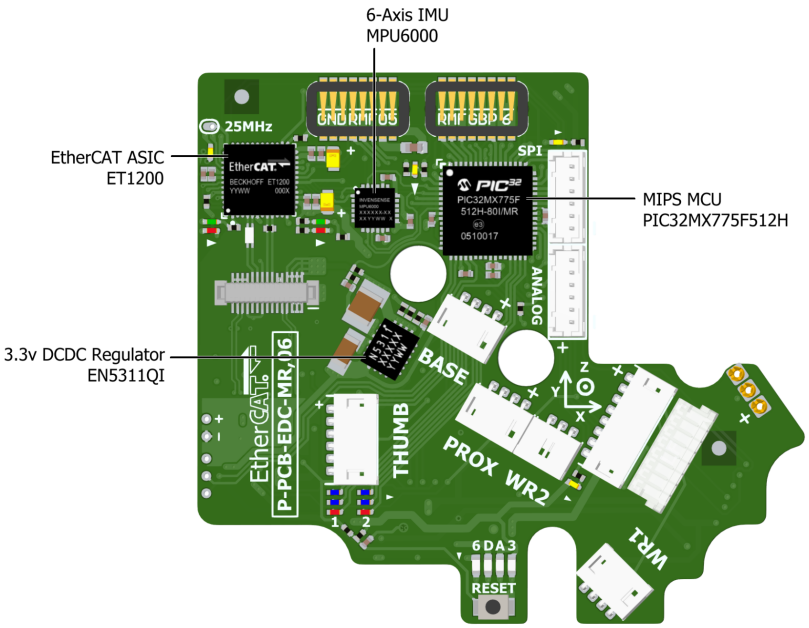


7

Electrical description

- *Chipset*
- *Data Flow*
- *Control Description*

7.1. Chipset



PIC32: This is a 32-bit MIPS CPU which performs all of the sensor sampling, and handles the flow of all the data in the hand.

ET1200: This is the EtherCAT ASIC, dealing with the reception and transmission of EtherCAT packets to and from the PC.

MCP3208: These are 12-bit ADCs used throughout the hand. There are two in the palm, and one in each finger. There are two on the underside of the Palm PCB, and one in each finger.

A1321: These are Hall effect sensors used to measure the position of all joints of the hand.

EN5311QI: This is an ultra high efficiency switching regulator which powers the PIC32 and the ET1200.

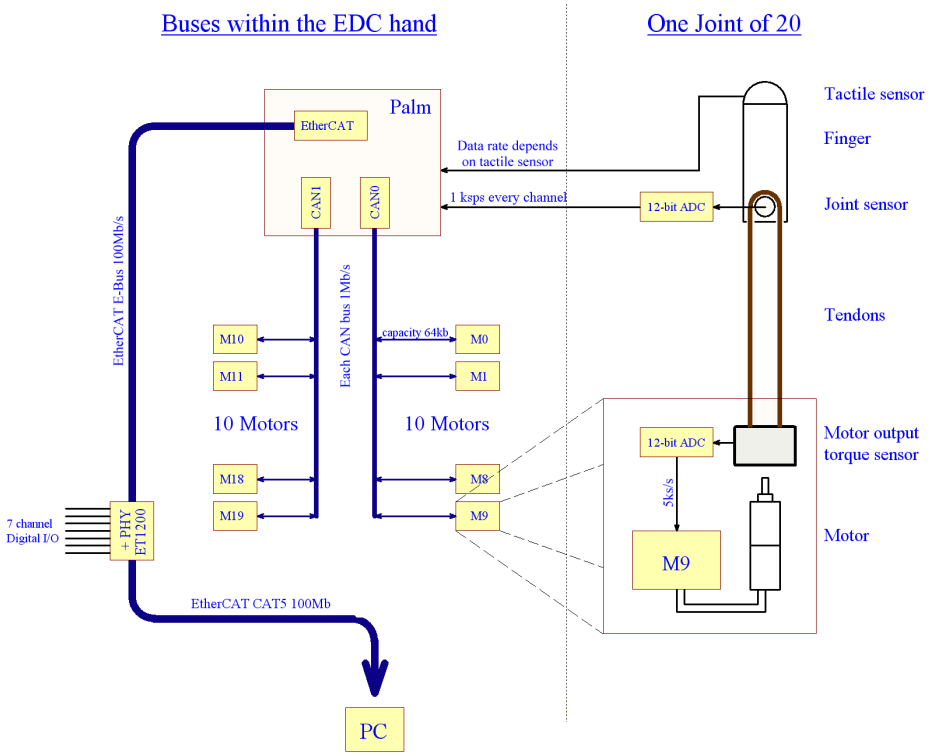
MPU-6000: (Optional) This is a 6-axis accelerometer / gyroscope which may be useful to some customers.

7.2. Data Flow

7.2.1. Overview

The only data connection between the PC and the hand is a 100Mbps EtherCAT cable. EtherCAT is an open high performance Ethernet-based fieldbus system. The EtherCAT cable enters the hand at the elbow, and connects to an EtherCAT bridge which converts the signal to E-Bus (LVDS) which is suitable for connection to the palm as it needs no magnetics.

Motor hand:



Two CAN buses are connected to the palm, and each serves 10 motors, giving a total of 8kB/s bandwidth to/from each motor. An SPI bus serves each finger, allowing the palm to sample all joint sensors at 1000Hz.

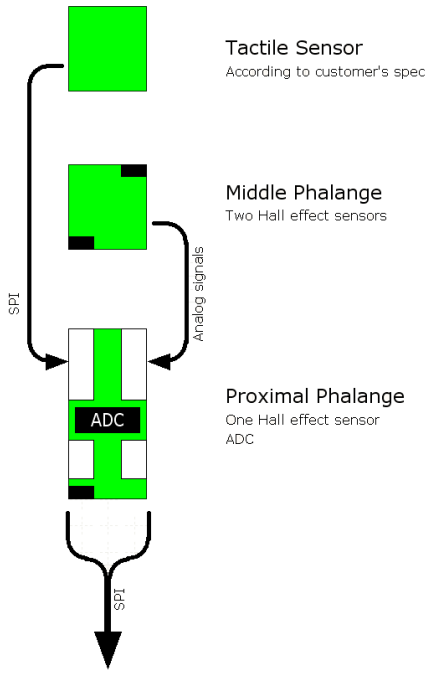
7.2.2. Palm

The palm acts as the main data hub, dealing with all EtherCAT, CAN, and SPI communication.

7.2.3. Fingers

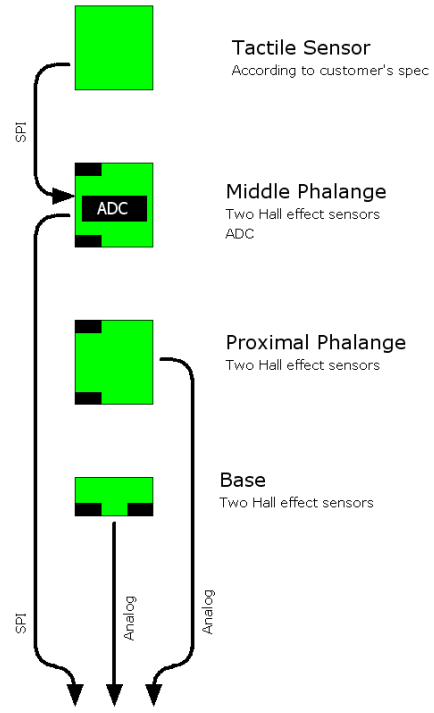
Each finger is identical. It contains four analogue Hall effect sensors which measure the position of each of its four joints. Three of these sensors are sampled by an ADC in the finger, while the fourth (J4) is sampled by an ADC in the palm.

The proximal phalange contains the ADC, which is connected to the palm by SPI. It also houses the connectors for the middle phalange and the tactile sensor.



7.2.4. Thumb

The thumb contains six Hall effect sensors to measure the position of its five joints. The base joint with its large angle range needs two sensors to fully cover the range.



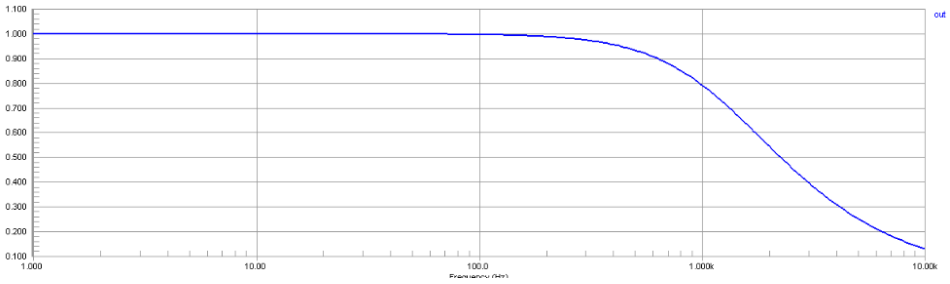
Only the middle phalange of the thumb contains an ADC. The other two phalanges return analog signals which are sampled by an ADC on the palm.

7.2.5. Motors

When in torque control mode, each motor MCU samples its own torque sensor at 5000Hz, and updates the motor PWM signal at the same rate, according to its internal PID control loop. New force demands are sent to the motor MCU every millisecond. When in position control mode, the force control loop is switched off, and the host simply sends PWM demand values to the MCU every millisecond. These are implemented immediately. Requests for sensor data are sent to the motor MCU every 2 milliseconds, and corresponding values are returned immediately.

