Ned User Manual

Ned

Robot

Ned is a 6-axis collaborative robot arm designed for Education and Research.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Ned is designed to reproduce all the movements required in the most advanced uses in industry 4.0, with a **precision of 0.5mm** and a **repeatability of 0.5mm.**

Ned's **aluminum structure** makes it exemplary robust, allowing it to accomplish with fluidity the movements required for your robotics projects.

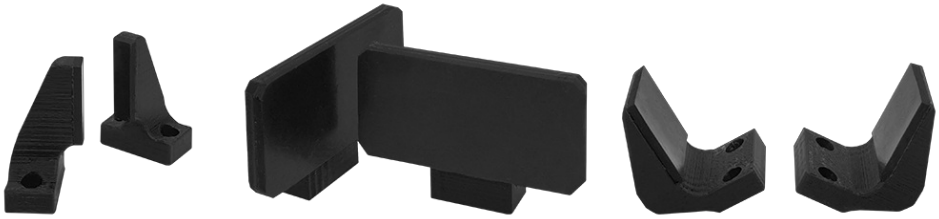
This **cobot** takes advantage of the capacities of the **Raspberry Pi 4**, with a 64-bit ARM V8 high performance processor, 2GB of RAM and an improved connectivity.

Ned is a **collaborative robot based on Ubuntu 18.04 and ROS** (Robot Operating System) Melodic, an open-source solution created for robotics. Through ROS, **Ned has multiple libraries** allowing you to conceive many programs, from the most simple to the most complex ones, responding then, in a flexible way, to your needs.

Ned tools technical specifications

Mounting a tool on Ned is made easy with our brand new **EasyConnect system**. Simply plug your tool, connect its cable, and it is ready to use.

Ned's package includes a **Custom Gripper**. Its standard jaws can manipulate small objects, and you can 3D print your own custom jaws or buy our set of jaws (index.html#overview-standard-jaws) that contains the precision jaws, the flat ones and the XL ones.



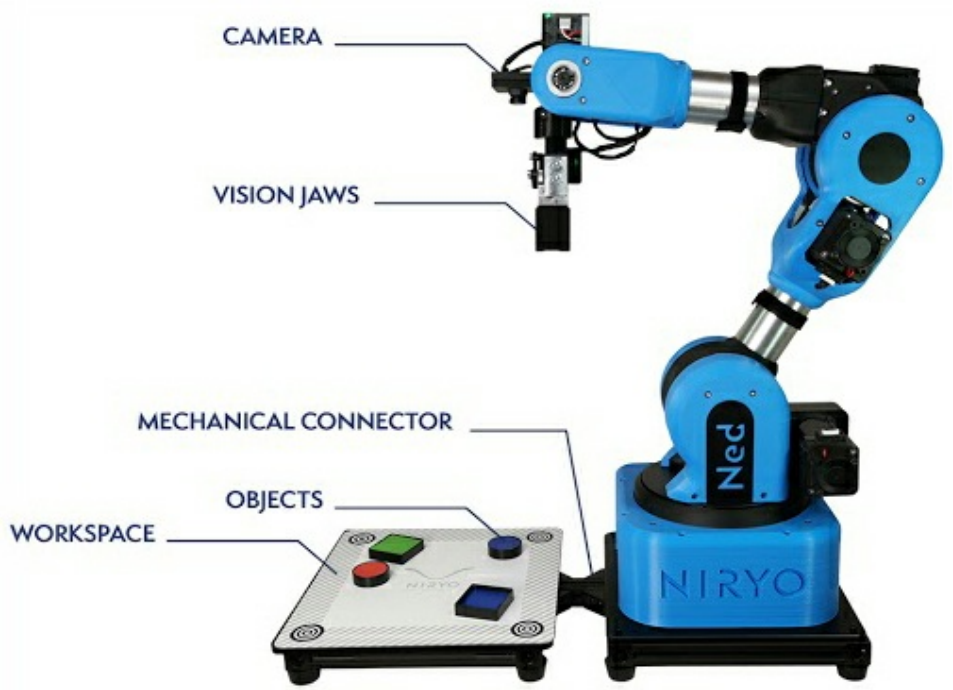
Ned can also be used with:

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

- The Large Gripper (index.html#large-gripper) which has the ability to grab larger objects while keeping the ability to close entirely;
- The Adaptive Gripper (index.html#adaptive-gripper) which allow Ned to grab non-standard objects with complex shapes (eg. an egg);
- The Vacuum Pump (index.html#vacuum-pump) to grab objects with plane and non-porous surfaces;
- The Electromagnet (index.html#electromagnet) that is useful to manipulate metallic objects, from one to many (eg. screws, bolts...).

Ned's Ecosystem is designed to let you reproduce advanced use cases of industry 4.0.

Vision Set



The Vision Set gives Ned the ability to see its environment and detect the objects to interact with based on their characteristics (shape and/or color). This set includes :

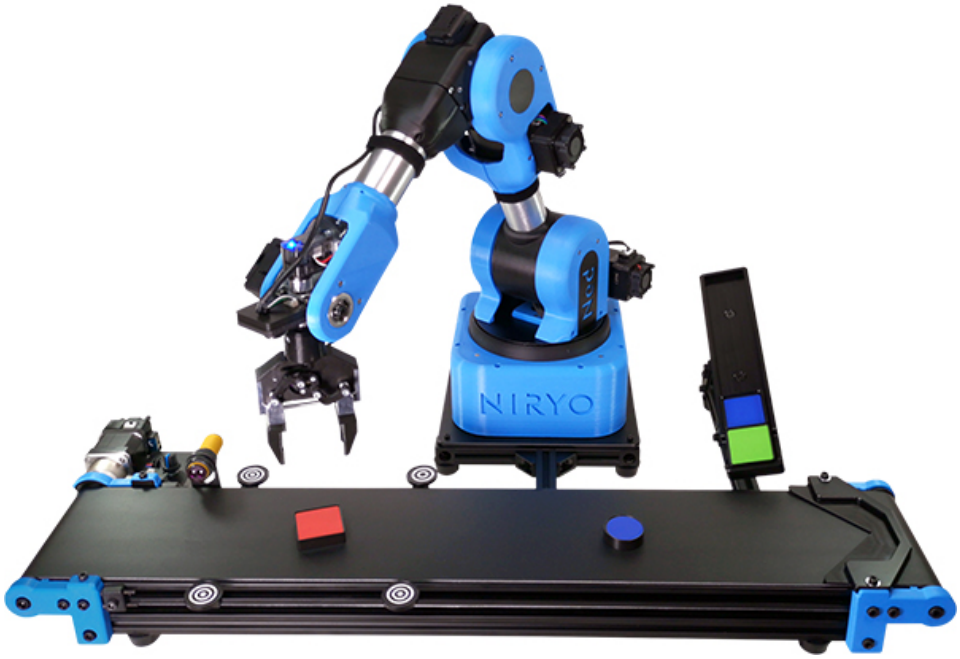
- A camera that is mountable on Ned's wrist to give it a dynamic field of view depending on Ned's position;
- A workspace which is designed around a specific repositionnable material;
- Supports for the workspace and the robot, to build a rigid environment for the vision;
- 6 objects (3 rounds, 3 squares) of different colors to use our built-in recognition system based on colors and shapes. The squares are designed to be able to act as containers for the circles. Easily create an industry 4.0 pick & pack application with no particular knowledge of programming;
- Vision jaws for the Custom Gripper, that are specifically designed for the provided objects;
- A calibration tip to set your workspace up.

Conveyor Belt

The Conveyor Belt is the key add-on to prototype industry 4.0 production lines.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

The **Standard version** includes the Conveyor Belt, its control box for autonomous use, and an IR sensor to detect the presence of an object on it.



The **Education version** adds the following components:

- A mountable workspace to use vision functions directly on the Conveyor Belt;
- A slope that can help to create complex or multi-robot processes;
- An end-stop to stop the objects when they reach the end of the Conveyor Belt;
- The 6 objects of the Vision Set to extend the quantity of objects you can use on your production line;
- The Vision jaws to manipulate them.

With Ned's ecosystem, you have the optimal solution to discover, learn and test industry 4.0 processes.

Overview of this manual

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

This manual contains instructions for:

- Safety Information about Ned;
- Mechanical and electrical installation of the robot;
- Software of the robot;
- Maintenance and troubleshooting.

This manual should be read before:

- Installation and electrical connections;
- First usage of the robot;
- Maintenance and repair.

Safety notice

> **Warning**
>
> You must read, understand, and follow all safety information in this manual.

It is important that the safety instruction on this manual and the electrical and the mechanical instructions are followed. The robot specifications should be respected to avoid all damage to the machine or to the user.

Additional content

You can find additional content by clicking on the linkhere (https://docs.niryo.com/).

Installation and commissioning

This section specifies the procedures to follow during unpacking and transportation.

What the box contains

When you order Ned, you receive a box with the following items:

- 1x Ned Robot
- 1x Custom Gripper
- 1x Power cable
- 1x Universal travel plug adapter
- 1x SD card reader
- 1x Allen Key set

Unpacking

1. Inspect Ned's packaging to make sure nothing is damaged;
2. Remove the packaging and check for any transport damage or loss;
3. Put Ned in a dedicated operating environment;
4. Make sure Ned is stable.

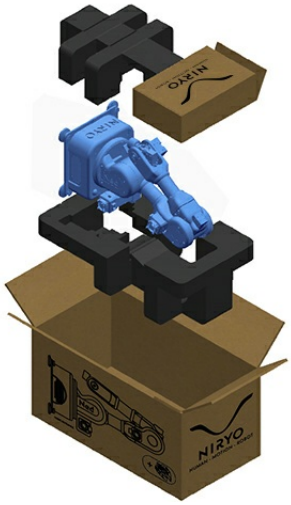
Operating conditions

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Parameters	Value
Minimum Ambient Temperature	+5°C
Maximum Ambient Temperature	+45°C

Shipping and transport position

This figure shows Ned in its shipping position, which is a recommended transport position to avoid Ned's damages.



Safety instructions

Safety signals and symbols

This section specifies all dangers that can arise when doing the work described in the User Manual. Each danger consists of:

- A caption specifying the danger level (DANGER, WARNING, or CAUTION) and the type of danger;
- A brief description of the danger;
- Instruction about how to eliminate danger.

Designation	Description
DANGER	Warns that an accident will occur if the instructions are not followed, resulting in an injury.
WARNING	Warns that an accident will occur if the instructions are not followed, resulting in a possible injury or product damage.
CAUTION	Warns of possible product damage.
NOTE	Describes an important fact or condition.
TIP	Describes additional information or an easy way to use the function.

Icon	Designation	Description
	ELECTROSTATIC DISCHARGE	Warns of a possible electrostatic discharge that may damage the robot.
	ELECTRICAL SHOCK	Warns of an electrical hazard.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Safety Overview

This section includes information on general safety risks to be considered when performing installation and service work on Ned.

These safety instructions must be read and followed by any person who deals with Ned.

Spare parts and special equipment

The installation and the use of external components is possible (such as Arduino, external sensors, devices). But the user must respect the electrical specifications and connections mentioned in the User Manual.

Niryo is not liable for damages or injuries caused by unauthorized modifications to the robot system or the misuse of the robot.

Complete robot

Danger

1. Before installing, using, or programming Ned, read the product specifications and Manual;
2. Make sure to install Ned and all electrical equipment according to the specifications and the warnings from chapters Installation and commissioning (index.html#installation-and-commissioning) and Technical specifications (index.html#technical-specifications);
3. Children are not allowed to operate Ned without adult supervision to prevent any possible injury or improper misuse of Ned;
4. Make sure that changing robot software will not cause hazards, damages to the Ned's system or injuries.

Warning

1. Motors and gearboxes are hot after running Ned for a long time. Avoid touching especially with a higher environment temperature;
2. Removing parts may result in the collapse of Ned. Take the necessary measures to ensure that Ned does not collapse as parts are removed;
3. Make sure to remove the power cable during assembly, wiring, or repair. Doing so will help to prevent equipment damage or accidental short-circuit;
4. The cable packages may be sensitive to mechanical damage;
5. Before using Ned, make sure to firmly fix it on a stable surface to avoid collapse or the fall of Ned;
6. Do not use Ned outside;
7. Do not put Ned in a humid environment or near water;
8. Do not install or operate Ned in dangerous environments (e.g., in the presence a strong magnetic field, dangerous gases, fire or flammables) to avoid dangers which may occur due to external conditions during operation;
9. Respect the technical specifications to avoid damaging the motors. Stepper motors and servo motors may be damaged if excessive force is used;
10. Make sure external change on the motors parameters or the robot software does not cause damage to Ned.
11. Do not open Ned base when the power cable is connected. Do not touch the electronic board, they may be hot.

Niryo is not liable for damages or injuries caused by unauthorized modifications to the robot system or the misuse of the robot.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Caution

1. Do not turn off Ned during a sequence or a motion unless absolutely necessary, this may reduce Ned operating age;
2. Try to install Ned arm on a stable surface with enough space to avoid any shock or vibration.

Voltage related risks, robot

A danger of high voltage is associated with the robot in:

- The power supply of the motors (11.1V, 8V);
- Ned's grippers are powered by 8V. A risk of short circuit and sparks exists. Be sure to make the installations with the power disconnected.

Safety actions

- Fire extinguishing

Note

Use a CARBON DIOXIDE (CO_2) extinguisher in the event of a fire in Ned's system.

- Manually stopping or overriding the arm: if needed, Ned arm can be stopped manually. This is possible since Ned arm is light, the arm force is limited, and the drivetrain power is limited.

To prevent unnecessary damage and wear of Ned arm, it is recommended to use the normal stopping functions of Ned (by the software or the top button).

Note

Ned is not equipped with brakes since its weight and design do not require a holding brake.

Technical specifications

This section describes the technical specifications of Ned.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Parameters	Value
Weight	6,5 kg
Payload	300 g
Reach	440 mm
Degree of freedom	6 rotating joints
Joints range	-170 ° ≤ Joint 1 ≤ 170 ° -120 ° ≤ Joint 2 ≤ 35 ° -77 ° ≤ Joint 3 ≤ 90 ° -120 ° ≤ Joint 4 ≤ 120 ° -100 ° ≤ Joint 5 ≤ 55 ° -145 ° ≤ Joint 6 ≤ 145 °
Joints range	-2,97 rad ≤ Joint 1 ≤ 2,97 rad -2,09 rad ≤ Joint 2 ≤ 0,61 rad -1.34 rad ≤ Joint 3 ≤ 1,57 rad -2,09 rad ≤ Joint 4 ≤ 2,09 rad -1,75 rad ≤ Joint 5 ≤ 0,96 rad -2,53 rad ≤ Joint 6 ≤ -2,53 rad
Joints speed limit	Joint 1 ≤ 150 °/s Joint 2 ≤ 115 °/s Joint 3 ≤ 140 °/s Joint 4 ≤ 180 °/s Joint 5 ≤ 180 °/s Joint 6 ≤ 180 °/s
Joints speed limit	Joint 1 ≤ 2.6 rad/s Joint 2 ≤ 2.0 rad/s Joint 3 ≤ 2.5 rad/s Joint 4 ≤ 3.14 rad/s Joint 5 ≤ 3.14 rad/s Joint 6 ≤ 3.14 rad/s
TCP max speed	1144 mm/s
Repeatability	0.5 mm

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Parameters	Value
Footprint	200x200 mm
Mounting	Table
I/O power supply	5V for digital GPIO 12V for Switches output
I/O ports	Digital inputs/outputs 5V: 6 Digital outputs 12V: 2
I/O interface	2 x USB2.0 2 x USB3.0 1 x ETHERNET GIGABIT
Communication	Modbus TCP (master) TCP/IP CAN (slave, compatible with NiryoStepper motors)
Materials	Aluminum Plastic PLA
Temperature	5 - 45°C
Power supply	DC 11.1V - 6A
Programming Environment	Niryo Studio PyNiryo C++ Python ROS

Tools technical specifications

Tools Overview

Ned is compatible with 5 different sorts of tools:

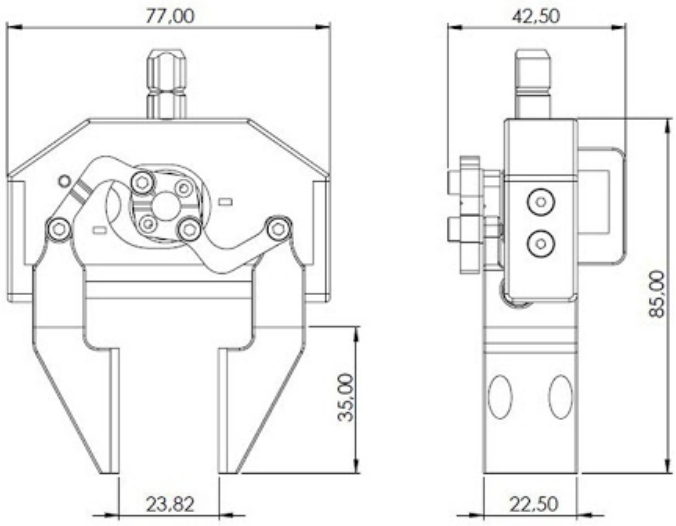
- 3 x Grippers
- 1 x Electromagnet
- 1 x Vacuum Pump

Specifications

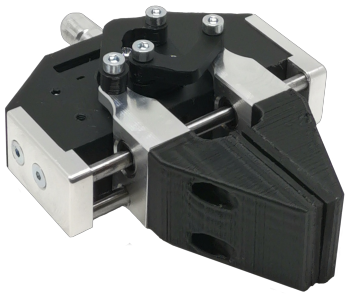
Custom Gripper

Overview - Standard Jaws

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS



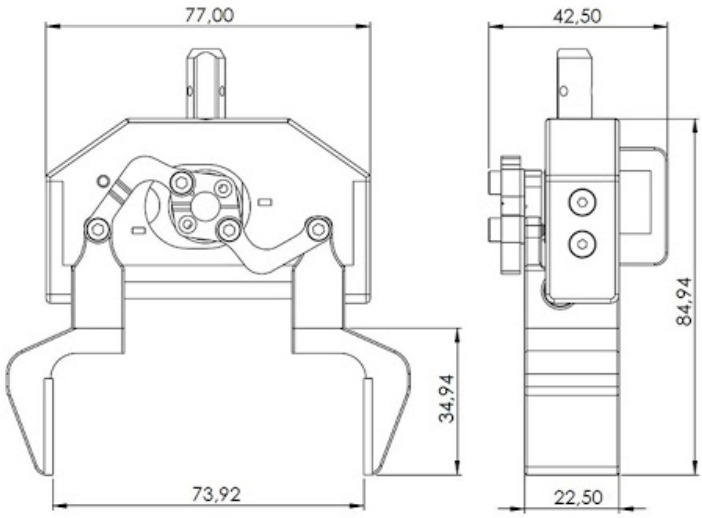
(_images/GRIPPER_STANDARD_SCHEMA_1.jpg)



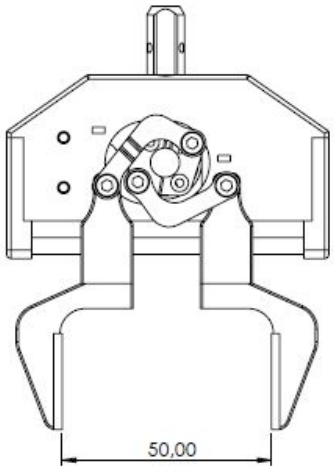
(_images/GRIPPER_MORS_STANDARD.png)

Overview - XL Jaws

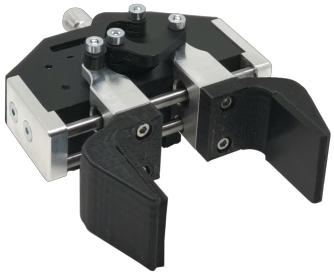
2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS



(_images/GRIPPER_XL_SCHEMA_1.jpg)



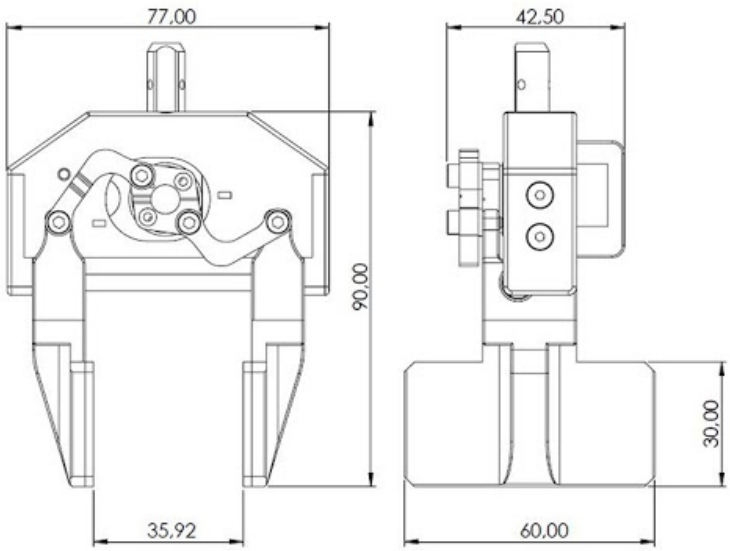
(_images/GRIPPER_XL_SCHEMA_2.jpg)

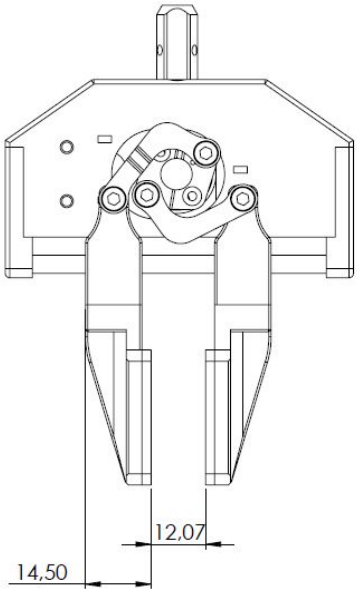


(_images/GRIPPER_MORS_XL.png)

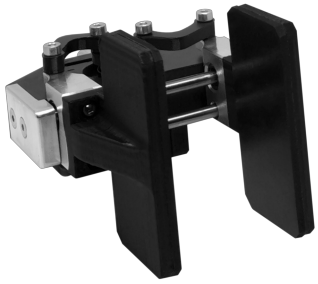
Overview - Flat Jaws

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS





(_images/GRIPPER_FLAT_SCHEMA_1.jpg)

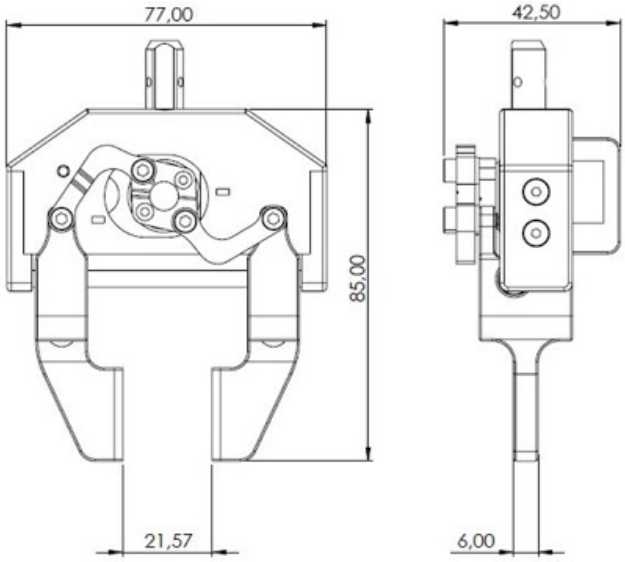


(_images/GRIPPER_FLAT_SCHEMA_2.jpg)

(_images/GRIPPER_MORS_FLAT.png)

Overview - Precision Jaws

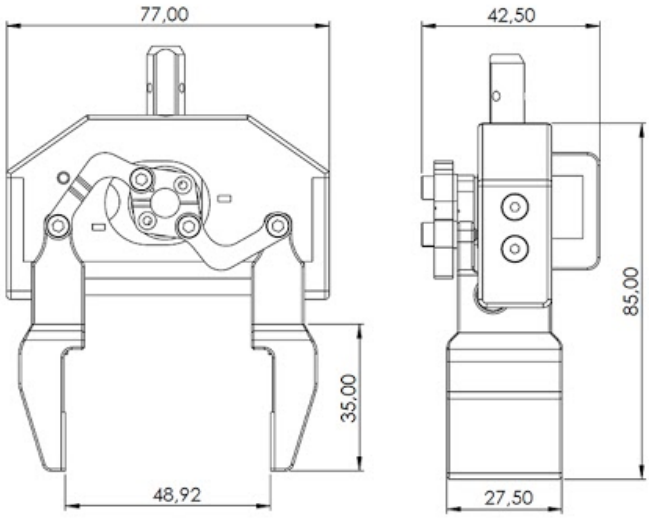
2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS



(_images/GRIPPER_PRECISION_SCHEMA_1.jpg)

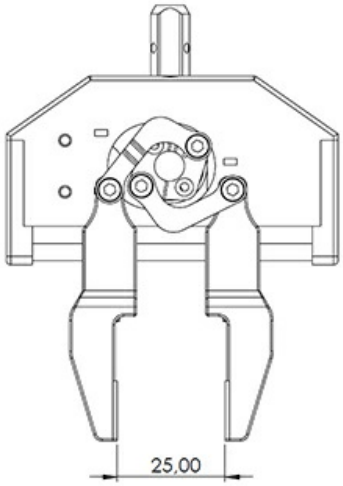
(_images/GRIPPER_MORS_PRECISION.png)

Overview - Vision Jaws

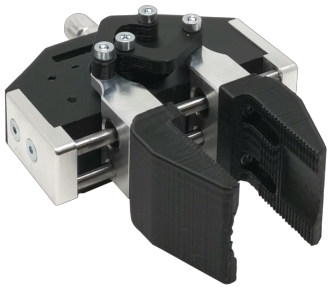


2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

(_images/GRIPPER_VISION_SCHEMA_1.jpg)



(_images/GRIPPER_VISION_SCHEMA_2.jpg)



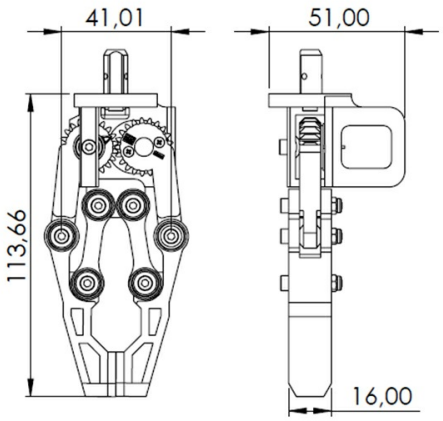
(_images/GRIPPER_MORS_VISION.png)

Custom Gripper's technical specifications

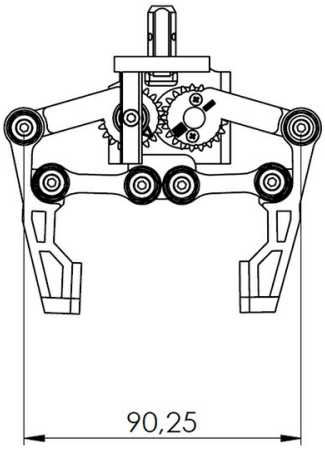
Parameter	Description
Max operating width	Standard Jaws: 23.82 mm XL Jaws: 73.92 mm Flat Jaws: 35.92 mm Precision Jaws: 21.57 mm Vision Jaws: 48.92 mm
Picking distance from end effector base	Standard Jaws: 85 mm XL Jaws: 85 mm Flat Jaws: 90 mm Precision Jaws: 85 mm Vision Jaws: 85 mm
Motor	XL320 Servo Motor
Weight	Standard Jaws: 109 g XL Jaws: 107 g Flat Jaws: 123 g Precision Jaws: 101 g Vision Jaws: 107 g
Power Supply	7.6 V
Operating temperature	5-45°C

Large Gripper

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS



(_images/GRIPPER_LARGE_SCHEMA_1.jpg)



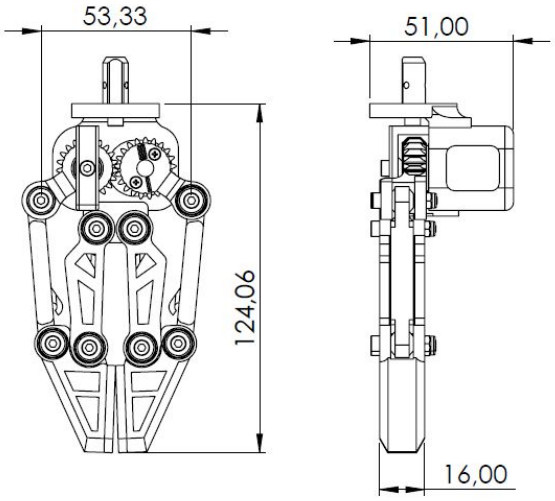
(_images/GRIPPER_LARGE_SCHEMA_2.jpg)

(_images/GRIPPER_LARGE.png)

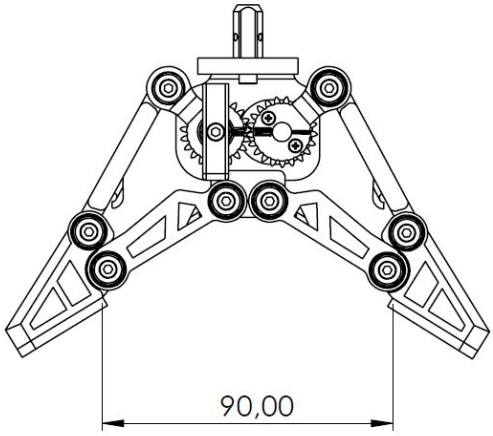
Parameter	Description
Max operating width	60 mm
Picking distance from end effector base	80 mm
Motor	XL320 Servo Motor
Weight	90 g
Power Supply	7.6 V
Operating temperature	5-45°C

Adaptive Gripper

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS



(_images/GRIPPER_ADAPTATIVE_SCHEMA_1.jpg)



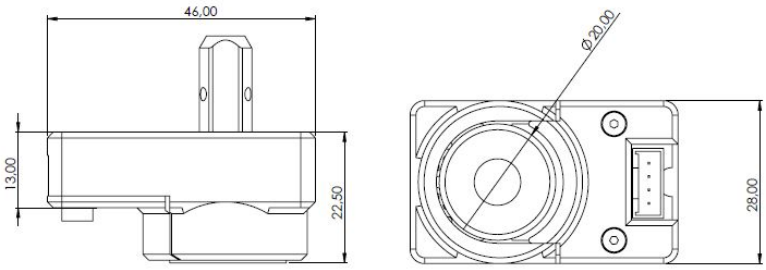
(_images/GRIPPER_ADAPTATIVE_SCHEMA_2.jpg)

(_images/GRIPPER_ADAPTATIF.png)

Parameter	Description
Max operating width	90 mm
Picking distance from end effector base	85 mm
Motor	XL320 Servo Motor
Weight	110 g
Power Supply	7.6 V
Operating temperature	5-45°C

Electromagnet

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

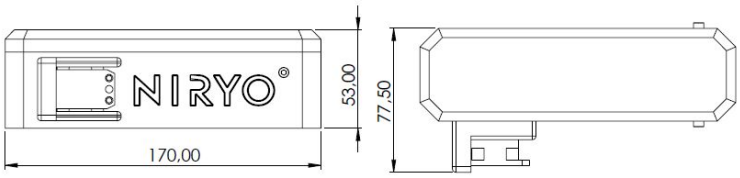


(_images/GRIPPER_ELECTROMAGNET_SCHEMA_1.jpg)

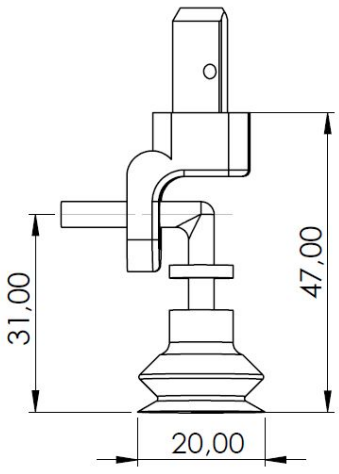
(_images/AIMANT.png)

Parameter	Description
Max operating width	20 mm
Picking surface	20 mm
Picking distance from end effector base	22.5 mm
Control interface	Digital output
Weight	42 g
Power Supply	5 V
Operating temperature	5-45°C

Vacuum Pump



(_images/GRIPPER_VACUUM_PUMP_SCHEMA_1.jpg)



2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

(_images/GRIPPER_VACUUM_PUMP_SCHEMA_2.jpg)

(_images/POMPE.png)

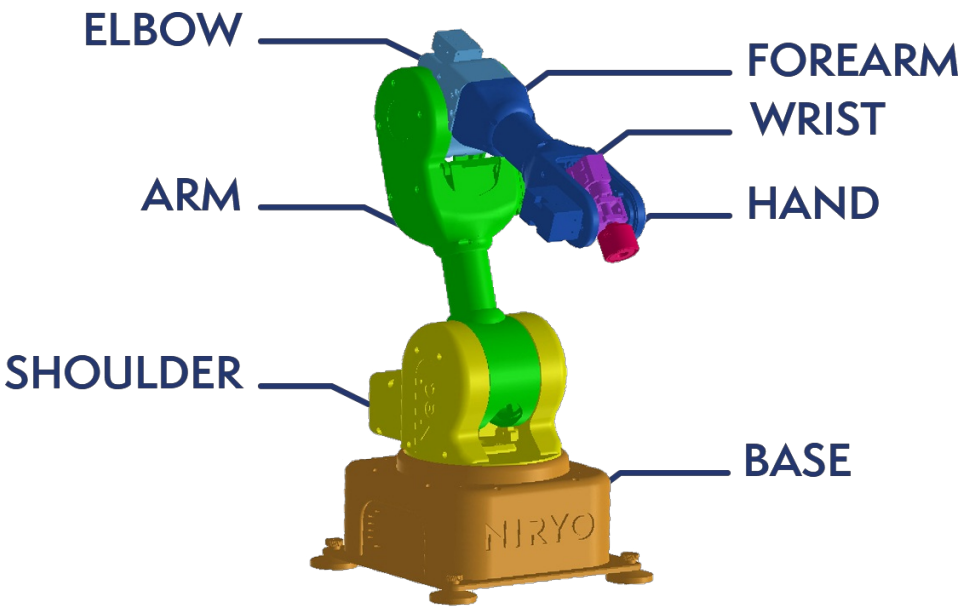
Parameter	Description
Vacuum Pump control	Integrated XL320 servo motor
Weight	10 g
Total weight	170 g
Payload	300 g
Power Supply	7.6 V
Operating temperature	5-45°C
Vacuum Tube dimension	20 mm
Picking distance from end effector base	47 mm

Mechanical interface

Mechanical Overview

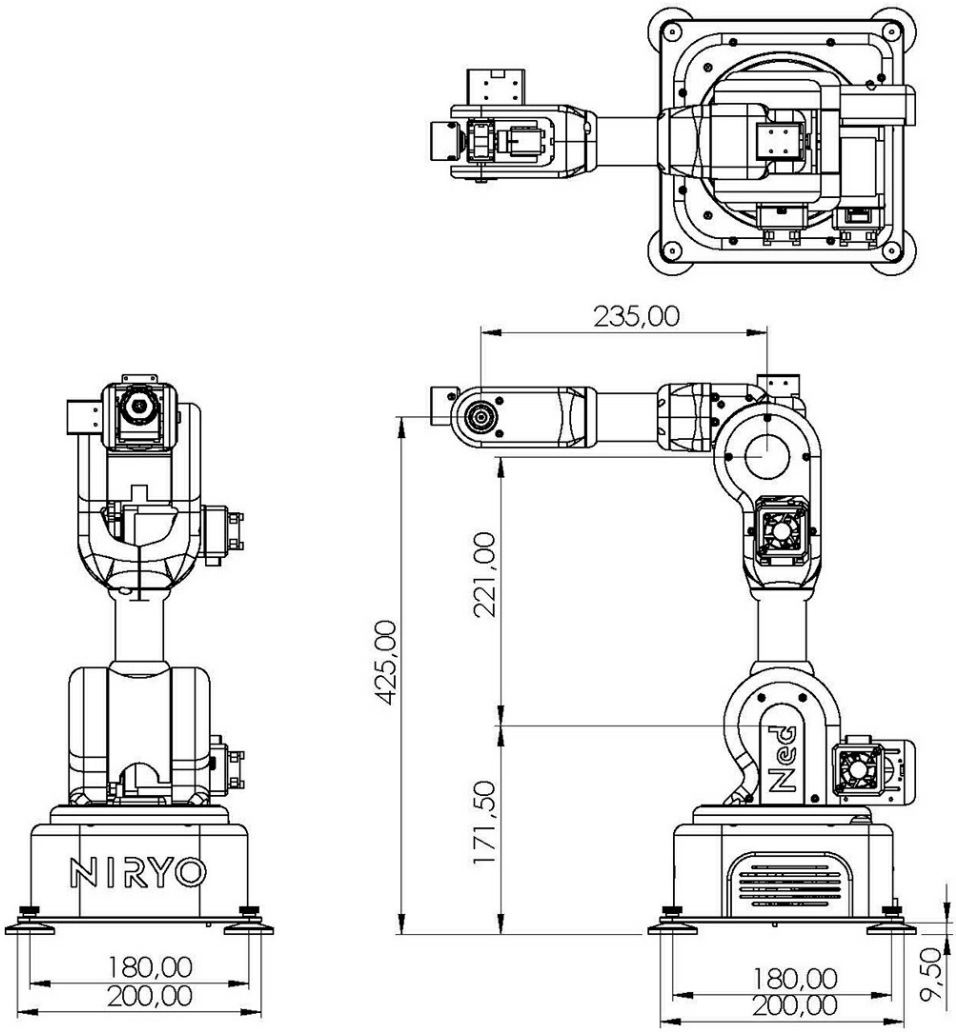
This chapter introduces the mechanical and the electrical interfaces of Ned.

Ned is a 6-axis collaborative robotic arm. It consists mainly of six robot joints of aluminum with plastic covers. Ned consists of 7 parts :

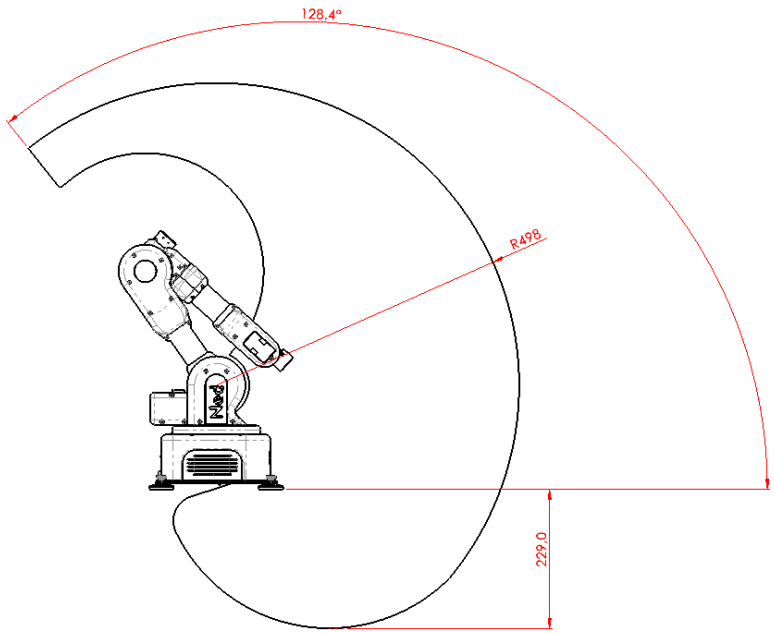


Three-view of the robot

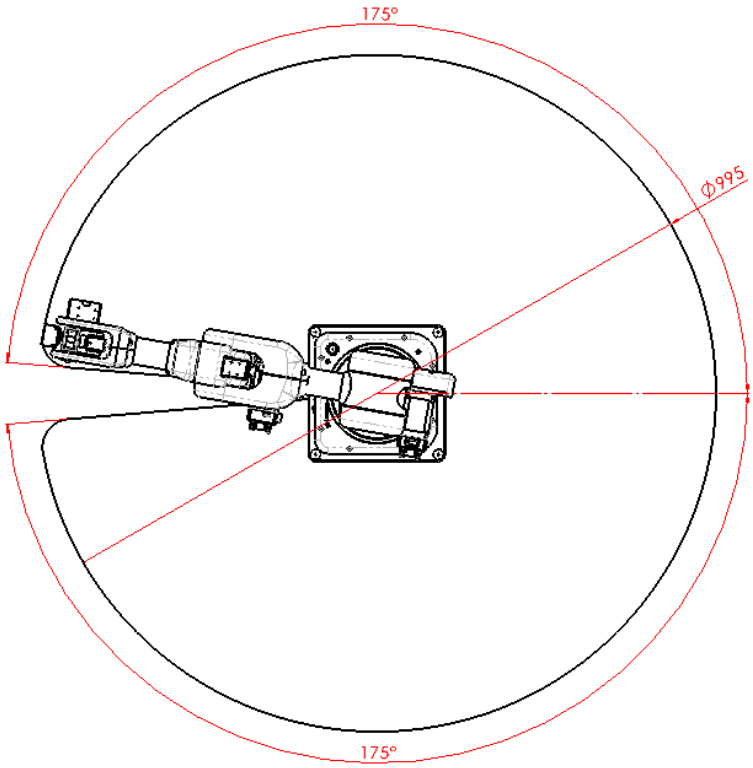
2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS



Robot workspace



2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS



Tool mounting

Each tool has the same mechanical connector interface. This option allows a quick-change option of the tool and good stability. See the picture below.

Vision Set mounting

To mount the Vision Set, all you need to do is attach the camera to Ned's wrist, set up and fix your workspace to Ned's structure to have a stable environment and put the vision jaws on the Custom Gripper.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Robot calibration

Ned is delivered without a mechanical calibration. Ned can not execute any motion properly without auto-calibration because of a mismatch between the origin of each motor and its corresponding origin stored in the controller.

Note

Make sure to process the calibration in order to align the origins of every joint.

Hint

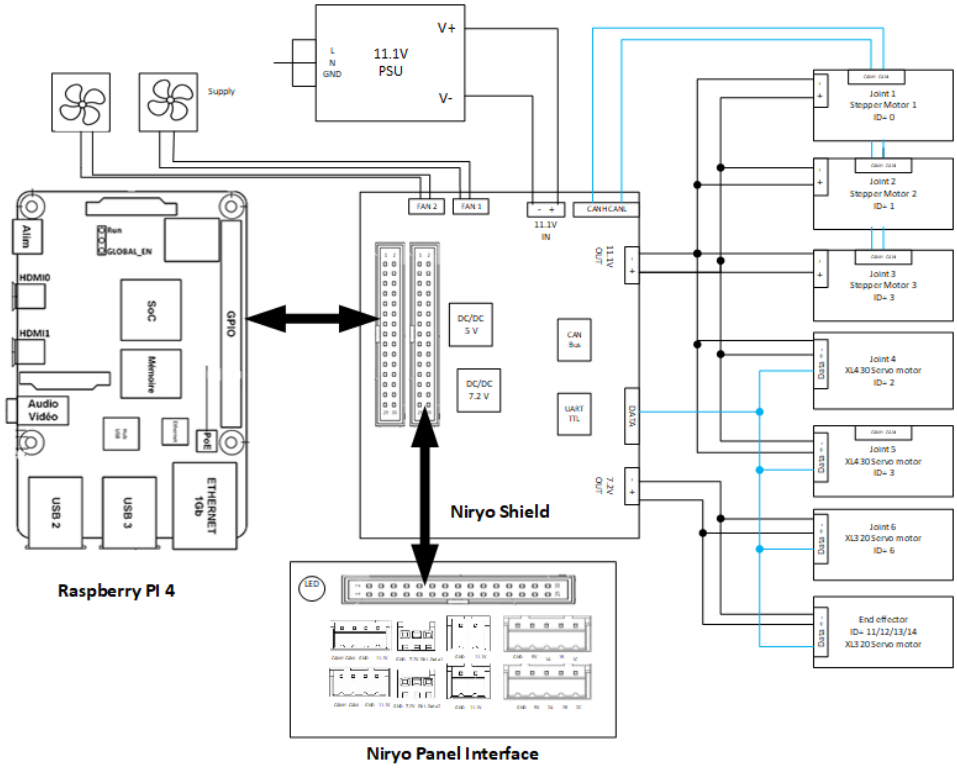
Check Niryo Studio Manual (https://docs.niryo.com/product/niryo-studio/index.html) to get more information about Ned's calibration.

Electrical interface

Electrical interface Overview

Electrical architecture:

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS



Electrical warnings and cautions

Danger

- Make sure that all the equipment and the wires are kept dry. If water enters the equipment, disconnect the power;
- Carefully follow the safety instructions of the next section to avoid damaging your robot.

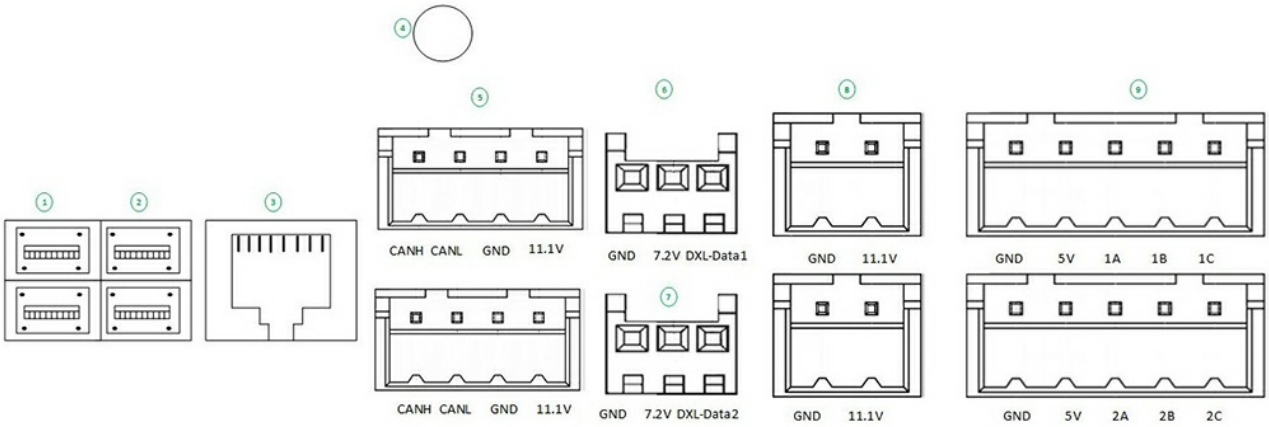
Warning

Do not unplug or power off the robot during use, you may damage the Raspberry Pi 4 inside the robot, or the micro SD card inside the Raspberry Pi 4

Panel connectors

Panel connectors Overview

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS



1	2 x USB port 2.0
2	2 x USB port 3.0
3	Ethernet port
4	Panel LED
5	2 x NiryoStepper connector
6	XL320 connector
7	XL430 connector (not used)
8	2 x SW : 12 V digital output
9	2 x GPIO panel
10	Top button

Power supply

Ned is powered with AC/DC adapter with 11.1V output

Parameters	value
Frequency	50/60 Hz
Input	AC 100-240 V , 1.5 A
Output	11.1 V – 6 A

Warning

1. When Ned is turned on, do not disconnect the power supply of Ned. You may damage the Raspberry Pi 4 inside Ned, or the micro-SD card.
2. If you modify the original power supply of the robot, you should respect the same specifications.
3. Before plugging the power connector and powering on Ned, first make sure that:
   - Ned is on a flat and stable surface;
   - The power switch is turned off (position 0);
   - Ned has enough space to move without hitting something or someone.

Power Switch

The power switch turns Ned on and off. It enables the power supply.

Always switch it off after shutting down Ned and turn it on to power on Ned.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Warning

Before plugging or unplugging the power supply, make sure to turn off the switch.
Do not keep the switch on after shutting down Ned, otherwise , the power will be present on the motors and on the shield

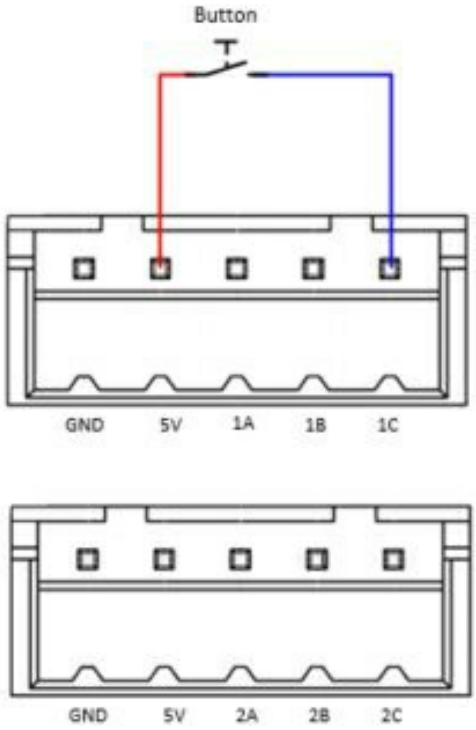
Digital inputs/outputs

- Ned has 6 inputs/outputs that can be configured either as digital input or digital output with Niryo Studio (see How to program the digital I/O on the Software Manual (https://docs.niryo.com/product/niryo-studio/index.html)).

Terminal	Parameters	Min	Max	Unit
Digital Input				
[1A-2A-1B-2B-1C-2B]	Voltage	0	5	V
Digital output				
[1A-2A-1B-2B-1C-2B]	Voltage	0	5	V

- Digital input connection:

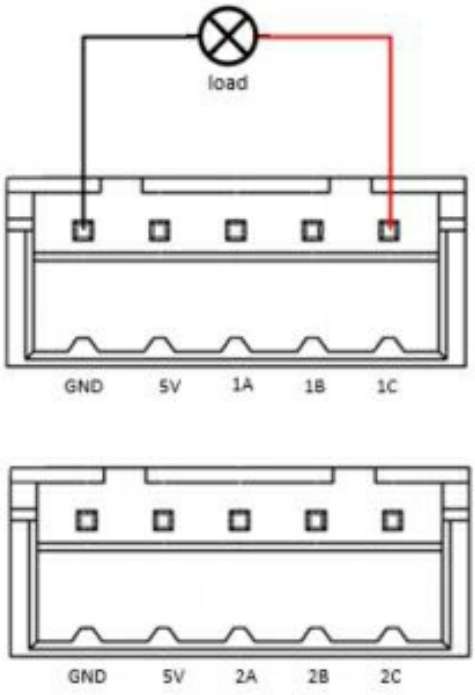
  This illustration below shows how to connect a button to a digital input.



- Digital output connection:

  This example shows how to connect a load in order to be controlled from a digital output (see below).

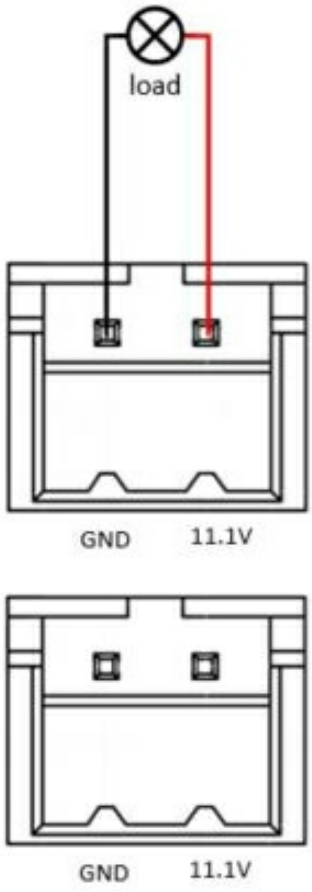
2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS



Switch connections

Terminal	Parameters	Min	Max	Unit
Digital Output				
SW1, SW2	Voltage	0	11.6	V
	Current	0	0.5	A

Below is an example of how to connect an external load to the switch output.



2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Ethernet

An Ethernet connection is provided at the panel connector of Ned.

The Ethernet interface can be used for the following:

- Modbus TCP/IP;
- PyNiryo;
- Connect Ned to your local area network.

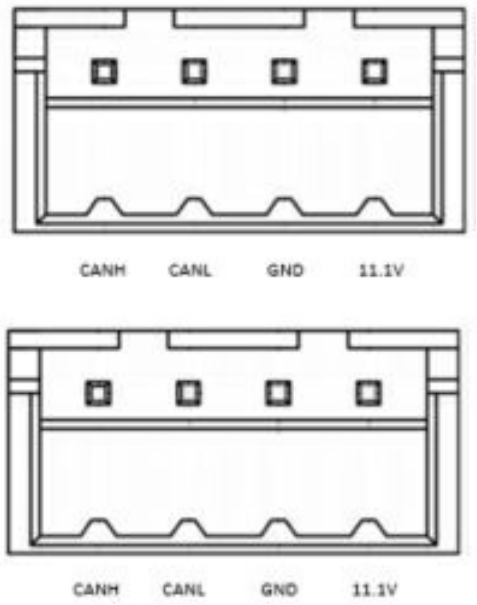
USB Port

The USB can be used to connect the keyboard, mouse, or Vision Set.

Find more information about Vision Set on the Vision Set Manual (https://docs.niryo.com/product/vision-set/index.html).

NiryoStepper connectors

NiryoStepper connectors interface can be used to drive a NiryoStepper motor. It is based on CAN Bus communication protocol.



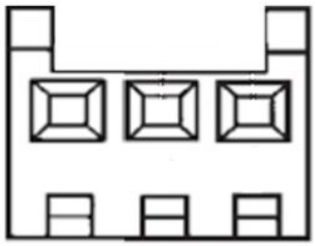
This interface can be used to drive Niryo's Conveyor Belt (https://docs.niryo.com/product/conveyor-belt/index.html).

DXL connection

XL320 Connector

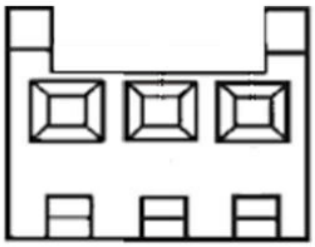
This connector can be used to connect Ned's Vacuum Pump tool.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS



XL430 Connector

Not used.



Panel LED

Ned's LED panel has several colors which represent different modes and error status. Refer to the following table:

Color / Blinking	Description	Troubleshooting
Solid blue	Hotspot mode, the robot has started and creates its own Wi-Fi network.	N/A
Solid green	Connected mode, the robot is fully started and connected to a Wi-Fi network.	N/A
Alternating between Green/Red	Motor error on connected mode	Check "HARDWARE STATUS" and logs to troubleshoot the error.
Alternating between Blue/Red	Motor error on hotspot mode.	Check "HARDWARE STATUS" and logs to troubleshoot the error.
Solid purple	Robot controller is shutting down	N/A
Alternating white – green	The robot is paused on connected mode	N/A
Alternating white – blue	The robot is paused on hotspot mode	N/A
Solid red	On start-up: the robot is booting; On power off: shutdown completed	N/A

Top button

The top button of Ned has multiple functionalities depending on how long you press the button and the number of presses.

Note

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Note

All functionalities are available for use only after the robot has been fully started (LED is green or blue).

Press count / Delay	Function
Program is executing	
1 press - 0.1 and 3s	Pause the robot program with Learning Mode off.
2 press – 0.1 and 3s	Pause the robot program with Learning Mode on.
Program is paused	
1 press - 0.1 and 3s	Continue the program.
2 press – 0.1 and 3s	Stop the program.
Program is set on autorun	
1 press - 0.1 and 3s	Start an autorun program.
Programming the robot with blockly	
1 press - 0.1 and 3s	Add / save a freedrive joint position.

Note

When you pause the robot's program with the top button, you should either press the button to continue or press twice to stop it.

Hint

You can test your robot hardware for the first time without using any software tool, you can follow the steps below:

- Plug the power supply;
- Turn on the power switch button;
- Wait until the LED panel turns blue (if it is the first time you use Ned this should take 2 minutes);
- Press the top button: Ned will start by calibrating and then it will execute a pre-saved sequence.

Tool wiring

Custom Gripper, Large Gripper and Adaptive Gripper:

There is a cable on the tool end of Ned with 3 pins. This connector can drive Custom Gripper, Large Gripper and Adaptive Gripper.

Note

Make sure to respect color order when you are wiring your gripper.

Warning

Not respecting the wiring instructions may damage your gripper or Ned.

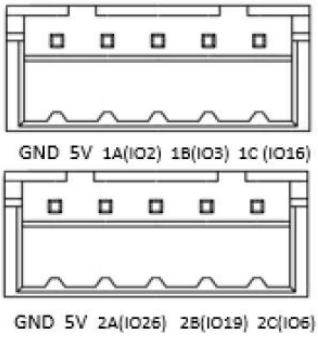
Vacuum pump:

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Plug the suction cup to Ned's hand and connect the motor wire to theXL320 Connector placed at the back of Ned.

Electromagnet:

Plug the electromagnet to Ned's hand and connect the cable on the GPIO1 or GPIO 2 connector on the panel connector.



Note

Information about programming and using the tool are available on the Software Manual (https://docs.niryo.com/product/niryo-studio/index.html)

Advanced Programming

There are many other ways to develop on Ned.

For more information, please refer to the following links:

- Niryo Studio (https://docs.niryo.com/product/niryo-studio/index.html)
- Python (https://docs.niryo.com/dev/pyniryo/index.html)
- ROS (https://docs.niryo.com/dev/ros/index.html)
- Modbus (https://docs.niryo.com/dev/modbus/index.html)

Suggest a modification | Download as PDF

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Niryo Studio User Manual

Niryo Studio is a graphical HMI. It allows a fast and direct control of Ned with an external computer.

Its purpose is to provide users with a complete and simple interface for Ned motion, programming environments and current status of Ned.

Through Niryo Studio application, users can:

- Manage and set Ned parameters
- Control Ned motion
- Program Ned using Blockly
- Execute a program
- Manage, set, control and program Ned tools
- Manage, set, control and program Ned addons

Users must read this manual in order to fully understand all Ned and Niryo Studio functionalities.

Note

If you encounter software problems or if you want to access latest updates for your Ned, please give an internet access to your robot (Ethernet or Wi-Fi) to be able to update it through NiryoStudio. Instructions here (index.html#ned-software-update)

Changelog

September release - New features batch

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Features

Direct control

- Jog Pose.

  Control the pose of the robot in position and orientation. The arrows show the direction of the translation and rotation command to apply to the robot's pose. You can adjust the command speed.

- Jog Joints.

  Control the joints of the robot in orientation. The arrows show the direction of the rotation command to apply to a joint. You can adjust the command speed.

State and 3D View

- Frames visualization.

  Visualize the TCP and the robot base frames. You can select which frames you want to display.

Programming

- **Try Except** block in *Logic* group.

  The function will try to run what is inside of it. A number of attempts can be set to repeat the operation if the robot failed to do it. You can set the function to continue or stop the program if the attempts do not succeed.

- **Linear** and **Try Linear in** *Move Pose* block, in *Movement* group.

  Linear function makes the robot move linearly. **Try Linear** function makes the robot move linearly but if no linear trajectory can be computed, it will do a *Standard* movement.

- **Linear** and **Try Linear** in *Shift pose* block, in *Movement* group.

  Linear function makes the robot do a translation movement linearly. **Try linear** makes the robot do a translation movement linearly but if no linear trajectory can be computed, it will do a *Standard* movement.

- **Move Trajectory** function in *Movement* group.

  This function builds a trajectory by giving it waypoints. The robot will try to reach way points in the given order. The *distance smoothing* parameter defines the distance from which the trajectory will be smoothed around each way point.

- **Set TCP** block in *Tool* group.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

This function changes the TCP (Tool Center Point) coordinates of the robot. TCP coordinates are used with *Pose*-type movements.

- **Activate/Deactivate TCP** block in *Tool* group.

  This function activates or deactivates TCP coordinates. When activated, TCP coordinates are defined by the user. When deactivated, the TCP coordinates are equal to the final point of the robot's arm.

Settings

- Angles unit selection.

  Angles unit selection between degree and radian from the top bar.

- Setup TCP coordinates from *Robot's settings* view.

  Change TCP (Tool Center Point) coordinates. TCP coordinates are used as references for *Pose* movements. *Activate* TCP allows the modification of the TCP coordinates. *Scan* will check if a tool is attached to the arm and will update the TCP coordinates according to the connected tool. *Reset* will reset the TCP coordinates to the actual tool (0 if no tool). You can manually modify the TCP coordinates by writing your orientation and translation. Click *Apply* to apply the new coordinates.

Saved positions

- Play position

  Move to the desired position from position list by clicking on the*Play* button.

Vision set

- Camera's settings

  Brightness, saturation and contrast adjustments for the camera.

Improvements

Programming

- Graphical improvement of **save program** function in the *Blockly program* view.
- Graphical improvement of **Load program from robot** function in the *Blockly program* view.
- Remove **Clear workspace** in the *Blockly program* view.
- Graphical improvement of **Add position** function in the *Blockly program* view.

Download Niryo Studio

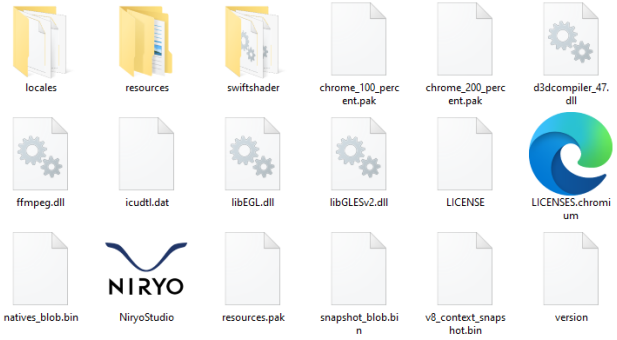
You can download the archive containing Niryo Studio from Niryo Websitehere (https://niryo.com/en/download/). Select the version you need depending on your computer operating system.

How to launch Niryo Studio

Once you have downloaded the .zip file you need to extract the archive. Then go into the new created folder and search for an executable named "NiryoStudio". Double click on the executable to start the application.

You do not need to install anything on your computer.

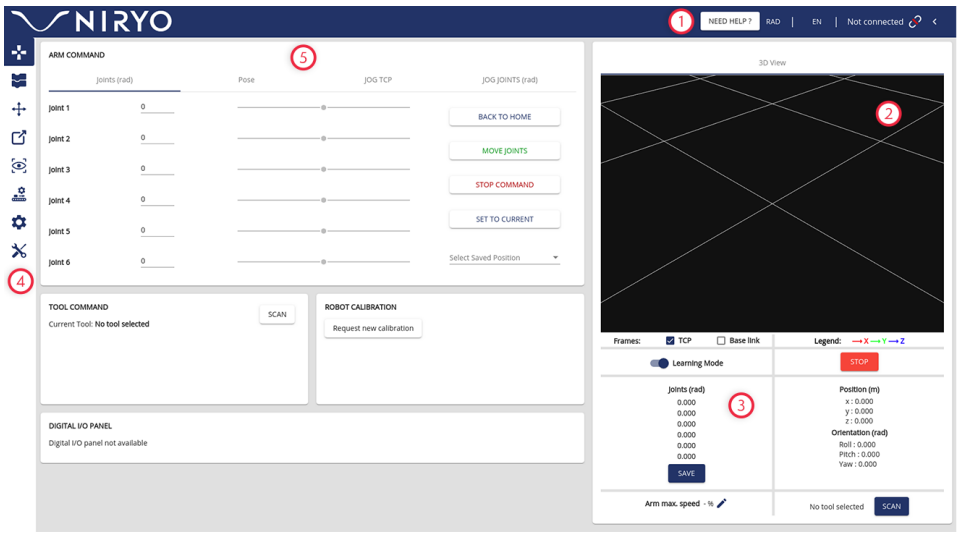
2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS



Overview

Operating interface

When the Niryo Studio launches, the window below appears.



1. The top toolbar allows you to change the angle units, the language, and connect to Ned
2. 3D Ned visualization or camera stream video
3. State section and mode selection
4. The left menu allows you to switch between the different sections of the application
5. The main window of the application

Navigation interface

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

The listed icons from top to bottom on the left menu are:

- Direct command: allows direct motion control of Ned and tools
- Blockly Programming: create, edit and execute a program in Blockly.
- Saved positions
- Program settings: run and upload the robot's program
- Vision Set: set and program the Vision Set
- Conveyor Belt: scan and control the Conveyor Belt
- Settings: robot and software settings
- Logs and Status: displays robot, Niryo Studio logs and the hardware status

Robot connection

There are three ways to connect Ned to Niryo Studio:

- Wired connection (Ethernet cable)
- Wireless connection (Wi-Fi):
    - Hotspot mode (the robot provides its own Wi-Fi network)
    - Connected mode (the robot is connected on your network)

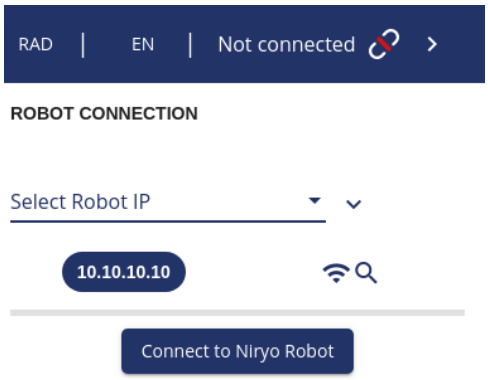
Using Ned in Hotspot mode

If the LED on the back panel is blue, that means you can connect to Ned via its own Wi-Fi. In this case Ned IP address is 10.10.10.10.

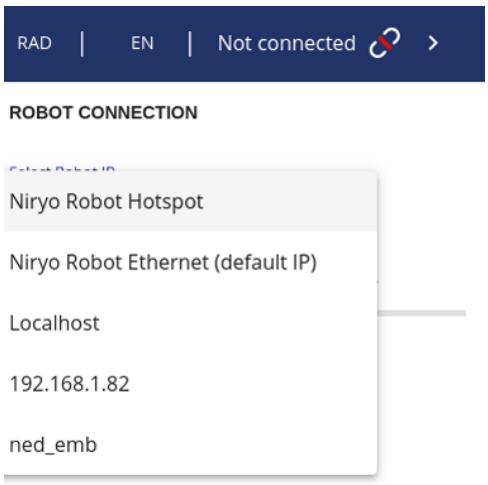
Ned has its own Wi-Fi network, which starts with "NiryoRobot" followed by a series of numbers and letters. To connect, follow the steps below:

- Connect to the Wi-Fi access point using your laptop with the password**"niryorobot"**
- Click the arrow near "Not connected" to access Ned connection

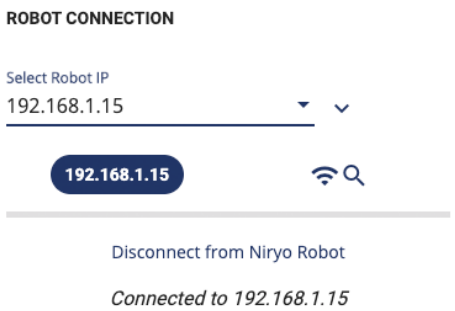
2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS



- Select “Niryo Robot Hotspot” and connect



- To disconnect from Ned, open the connection panel again and click on “Disconnect”



Using Ned on your Wi-Fi network

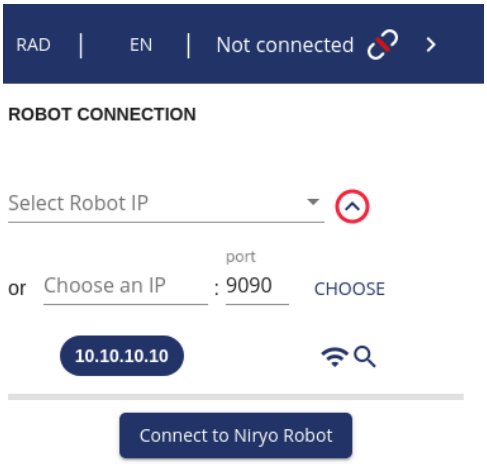
If Ned is connected to the Wi-Fi network (see therobot settings (index.html#ned-settings) section), the LED on the back panel is green.

To connect Ned to Niryo Studio, please follow the following steps:

- Connect your computer to the same Wi-Fi network as Ned,

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

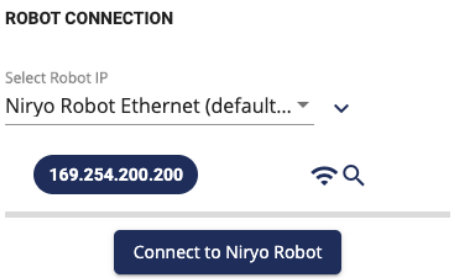
- If you do not know your robot IP address, open the connection panel and click on the “Wi-Fi + Search” button. This will search for connected robots inside the current Wi-Fi network. Choose the correct IP address and click “Connect Niryo Robot”,
- If you gave a custom name to Ned, select it by its name,
- If you already know your Ned’s IP address, click the arrow, enter your IP address and connect to the robot.



Using Ned with an Ethernet cable

You can also use an Ethernet cable to physically connect Ned with your computer.

- Connect Ned to your local network (computer or switch) using the LAN port,
- Modify the selected IP address on the connection panel to “Niryo Robot Ethernet (default IP)”,



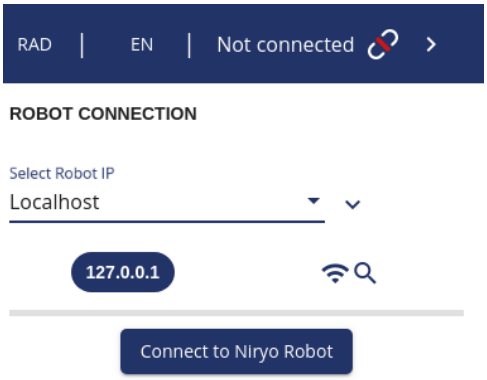
- Connect to Ned.

Using Ned in simulation with Niryo Studio

Simulators such as Rviz (https://docs.niryo.com/dev/ros/source/simulation.html#control-with-ros) or Gazebo (https://docs.niryo.com/dev/ros/source/simulation.html#launch-simulation) must be launched on the same computer as the one running Niryo Studio.

- Click the arrow near “Not connected” to access the robot connection,

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS



- Select “Localhost” and connect.

Robots specificities

Calibration

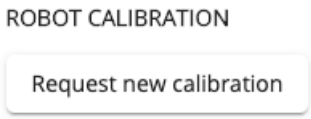
Ned needs to be calibrated in three cases:

- At the start-up
- If the steppers motors lose their offset positions and need to re-calibrate
- If the 3D view and Ned physical movements are not similar

In most of the cases, Niryo Studio will alert you by showing “CALIBRATION NEEDED” on the top right.

> **Note**
>
> In the case of “CALIBRATION NEEDED”, you can’t perform any command on Ned without calibrating it.

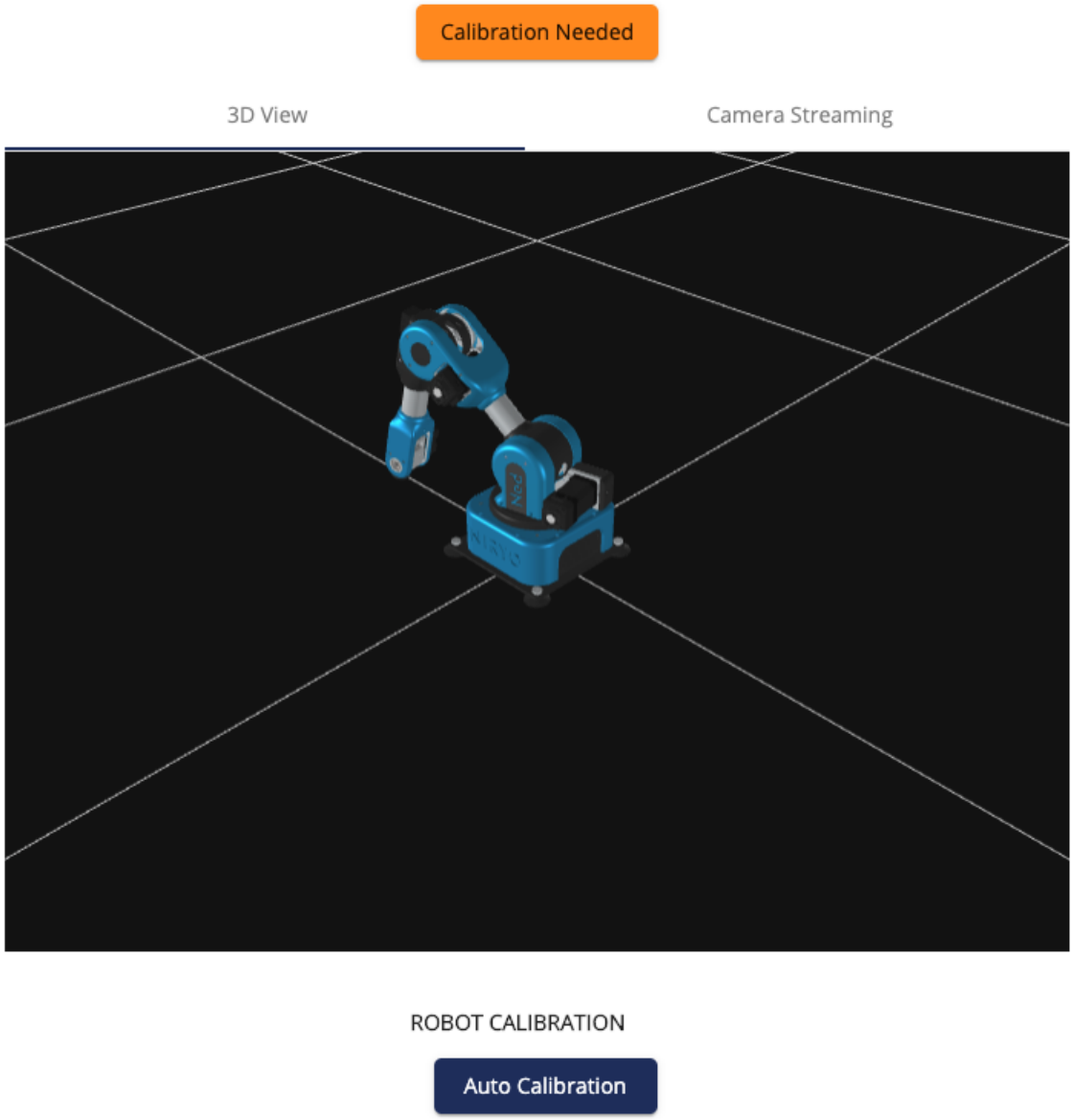
To perform a calibration, click on the button “Request new calibration” in the “ROBOT CALIBRATION” section.