7.2.6. Analogue sensor filtering

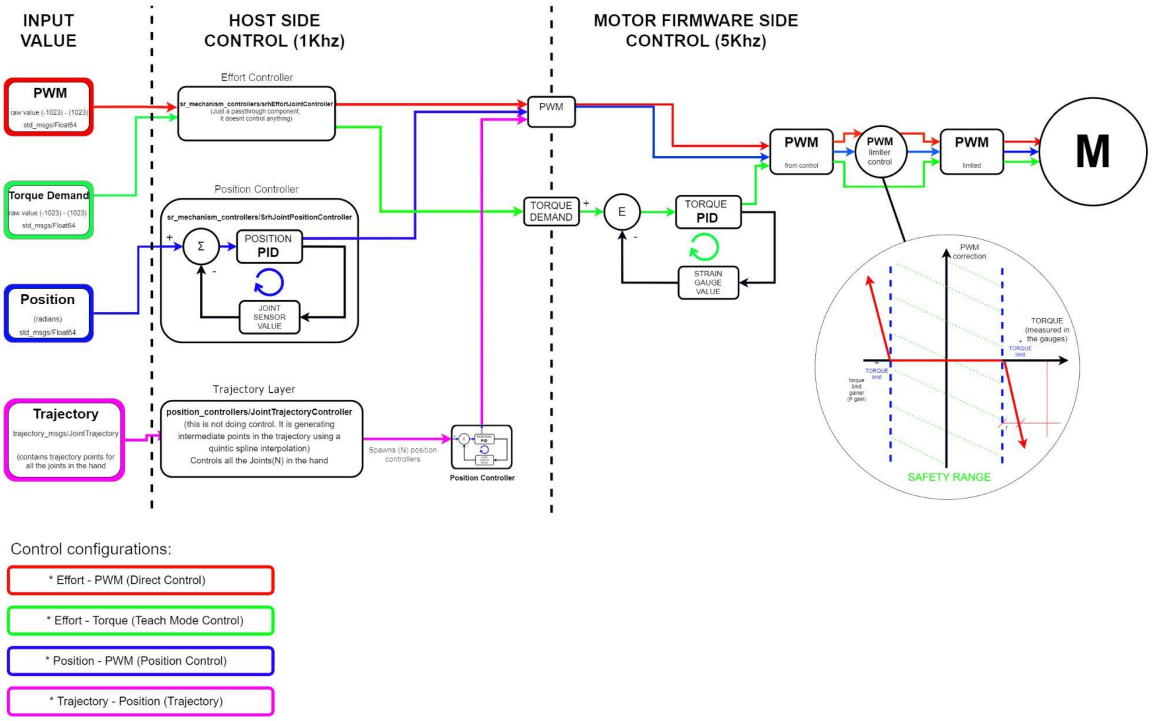
Hall effect sensors naturally produce noise, and so need a simple RC filter. The filter produces a flat response to 100Hz, allowing for an undisturbed joint angle signal up to rotation speeds of 90° per 0.025sec.



7.3. Control Description

7.3.1. Control Implementation

Host side loop (1KHz)	Motor side loop (5Khz)	Secondary Motor loop (5KHz)
“P”, “I” & “D”: Control gain parameters	“F”: Torque demand fed forward to the output	Torque_limit: Control torque demand limitation
“Max_force”: Output limitation	“P”, “I” & “D”: Control gain parameters	Torque_limiter_gain: Proportional term parameter
“Position_Deadband”: Band where error is considered zero	“Max_PWM”: Motor PWM limitation	
	“Deadband”: Band where error is considered zero	
	“Sign”: Define the flex and the extend tendon	



7.3.2. Control Options

PWM	Teach Mode	Position	Trayectory(*)
Control input: PWM demand	Control input: Torque (Effort) demand	Control input: Position demand	Control input: Position demand + time (*) Position Control with the addition of one algorithm on the top which splits the position target into a collection of points, creating a spline which controls the speed of the joint
Input refreshment: 1kHz	Input refreshment: 1kHz	Input refreshment: 1kHz	
Implemented in: Motor side	Implemented in: Motor side	Implemented in: Host side	
Control output: PWM demand	Control output: PWM demand	Control output: PWM demand	
	Control loop: 5KHz	Control loop: 1KHz	
	Sensor feedback: Motor strain gauges	Sensor feedback: Joint position sensors	

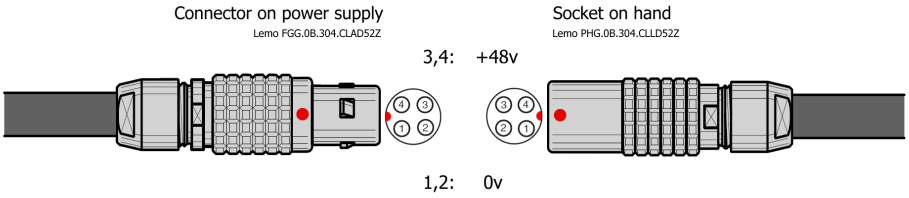
8

Connectors and Pinouts

- *External Connectors*
- *Internal Connectors*

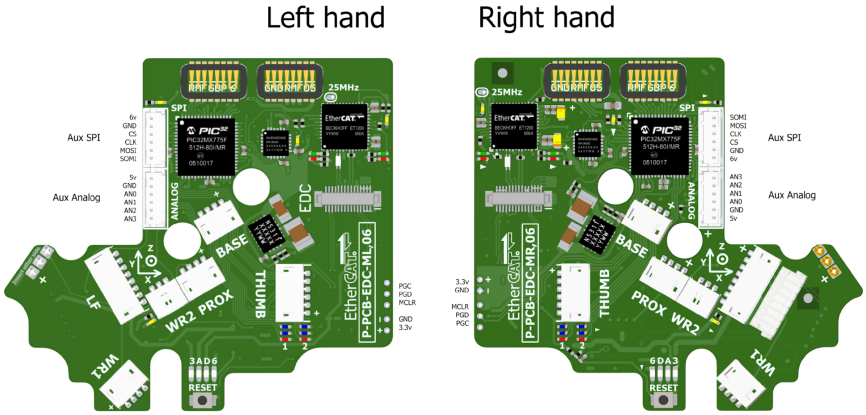
8.1. External Connectors

8.1.1. Power



8.1.2. Peripheral Connectors

The following connectors are available for connecting external peripherals.



Auxiliary Analog: This connector allows you to add up to four extra analog sensors to the hand. These sensor channels are always sampled at 1000Hz, and are available at the host. They are currently pub-

lished in /debug_etherCAT_data/sensors[26 .. 28]. To assemble a connector for this socket, use the following part numbers:

Harwin M30F1100600, Harwin M30-1010046.

1. 5v regulated supply
2. GND regulated supply
3. Analog input channel 0
4. Analog input channel 1
5. Analog input channel 2
6. Analog input channel 3

Auxiliary SPI:

> **Warning:** The Auxiliary SPI is not supported by the current firmware version of the hand. We are working on this issue and will add the functionallity in future releases.

An external SPI device may be connected here, e.g. an ADC, DAC, or I/O expander. The palm may be able to auto-detect the type of device connected, and inform the host. Currently, the palm supports only three devices:

- MCP3208 - 8 channel, 12-bit ADC
- MCP3204 - 4 channel, 12-bit ADC
- MCP3202 - 2 channel, 12-bit ADC

The auto-detection of these devices is not completely reliable, since the MCP320x chips have no explicit autodetection mechanism. The palm attempts to read all eight channels. If it reads values other than 0x000 or 0xFFF, it assumes an ADC exists. Therefore, if the analogue sensors are giving exactly 0v or 5v on every channel, the palm may fail to autodetect.

If the palm fails to autodetect a device, it assumes the device is an MCP3208.

To assemble a connector for this socket, use the following part numbers: Harwin M30F1100600, Harwin M30-1010046.

1. 6v supply
2. GND supply
3. Chip Select
4. Clock
5. Master out, Slave In
6. Slave Out, Master In

6v supply

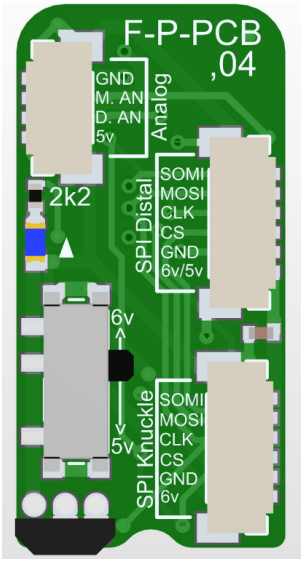
ICD3 Socket: This allows you to re-program new firmware into the Palm's PIC32 MCU. See the section on re-programming the palm. You have been supplied with an adaptor to connect the ICD3 programmer here.

8.2. Internal Connectors

The following connectors are detailed here for your information, but are not available without dismantling the hand.

8.2.1. Fingers

The connectors in the fingers are can be exposed by removing the four screws on the back of the proximal phalange, using an orange hex driver. The board is available in two versions



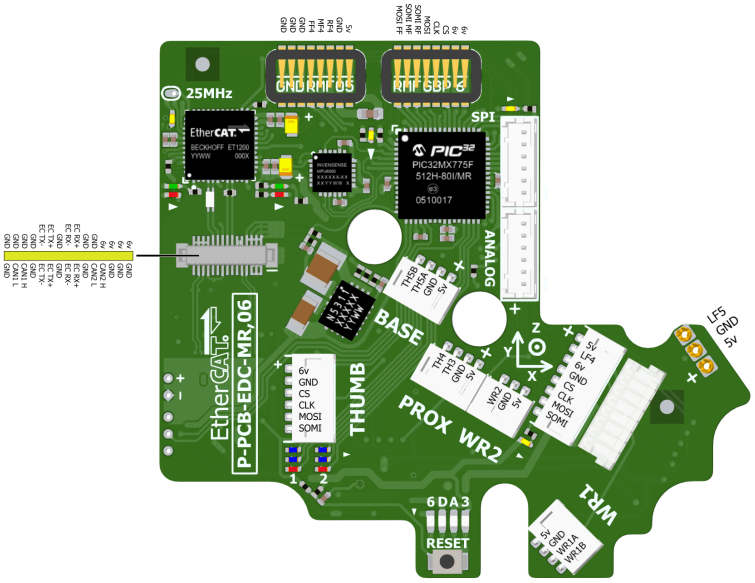
SPI to Palm: This consists of two 3-way connectors, and connects the Palm MCU to the finger.

MID Analog: This connects the two Hall effect sensors in the middle phalange to this PCB.

SPI to Tactile Sensor: This connects the tactile sensor to this PCB. The plug is JST 06SUR-32S. I/O signals are expected to be 3.3V compatible. The 5V/6V selector switch sets the power supplied to the Tactile Sensor

Tactile Sensor Connector: This is an analogue input, accepting three resistive sensors.

8.2.2. Palm



EtherCAT Dual CAN connector: This connector supplies power to the palm, as well as connecting the EtherCAT LVDS and both CAN buses.

9

Maintaining the system

- *Mechanical Maintenance*
- *Electronic Maintenance*
- *Reinstalling The Software*

9.1. Mechanical Maintenance

9.1.1. Inspecting for wear and tear

There are three types of wear and tear that might happen to your hand over the course of its use. **Settling:** Parts and fittings may settle over time. In particular, tendons may stretch slightly, and the spools on the motors may shift, increasing the amount of slack in the system. This can be remedied by re-tensioning the tendons. Hall effect sensors may shift in position slightly. This may be remedied by re-calibrating the joint.

Wearing: In use, some tendons are subject to wear. In particular, wherever a tendon travels over a pulley, it may show signs of wear, appearing slightly fluffy. Normally, this isn't a problem, but if excessive wear is apparent, then the tendon might need to be changed.

Failing: As many parts of the hand are subject to very high loads at times, it is possible for some parts to fail. The exact symptoms of the failure depend on the particular part in question. Most such failures will need to be fixed by service personnel.

9.1.2. Re-tensioning

From time to time, you may find that the amount of slack in the tendons has increased, causing backlash and deteriorating the control. The tendons can be re-tightened at the motor.

Look at the Motor Layout diagram in this document to find the position of the motor that needs tightening. Using the green hex driver, remove the black motor cover; you can unplug the fan.