After clicking, a “Calibration Needed” warning will appear on top of the 3D view.

When the “Calibration Needed” warning appears, you can click on it to perform an auto calibration or you can click on the “Auto calibration” button in the “ROBOT CALIBRATION” section.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS



By choosing auto-calibration, Ned will execute an automatic sequence by moving its axis that needs to be calibrated until they reach their maximum position, so a correct offset can be applied to each motor. This sequence should take about 20-30 seconds. When the calibration is over, Ned will come back to a resting position.

Caution

Before launching the auto-calibration, make sure that there is no obstacle around Ned. You should put Ned in a proper position before performing the calibration. The position in the picture below is recommended. This will avoid damaging Ned, collision, and any hazard motion.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Note

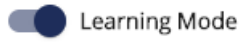
If the calibration is not successful, the “Calibration Needed” warning will still be displayed. Try the calibration again and if this new calibration does not succeed, it can mean that your robot has a mechanical/hardware issue.

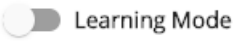
Hint

You can check if the calibration has succeeded by moving the robot and examine the robot motion in the 3D view.

Learning mode

Niryo robots have two operating modes: the manual mode (Learning mode) and the automatic mode. The current mode can be determined visually in Niryo Studio, via the Learning Mode button position.





- **Manual mode/learning mode: when Ned is in learning mode, the toggle on Niryo Studio is on.**

  In this mode, the user can guide Ned by hand. In this case, there is no torque applied on the joints and motors are in relaxed mode. This mode is recommended when no program is running.

- **Automatic Mode (Learning Mode is deactivated): The Learning Mode does not work under Automatic Mode.**

  It is not possible to move Ned by hand. This mode is active when a program is running or when Ned is performing a motion.

Caution

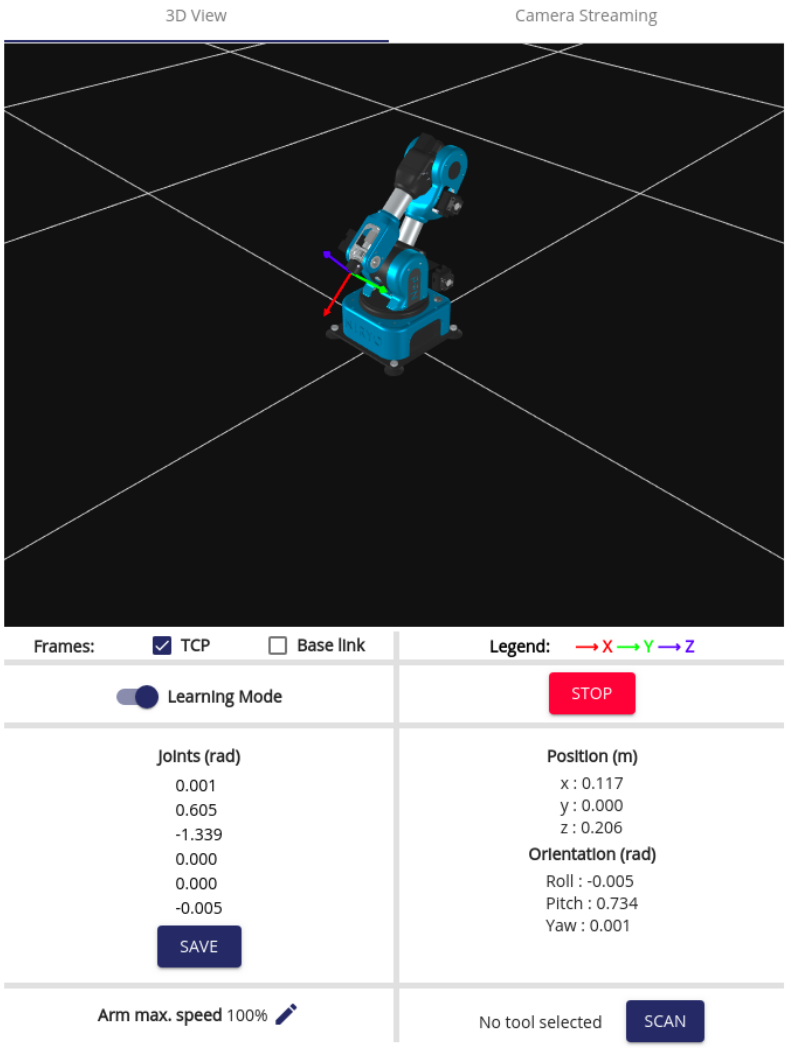
2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Do not try to override the motor's torque when learning mode is disabled: you could damage the robot and the motors. You can force to eliminate the risk or you can unplug the power supply ONLY in case of emergency (Ned blocked or application freezed)

Hint

Activate the learning mode when the program is paused or stopped.

State and 3D View



3D View

You can see here the 3D representation of the robot. You can display TCP (Tool Center Point) and Base link frames.

States and commands

From here, you can:

- View the current joints angle.
- View the current TCP position and orientation.
- Activate or deactivate the Learning mode.
- Stop the current movement.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

- Save the current position of the robot.
- Set the arm speed limit.

Direct control

Arm command

On the ARM COMMAND section, you can move Ned's arm directly either by moving the robot joint individually or by translating / rotating Ned pose.

You will find 4 main subsections that this documention will cover below, but we will start with the main buttons available.

Home Position

Click the "BACK TO HOME" button in order to move Ned to its home position.

Select Saved position

If you already saved a position, you can select it via the "Select Saved Position" scrolling menu. Once selected, it will upload joints value if your are in joints view, or pose if you are in pose view.

Then you can click on "MOVE JOINTS" or "MOVE POSE" to move the robot according to the selected joints/pose.

Stop command

The "STOP COMMAND" button stops Ned's motion.

Joints

This method allows you to control each joint individually from its minimum to its maximum value (see the Robots specificities (index.html#document-source/specification) section).

If a joint reaches its joint limit, it can't be driven any further.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

By clicking on the “SET TO CURRENT” button, you will refresh the interface to the actual joints values.

To perform a move joints, you need to:

- Modify the values of the joints either by writing the desired joint value or by moving the cursor,
- Click on the “MOVE JOINTS” button to execute the move command.

Once the command is done (success or not), you will get a notification at the bottom of the screen.

Important

At any time, you can cancel the current command execution by clicking on the “STOP COMMAND” button.

Pose

This panel gives you the possibility to move the pose of Ned’s TCP (Tool Center Point).

You can edit the values by clicking on them then use the “MOVE POSE” button.

JOG TCP

This panel will help you move Ned in a more visual way.

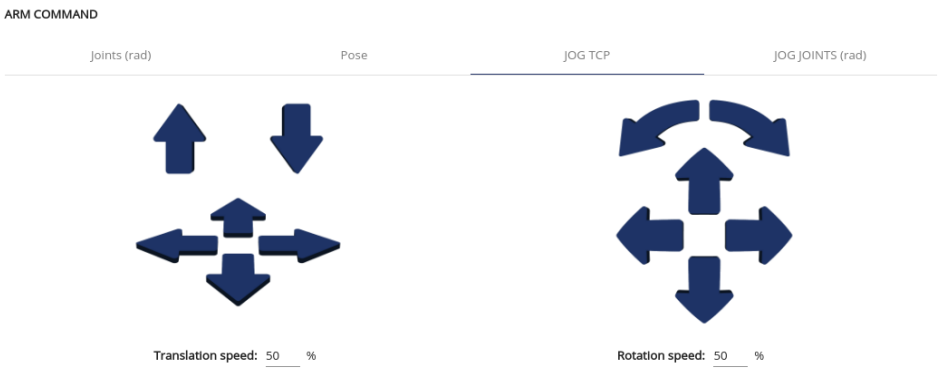
The TCP (Tool Center Point) of the robot is the center of the end effector.

- Holding a translational arrow will make the robot follow its direction,
- Holding a rotational arrow, will make the robot rotate around the TCP.

The robot will move until you release the button.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

You can modify the speed of the jogs. A low value will give you more precision while a higher value will bring you faster to the position.



Hint

The performance of the jog TCP can be impacted by your network connection. We advise you to use an Ethernet connection to get the optimal communication and the best experience with the jog TCP feature.

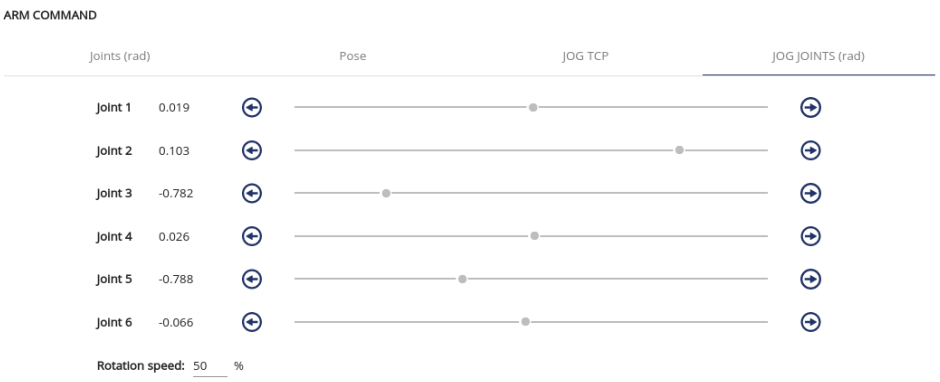
JOG JOINTS

You can also switch to jog Joints on the ARM COMMAND panel.

The jog Joints allows you to control each joints of the robot by holding down one arrow.

The robot will move until you release the button.

You can also change the movement speed. A low value will give you more precision while a higher value will bring you faster to the position.



Hint

The performance of the jog joints can be impacted by your network connection. We advise you to use an Ethernet connection to get the optimal communication and the best experience with the jog Joints feature.

Tool command

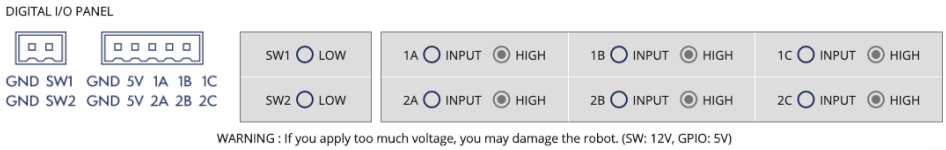
- The “SCAN” button will detect and add a tool if there is already a connected tool (with motor).
- **If you have a gripper:**

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

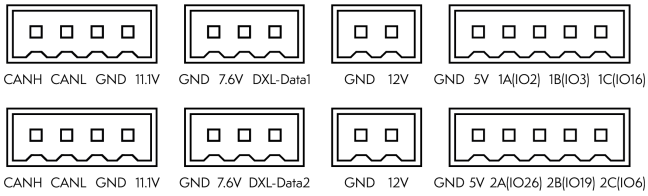
- You can set the opening and closing speed by writing the value or by dragging the cursor (from 100 and 1000),
- You can send open and close commands to the gripper by clicking “OPEN” and “CLOSE” buttons.

- If you have a Vacuum Pump, the “ON” button will pull the air with the Vacuum Pump and the “OFF” button will pull the air out of the pump.

Digital I/O panel



You can monitor and set I/O signals from or to Ned. The “DIGITAL I/O PANEL” displays the current state of the I/O “SW1”, “SW2”, “GPIO1”, and “GPIO2” connectors on the back of Ned.



Note

Check the Electrical Interface (https://docs.niryo.com/product/ned/source/hardware/electrical_interface.html#electrical-interface) for electrical and connection details.

- “SW1” and “SW2” connectors can be used to plug a fan, a motor, etc.
- For “GPIO1” and “GPIO2”, you have, from left to right: GND, 5V, and 3 digital pins. The digital pins can be used to communicate with another device (ex: an Arduino board. Check out this tutorial (https://docs.niryo.com/applications/ned/source/tutorials/control_ned_arduino.html) to learn more about that).

Switches

The switches are already set as OUTPUT mode.

You can change the “SW1” and “SW2” states to “HIGH” or “LOW” by clicking the corresponding button.

GPIO

Warning

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Make sure to read the Safety Precautions (https://docs.niryo.com/product/ned/source/hardware/safety_instructions.html) and the hardware manual (https://docs.niryo.com/product/ned/source/hardware/electrical_interface.html#digital-inputs-outputs) before using the digital I/O of Ned.

A digital pin (GPIO) can be set as an INPUT or OUTPUT mode.

You can change the mode (INPUT or OUTPUT) and the state (LOW or HIGH) for each pin, by clicking on the corresponding mode and state on the “DIGITAL I/O PANEL”.

- When the pin is set as an INPUT, you can read the state in this panel.
- In OUTPUT mode, you can set the OUTPUT state of the pin (LOW or HIGH) by clicking on the corresponding button.

Note

By default, all the GPIO pins are in INPUT mode and HIGH state.

Electromagnet control

Make sure to read the Safety Precautions (https://docs.niryo.com/product/ned/source/hardware/safety_instructions.html) and the Hardware Manual (https://docs.niryo.com/product/ned/source/hardware/electrical_interface.html#digital-inputs-outputs) before using the digital I/O of Ned.

The Electromagnet can be set up and controlled on the “DIGITAL I/O PANEL”.

You can connect the electromagnet on the GPIO1 or GPIO2 at the back of Ned (seeHardware Manual (https://docs.niryo.com/product/ned/source/hardware/electrical_interface.html#electromagnet)).

Then, you need to set up the digital I/O pin.

- Select the pin where you connected your electromagnet cable.
- Set this pin to OUTPUT.
- Control your electromagnet by changing the state of the selected pin to LOW or HIGH.

Hint

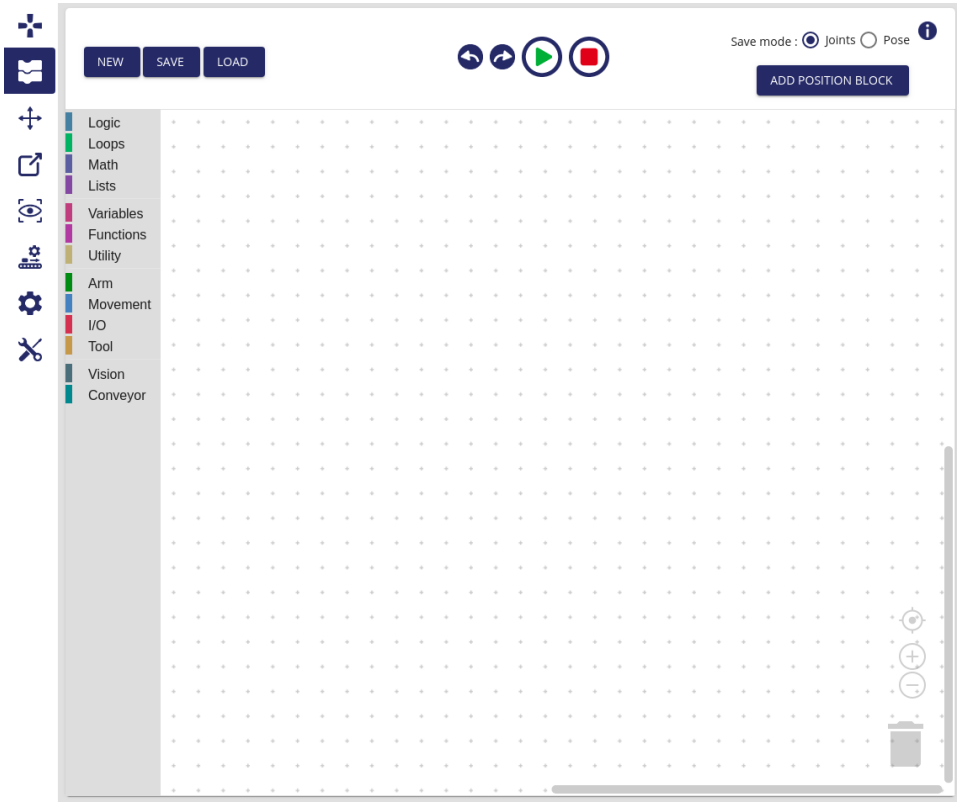
If you are using the connector provided with the electromagnet, **select 1A pin if you connect to the GPIO1** and **2A if you connect to the GPIO2**.

Programming

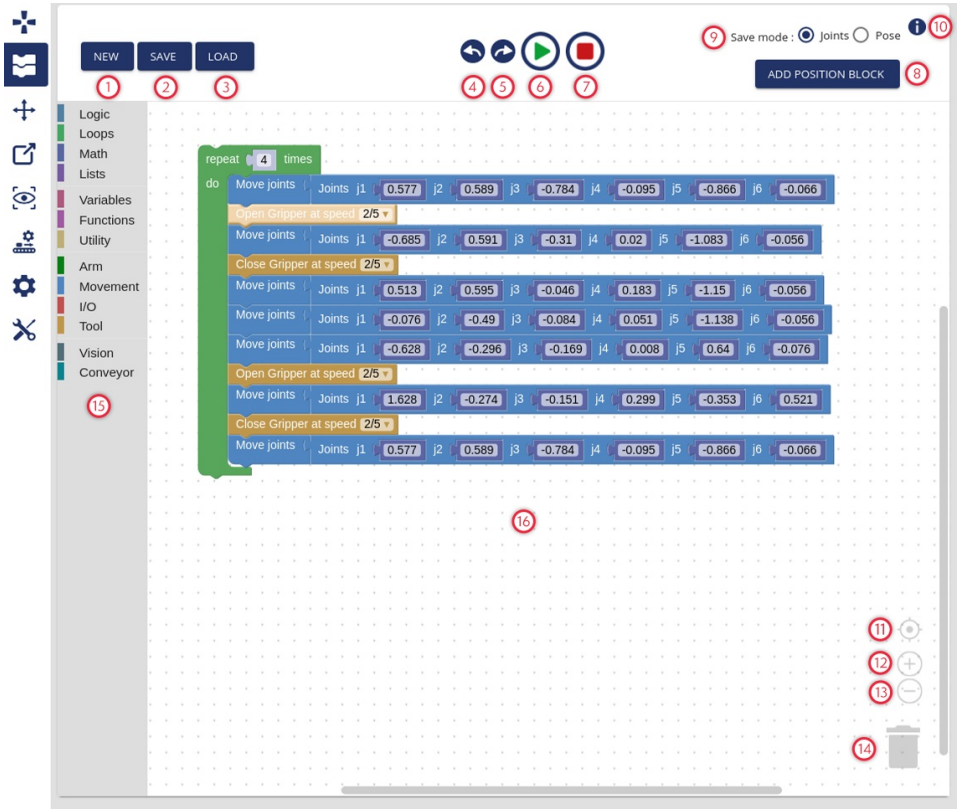
Project panel

Click the project button on the left menu to start creating and editing Blocky programs.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS



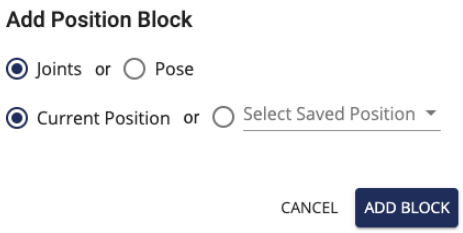
In the figure below, the top is the project Toolbar, and the left is the Blockly menu.



1. Create a new project: click the “NEW” button to create a new project.
2. Save program: click the “SAVE” button, to save a program. You can save a program on your computer or directly on Ned as an XML file.
3. Load project: click the “LOAD” button to import a Blockly program either from computer or from Ned. The file is a xml.
4. Undo. You can also use CTRL + Z.
5. Redo. You can also use CTRL + MAJ + Z.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

6. Play the program displayed on the workspace. Once the program is done (success or not), you will get a notification on the bottom of the screen.
7. Stop the current program execution.
8. Add a position block. This will open a pop-up:



This is the most useful way to add a position command block to your workspace. You can choose between a “Move Joint” block and a “Move Pose” block, and use either the current robot state, or a previously saved position.

9. Select the save mode when using the top button on Ned.

Hint

While the Niryo Block interface is open, you can add a “Move joint” or “Move pose” block with current joints values and pose values just by pressing and releasing the top button of the robot.

10. Help.
11. Clicking on this icon will center the workspace on your blocks.
12. Workspace zoom control.
13. Workspace zoom control.
14. To delete a block, simply drag it and drop it onto the trash. You can also select it and press the delete key on your keyboard.
15. The “Niryo Robot” functions in the block library cover almost all of the possible commands you can execute on Ned:
    - Arm commands
    - Movement commands
    - Tool commands
    - I/O setup and control
    - Vision commands
    - Conveyor control

Note

The “NIRYO BLOCKS” panel contains everything you need to create complete programs for Ned. This is probably the panel on which you will spend most of your time.

16. This is the workspace. Your whole program will be there.

Note

You cannot execute a new sequence while another sequence is running. If you want to start a new sequence, you will have to stop the previous one.

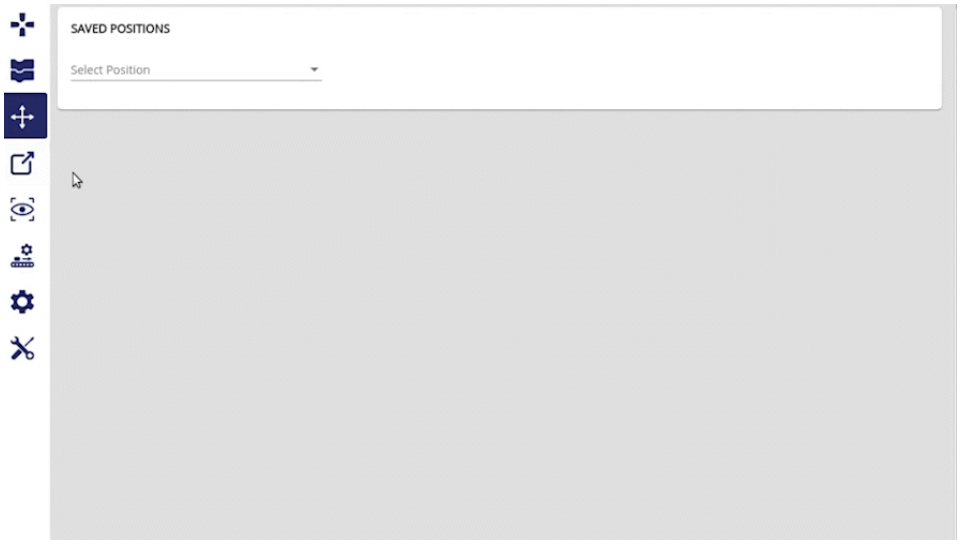
Hint

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Niryo blocks along with the learning mode is the perfect combination to easily create programs in no time. Activate the learning mode, then for each position you want, add a position block corresponding to the current state, and that's it.

Saved positions

In the "SAVED POSITION" panel, you can find all the saved positions.



When you select a position, you can see its properties:

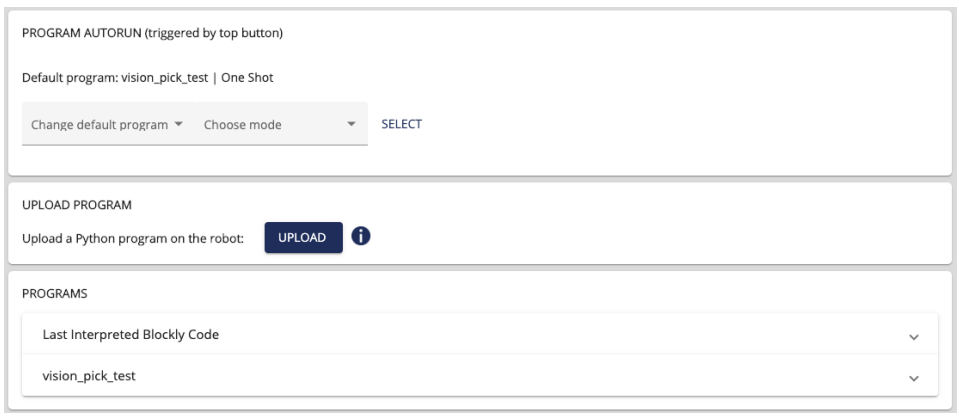
- Name
- Description
- Values: list of joints values in radian and the position and orientation of the TCP (Tool Center Point).

You can:

- Edit the position's name and description,
- Move the robot to the desired position,
- Delete the position.

Program settings

In the "Program Settings", you can set an autorunning program, upload a Python program and display a program.