1. Insert the blue hex driver through the hole in the wrist plate, and push it down until you feel it engage with the hex socket in the screw in the motor spool.
2. Rotate the hex driver clockwise a little way. This will tighten the left tendon, and loosen the right tendon.
3. Insert the spool-tool into one of the 2mm holes in the top of the spool.
4. Use the hex driver to loosen the screw. The top of the spool should now be free to rotate.

5. Use the spool-tool to rotate the top of the spool to the right. You should see the right tendon tighten as you do this.
6. Re-tighten the bolt using the hex driver.

9.1.3. Broken tendon

Fixing a broken tendon can be easy or extremely difficult depending on the joint involved. Complete instructions for replacing tendons is beyond the scope of this document. Please contact the Shadow Robot Company if this happens support@shadowrobot.com.

9.1.4. Tangled tendon

If a pair of tendons has become extremely slack, then they may become entangled at the spool. If this happens, rotate the motor until they are no longer entangled, and follow the procedure for re-tensioning.

9.1.5. Re-calibrating

The calibration tables in the host allow it to convert from the raw ADC readings sent by the Palm, into real angles in radians required by ROS. It is possible that eventually some joints may need to be re-calibrated. If this is the case, please contact Shadow Robot support@shadowrobot.com,

9.2. Electronic Maintenance

9.2.1. Strain Gauge failure

If one strain gauge appears to have failed, it is possible that the connector has come loose. Swing out the motor, and examine the connectors. Re-insert with a small pair of tweezers. If this was not the cause of the problem, please contact Shadow Robot for advice sending an email to support@shadowrobot.com.

9.2.2. Position sensor failure

If a position sensor appears to be returning incorrect values, or a constant value, first check that the raw sensor data is working: rostopic echo /debug_etherCAT_data/sensor[?] You can also check the state of the raw sensor from the Shadow Hand Calibration plugin in the GUI.

9.3. Reinstalling The Software

Note: You will have received the laptops and NUC with all the software installed and ready to use.

However, If you need to reinstall the software, please use the one-liner given to you in the "Delivery Instructions" sheet. If you want to upgrade your installed product version please contact support@shadowrobot.com and we will provide an updated one-liner.

10

FAQ & Changelog

- *Hand Frequently Asked Questions*
- *Why Do We Use A NUC?*
- *Changelog*

10.1. Hand Frequently Asked Questions

A list of common issues and how to resolve them.

10.1.1. 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

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.

10.1.2. Software

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

A: Check that the Hand is powered on and that the Ethernet cable is connected. Go to the "Shadow Advanced Launchers" folder on the desktop and click on the "Launch NUC Container" icon. This will open a terminal window. Type the following command:

```
sr_hand_detector_node
```

If the Hand is connected, you should see a message saying that a Hand is connected and the ethernet adapter is connected to. If the Hand is not connected, you will see a message saying that no Hand is detected. If this is the case, unplug and replug the Ethernet cable and 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

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

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

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 Aurora command following your Delivery Instructions. If you have any questions, please contact us at support@shadowrobot.com.

Q: Nothing happens when I click on the icons on the desktop.

A: This could be because the NVidia driver is not correctly installed and configured. Firstly, confirm the correct driver in installed by opening the Additional Drivers application. Confirm the driver used is the NVIDIA driver metapackage from nvidia-driver-510 (proprietary). If this is not the case, select this driver, click Apply Changes and restart the computer. If this is the case, 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.

10.1.3. 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 support@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.

10.2. 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.

10.2.1. Testing the overuns

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.

10.2.2. 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.

10.2.3. 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.

10.3. Changelog

10.3.1. ROS Noetic

Version 1.0.29 (Aurora 2.2.4) Current Noetic Release

- Removing static UR paths
- Adding axis scaling depending on noise
- Bug: Fix world scene creation
- Normalising velocity scaling and acceleration scaling parameters consistent among hand(s) & arm(s) combination
- Versioning universal robot dependencies
- Adding display of TF by default unenabled and RobotModel
- Bug: Teach mode node initialisation fix
- Add type hinting and tactile detect prevention
- Adding robot safe store model, worlds and scenes
- Adding support for new stand
- Adding export of sr_hand_finder (and sr_arm_finder) libraries such that they can be resolved externally

Version 1.0.28

- Removing static UR paths

Version 1.0.27

- Bug: Fix hand inertia values (Little Finger metacarpal)
- Increase timeout for joint states in teach mode node
- Add missing catkin package angles
- Updating demos

Version 1.0.26 (Aurora 2.2.0) - Previous noetic-release

- Enable autodetection params determine the pack position for different hands
- Updating the controller stopper node and urscript file location
- Updating demo (for release)
- Updating robot states service and increasing demo sleep time
- Adding E4 to the list of correct options

Version 1.0.25

- Changing demo scripts
- Fixing bug introduced by lint changes that broke the generation of group states for multi-robot move groups.
- Updated the close and open demo
- Fixed issues with sr ur unlock test
- Changed accepted joints for hand_extra_lite
- Adding new position-only fingertip IK example
- Changed finger in hand_lite grasp demo causing KeyErrors
- Fixed how sr_bimanual.launch deals with product types
- Update ports for ur fix
- Enabling coverage in robot commander
- Bug: Removing the axis from the link visual and collision. The join deals with the axis not the link.
- Add latest MST ROS message
- New version E4: Includes new meshes for lfmetacarpal and palm
- Removed thumb distal material
- Fixing relative lfknuckle position
- Fixing dae files
- Fixing hand e urdf convention
- Update hand icon
- Added Timeout to calibrate_hand_finder
- Adding getters for private variables in HandFinder
- Added a rosparam and launch file to run calibration script.

- Update hand icon
- Fetching changes from upsteram
- Update known_good_firmware.txt
- Fixing arm calibration loader.

Version 1.0.24

- Fixing bootloader
- Fixed issue with bootloader GUI when in server_and_nuc deployed scenario
- Added motor check
- Change toolset branch
- Fix licence and linter changes
- Demo_A calibration and tuning update.
- New Calibration.
- Fix snapping to the next ms period after an overrun. By making the period be an integer, the existing algoritm works wellGitHub

Version 1.0.23

- Migrating config to autodetection

Version 1.0.22

- Fixing bootloader

Version 1.0.21 (Aurora 2.1.6) - previous noetic-release

- Used for release

Version 1.0.20

- Fixing TF re-publisher to prevent REPEATED_TF messages
- Updated and fixed issue with error reported by bag_rotate node
- Updated bag rotate
- Corrected order of fingers moving when using demo icons

Version 1.0.19

- Update known good firmware file - UR10 arms
- Fixing rosbag rotate

Version 1.0.18 (Aurora 2.1.5) - previous 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 teach mode 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 spawningFixing 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

10.3.2. 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 Noeticoetic
- 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 rostest
- Commenting trac_ik and replacing it to kdl until it is available in Noeticoetic

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 rostest
- 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 repositor
- 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

10.3.3. 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

11

Support & Teamviewer

- *Support And Teamviewer*
- *Shadow Backup USB Stick*

11.1. Support And Teamviewer

11.1.1. Support Contact

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

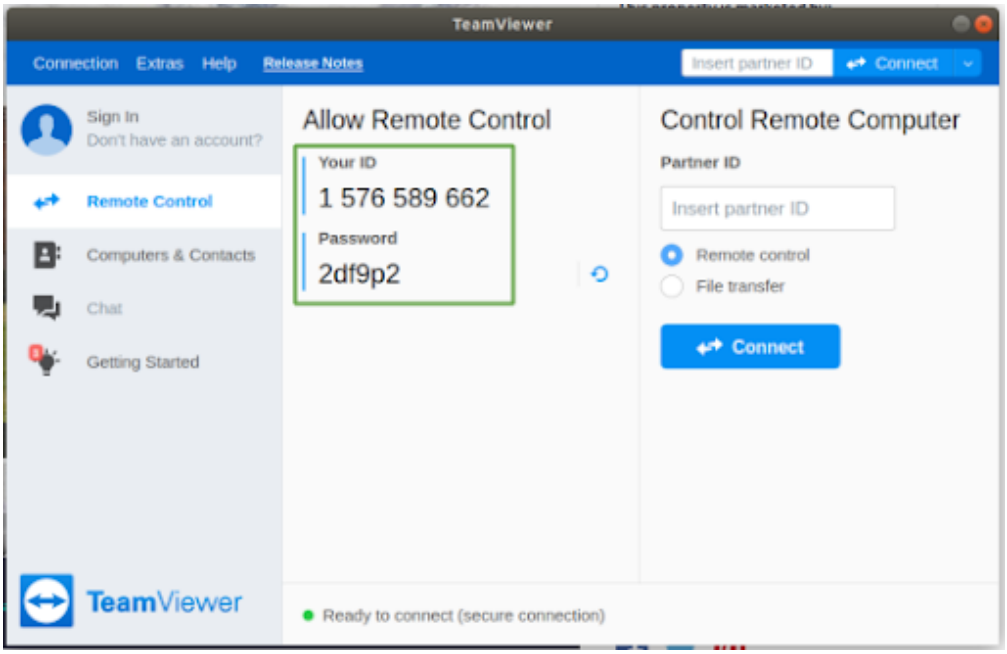
11.1.2. Teamviewer

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.

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

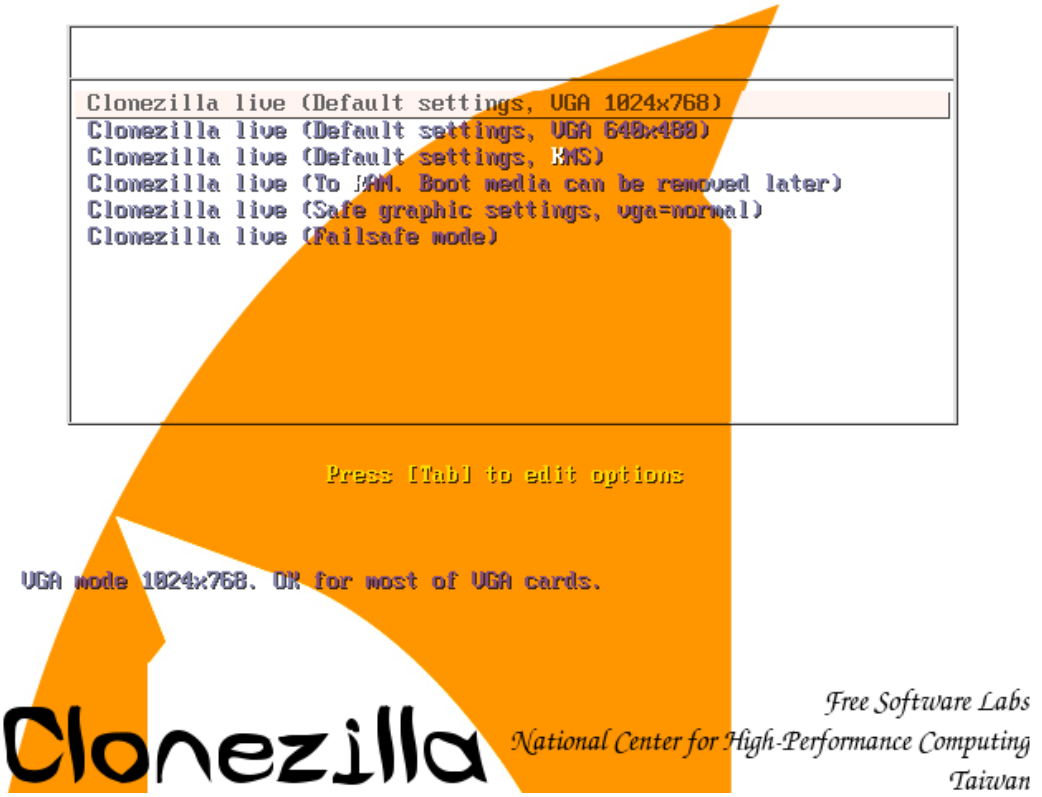
If you are restoring a NUC backup, you will need additional equipment (not provided by Shadow): A monitor with HDMI connector, an HDMI cable and a USB keyboard.

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.

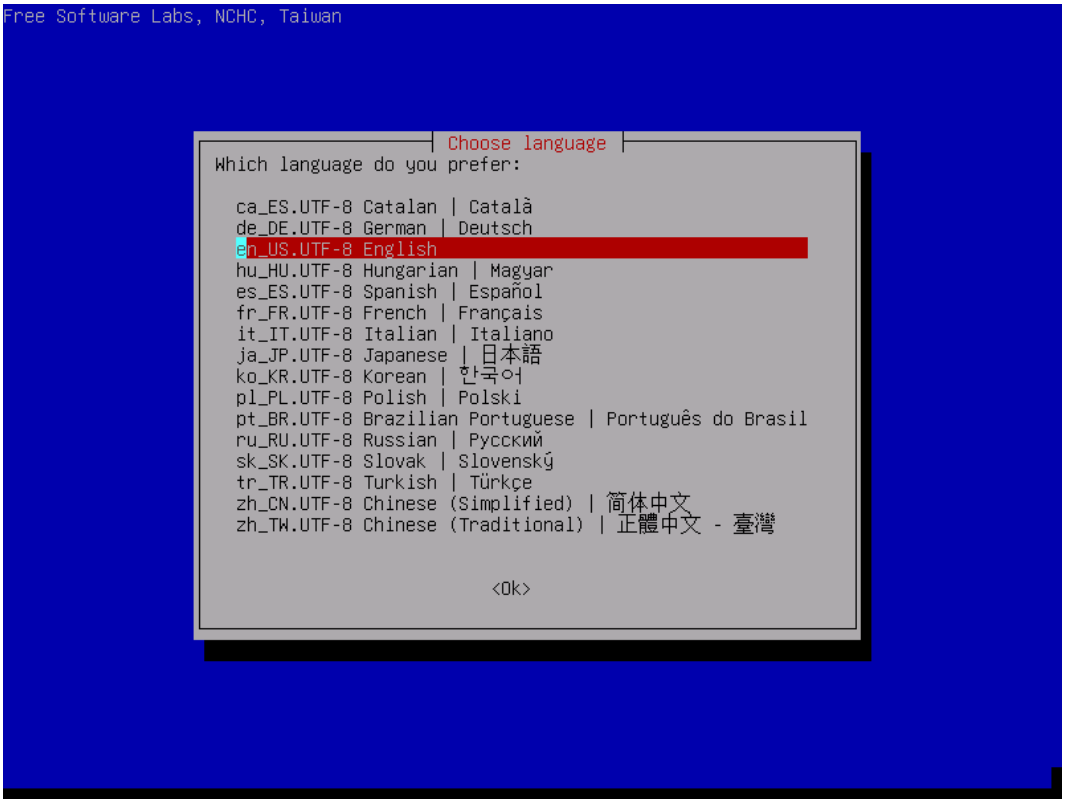
You don't have to restore both NUC and laptop. You can choose to restore just 1 of them.

11.2.1. Clonezilla backup restore steps

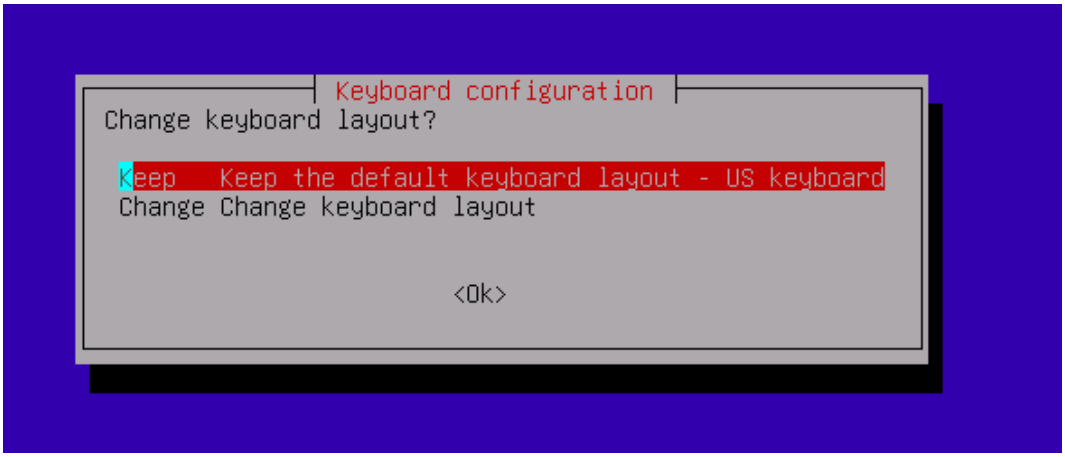
1. Connect the Shadow USB stick to the device you want to restore
2. Power on the device while tapping F7 and F8 on the keyboard. A blue dialog box will appear asking you which disk to boot from (if this doesn't work, try tapping F2/F12 and accessing the boot menu/boot override via BIOS): select the 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



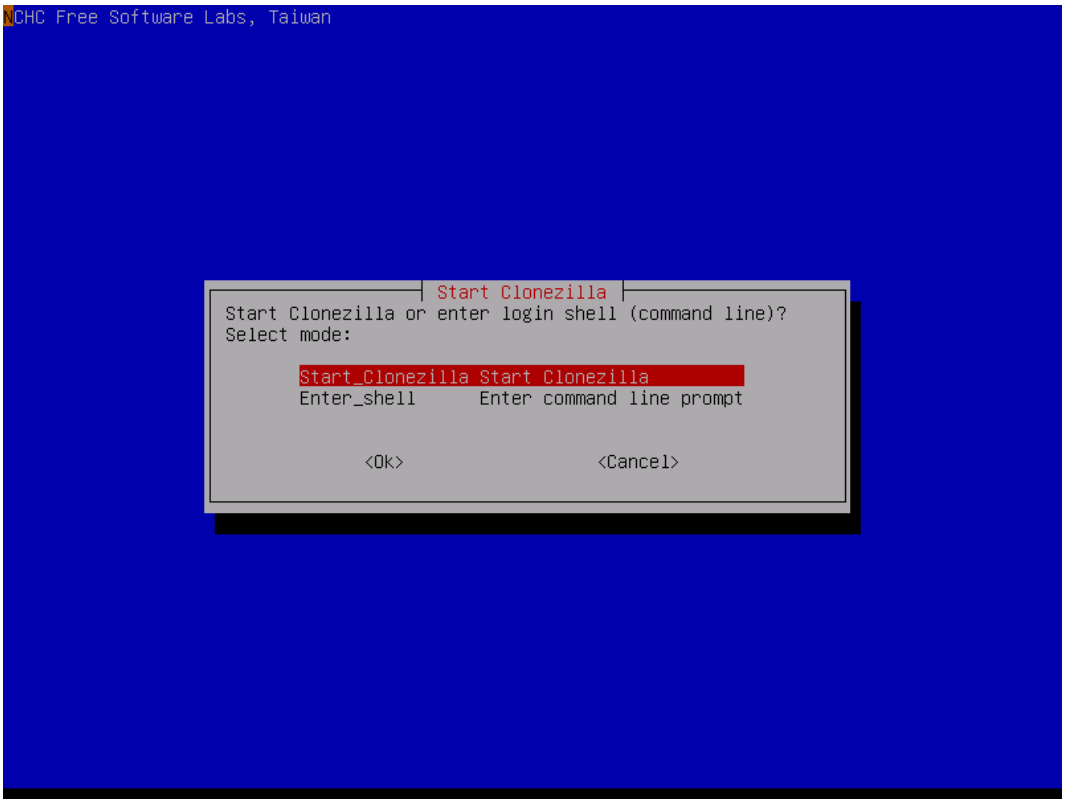
4. Select Clonezilla live (Default settings, VGA 1024x768)
5. In the next menu, select English for the language: (just press Enter)



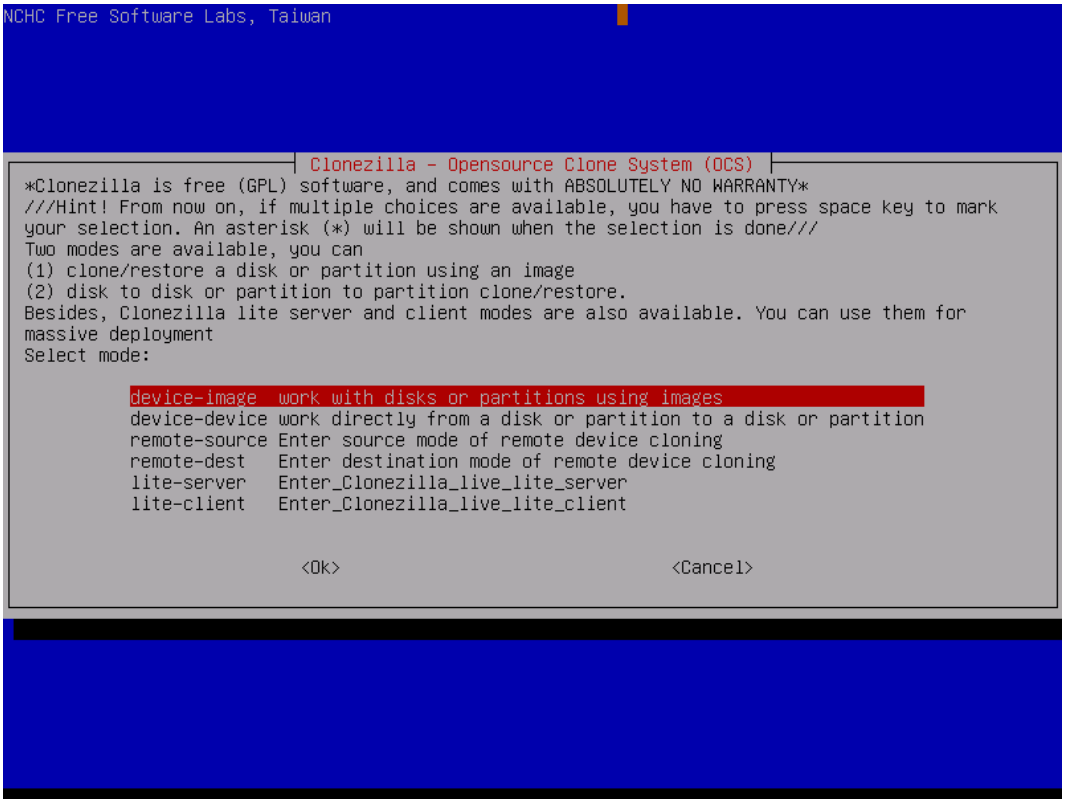
6. In the next screen, the default keyboard layout is US keyboard, and it's fine for our purposes here, so just press Enter