Program Autorun

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

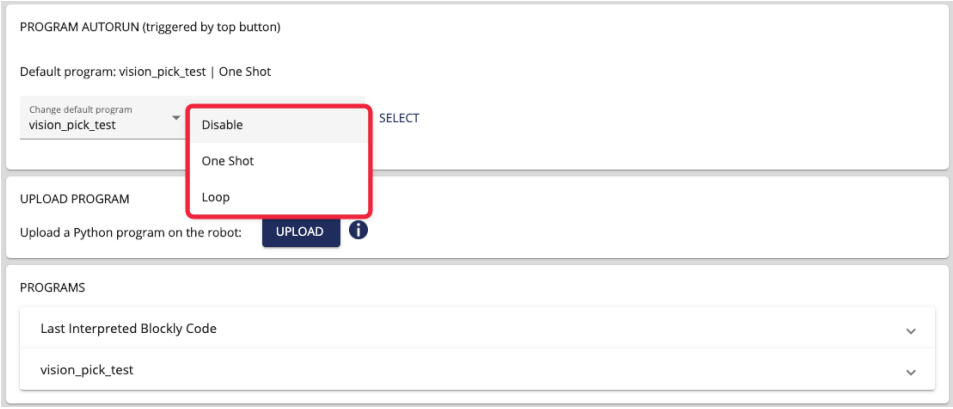
The “PROGRAM AUTORUN” functionality allows you to execute a set of sequences, directly by pressing the top button of the robot.

Here is how it works:

- Program a sequence with Blockly.
- Save the sequence on Ned.
- Add the sequence to the sequence autorun.
- Now trigger the sequence with the top button, without having to open Niryo Studio.

Hint

This is very useful when you use Ned in an exhibition event or for a demo. You will just need to power on Ned, wait for the LED to turn blue or green, and then press the button to run a given set of programs (that you previously programmed).



1. Display the default program triggered with the button
2. Select the program
3. Choose the “PROGRAM AUTORUN” mode. You have choices:
    a. ONE SHOT: when you press and release the top button, the given set of sequences will be run once. After that, Ned will go back to a “resting position” and activate the “Learning Mode”. You can start another run by pressing the button again. If you press the button while the set of sequences is running, it will stop the execution.
    b. LOOP: when you press and release the top button, the selected program will run in a loop, forever. You can press the button again to stop the execution.
4. SELECT button: select the program autorun.

Upload program

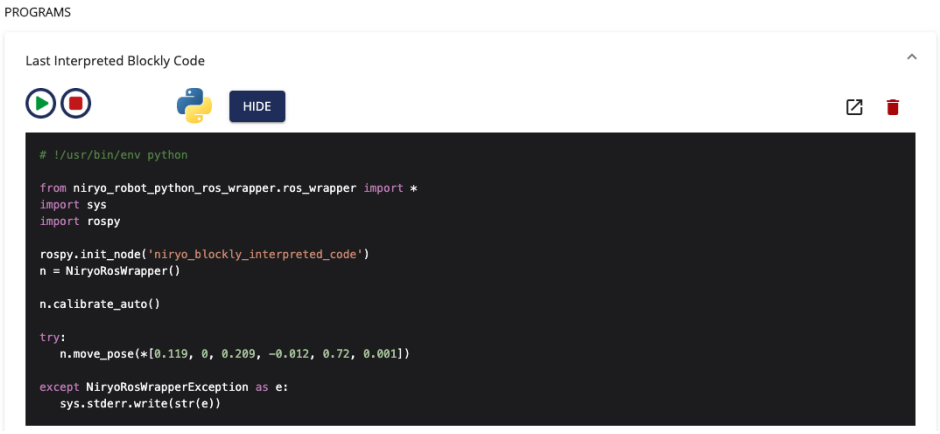
You can upload a Python script from your computer to Ned.

Programs list

You can pick a previously saved program from the select box. Click on a sequence to see all the properties and actions available for this program.

You can display the Python code of the program.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS



After the program details, you get a series of available actions:

1. Play the selected program (same as the “PLAY” button in Niryo blocks).
2. Stop the current program execution (same as the “STOP” button in Niryo blocks).
3. Open the program in Niryo blocks. This will make you switch to the “Niryo blocks” panel, and the program will be added to the current workspace.

> **Hint**
>
> This functionality is very useful when you want to duplicate and create a new program from an existing one.

4. Edit the program. You can modify the name, the description, and the Blockly XML.
5. Delete the program.

Addons

Conveyor Belt

The use of the Conveyor Belt is detailed in its documentation (https://docs.niryo.com/product/conveyor-belt/source/how_to_use.html#on-niryo-studio).

Vision Set

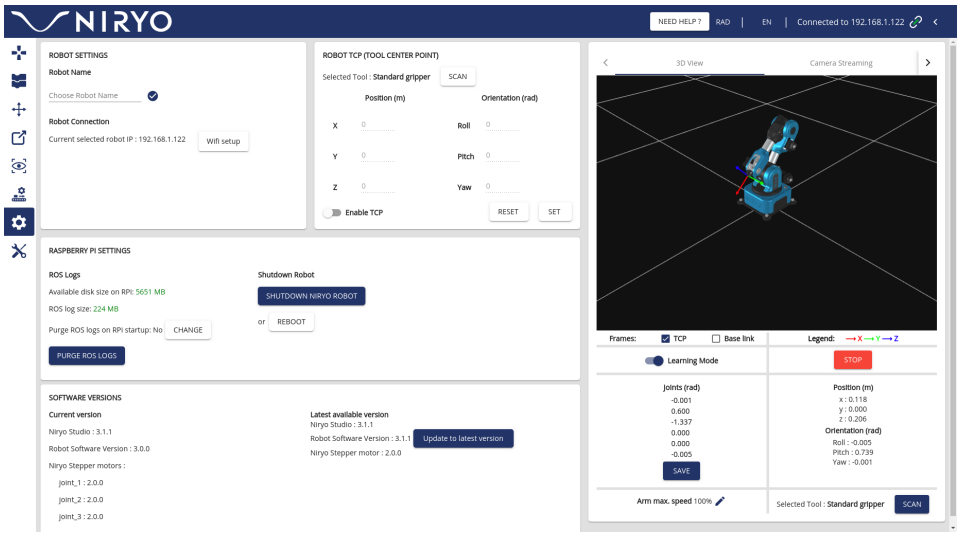
Discover how to use the vision in theVision Set documentation (https://docs.niryo.com/product/vision-set/source/setup.html#software).

Robot’s settings and update

Ned settings

Ned’s related parameters can be set in the “ROBOT SETTINGS” panel.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS



Robot Name

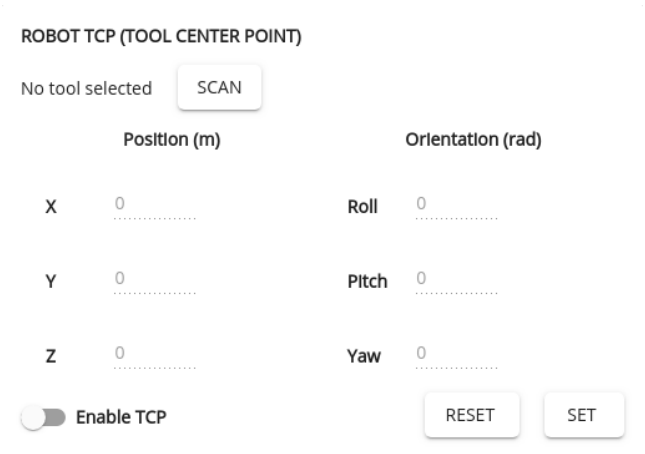
You can give a custom name to Ned.

- This name will be used when Ned is in hotspot mode to create the Wi-Fi network name.
- In Niryo Studio, when Ned is in connected mode, you will see that Ned is represented both by its IP address and its custom name.

You can set a custom name under “ROBOT SETTINGS”. After choosing the name, you need to reboot Ned.

Robot TCP (Tool Center Point)

The TCP (Tool Center Point) of the robot corresponds to the working point of the tool. From here, you can change the frame of the TCP.



When the TCP is enabled, every position move is set corresponding to the TCP frame.

When you plug a tool and if the TCP is enabled, the TCP frame will automatically update to the corresponding tool TCP.

Ned Wi-Fi setting

If Ned is already connected to a Wi-Fi network and you want to change your Wi-Fi network:

- Connect to Ned in “Hotspot mode” or in “Connected mode”

Note

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Note

If you are not connected to Ned, you will not be able to change its Wi-Fi network.

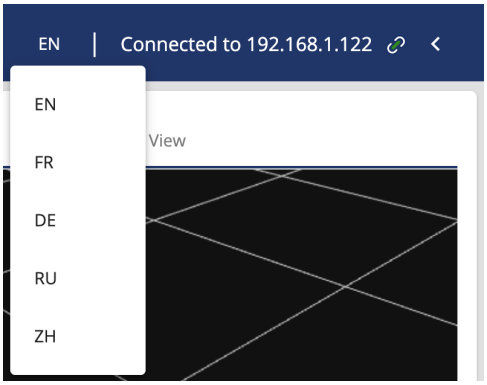
- Under “ROBOT SETTINGS”, check your current IP address.
- Click the Wi-Fi setup button and fill in the Wi-Fi name (SSID) and password of the Wi-Fi network you want Ned to connect to. Make sure to avoid typos in the name and password. Then, click on “Connect robot to Wi-Fi”.
- Ned will try to connect to the Wi-Fi, depending on the result, the robot’s reaction changes:
  - If successfully connected: the robot will reboot itself, the LED will become red then green. Connect to the same wifi network and then connect to Ned (Help: Using Ned on your Wi-Fi network (index.html#using-ned-on-your-wi-fi-network).)
  - If failed to connect to the SSID: the LED will remain blue (the LED didn’t change after ~30 seconds). Verify your password and try to connect again.
  - If the SSID can’t be found currently by the Raspberry Pi: you won’t be disconnected from Niryo Studio, but an error notification will be shown. You can retry within a few seconds (the wi-fi card may be disturbed by other wi-fi devices) or move your Raspberry Pi closer to the Wi-fi router.

Warning

If you changed Ned network configuration from Wi-fi to hotspot, troubles can occur when reconnecting via Niryo Studio, please reboot your robot in this case.

Language selection

The default language of Niryo Studio is English but you can change the current language with the language button on the top toolbar.



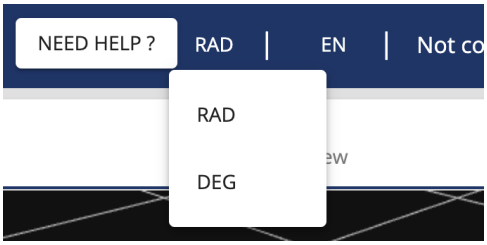
Note

The Blockly editor and the logs remain in English.

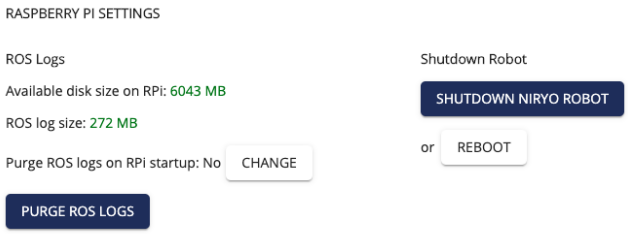
Angle unit selection

The default angle unit in robotics is radian. But you can change to degree with the corresponding button in the top toolbar.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS



Raspberry PI settings



ROS logs:

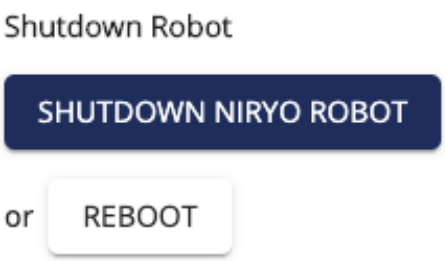
In this section, you can see:

- The available disk size on the Raspberry Pi. If the number is displayed in orange or red, it means there is no space enough on the micro-SD card.
- The size of the stored ROS logs: after some times, they can take a lot of space.
- You can choose to remove all previous ROS logs when Ned boots. This is very useful for a normal usage of Ned. Basically, you do not need all the ROS logs that are produced and stored, unless you are developing or debugging a specific point (in this case you might want to switch the option to “NO” and then turn it back to “YES” when you’re done).

Caution

If you never purge ROS logs (on boot or manually), the micro-SD card will eventually become full and might not work as expected. So only change the option to “NO” if you know what you are doing. In any case, if you do not really understand this functionality, always leave the option to “YES”.

- You can also remove all ROS logs immediately by clicking on the “Purge ROS logs” button.
- Shutdown Ned.



- You can shut down Ned with Niryo Studio. The “SHUTDOWN NIRYO ROBOT” will shut down the Raspberry Pi (same behaviour as if you press the physical top button for 3 seconds). Be sure to wait for the LED to turn red before completely powering off Ned.
- You can reboot Ned by clicking “Reboot”.

Note

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

If you reboot Ned, do not power it off. The LED will turn to purple, then red, then will become blue or green again.

Software versions

You can check that Ned and relative softwares are up to date by comparing the current and latest version for each of them.

- Niryo Studio: the desktop app to control Ned.
- Robot Software: the current version of the Niryo robot ROS stack running on Ned.
- Niryo Steppers motors: a list of the firmware versions running on each NiryoStepper attached to Ned. Click here to update the firmware version (https://niryo.com/docs/niryo-one/update-your-robot/update-niryo-steppers/).

Note

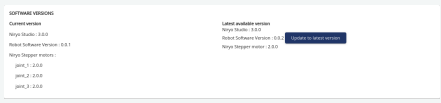
All the NiryoStepper motors must have the same version.

- Latest available versions (directly updated from our website).

Note

If an incompatible software version is detected, if the version you have is not supported anymore or an update is available, you will get a pop-up notification when you connect to Ned. Please follow the displayed instructions.

a. Message displayed when Ned is not connected to internet.



b. An update is available to update Ned software.

Ned software Update

Ned software inside the robot can be updated through Niryo Studio in only a few steps.

First, your Ned must have an access to internet, so you have to connect it to an ethernet (you have to edit via ssh the network configuration of Ned) or a Wi-fi (index.html#using-ned-on-your-wi-fi-network) network that provides internet.

When your robot is properly configured, at the next boot of the robot, it will check if an update is available. A pop-up will then be displayed at each following connection from Niryo Studio with Ned to inform you that an update is available for your robot.

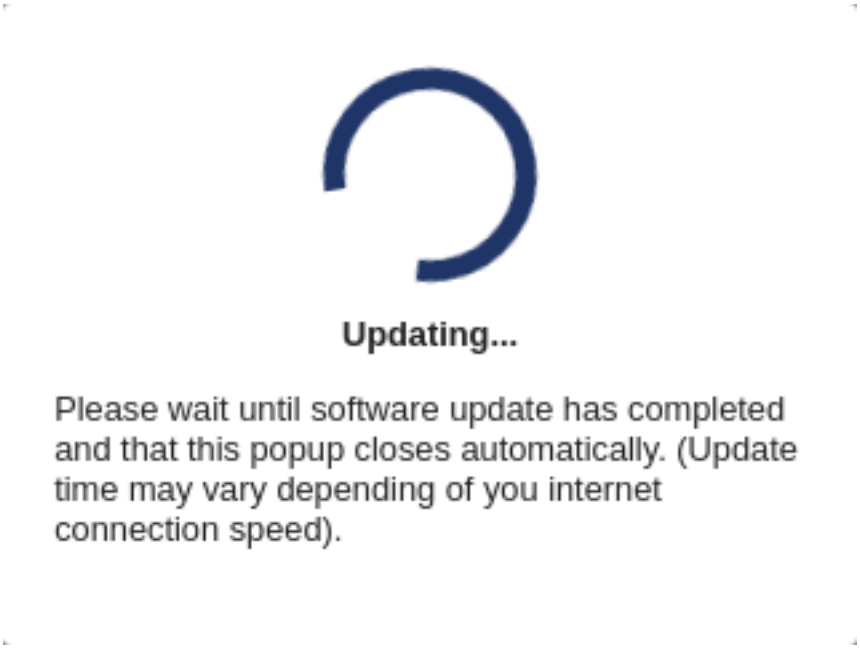
You then only have to click on the update button of the pop-up or the update button in thesoftware versions panel.

Warning

Ned will reboot itself at the end of the process (in case of update success) so it's recommended **not** to do any other operation with Ned before starting an update.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

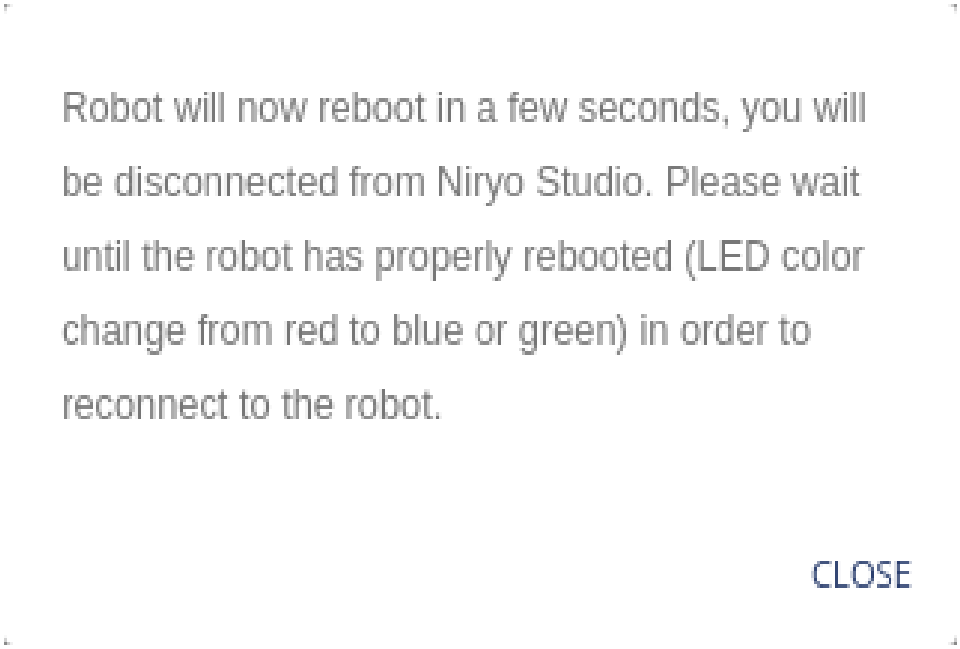
Once the update has started, a pop-up will indicate that Ned is updating and that it's downloading / applying the update.



Note

Update time is heavily impacted by the internet speed of your connection so if you connect to a wi-fi network, be sure that the raspberry is not too far away from the wi-fi modem.

Once the update has successfully ended, another pop-up will appear (closing the previous one) telling that your robot will reboot and you may be disconnected from Niryo Studio. Now you just have to wait for the robot to initialize properly (the red LED turning into green or blue) before reconnecting through Niryo Studio, the update process is done.



Warning

In case the update failed, it could be a faulty connection, try to do the update once more and if you still encounter a problem, please contact us at contact@niryo.com (mailto:contact%40niryo.com).

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Logs and Status

Logs

At the top of the Robot Status page, you can find 2 sections:

- Niryo Studio logs: logs produced by the desktop application.
- ROS logs: logs produced internally by Ned.

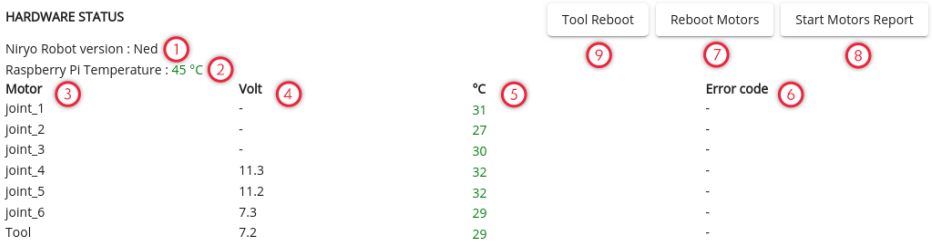
Hint

- You can save Ned logs or Niryo Studio logs on a text file.
- You can send this file to our technical support to help you in case of problem.

Hardware status

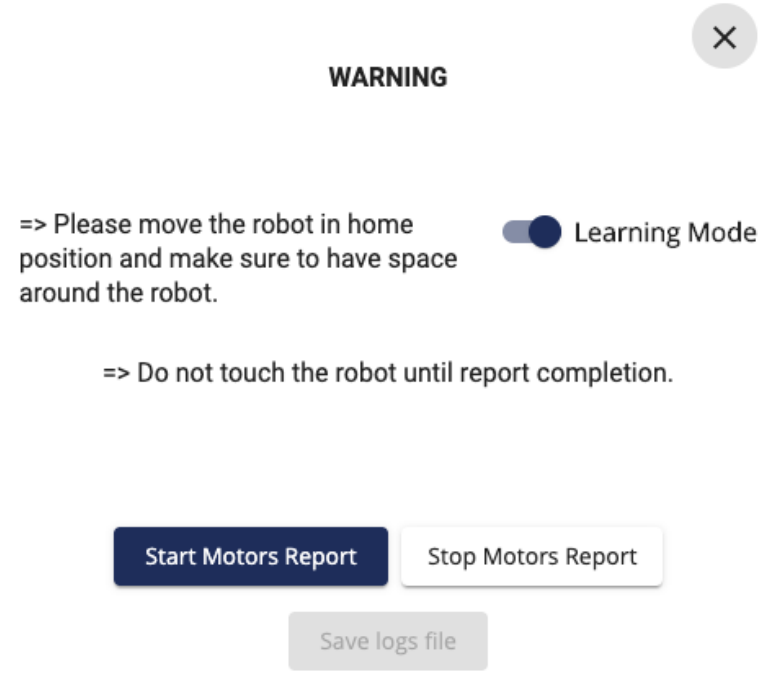
In the “HARDWARE STATUS” panel, you can find the hardware status of Ned.

In this section, you can find details about the controller temperature, motors voltage, temperature, and error code.

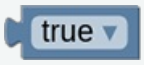
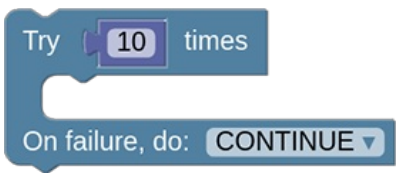


1. Ned hardware version.
2. Raspberry Pi temperature: the temperature of the Raspberry Pi is displayed in real time. If it’s displayed in orange or red, that means that something is not normal and the board is too hot. Make sure that your board temperature does not exceed 70°C.
3. List of connected motors: you should have 6 motors (from Joint_1 to Joint_6), if you have a connected tool, it should appear on the list.
4. Displayed measured voltage of servo motors (Dynamixels).
5. Measured temperature of each joint.
6. Error code: displays error code if there is any. Check troubleshooting chapter to resolve the problem.
7. Reboot motors: this button reboots servo motors without rebooting the entire robot, this can help you to resolve motors errors.
8. Start motors report: this option helps you to debug Ned. Click the Motor Report button and Start Motors Report.
9. Tool reboot: this option allows you to reboot the tool’s motor. This button is useful when a motor error occurs.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

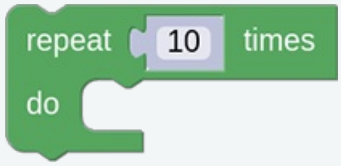
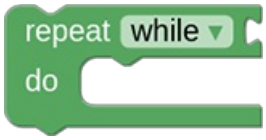
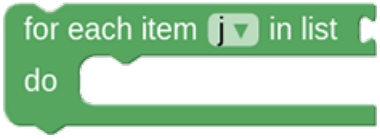


Logic

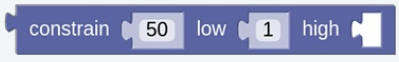
Concept	Explanation	Example
If/Else If/Else block	If the first value is true, then does the first block of statements. Otherwise, if the second value is true, does the second block of statements. If none is true, does the the last block of statements.	
Mathematical comparison operator block	Compares two values with an operator [=, ≠, <, ≤, >, ≥]. Returns true if the comparison is true.	
Logical comparision operator block	Compares two values with an operator [*and* / *or*]. Returns true if the comparison is true.	
Not block	Returns true if the input is false, and false if the input is true.	
True/False block	Returns either true or false.	
Null block	Returns null.	
Test block	Checks the condition in test block statement. If the condition is true, returns the value of the 'if true' value. Otherwise, returns the 'if false' value.	
Try/Except block	On the "Try" statement you can ask Ned to try to execute any type of action a certain amount of time You can then choose if Ned should CONTINUE or STOP in case he is "On failure". If not Ned will execute the next block.	

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Loops

Concept	Explanation	Example
Time loop block	Does one or several statements multiple times.	
While loop block	While a value is true, then do some statements.	
For loop block	Executes a loop based on a variable, from a starting number to an ending number with a specified interval, and executes the specified blocks.	
For each item in list loop block	For each item in a list, sets the variable “v” to the item, and then does some statements.	
Break block	Breaks out of the containing loop. Must be in a loop block.	