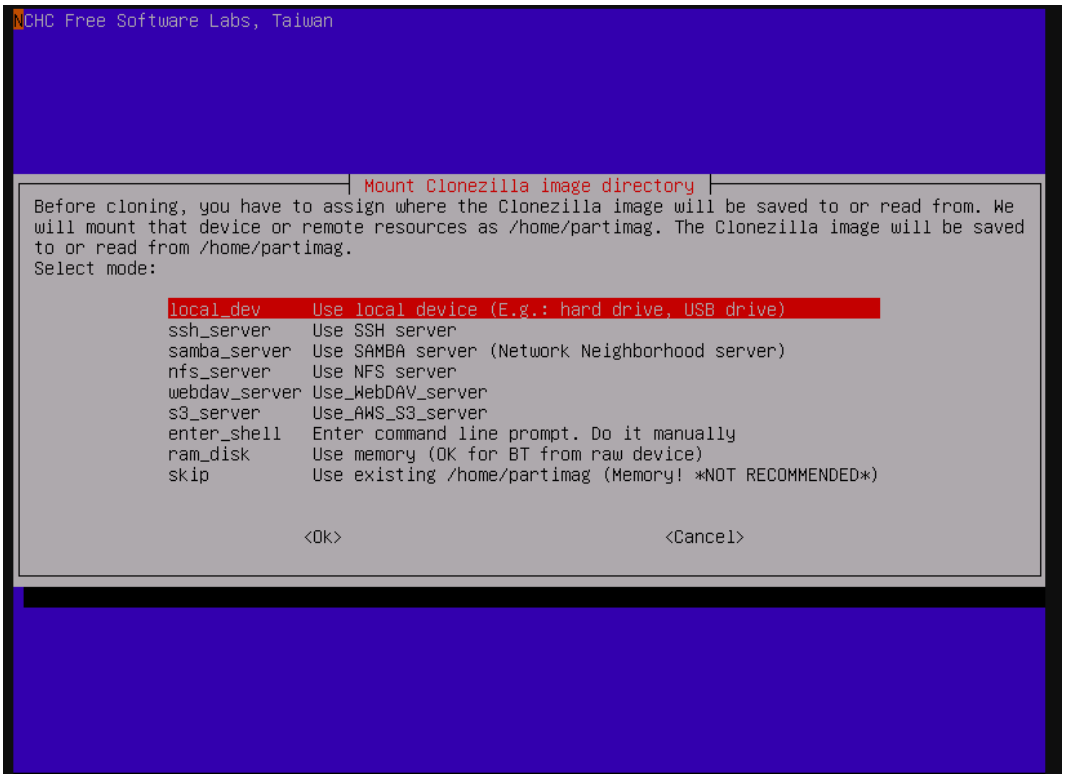
7. Select Start Clonezilla



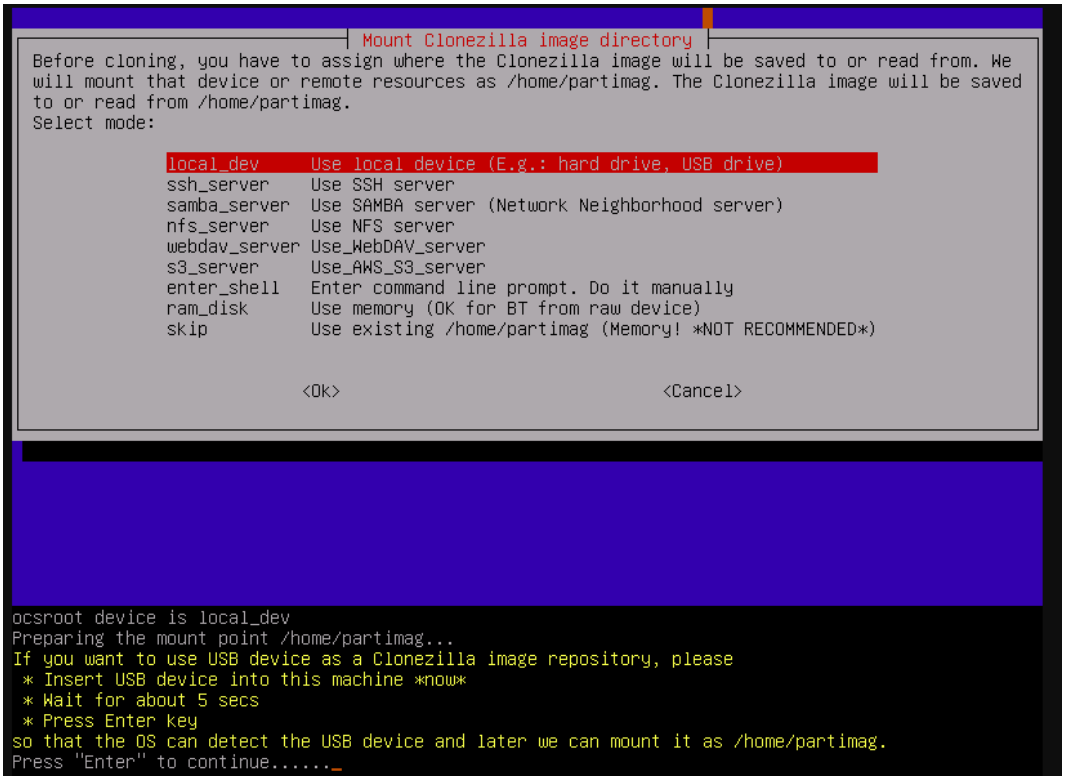
8. Choose device-image



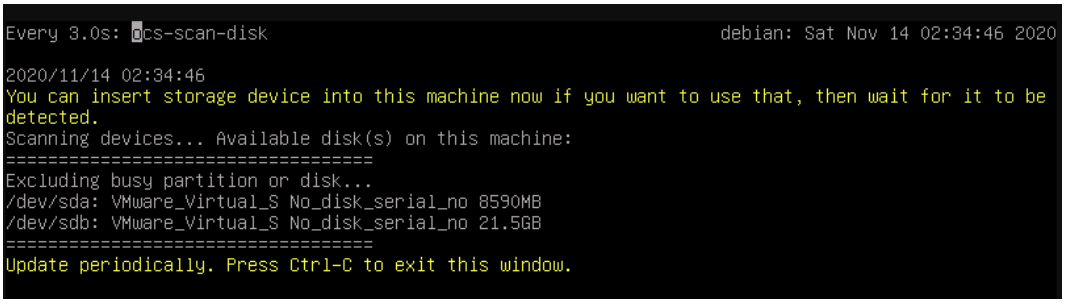
9. Choose local device:



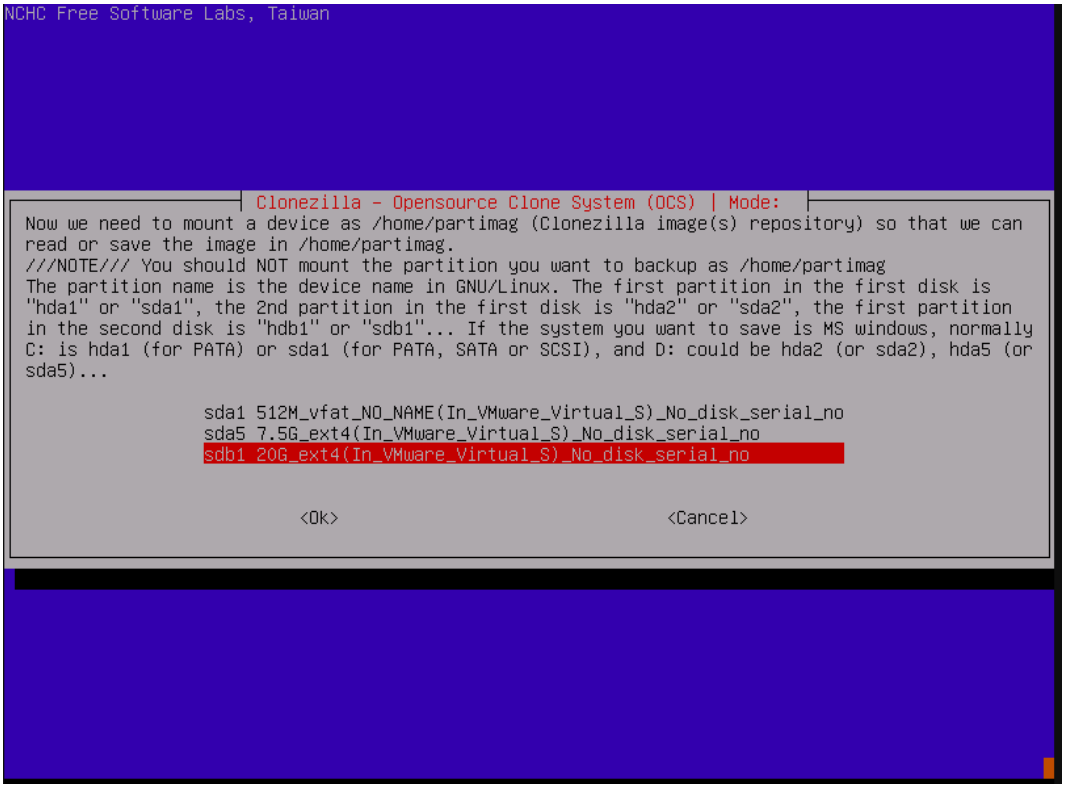
10. Press enter when you see this screen:



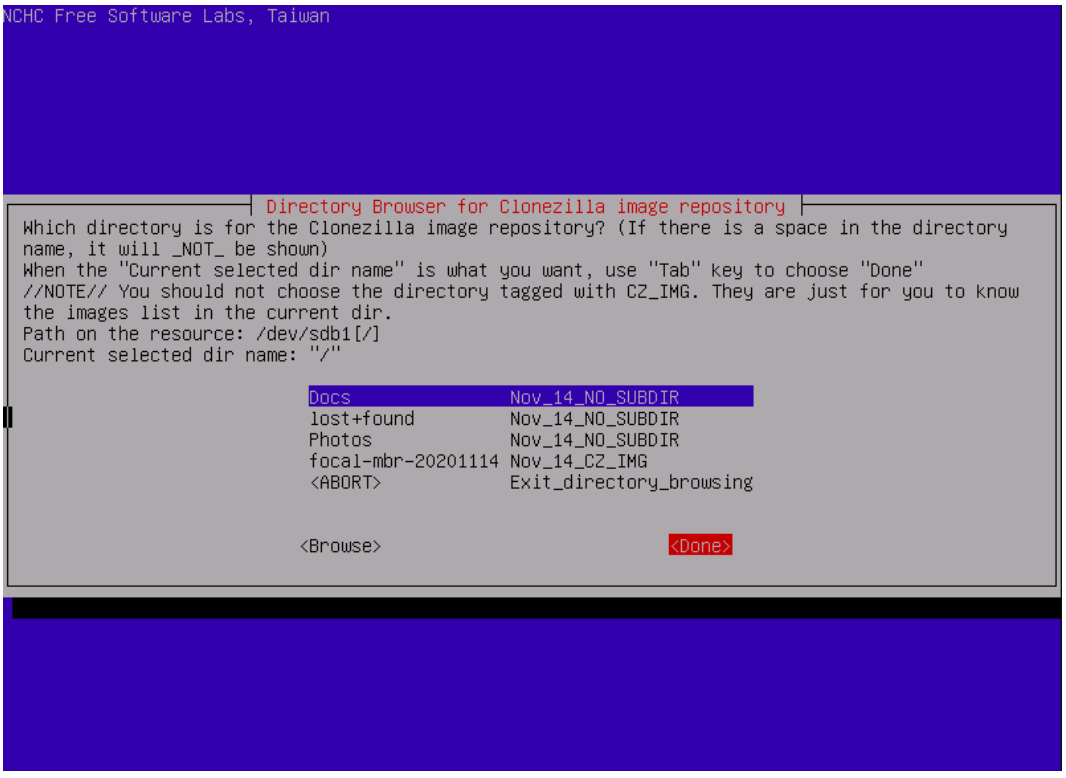
11. If the disk you are backing up from appears in the list on this screen press Ctrl+C otherwise reconnect the drive and wait 30secs for it to appear in the list.



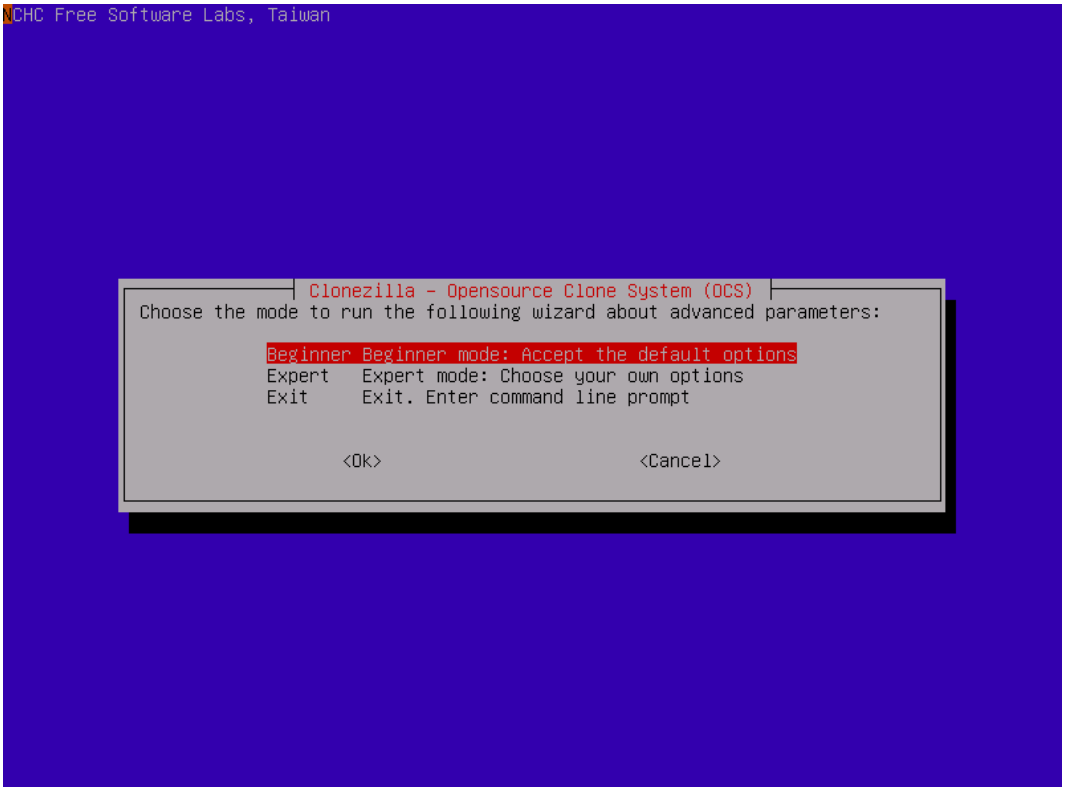
12. You should see a disk menu like this (not the exact image), but you need to select your Shadow Backup USB stick large partition with several GB (where the disk images are) (it might show up as a SanDisk device, or a similar brand matching the model of the USB stick, but with several GB of space)



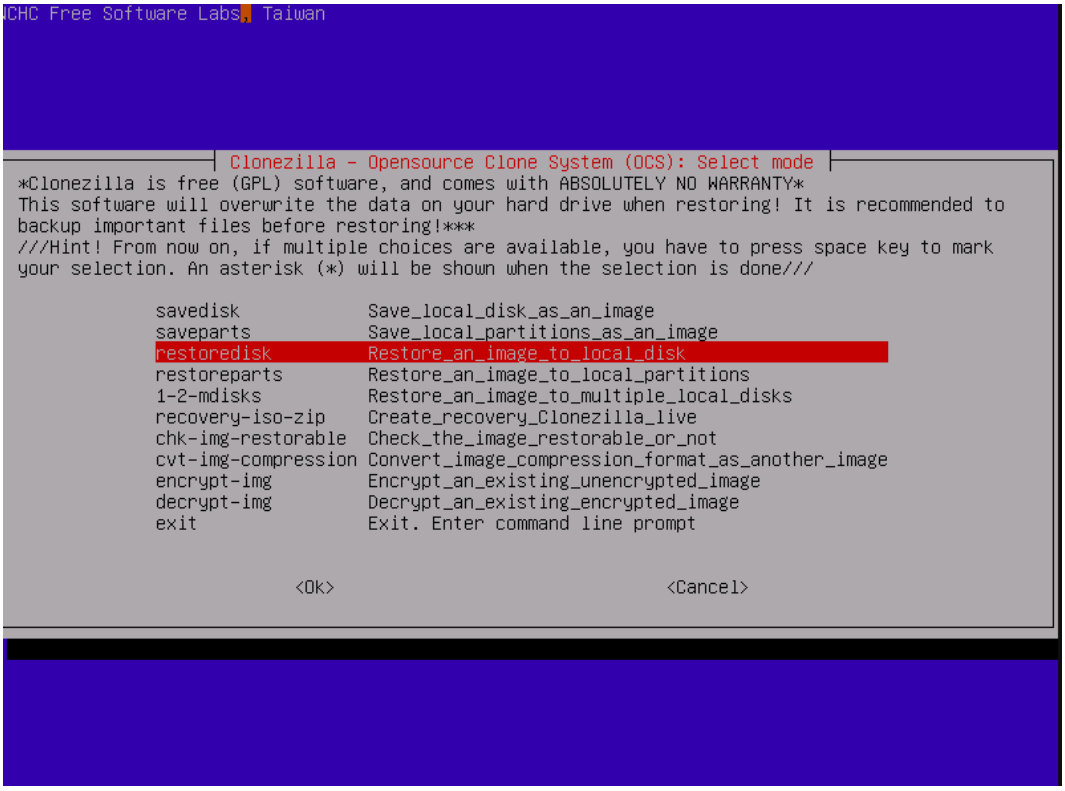
13. In the folder selection screen below you should see 2 folders for the clonezilla images for NUC and laptop, don't select them, just select Done