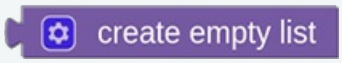
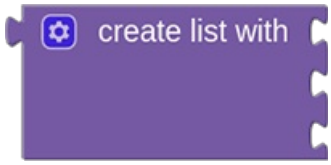
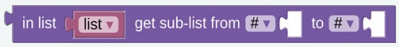
Math

Concept	Explanation	Example
Set number block	Sets a value.	
Mathematical operation block	Returns the value of the specified operation with two numbers. [+, -, *x*, ÷, ^].	
Check operation block	Checks if a number is even, odd, prime, whole, positive, negative or divisible.	
Round block	Rounds a number up or down.	
Mathematical list operations block	Returns the mathematical operation value of all number in the list [*sum*, *min*, *max*, *average*, *median*, *modes*, *standard deviation*, *random item*].	
Division remainder block	Returns the remainder from the division of two numbers.	
Constrain number block	Constrains a number to be between the specified limits (inclusive).	
Random integer block	Returns a random integer between the two specified limits (inclusive).	

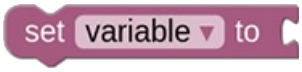
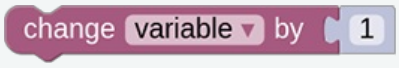
Lists

Concept	Explanation	Example

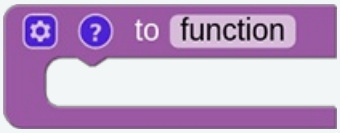
2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Concept	Explanation	Example
Create list block	Creates an empty list.	
Create list with items block	Creates a list with a specified number of items.	
Create list with one item block	Creates a list consisting of the given value repeated the specified number of times.	
List length block	Returns the length of a list.	
Is list empty block	Returns true if the list is empty.	
Item index finder block	Returns the index of the *first* or the *last* occurrence of the item in the list. Returns 0 if the item is not found.	
Get item block	*Returns / returns and removes / removes* the item at the specified position in a list. #1 is the first item.	
Set item index block	*Sets / inserts at* the item at the specified position in a list. #1 is the first item.	
Copy list-portion block	Creates a copy of the specified portion of a list.	
Sort list block	Sorts *numeric / alphabetic / alphabetic ignore case* by *ascending / descending* a copy of a list.	

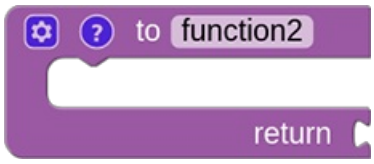
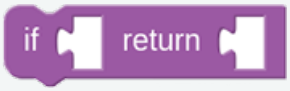
Variables

Concept	Explanation	Example
Create variable block	Creates a variable by its name.	
Set variable block	Sets this variable to be equal to the input.	
Change variable block	Changes this variable by the input.	
Get variable block	Returns the variable.	

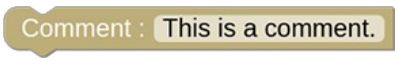
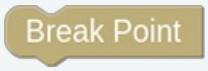
Functions

Concept	Explanation	Example
Create no output function block	Creates a function with no ouput. You can add input arguments.	

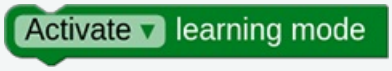
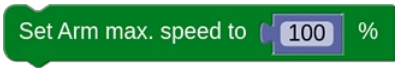
2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Concept	Explanation	Example
Create function with output block	Creates a function with an ouput. You can add input arguments.	
Function return block	If a value is true, then ends the function and returns the return value. Must be inside a function.	
Call function block	Runs the function.	

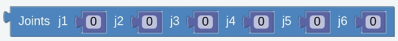
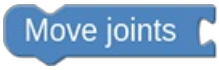
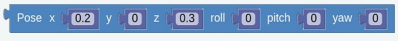
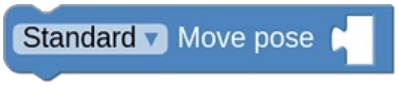
Utility

Concept	Explanation	Example
Wait time block	Creates a break time in the program.	
Comment block	Adds comments to the code. This block will not be executed. Note: accents are not accepted (é,à,è,...).	
Break Point block	Stops the execution of the program. Press the “Play” button to resume.	

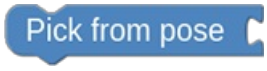
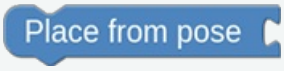
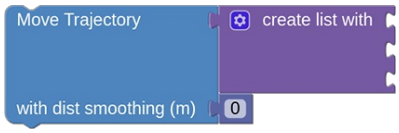
Arm

Concept	Explanation	Example
Learning mode block	*Activates* / *Deactivates* the learning mode.	
Set arm speed block	Sets the arm speed.	

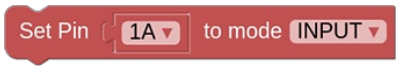
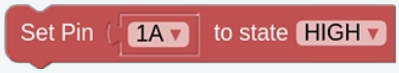
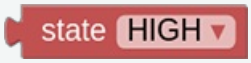
Movement

Concept	Explanation	Example
Joints block	Creates an object pose according to the robot’s joints values.	
Move joints block	Moves the robot according to a Joints block.	
Pose block	Creates an object pose according to the end effector’s cartesian coordinates.	
Move pose block	Moves the robot according to a pose block. You can change between “Standard” move (Point to Point), “Linear”, the robot will follow a linear trajectory from its position to the desired position, or “Try linear”, the robot will try to do a “Linear” trajectory, but if it can’t compute the linear trajectory, it will do a “Standard” move.	

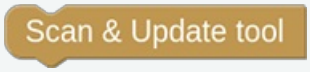
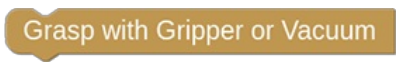
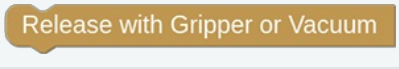
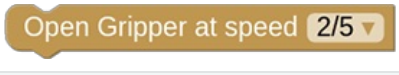
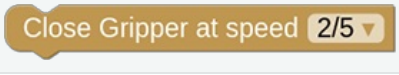
2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Concept	Explanation	Example
Shift block	Shifts the robot pose according to an axe *x* / *y* / *z* / *roll* / *pitch* / *yaw*. You can change between “Standard” move (Point to Point), “Linear”, the robot will follow a linear trajectory from its position to the desired position, or “Try linear”, the robot will try to do a “Linear” trajectory, but if it can’t compute the linear trajectory, it will do a “Standard” move.	
Pick from pose block	Moves the robot’s TCP (arm’s end point + tool coordinate) to a specified pose and activates the tool to pick an object.	
Place from pose block	Moves the robot’s TCP (arm’s end point + tool coordinate) to a specified pose and deactivates the tool to place an object.	
Move Trajectory block	The robot will pass through the list of goals with the desired distance smooth. The distance smooth is the radius from the goal where the robot will start to go towards the next point.	

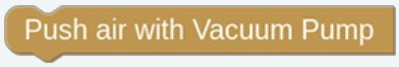
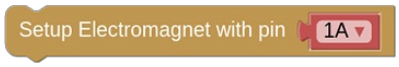
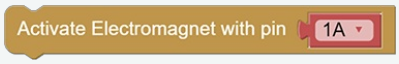
I/O

Concept	Explanation	Example
Get I/O block	Returns the number of the pin.	
Set I/O mode block	Sets I/O pin mode to *input* or *output*.	
Set output state block	Sets output pin state to *high* or *low*.	
Get input state block	Returns the input pin state.	
State block	Returns the state value *high* or *low*.	
Set Switch state block	Sets the 12V switch state to *high* or *low*.	

Tool

Concept	Explanation	Example
Scan tool block	Scans and updates current tool.	
Grasp block	Activates the *gripper* / *vacuum*.	
Release block	Deactivates the *gripper* / *vacuum*.	
Open gripper block	Opens the gripper at a certain speed.	
Close gripper block	Closes the gripper at a certain speed.	

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Concept	Explanation	Example
Pull air block	Pulls the air in the vacuum pump.	
Push air block	Pushes the air in the vacuum pump.	
Setup electromagnet block	Selects on which pin the electromagnet is connected.	
Activate electromagnet block	Activates the electromagnet power.	
Deactivate electromagnet block	Deactivates the electromagnet power.	
Activate / Deactivate TCP block	Activates or deactives the TCP.	
Set TCP block	Sets TCP frame	

Vision

The Vision blocks and Vision templates are detailed in the Vision Set Documentation (https://docs.niryo.com/product/vision-set/source/how_to_use_blockly.html#blocks-description).

Conveyor blocks

The Conveyor blocks are detailed in the Conveyor Belt documentation (https://docs.niryo.com/product/conveyor-belt/source/how_to_use.html#blockly).

Advanced Programming

There are many other ways to develop on Ned.

For more information, please refer to the following links:

- Python (https://docs.niryo.com/dev/pyniryo/index.html)
- ROS (https://docs.niryo.com/dev/ros/index.html)
- Modbus (https://docs.niryo.com/dev/modbus/index.html)

Suggest a modification | Download as PDF

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Ned ROS documentation

This documentation contains everything you need to understand Ned's functioning & how to control it through ROS.
It is made as well for users who are using the "physic" robot as those who want to use a virtual version.

Preamble

Before diving into the software documentation, you can learn more about the robot development in theOverwiew (index.html#document-source/overview) section.

Also, if you want to use a simulated version of Ned, browse the Simulation (index.html#document-source/simulation) section.

Ned Control via ROS

Ned is fully based on ROS.

ROS Direct control

Important

To control the robot directly with ROS, you will need either to be connected in SSH to the physic robot, or to use the simulation.

ROS is the most direct way to control the robot. It allows:

- to send command via the terminal in order to call services, trigger action, ...
- to write a entire Python/C++ node in a script to realize a full process.

See ROS (index.html#document-source/ros) section to see all Topic & Services available.

Python ROS Wrapper

Important

To use Python ROS Wrapper, you will need either to be connected in SSH to the physic robot, or to use the simulation.

The Python ROS Wrapper is built on top of ROS to allow a faster development than ROS. Programs are run directly on the robot which allows to trigger them with the robot's button once a computer is no longer needed.

See Python ROS Wrapper (index.html#document-source/ros_wrapper) to see which functions are accessible and examples on how to use them.

More ways

Other methods are available to control the robot allowing the user to code and run programs outside his terminal.

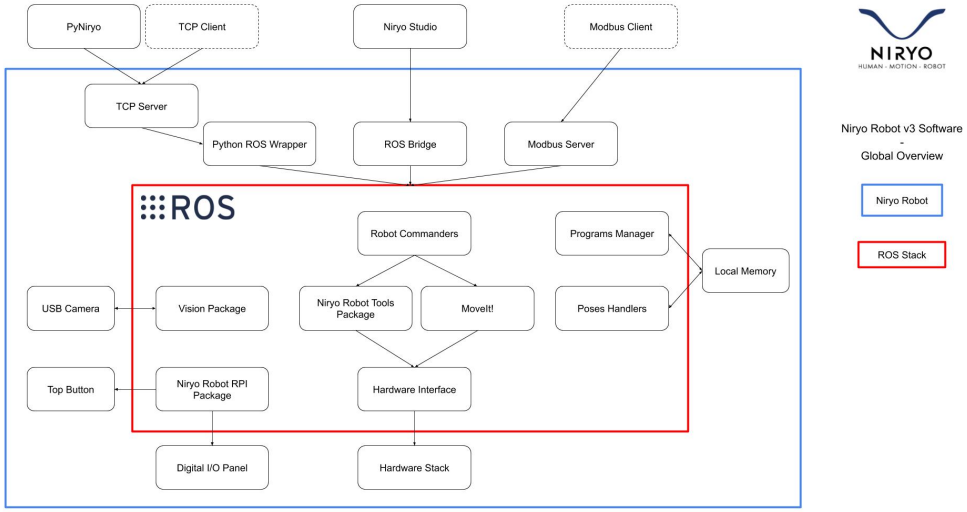
Learn more on this section (index.html#document-source/more).

ROS Stack overview

Ned is a robot based on Raspberry, Arduino & ROS. It uses ROS to make the interface between Hardware and high-level bindings.

On the following figure, you can see a global overview of the Niryo's robot software in order to understand where are placed each part of the software.

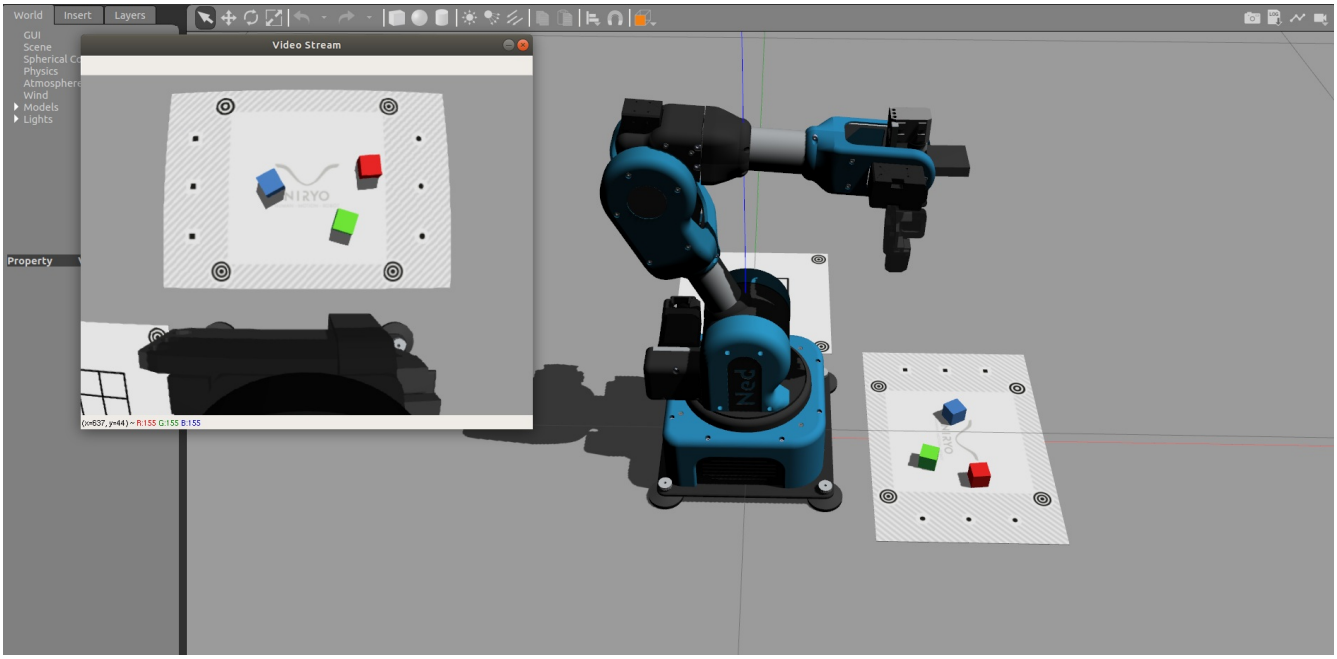
2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS



Niryo robot v3 software

Use Niryo robot through simulation

The simulation allows to control a virtual Ned directly from your computer.



Ned in Gazebo Simulation

In this tutorial, you will learn how to setup simulation on a computer.

Note

You can use Niryo Studio with the simulation (https://docs.niryo.com/product/ned/source/software/niryo_studio.html#connecting-simulation-to-niryo-studio/). To do so, you just have to connect Niryo Studio to "Localhost".

Simulation environment installation

Attention

The whole ROS Stack is developed and tested on ROS **Melodic** which requires **Ubuntu 18.04** to run correctly. The using of another ROS version or OS may lead to malfunctions of some packages.

To allow the simulation to run on your computer, you will need to install ROS and some packages.

Installation index:

Prepare environment

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

> **Note**
>
> All terminal command listed are for Ubuntu users.

Place yourself in the folder of your choice and create a folder **catkin_ws_niryo_ned** as well as a sub-folder **src**:

```
mkdir -p catkin_ws_niryo_ned/src
```

Then go to the folder **catkin_ws_niryo_ned** and clone Ned repository in the folder **src**. For the future operation, be sure to stay in the **catkin_ws_niryo_ned** folder:

```
cd catkin_ws_niryo_ned
git clone https://github.com/NiryoRobotics/ned_ros src
```

Install ROS dependencies

Install ROS¶
You firstly need to install ROS Melodic. To do so, follow the ROS official tutorial here (http://wiki.ros.org/melodic/Installation/Ubuntu) and chose the **Desktop-Full Install**.

Install additional packages¶
To ensure the functioning of all Ned's packages, you need to install several more packages:

Method 1: Quick installation via ROSDep¶
For each packages, we have referenced all the dependencies in their respective *package.xml* file, which allow to install each dependency via *rosdep* command:

```
rosdep update
rosdep install --from-paths src --ignore-src --default-yes --rosdistro melodic --skip-keys "python-rpi.gpio"
```

Method 2: Full installation¶
ROS packages needed are:

- build-essential
- catkin
- python-catkin-pkg
- python-pymodbus
- python-rosdistro
- python-rospkg
- python-rosdep-modules
- python-rosinstall python-rosinstall-generator
- python-wstool

To install a package on Ubuntu:

```
sudo apt install <package_name>
```

Melodic specific packages needed are:

- moveit
- control
- controllers
- tf2-web-republisher
- rosbridge-server
- joint-state-publisher-gui

To install a ROS Melodic's package on Ubuntu:

```
sudo apt install ros-melodic-<package_name>
```

Setup Ned ROS environment

> **Note**
>
> Be sure to be still placed in the **catkin_ws_niryo_ned** folder.

Then perform the **make** of Ned's ROS Stack via the command:

```
catkin_make
```

If no errors occurred during the **make** phase, the setup of your environment is almost complete!

It is necessary to source the configuration file to add all Ned packages to ROS environment. To do so, run the command:

```
source devel/setup.bash
```

It is necessary to run this command each time you launch a new terminal. If you want to make this sourcing appends for all your futur terminals, you can add it to your **bashrc** file:

```
echo "source $(pwd)/devel/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Installation is now finished!

Simulation utilization

Important

- If you haven't follow the step of Simulation Environment Installation. you won't be able to use the simulation.
- Hardware features won't be accessible.

The simulation is a powerful tool which allow to test new programs directly on your computer which prevent to transfer new code on the robot.
It also helps for developing purpose → no need to transfer code, compile and restart the robot which is way slower than doing it on a desktop computer.

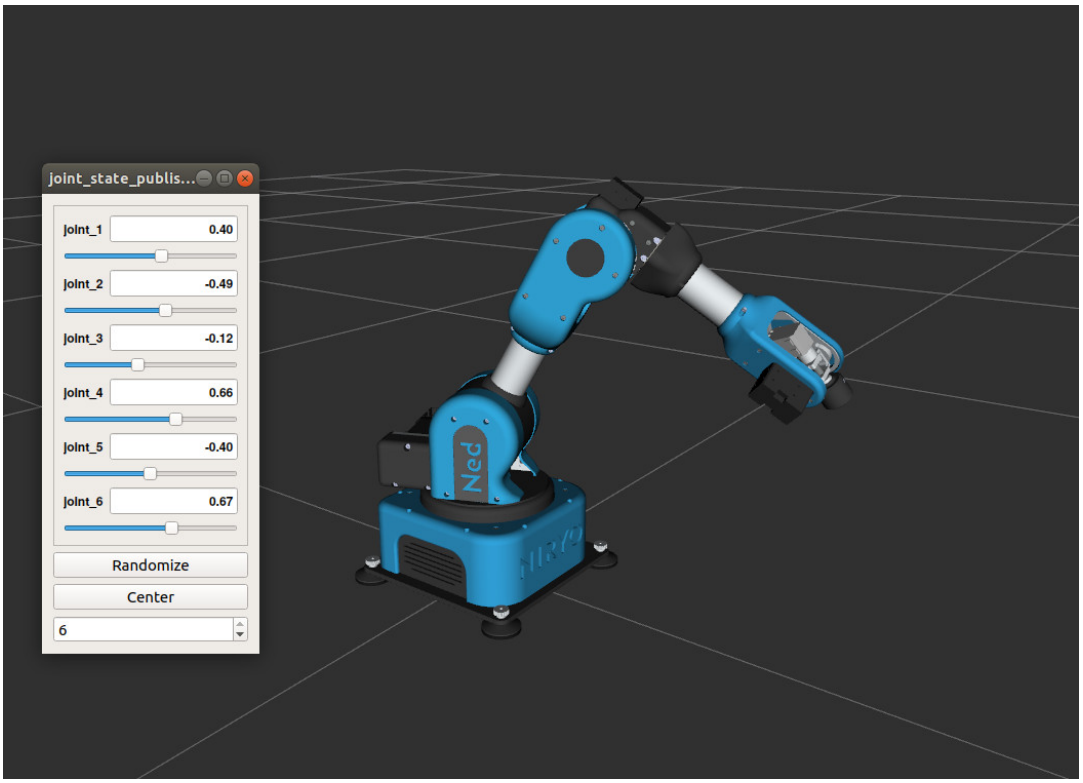
Without physics - Visualization

The visualization happens with Rviz which is a powerful tool.

Control with trackbar¶
This visualization allows an easy first control of the robot, and helps to understand joints disposal. You can access it by using the command:

```
roslaunch niryo_robot_description display.launch
```

Rviz should open with a window containing 6 trackbars. Each of these trackbars allows to control the corresponding joint.



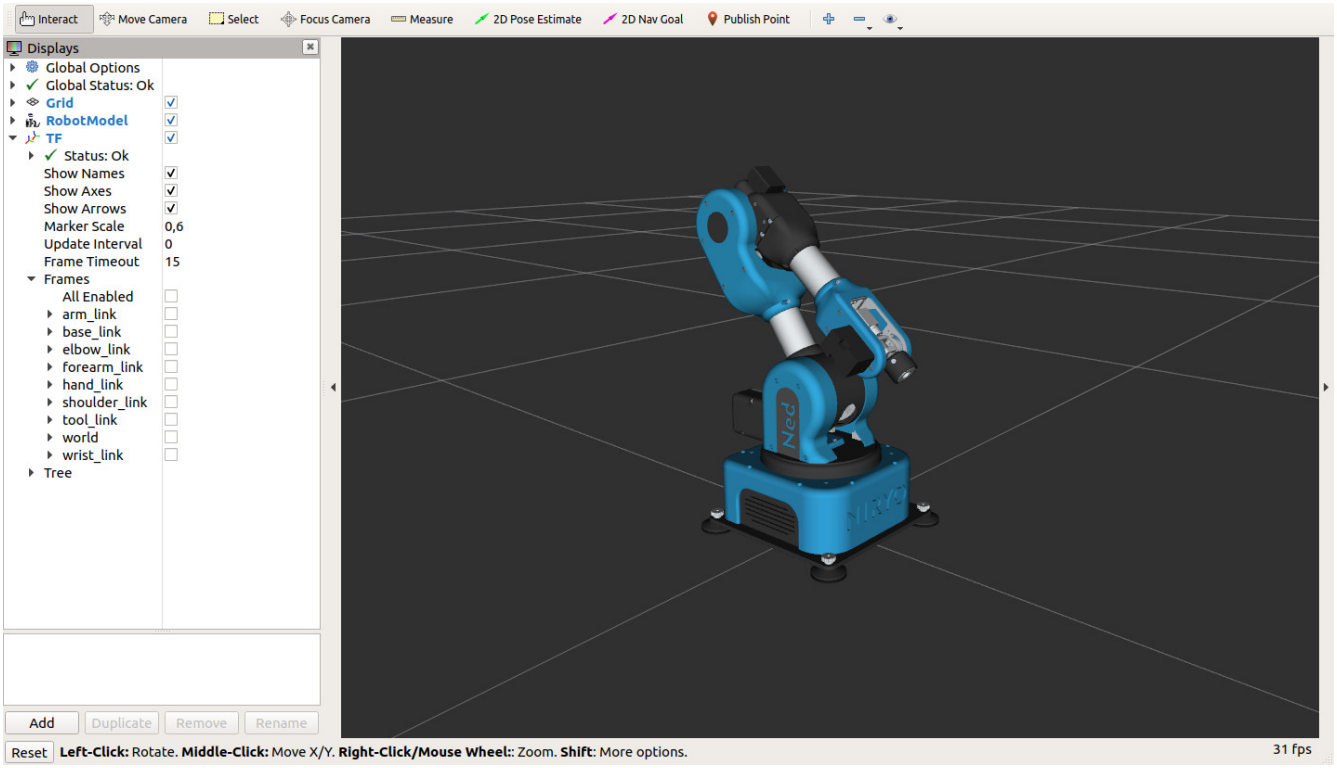
Example of trackbars use.

Control with ROS¶
Not only Rviz can display the robot, it can also be linked with ROS controllers to show robot's actions from ROS commands!
This method can help you debugging ROS topics, services and also, API scripts.

To run it:

```
roslaunch niryo_robot_bringup desktop_rviz_simulation.launch
```

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS



Rviz opening, with the robot ready to be controlled with ROS!

With physics - Simulation

For the simulation, Ned uses Gazebo, a well known tool among the ROS community. It allows:

- Collision.
- World creation → An virtual environment in which the robot can deal with objects.
- Gripper & Camera using.

The Niryo Gripper 1 has been replicated in Gazebo. The Camera is also implemented.

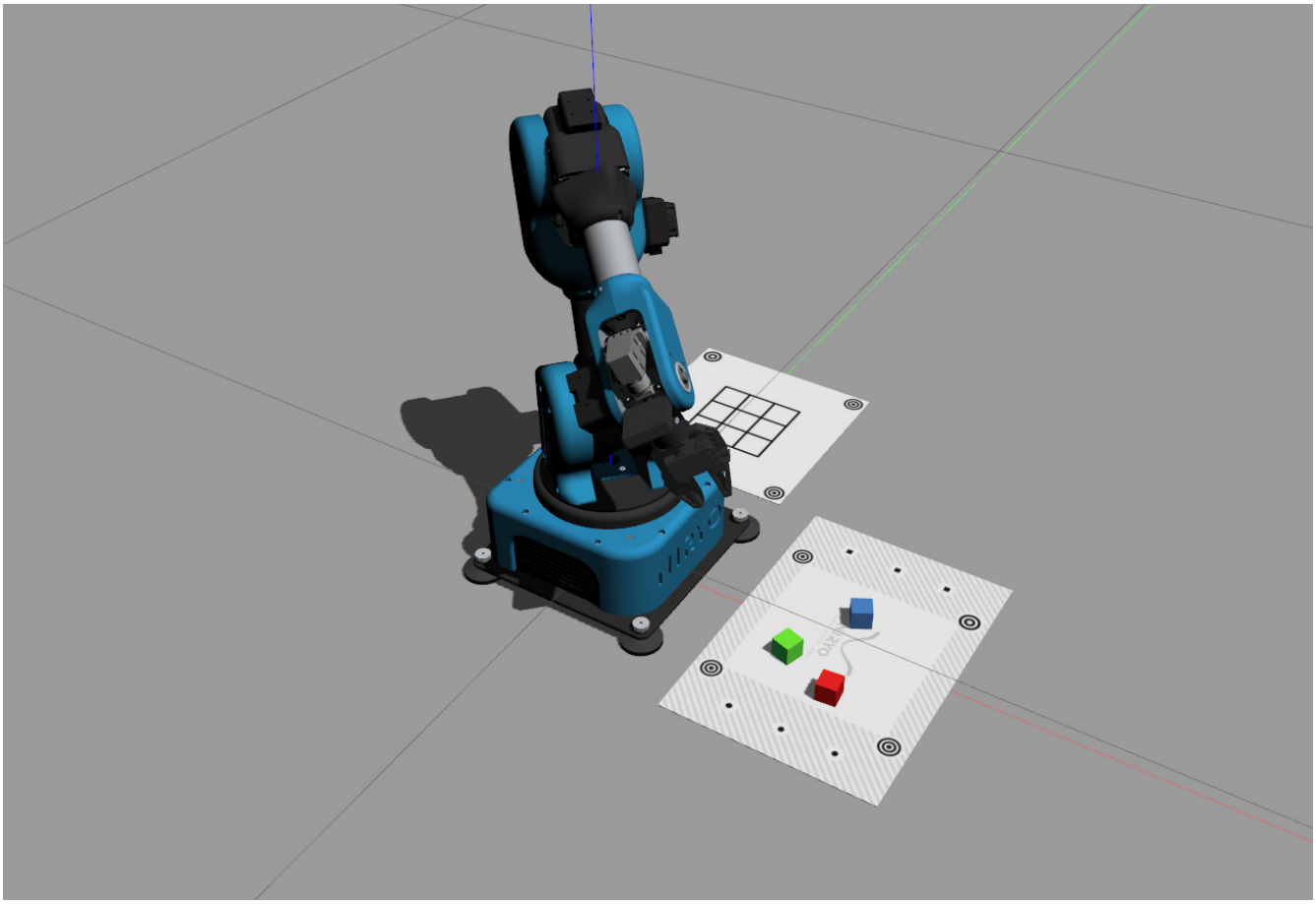
> **Note**
>
> Gazebo also generates camera distortion, which brings the simulation even closer from the reality!

Launch simulation¶
A specific world has been created to use Ned in Gazebo with 2 workspaces.

To run it:

```
roslaunch niryo_robot_bringup desktop_gazebo_simulation.launch
```

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS



Gazebo view, with the robot ready to be controlled with ROS!

Note

You can edit Gazebo world to do your own! It's placed in the folder*worlds* of the package niryo_robot_gazebo.

Simulation option¶
The user can disable 3 things by adding the specific string to the command line:

- the Gazebo graphical interface:*gui:=false*.
- the Camera & the Gripper - Vision & Gripper wise functions won't be usable:*gripper_n_camera:=false*.

Hint

Gazebo can be very slow. If your tests do not need Gripper and Camera, consider using Rviz to alleviate your CPU.

ROS Stack documentation

ROS Logo

ROS (Robot Operating System) is an Open-Source Robotic Framework which allows to ease robot software development. The framework is used in almost each part of Ned software.

The high-level packages (motion planner, vision, ...) are coded in Python to give good readability whereas communication with Hardware is developed in C++ to ensure speed.

Note

To learn more about ROS, go on Official ROS Wiki (http://wiki.ros.org/).

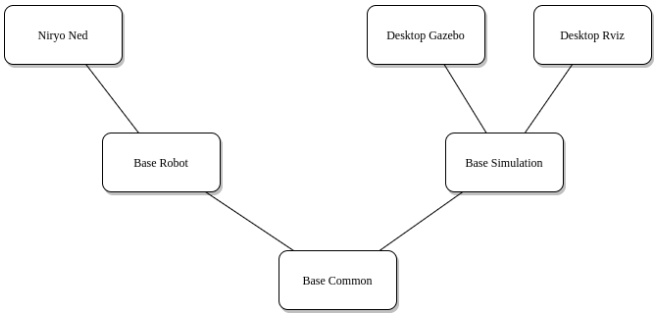
In this section, you will have access to all information about each Niryo Robot's ROS packages.

Niryo_robot_bringup

This packages provides config and launch files to start Ned and ROS packages with various parameters.

Launch files are placed in the*launch* folder. Only files with **.launch** extension can be executed.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS



Bring Up Launch Files' organization

On RaspberryPI

Ned¶
The file **niryo_ned_robot.launch** allows to launch ROS on a Raspberry Pi 4.
This file is automatically launched when Ned boots (Ned RPi4B image).

Command to launch Ned's ROS Stack:

roslaunch niryo_robot_bringup niryo_ned_robot.launch

On Desktop (Simulation)

As the simulation happens on a computer, the hardware-related stuff is not used.

For both of following launch files, you can set:

- *gui* to "false" in order to disable graphical interface.

Gazebo simulation¶
Run Gazebo simulation. The robot can do everything that is not hardware-related:

- move, get_pose.
- use the camera (to disable it, set "camera" parameter to 'false').
- use the Gripper 1 (to disable it, set "simu_gripper" parameter to 'false').
- save/run programs, go to saved pose, ...

Command to launch the simulation:

```
roslaunch niryo_robot_bringup desktop_gazebo_simulation.launch
```

To disable camera & gripper:

```
roslaunch niryo_robot_bringup desktop_gazebo_simulation.launch gripper_n_camera:=false
```

Rviz simulation¶
Run Rviz simulation. You can access same features as Gazebo except Camera & Gripper.

To run it, use the command:

```
roslaunch niryo_robot_bringup desktop_rviz_simulation.launch
```

Notes - Ned Bringup

niryo_robot_base files setup many rosparams, these files should be launched before any other package.

Niryo_robot_arm_commander

This package is the one dealing with all commander related stuff.
It is composed of only one node, which is running separately the arm commander and the tool commander.

Commander node

The ROS Node is made to interact with:

- The Arm through MoveIt!
- The tools through the tool controller.

All commands are firstly received on the actionlib server which:

- Handles concurrent requests.
- Checks if the command can't be processed due to other factors (ex: learning mode).
- Validates parameters.
- Calls required controllers and returns appropriate status and message.

The namespace used is: `/niryo_robot_arm_commander/`

Parameters - Commander¶

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Commander's Parameters

Name	Description
`reference_frame`	Reference frame used by MoveIt! for moveP. Default : 'world'
`move_group_commander_name`	Name of the group that MoveIt is controlling. By default: "arm"
`jog_timer_rate_sec`	Publish rate for jog controller
`simu_gripper`	If you are using the simulated Gripper and want to control the Gripper

Action Server - Commander¶

Commander Package Action Servers

Name	Message Type	Description
`robot_action`	RobotMove	Command the arm and tools through an action server

Services - Commander¶

Commander Package Services

Name	Message Type	Description
`is_active`	GetBool	Indicate whereas a command is actually running or not
`stop_command`	Trigger	Stop the actual command
`set_max_velocity_scaling_factor`	SetInt	Set a percentage of maximum speed
`/niryo_robot/kinematics/forward`	GetFK	Compute a Forward Kinematic
`/niryo_robot/kinematics/inverse`	GetIK	Compute a Inverse Kinematic

Messages - Commander¶

Commander Package Messages

Name	Description
ArmMoveCommand	Message to command the arm
ShiftPose	Message for shifting pose
PausePlanExecution	Pause movement execution

All these services are available as soon as the node is started.

Dependencies - Commander¶

- actionlib
- actionlib_msgs
- control_msgs
- geometry_msgs
- MoveIt!
- moveit_msgs
- Niryo_robot_msgs
- Niryo robot tools commander package
- python-numpy
- ros_controllers
- rosbridge_server
- sensor_msgs
- std_msgs
- tf2_web_republisher
- trajectory_msgs

Action, services & messages files - Commander

RobotMove (Action)¶

```
# goal
niryo_robot_arm_commander/ArmMoveCommand cmd
---
# result
int32 status
string message
---
# feedback
niryo_robot_msgs/RobotState state
```

GetFK (Service)¶

```
float32[] joints
---
niryo_robot_msgs/RobotState pose
```

GetIK (Service)¶

```
niryo_robot_msgs/RobotState pose
---
bool success
float32[] joints
```

JogShift (Service)¶

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
int32 JOINTS_SHIFT = 1
int32 POSE_SHIFT = 2

int32 cmd

float32[] shift_values

---
int32 status
string message
```

ArmMoveCommand (Message)¶

```
int32 JOINTS = 0            # uses joints
int32 POSE = 1              # uses position and rpy
int32 POSITION = 2          # uses position
int32 RPY = 3               # uses rpy
int32 POSE_QUAT = 4         # uses position and orientation
int32 LINEAR_POSE = 5       # uses position and rpy
int32 SHIFT_POSE = 6        # uses shift
int32 SHIFT_LINEAR_POSE = 7 # uses shift
int32 EXECUTE_TRAJ = 8      # uses dist_smoothing, list_poses
int32 DRAW_SPIRAL = 9

int32 cmd_type

float64[] joints
geometry_msgs/Point position
niryo_robot_msgs/RPY rpy
geometry_msgs/Quaternion orientation
niryo_robot_arm_commander/ShiftPose shift

geometry_msgs/Pose[] list_poses
float32 dist_smoothing
```

PausePlanExecution (Message)¶

```
int8 STANDBY = 0
int8 PLAY = 1
int8 PAUSE = 2
int8 RESUME = 3
int8 CANCEL = 4

int8 state
```

ShiftPose (Message)¶

```
int32 axis_number
float64 value
```

Niryo_robot_description

This package contains URDF files and meshes (collada + stl) for Ned.

To display Ned on Rviz:

```
roslaunch niryo_robot_description display.launch
```

Note : 3D visualization is not available on Ned Raspberry Pi4 image. To use the following commands, you must have setup Ned ros stack on your computer.

Niryo_robot_gazebo

Usage

This package contains models, materials & Gazebo worlds.

When launching the Gazebo version of the ROS Stack, the file**niryo_robot_gazebo_world.launch.xml** will be called to generate the Gazebo world.

Create your own world

Create your world's file and put it on the folder*worlds*. Once it is done, you have to change the parameter **world_name** in the file **niryo_robot_gazebo_world.launch.xml**.

You can take a look at the Gazebo world by launching it without robot by precising the world name in the arg*world_name*:

```
roslaunch niryo_robot_gazebo niryo_gazebo_world.launch world_name:=niryo_cube_world
```

Niryo_robot_msgs

This package contains standard messages which can be used by all other packages.

Niryo messages

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Ned Messages

Name	Description
CommandStatus	Enum-wise message for status code
ObjectPose	x, y, z, roll, pitch, yaw
RobotState	position, rpy, quaternion
RPY	roll, pitch, yaw
HardwareStatus	several hardware informations
SoftwareVersion	several software version

Niryo services

Ned Services

Name	Description
GetBool	Return a bool
GetInt	Return a integer
GetStringList	Return a list of string
SetBool	Set a bool and return status
SetInt	Set a integer and return status
SetString	Set a string and return status
Trigger	Trigger a task

Niryo message dependencies

- geometry_msgs

Niryo message files

CommandStatus¶

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
int32 val

# overall behavior
int32 SUCCESS = 1
int32 CANCELLED = 2
int32 PREEMPTED = 3

int32 FAILURE = -1
int32 ABORTED = -3
int32 STOPPED = -4

int32 ROS_ERROR = -20

int32 FILE_ALREADY_EXISTS = -30

int32 UNKNOWN_COMMAND = -50
int32 NOT_IMPLEMENTED_COMMAND = -51
int32 INVALID_PARAMETERS = -52

# - Hardware
int32 HARDWARE_FAILURE = -110
int32 HARDWARE_NOT_OK = -111
int32 LEARNING_MODE_ON = -112
int32 CALIBRATION_NOT_DONE = -113
int32 DIGITAL_IO_PANEL_ERROR = -114
int32 LED_MANAGER_ERROR = -115
int32 BUTTON_ERROR = -116
int32 WRONG_MOTOR_TYPE = -117
int32 DXL_WRITE_ERROR = -118
int32 DXL_READ_ERROR = -119
int32 CAN_WRITE_ERROR = -120
int32 CAN_READ_ERROR = -121
int32 NO_CONVEYOR_LEFT = -122
int32 NO_CONVEYOR_FOUND = -123
int32 CONVEYOR_ID_INVALID = -124
int32 CALIBRATION_IN_PROGRESS = -125

# - Vision
int32 VIDEO_STREAM_ON_OFF_FAILURE = -170
int32 VIDEO_STREAM_NOT_RUNNING = -171
int32 OBJECT_NOT_FOUND = -172
int32 MARKERS_NOT_FOUND = -173

# - Commander
# Arm Commander
int32 ARM_COMMANDER_FAILURE = -220
int32 GOAL_STILL_ACTIVE = -221
int32 JOG_CONTROLLER_ENABLED = -222
int32 CONTROLLER_PROBLEMS = -223
int32 SHOULD_RESTART = -224
int32 JOG_CONTROLLER_FAILURE = -225

int32 PLAN_FAILED = -230
int32 NO_PLAN_AVAILABLE = -231
int32 INVERT_KINEMATICS_FAILURE = -232

# Tool Commander
int32 TOOL_FAILURE = -251
int32 TOOL_ID_INVALID = -252
int32 TOOL_NOT_CONNECTED = -253
int32 TOOL_ROS_INTERFACE_ERROR = -254

# - Pose Handlers
int32 POSES_HANDLER_CREATION_FAILED = -300
int32 POSES_HANDLER_REMOVAL_FAILED = -301
int32 POSES_HANDLER_READ_FAILURE = -302
int32 POSES_HANDLER_COMPUTE_FAILURE = -303

int32 WORKSPACE_CREATION_FAILED = -308

# - Programs Manager
int32 PROGRAMS_MANAGER_FAILURE = -320
int32 PROGRAMS_MANAGER_READ_FAILURE = -321
int32 PROGRAMS_MANAGER_UNKNOWN_LANGUAGE = -325
int32 PROGRAMS_MANAGER_NOT_RUNNABLE_LANGUAGE = -326
int32 PROGRAMS_MANAGER_EXECUTION_FAILED = -327
int32 PROGRAMS_MANAGER_STOPPING_FAILED = -328
int32 PROGRAMS_MANAGER_AUTORUN_FAILURE = -329
int32 PROGRAMS_MANAGER_WRITING_FAILURE = -330
int32 PROGRAMS_MANAGER_FILE_ALREADY_EXISTS = -331
int32 PROGRAMS_MANAGER_FILE_DOES_NOT_EXIST = -332


# - Serial
int32 SERIAL_FILE_ERROR = -400
int32 SERIAL_UNKNOWN_ERROR = -401


# - System Api Client
int32 SYSTEM_API_CLIENT_UNKNOWN_ERROR = -440
int32 SYSTEM_API_CLIENT_INVALID_ROBOT_NAME = -441
int32 SYSTEM_API_CLIENT_REQUEST_FAILED = -442
```

ObjectPose¶

```
float64 x
float64 y
float64 z

float64 roll
float64 pitch
float64 yaw
```

RobotState¶

```
geometry_msgs/Point position
niryo_robot_msgs/RPY rpy
geometry_msgs/Quaternion orientation
```

RPY¶

```
# roll, pitch and yaw

float64 roll
float64 pitch
float64 yaw
```

HardwareStatus¶

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
std_msgs/Header header

# Raspberry Pi board
int32 rpi_temperature

# Robot version : 1 Ned (only one until needed modification)
int32 hardware_version

# Motors
bool connection_up
string error_message
bool calibration_needed
bool calibration_in_progress

string[] motor_names
string[] motor_types

int32[] temperatures
float64[] voltages
int32[] hardware_errors
string[] hardware_errors_message
```

SoftwareVersion¶

```
string rpi_image_version
string ros_niryo_robot_version

string[] motor_names
string[] stepper_firmware_versions
```

Niryo Service files

GetBool¶

```
---
bool value
```

GetInt¶

```
---
int32 value
```

GetNameDescriptionList¶

```
---
string[] name_list
string[] description_list
```

GetStringList¶

```
---
string[] string_list
```

SetBool¶

```
bool value
---
int32 status
string message
```

SetInt¶

```
int32 value
---
int32 status
string message
```

SetString¶

```
string value
---
int32 status
string message
```

Trigger¶

```
---
int32 status
string message
```

Niryo_robot_modbus

Niryo_robot_poses_handlers

This package is in charge of dealing with transforms, workspace, grips and trajectories.

Poses handlers node

Description - Poses handlers¶
The ROS Node is made of several services to deal with transforms, workspace, grips and trajectories.

The namespace used is: `/niryo_robot_poses_handlers/`

Workspaces¶
A workspace is defined by 4 markers that form a rectangle. With the help of the robot's calibration tip, the marker positions are learned. The camera returns poses (x, y, yaw) relative to the workspace. We can then infer the absolute object pose in robot coordinates.

Grips¶

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

When we know the object pose in robot coordinates, we can't directly send this pose to the robot because we specify the target pose of the tool_link and not of the actual TCP (tool center point). Therefore we introduce the notion of grip. Each end effector has its own grip that specifies where to place the robot with respect to the object. Currently, the notion of grip is not part of the python/tcp/blockly interface because it would add an extra layer of complexity that is not really necessary for the moment. Therefore we have a default grip for all tools that is selected automatically based on the current tool id. However, everything is ready if you want to define custom grips, e.g. for custom tools or for custom grip positions.

The vision pick loop¶
1. Camera detects object relative to markers and sends x_{rel}, y_{rel}, yaw_{rel}.
2. The object is placed on the workspace, revealing the object pose in robot coordinates x, y, z, roll, pitch, yaw.
3. The grip is applied on the absolute object pose and gives the pose the robot should move to.

Poses & trajectories¶
List of poses

Parameters - Poses handlers¶

Poses Handlers' Parameters

Name	Description
`workspace_dir`	Path to the Workspace storage mother folder
`grip_dir`	Path to the Grip storage mother folder
`poses_dir`	Path to the Poses storage mother folder
`trajectories_dir`	Path to the Trajectory storage mother folder

Services - Poses handlers¶

Poses Handlers' Services

Name	Message Type	Description
`manage_workspace`	ManageWorkspace	Save/Delete a workspace
`get_workspace_ratio`	GetWorkspaceRatio	Get ratio of a workspace
`get_workspace_list`	GetNameDescriptionList	Get list of workspaces name & description
`get_workspace_poses`	GetWorkspaceRobotPoses	Get workspace's robot poses
`get_target_pose`	GetTargetPose	Get saved programs name
`manage_pose`	ManagePose	Save/Delete a Pose
`get_pose`	GetPose	Get Pose
`get_pose_list`	GetNameDescriptionList	Get list of poses name & description
`manage_trajectory`	ManageTrajectory	Save/Delete a Trajectory
`get_trajectory`	GetTrajectory	Get Trajectory
`get_trajectory_list`	GetNameDescriptionList	Get list of trajectories name & description

All these services are available as soon as the node is started.

Dependencies - Poses handlers¶
- geometry_msgs
- moveit_msgs
- Niryo_robot_msgs
- tf

Services & messages files - Poses handlers

GetPose (Service)¶

```
string name
---
int32 status
string message
niryo_robot_poses_handlers/NiryoPose pose
```

GetTargetPose (Service)¶

```
string workspace
float32 height_offset
float32 x_rel
float32 y_rel
float32 yaw_rel
---
int32 status
string message
niryo_robot_msgs/RobotState target_pose
```

GetTrajectory (Service)¶

```
string name
---
int32 status
string message
geometry_msgs/Pose[] list_poses
```

GetWorkspaceRatio (Service)¶

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
string workspace
---
int32 status
string message
float32 ratio  # width/height
```

GetWorkspaceRobotPoses (Service)¶

```
string name
---
int32 status
string message
niryo_robot_msgs/RobotState[] poses
```

ManagePose (Service)¶

```
int32 cmd
int32 SAVE = 1
int32 DELETE = -1

niryo_robot_poses_handlers/NiryoPose pose
---
int32 status
string message
```

ManageTrajectory (Service)¶

```
int32 cmd
int32 SAVE = 1
int32 DELETE = -1

string name
string description

geometry_msgs/Pose[] poses
---
int32 status
string message
```

ManageWorkspace (Service)¶

```
int32 SAVE = 1
int32 SAVE_WITH_POINTS = 2
int32 DELETE = -1

int32 cmd

niryo_robot_poses_handlers/Workspace workspace

---
int32 status
string message
```

NiryoPose (Message)¶

```
string name
string description

float64[] joints
geometry_msgs/Point position
niryo_robot_msgs/RPY rpy
geometry_msgs/Quaternion orientation
```

Trajectory (Message)¶

```
string name
geometry_msgs/Pose[] poses
```

Workspace (Message)¶

```
string name # maximum lenght of workspace's name is 30 characters
string description

geometry_msgs/Point[] points
niryo_robot_msgs/RobotState[] poses
```

Niryo_robot_programs_manager

This package is in charge of interpreting/running/saving programs. It is used by Niryo Studio.

Programs manager node

The ROS Node is made of several services to deal with the storage and running of programs.

Calls are not available from the Python ROS Wrapper, as it made to run its programs with the Python ROS Wrapper.

The namespace used is: `/niryo_robot_programs_manager/`

Parameters - Programs manager¶

Programs Manager's Parameters

Name	Description
`autorun_file_name`	Name of the file containing auto run infos
`programs_dir`	Path to the Program storage mother folder

Services - Programs manager¶

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Programs manager Services

Name	Message Type	Description
`execute_program`	ExecuteProgram	Executes a program
`execute_program_autorun`	Trigger	Executes autorun program
`get_program`	GetProgram	Retrieves saved program
`get_program_autorun_infos`	GetProgramAutorunInfos	Gets autorun settings
`get_program_list`	GetProgramList	Gets saved programs' name
`manage_program`	ManageProgram	Saves and Deletes programs
`set_program_autorun`	SetProgramAutorun	Sets autorun settings
`stop_program`	Trigger	Stops the current running program

All these services are available as soon as the node is started whereas on standalone mode or not.

Dependencies - Programs manager¶

- Niryo_robot_msgs
- python-yaml
- std_msgs

Services & messages files - Programs manager

ExecuteProgram (Service)¶

```
bool execute_from_string

string name
string code_string

niryo_robot_programs_manager/ProgramLanguage language
---
int16 status
string message
```

GetProgram (Service)¶

```
string name

niryo_robot_programs_manager/ProgramLanguage language
---
int32 status
string message

string code
string description
```

GetProgramAutorunInfos (Service)¶

```
---
int32 status
string message

niryo_robot_programs_manager/ProgramLanguage language
string name

# Mode
int8 ONE_SHOT = 1
int8 LOOP = 2

int8 mode
```

GetProgramList (Service)¶

```
niryo_robot_programs_manager/ProgramLanguage language
---
string[] programs_names
niryo_robot_programs_manager/ProgramLanguageList[] list_of_language_list
string[] programs_description
```

ManageProgram (Service)¶

```
# Command
int32 SAVE = 1
int32 DELETE = -1
int8 cmd

# Program Name
string name

# - Creation
niryo_robot_programs_manager/ProgramLanguage language

string code
string description

bool allow_overwrite
---
int16 status
string message
```

SetProgramAutorun (Service)¶

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
# Program language
niryo_robot_programs_manager/ProgramLanguage language

# Program Name
string name

# Mode
int8 DISABLE = 0
int8 ONE_SHOT = 1
int8 LOOP = 2

int8 mode

---
int16 status
string message
```

ProgramLanguage (Message)¶

```
int8 NONE = -1

int8 ALL = 0

# Runnable
int8 PYTHON2 = 1
int8 PYTHON3 = 2

# Not Runnable
int8 BLOCKLY = 66

int8 used
```

ProgramLanguageList (Message)¶

```
niryo_robot_programs_manager/ProgramLanguage[] language_list
```

ProgramList (Message)¶

```
string[] programs_names
niryo_robot_programs_manager/ProgramLanguageList[] list_of_language_list
string[] programs_description
```

Niryo_robot_rpi

This package deals with Raspberry Pi related stuff (Button, fans, I/O, leds, ...).