14. Choose Beginner mode



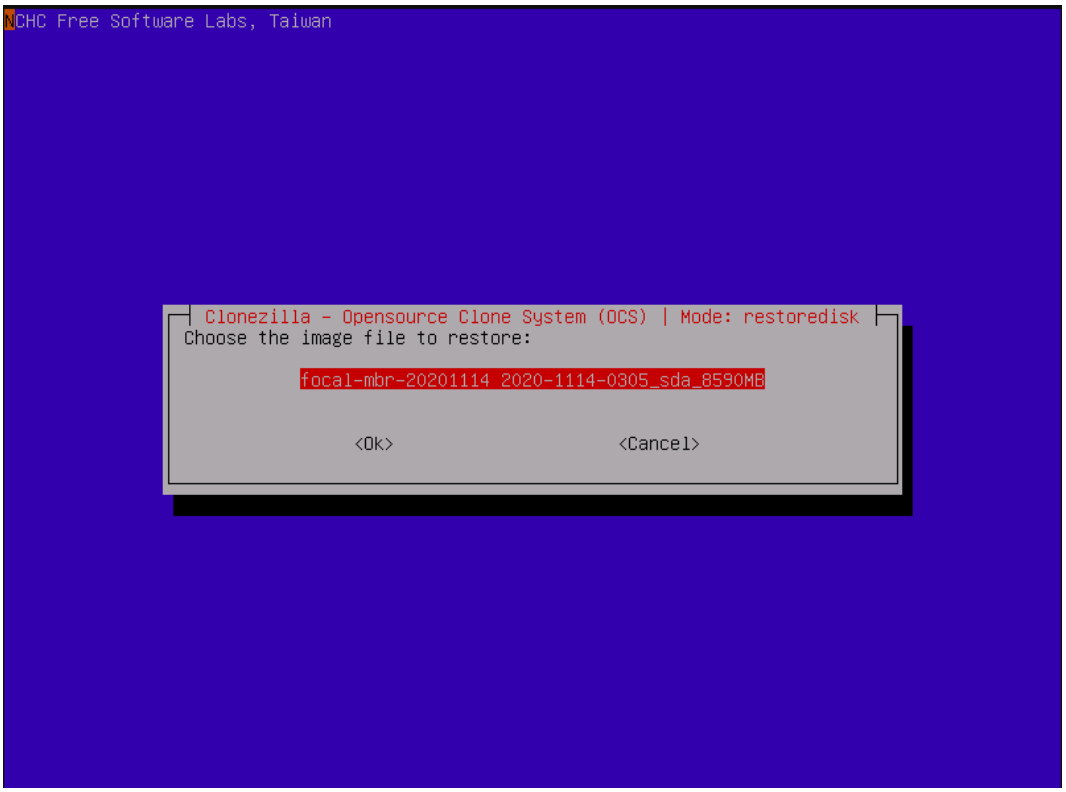
15. Choose restoredisk option



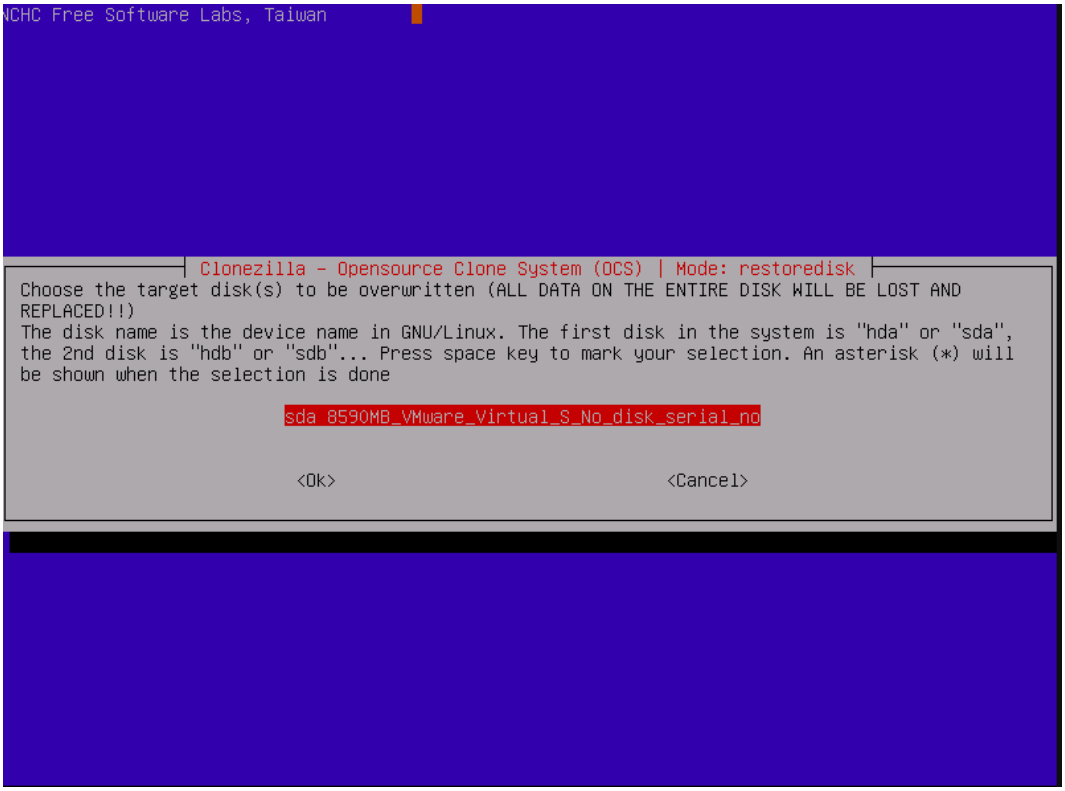
16. Now is the time to select whether you want to restore a NUC image or a laptop image. Depending on which device you have connected the Shadow backup USB stick to, select either the NUC image (may be labelled with your customer name and NUC or NUC-CONTROL and the size of the nuc's internal disk) or the laptop image (may be labelled with your customer name and LAPTOP

or SERVER and the size of the laptops internal disk).

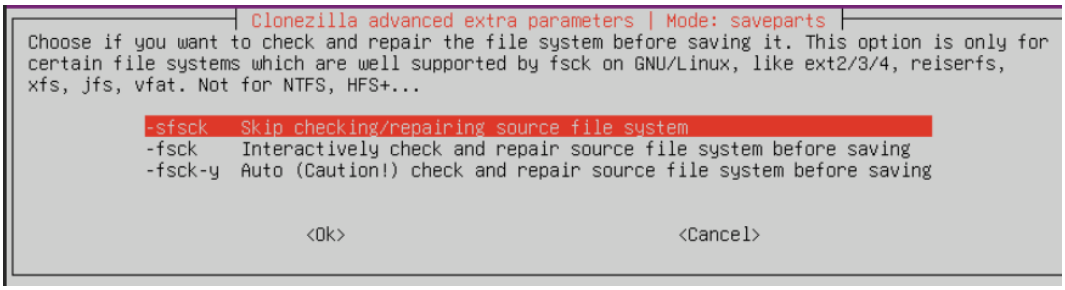
Warning: PLEASE ENSURE YOU ARE RESTORING THE CORRECT IMAGE AT THIS POINT AS IT CAN DAMAGE THE DRIVE OR FAIL ENTIRELY



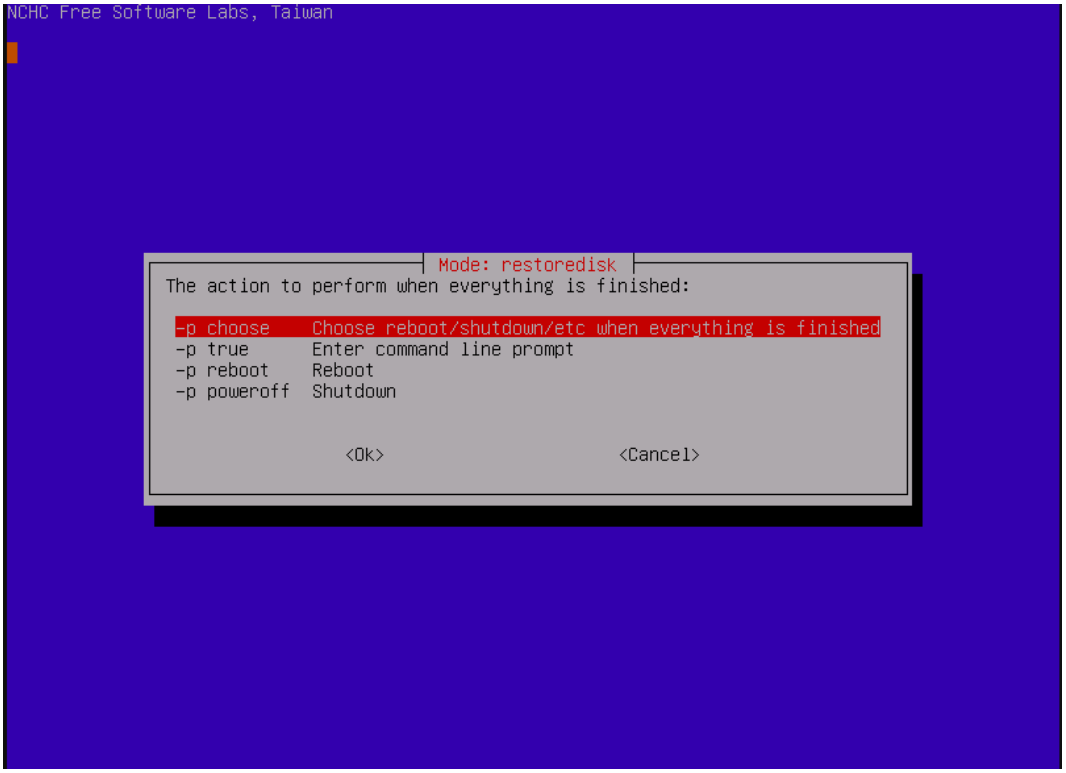
17. Just press enter on the destination disk:



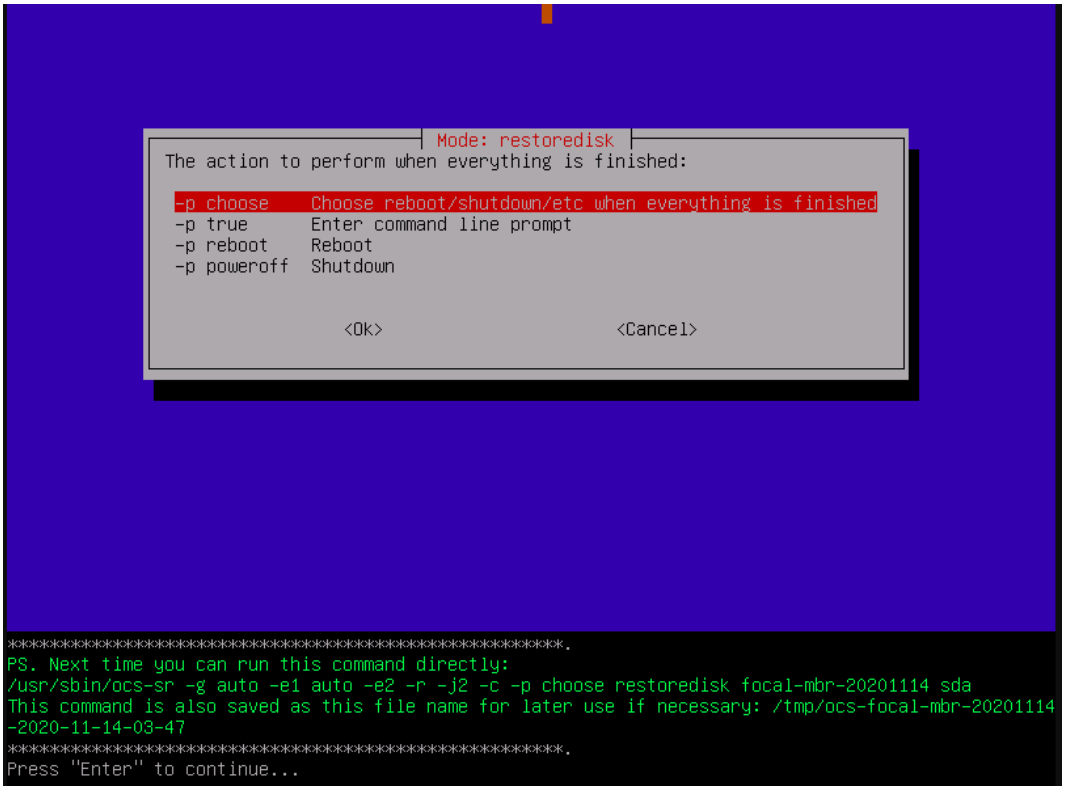
18. Select “No, skip checking the image before restoring”



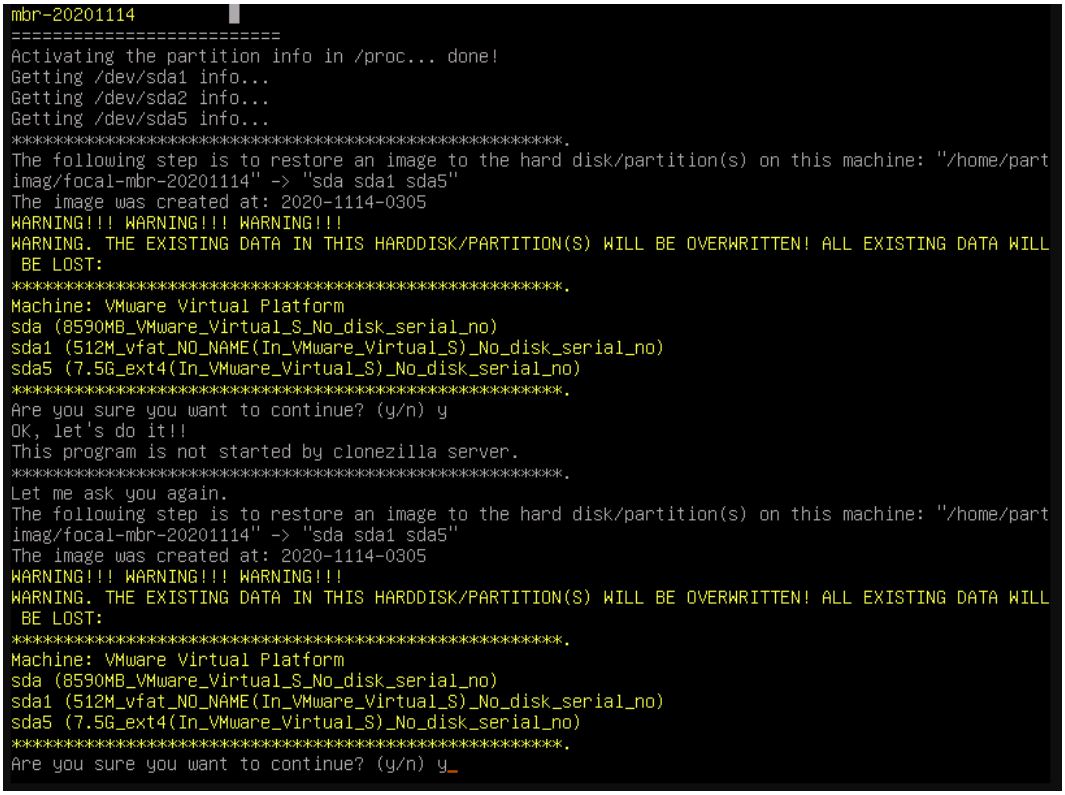
19. Select -p choose



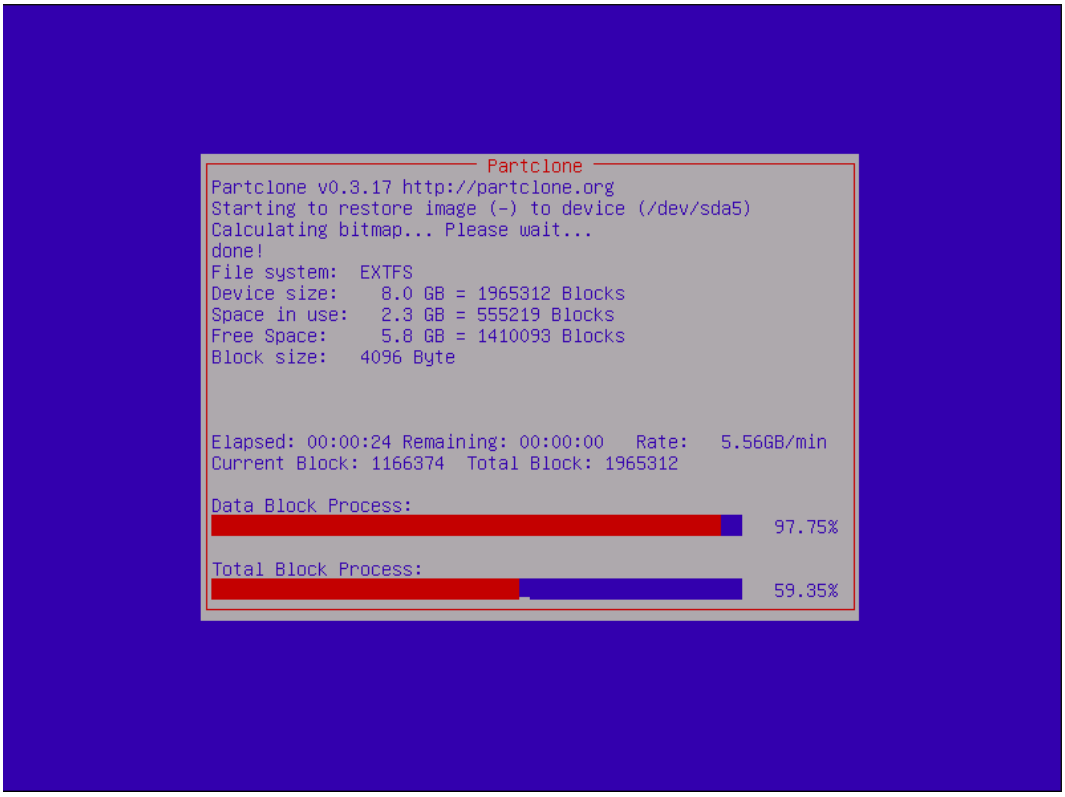
20. Press enter:



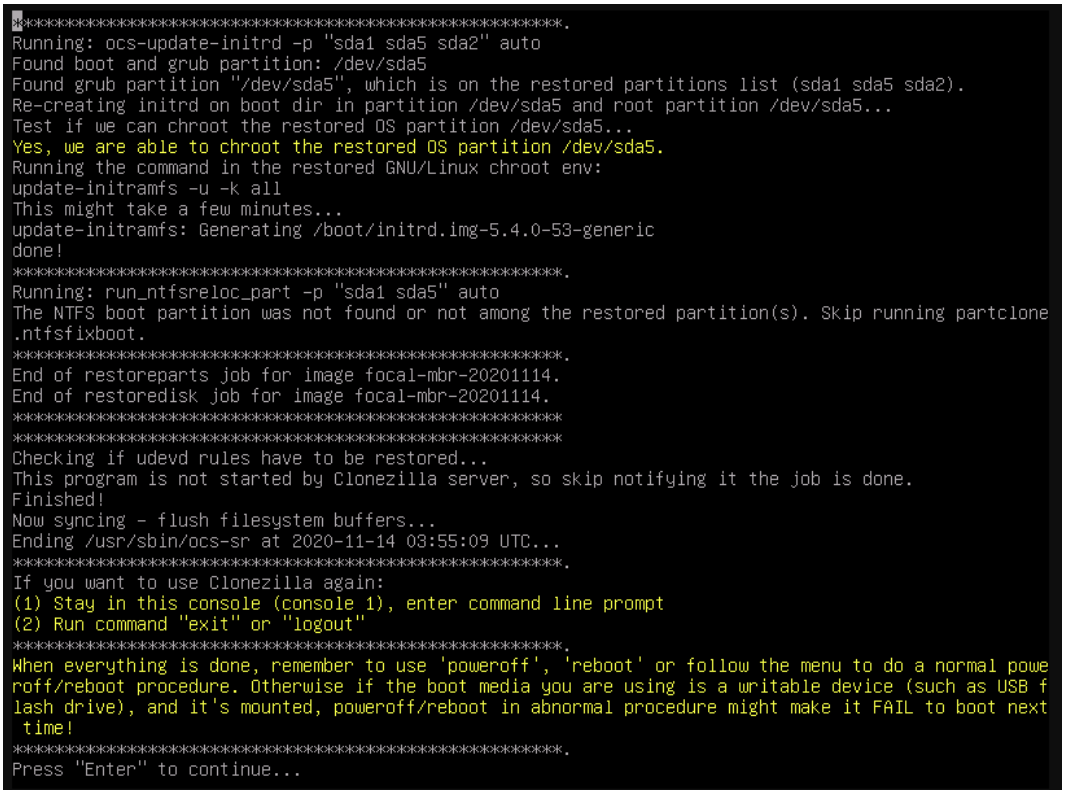
21. There will be 2 prompts and you have to press y and press Enter to each of them



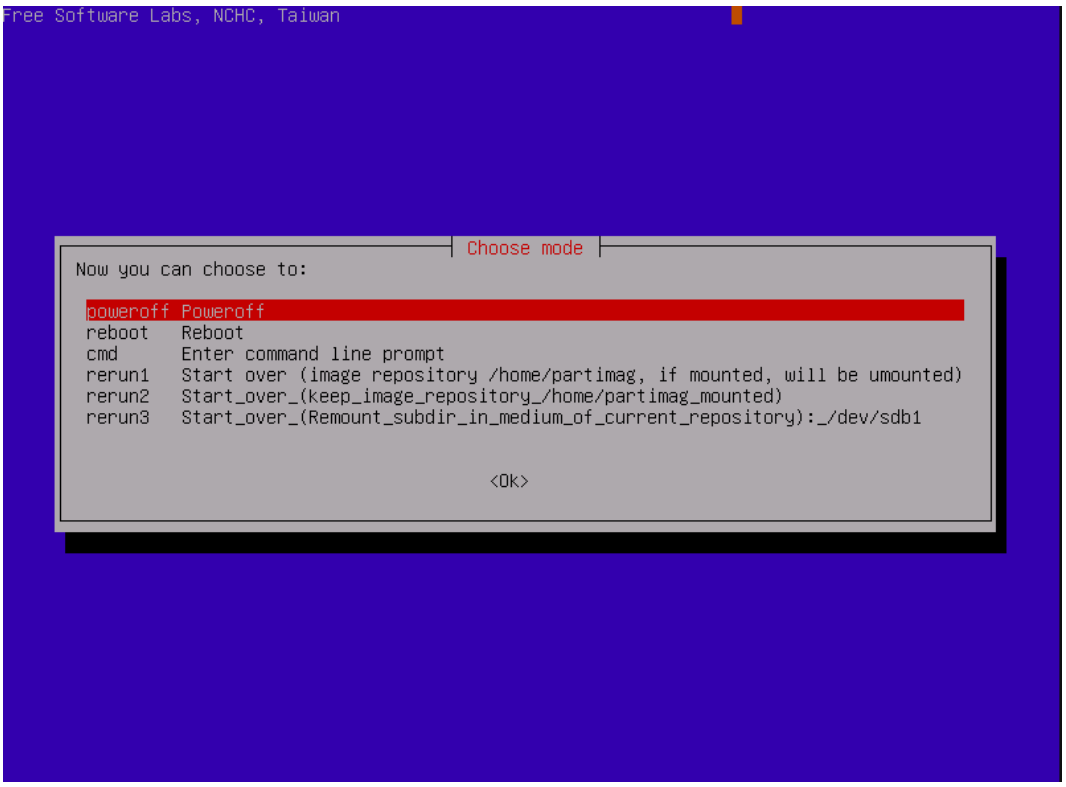
22. Clonezilla is now restoring the device image:



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



24. Choose poweroff



25. Unplug the Shadow Backup USB stick from the device
26. Power on the device. The device is now restored.