Raspberry Pi Node

The ROS Node manages the following components:

- Physical top button: executes actions when the button is pressed.
- Digital I/O panel: gets commands and sends the current state of digital I/Os. Also controls tools like the Electromagnet.
- Robot fans.
- Led: sets the LED color.
- ROS log: can remove all previous logs on startup to prevent a lack of disk space in the long run (SD cards do not have infinite storage).

The namespace used is: `/niryo_robot_rpi/`

Note that this package should not be used when you are using Ned ROS stack on your computer in simulation mode. Executes actions when the button is pressed.

Publisher - Raspberry Pi¶

RPI Package's Publishers

Name	Message Type	Description
`pause_state`	PausePlanExecution	Publish the current execution state launched when button is pressed
`/niryo_robot/blockly/save_current_point`	std_msgs/Int32	Publish current point when user is in Blockly page to save block by pressing button
`/niryo_robot/rpi/is_button_pressed`	std_msgs/Bool	Publish the button state (true if pressed)
`digital_io_state`	DigitalIOState	Publish the I/Os state by giving for each it's pin / name / mode / state
`/niryo_robot/rpi/led_state`	std_msgs/Int8	Publish the current led color
`ros_log_status`	LogStatus	Publish the current log status (log size / available disk / boolean if should delete ros log on startup)

Services - Raspberry Pi¶

RPI Services

Name	Message Type	Description
`shutdown_rpi`	SetInt	Shutdown the Raspberry Pi
`/niryo_robot/rpi/change_button_mode`	SetInt	Change top button mode (autorun program, blockly, nothing, ...)
`get_digital_io`	GetDigitalIO	Get digital IO state list
`set_digital_io_mode`	SetDigitalIO	Set a digital IO to the mode given
`set_digital_io_state`	SetDigitalIO	Set a digital IO to the state given
`set_led_state`	std_msgs/SetInt	Set led state
`set_led_custom_blinker`	LedBlinker	Set the led in blink mode with the color given
`purge_ros_logs`	SetInt	Purge ROS log
`set_purge_ros_log_on_startup`	SetInt	Modify the permanent settings that tells if robot should purge it's ROS log at each boot

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Dependencies - Raspberry Pi¶

- std_msgs
- actionlib_msgs
- sensor_msgs
- Niryo_robot_msgs
- Niryo_robot_arm_commander

Services & Messages files - Raspberry Pi

ChangeMotorConfig (Service)¶

```
int32[] can_required_motor_id_list
int32[] dxl_required_motor_id_list
---
int32 status
string message
```

GetDigitalIO (Service)¶

```
int32 pin
---
int32 status
string message

int32 pin
string name
int32 mode
int32 state
```

LedBlinker (Service)¶

```
uint8 LED_OFF = 0
uint8 LED_BLUE = 1
uint8 LED_GREEN = 2
uint8 LED_BLUE_GREEN = 3
uint8 LED_RED = 4
uint8 LED_PURPLE = 5
uint8 LED_RED_GREEN = 6
uint8 LED_WHITE = 7

bool activate
uint8 frequency # between 1hz and 100Hz
uint8 color
float32 blinker_duration # 0 for infinite
---
int32 status
string message
```

SetDigitalIO (Service)¶

```
uint8 pin
uint8 value
---
int32 status
string message
```

DigitalIOState (Service)¶

```
# GPIO pin
int32[] pins
# PIN names seen by user to make it simpler
string[] names
# IN/OUT
int32[] modes
# HIGH/LOW
int32[] states
```

LogStatus (Service)¶

```
std_msgs/Header header

# in MB
int32 log_size
int32 available_disk_size
bool purge_log_on_startup
```

Niryo robot tools commander package

Provides functionalities to control end-effectors and accessories for Ned.

This package allows to manage the TCP (Tool Center Point) of the robot. If the functionality is activated, all the movements (in Cartesian coordinates [x, y, z, roll, pitch, yaw]) of the robot will be performed according to this TCP. The same program can then work with several tools by adapting the TCP transformation to them. By default this feature is disabled, but can be enabled through the robot services.

Tools Commander node

The ROS Node is made of services to equip tool, an action server for tool command and topics for the current tool or the tool state.

The namespace used is: `/niryo_robot_tools_commander/`

Action server - tools¶

Tools Package Action Server

Name	Message Type	Description
`action_server`	ToolAction	Command the tool through an action server

Publisher - tools¶

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Tools Package Publishers

Name	Message Type	Description
current_id	std_msgs/Int32	Publish the current tool ID
tcp	TCP	Publish if the TCP (Tool Center Point) is enabled and transformation between the tool_link and the TCP

Services - tools¶

Tools Package Services

Name	Message Type	Description
update_tool	std_srvs/Trigger	Ping/scan for a dxl motor flashed with an ID corresponding to a tool and equip it (if found)
equip_electromagnet	SetInt	Equip the electromagnet with the motor ID given as parameter
enable_tcp	SetBool	Enable or disable the TCP (Tool Center Point) functionality. When we activate it, the transformation will be the last one saved since the robot started. By default it will be the one of the equipped tool.
set_tcp	SetTCP	Activate the TCP (Tool Center Point) functionality and defines a new TCP transformation.
reset_tcp	std_srvs/Trigger	Reset the TCP transformation. By default it will be the one of the equipped tool.

Dependencies - tools¶

- Niryo_robot_msgs
- std_msgs
- geometry_msgs

Action files - tools

ToolAction (Action)¶

```
# goal
niryo_robot_tools_commander/ToolCommand cmd
---
# result
int32 status
string message
---
# feedback
int32 progression
```

Messages files - tools

ToolCommand (Message)¶

```
# Gripper
uint8 OPEN_GRIPPER = 1
uint8 CLOSE_GRIPPER = 2

# Vacuump pump
uint8 PULL_AIR_VACUUM_PUMP = 10
uint8 PUSH_AIR_VACUUM_PUMP = 11

# Tools controlled by digital I/Os
uint8 SETUP_DIGITAL_IO = 20
uint8 ACTIVATE_DIGITAL_IO = 21
uint8 DEACTIVATE_DIGITAL_IO = 22

uint8 cmd_type

# Gripper1= 11, Gripper2=12, Gripper3=13, VaccuumPump=31, Electromagnet=30
uint8 tool_id

# if gripper close
uint16 gripper_close_speed

# if gripper open
uint16 gripper_open_speed

# if vacuum pump or electromagnet grove
bool activate

# if tool is set by digital outputs (electromagnet)
# if gpio<0 get value in memory
int8 gpio
```

TCP (Message)¶

```
bool enabled

geometry_msgs/Point position
niryo_robot_msgs/RPY rpy
geometry_msgs/Quaternion orientation
```

Services files - tools

SetTCP (Service)¶

```
geometry_msgs/Point position

#Only one of the two is required.
#If both are filled, the quaternion will be chosen by default
niryo_robot_msgs/RPY rpy
geometry_msgs/Quaternion orientation
---
int32 status
string message
```

Niryo_robot_unit_tests

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

This package provides config and launches files to start Ned unit tests.

Currently, tested packages are:

- Niryo_robot_poses_handlers
- Niryo_robot_programs_manager
- Python ROS Wrapper

Run tests with launch file

Using a launch file make the task easy as you do not have to bother about your current directory. Nevertheless, it starts a roscore, and so, the process won't end as "/rosout" will be still alive. To overcome this issue, the test node is set as **required**, so do not worry if you see some red lines at the execution's end.

Launch file with Rviz¶
It will start the ROS Stack and Rviz, then run the tests:

```
roslaunch niryo_robot_unit_tests simulation_rviz_tests.launch
```

Launch file with Gazebo¶
It will start the ROS Stack and Gazebo, then run the tests. Gazebo add Physics, Gripper and Camera to the tests:

```
roslaunch niryo_robot_unit_tests simulation_gazebo_tests.launch
```

Launch file Headless¶
It will start the ROS Stack then run the tests in a headless version (no display). It is very useful for CI purposes:

```
roslaunch niryo_robot_unit_tests simulation_headless_tests.launch
```

Run tests With Python file

Using a python file implies to use the correct relative directory, but it also allows to get status code at the end of the script.

Python file with Rviz¶

```
python niryo_robot_unit_tests/scripts/script_test_rviz.py
```

Python file with Gazebo¶

```
python niryo_robot_unit_tests/scripts/script_test_gazebo.py
```

Python file Headless¶

```
python niryo_robot_unit_tests/scripts/script_test_headless.py
```

Niryo_robot_user_interface

This packages handle high-level user interface commands coming TCP requests and also system-related features like I/Os, LED and fans.

You can find their documentations here:

- TCP Server

Use Ned's TCP server

Ned is permanently running a TCP Server to acquire requests. This server is built on top of the Ned Python ROS Wrapper (index.html#document-source/ros_wrapper).

It offers a simple way for developers to create programs for robot to control them via remote communication on a computer, on a mobile or any device with network facilities.

Programs can communicate through network TCP with the robots in any language available.

Connection¶
To access the server, you will have to use to robot's IP adress and communicate via the **port 40001**.

Communication¶
Only one client can communicate with the server (reconnection effective but no multi clients).

The server answers only after the command is done, so it can't deal with multiple commands at the same time.

Packet convention¶
General format¶
For easier usage and easier debugging, the communication is based on JSON format.

Every package have this following shape: `<json_packet_size>{<json_content>}<payload>` .

The JSON packet size is an unsigned short coded on 2 bytes.

The JSON contains command's name & params.

Payload contains *heavy* data like an image.

Request¶
Format - Request¶

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

As no function requests a payload in input, requests have the following.

Format: `<json_packet_size>{'param_list': [<param_1>, <param_2>, ....], 'command': <command_str>}`

Examples - Request¶
Calibrate auto: `{'param_list': ['AUTO'], 'command': 'CALIBRATE'}`

Move joints: `{'param_list': [0.0, 0.0, 0.0, 0.0, 0.0, 0.0], 'command': 'MOVE_JOINTS'}`

Answer¶
Format - Answer¶
Firstly, answers indicate to the user if its command has been well executed. This is indicated in the JSON by the parameter “status”.

A successful answer will have the format:

`{'status': 'OK', 'list_ret_param': [<param_1>, <param_2>, ....], 'payload_size': <payload_size_int>, 'command': <command_str>}<payload_str>`

An unsuccessful answer will have the format: `{'status': 'KO', 'message': <message_str>}`

Examples - Answer¶
Calibrate Auto: `{'status': 'OK', 'list_ret_param': [], 'payload_size': 0, 'command': 'CALIBRATE'}`

Get Pose: `{'status': 'OK', 'list_ret_param': [0.2, 0.15, 0.35, 0.5, -0.6, 0.1], 'payload_size': 0, 'command': 'GET_POSE'}`

Commands enum for TCP server¶

class **CommandEnum**

Enumeration of all commands used

CALIBRATE= *0*

SET_LEARNING_MODE= *1*

GET_LEARNING_MODE= *2*

SET_ARM_MAX_VELOCITY= *3*

SET_JOG_CONTROL= *4*

GET_JOINTS= *10*

GET_POSE= *11*

GET_POSE_QUAT= *12*

MOVE_JOINTS= *20*

MOVE_POSE= *21*

SHIFT_POSE= *22*

MOVE_LINEAR_POSE= *23*

SHIFT_LINEAR_POSE= *24*

JOG_JOINTS= *25*

JOG_POSE= *26*

FORWARD_KINEMATICS= *27*

INVERSE_KINEMATICS= *28*

GET_POSE_SAVED= *50*

SAVE_POSE= *51*

DELETE_POSE= *52*

GET_SAVED_POSE_LIST= *53*

PICK_FROM_POSE= *60*

PLACE_FROM_POSE= *61*

PICK_AND_PLACE= *62*

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

GET_TRAJECTORY_SAVED= *80*

EXECUTE_TRAJECTORY_FROM_POSES= *81*

EXECUTE_TRAJECTORY_SAVED= *82*

SAVE_TRAJECTORY= *83*

DELETE_TRAJECTORY= *84*

GET_SAVED_TRAJECTORY_LIST= *85*

EXECUTE_TRAJECTORY_FROM_POSES_AND_JOINTS= *86*

UPDATE_TOOL= *120*

OPEN_GRIPPER= *121*

CLOSE_GRIPPER= *122*

PULL_AIR_VACUUM_PUMP= *123*

PUSH_AIR_VACUUM_PUMP= *124*

SETUP_ELECTROMAGNET= *125*

ACTIVATE_ELECTROMAGNET= *126*

DEACTIVATE_ELECTROMAGNET= *127*

GET_CURRENT_TOOL_ID= *128*

GRASP_WITH_TOOL= *129*

RELEASE_WITH_TOOL= *130*

ENABLE_TCP= *140*

SET_TCP= *141*

RESET_TCP= *142*

TOOL_REBOOT= *145*

SET_PIN_MODE= *150*

DIGITAL_WRITE= *151*

DIGITAL_READ= *152*

GET_DIGITAL_IO_STATE= *153*

GET_HARDWARE_STATUS= *154*

SET_CONVEYOR= *180*

UNSET_CONVEYOR= *181*

CONTROL_CONVEYOR= *182*

GET_CONNECTED_CONVEYORS_ID= *183*

GET_IMAGE_COMPRESSED= *200*

GET_TARGET_POSE_FROM_REL= *201*

GET_TARGET_POSE_FROM_CAM= *202*

VISION_PICK= *203*

MOVE_TO_OBJECT= *205*

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

> **DETECT_OBJECT**= *204*
>
> **GET_CAMERA_INTRINSICS**= *210*
>
> **SAVE_WORKSPACE_FROM_POSES**= *220*
>
> **SAVE_WORKSPACE_FROM_POINTS**= *221*
>
> **DELETE_WORKSPACE**= *222*
>
> **GET_WORKSPACE_RATIO**= *223*
>
> **GET_WORKSPACE_LIST**= *224*
>
> **SET_IMAGE_BRIGHTNESS**= *230*
>
> **SET_IMAGE_CONTRAST**= *231*
>
> **SET_IMAGE_SATURATION**= *232*
>
> **GET_IMAGE_PARAMETERS**= *235*

Niryo_robot_vision

This package is the one dealing with all vision related stuff.

Vision Node

The ROS Node is made of several services to deal with video streaming, object detection... The node is working exactly the same way if you chose to use it on simulation or reality.

This node can be launched locally in a standalone mode via the command:

```
roslaunch niryo_robot_vision vision_node_local.launch
```

Configuration (Frame Per Second, Camera Port, Video Resolution) can be edited in the config file:

- For "standard" Node: *niryo_robot_vision/config/video_server_setup.yaml*
- For local Node: *niryo_robot_vision/config/video_server_setup_local.yaml*

The namespace used is: `/niryo_robot_vision/`

Parameters - Vision¶

Vision Package's Parameters

Name	Description
`frame_rate`	Streams frame rate
`simulation_mode`	Sets to true if you are using the gazebo simulation. It will adapt how the node get its video stream
`debug_compression_quality`	Debugs Stream compression quality
`stream_compression_quality`	Streams compression quality
`subsampling`	Streams subsampling factor

Publisher - Vision¶

Vision Package's Publishers

Name	Message Type	Description
`compressed_video_stream`	sensor_msgs/CompressedImage	Publishes the last image read as a compressed image
`video_stream_parameters`	ImageParameters	Publishes the brightness, contrast and saturation settings of the video stream

Services - Vision¶

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Programs manager Services

Name	Message Type	Description
`debug_colors`	DebugColorDetection	Returns an annotated image to emphasize what happened with color detection
`debug_markers`	DebugMarkers	Returns an annotated image to emphasize what happened with markers detection
`obj_detection_rel`	ObjDetection	Object detection service
`start_stop_video_streaming`	SetBool	Starts or stops video streaming
`take_picture`	TakePicture	Saves a picture in the specified folder
`set_brightness`	SetImageParameter	Sets the brightness of the video stream
`set_contrast`	SetImageParameter	Sets the contrast of the video stream
`set_saturation`	SetImageParameter	Sets the saturation of the video stream

All these services are available as soon as the node is started.

Dependencies - Vision¶
- Niryo_robot_msgs
- sensor_msgs

Topics files - Vision

ImageParameters (Topic)¶

```
float64 brightness_factor
float64 contrast_factor
float64 saturation_factor
```

Services files - Vision

DebugColorDetection (Service)¶

```
string color
---
sensor_msgs/CompressedImage img
```

DebugMarkers (Service)¶

```
---
bool markers_detected
sensor_msgs/CompressedImage img
```

ObjDetection (Service)¶

```
string obj_type
string obj_color
float32 workspace_ratio
bool ret_image
---
int32 status

niryo_robot_msgs/ObjectPose obj_pose

string obj_type
string obj_color

sensor_msgs/CompressedImage img
```

TakePicture (Service)¶

```
string path
---
bool success
```

SetImageParameter (Service)¶

```
float64 factor
---
int32 status
string message
```

Hardware ROS Stack documentation

ROS Logo

ROS (Robot Operating System) is an Open-Source Robotic Framework which allows to ease robot software development. The framework is used in almost each part of Ned's software.

The high-level packages (motion planner, vision, ...) are coded in Python to give good readability whereas communication with Hardware is developed in C++ to ensure speed.

Note

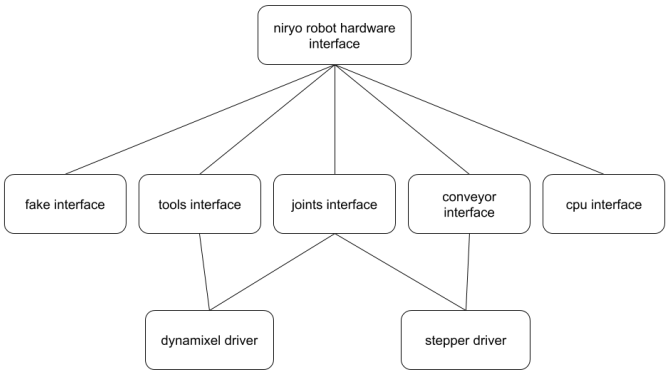
To learn more about ROS, go on Official ROS Wiki (http://wiki.ros.org/).

In this section, you will have access to all information about each Niryo robot's ROS hardware stack packages.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Niryo_robot_hardware_interface

This package handles packages related to the robot's hardware.
It launches hardware interface nodes, motors communication and driver.



Global overview of hardware stack packages organization.

Hardware interface node

The ROS Node is made to launch hardware interface and communication:

- Conveyor Belt interface
- Joints interface
- Tools interface
- Fake interface
- Dynamixel driver
- Stepper driver

The namespace used is: `/niryo_robot_hardware_interface/` .

Parameters - hardware interface¶

Hardware Interface's Parameters

Name	Description
`publish_hw_status_frequency`	Publishes rate for hardware status. Default : '2.0'
`publish_software_version_frequency`	Publishes rate for software status. Default : '2.0'

Published topics - hardware interface¶

Hardware Interface's Published Topics

Name	Message Type	Description
`hardware_status`	niryo_robot_msgs/HardwareStatus	Motors, bus, joints and CPU status
`software_version`	niryo_robot_msgs/SoftwareVersion	Raspberry and stepper software version

Services - hardware interface¶

Hardware Interface Package Services

Name	Message Type	Description
`launch_motors_report`	Trigger	Starts motors report
`reboot_motors`	Trigger	Reboots motors
`stop_motors_report`	Trigger	Stops motors report

Dependencies - hardware interface¶

- Tools_interface
- Joints_interface
- Conveyor_interface
- CPU_interface
- Fake_interface
- Niryo_robot_msgs

Joints_interface

This package handles packages related to the robot's joints controller.
It provides an interface to ros_control.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Joints interface node

The ROS Node is made to:

- Interface robot's motors to joint trajectory controller, from ros_control package.
- Create a controller manager, from controller_manager package, provides the infrastructure to load, unload, start and stop controllers.
- Interface with motors calibration.
- Initialize motors parameters.

Parameters - joints interface¶

Joint Interface's Parameters

Name	Description
`ros_control_loop_frequency`	Controls loop frequency. Default: '100.0'
`publish_learning_mode_frequency`	Publishes rate for learning mode state. Default: '2.0'
`calibration_timeout`	Waiting time between 2 commands during the calibration process. Default: '30'
`calibration_file`	File directory where is saved motors calibration value. Default: '/home/niryo/niryo_robot_saved_files/stepper_motor_calibration_offsets.txt'

Published topics - joints interface¶

Joint Interface's Published Topics

Name	Message Type	Description
`/niryo_robot/learning_mode/state`	std_msgs/Bool	Learning mode state

Services - joints interface¶

Joint Interface Package Services

Name	Message Type	Description
`/niryo_robot/joints_interface/calibrate_motors`	SetInt	Starst motors calibration - value can be 1 for auto calibration, 2 for manual
`/niryo_robot/joints_interface/request_new_calibration`	Trigger	Unsets motors calibration
`niryo_robot/learning_mode/activate`	Trigger	Either activates or deactivates learning mode

Dependencies - joints interface¶

- hardware_interface
- controller_manager
- Dynamixel_driver
- Stepper_driver
- Niryo_robot_msgs

Conveyor_interface

This package handles Niryo's Conveyor Belt.
You can control two Conveyors Belt.

Conveyor Belt interface node

The ROS Node is made to:

- Initialize Conveyor Belt motor parameters.
- Set and control Conveyors Belt.
- Publish Conveyors Belt state.

The namespace used is: `/niryo_robot/conveyor/`

Parameters - Conveyor Belt interface¶

Conveyor Belt Interface's Parameters

Name	Description
`publish_frequency`	Publishes rate for conveyors state. Default: '2.0'

Published topics - Conveyor Belt interface¶

Conveyor Belt Interface's Published Topics

Name	Message Type	Description
`feedback`	ConveyorFeedbackArray	Conveyors Belt states

Services - Conveyor Belt interface¶

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Conveyor Belt Interface Package Services

Name	Message Type	Description
control_conveyor	ControlConveyor	Sends a command to the desired Conveyor Belt
ping_and_set_conveyor	SetConveyor	Scans and sets a new Conveyor Belt

Dependencies - Conveyor Belt interface¶

- std_msgs
- stepper_driver

Services & messages files - Conveyor Belt interface

ControlConveyor (Service)¶

```
uint8 id

bool control_on
int16 speed
int8 direction
---
int16 status
string message
```

SetConveyor (Service)¶

```
uint8 cmd
uint8 id

uint8 ADD = 1
uint8 REMOVE = 2

---
int16 id
int16 status
string message
```

ConveyorFeedbackArray (Message)¶

```
conveyor_interface/ConveyorFeedback[] conveyors
```

ConveyorFeedback (Message)¶

```
#Conveyor id ( either 12 or 18)
uint8 conveyor_id
#Conveyor Connection state ( if it is enabled)
bool connection_state
# Conveyor Controls state : ON or OFF
bool running
# Conveyor Speed ( 1-> 100 %)
int16 speed
# Conveyor direction ( backward or forward)
int8 direction
```

Tools_interface

This package handles Niryo's tools.

Tools interface node

The ROS Node is made to:

- Set and control tools.
- Publish tool connection state.

Parameters - tools interface¶

Tools Interface's Parameters

Name	Description
check_tool_connection_frequency	Publishes and controls rate for tools connection state. Default: '2.0'
id_list	List of tools id Default: '[11,12,13,14,31]'
motor_type_list	List of motor tools type Default: '["xl320","xl320","xl320","xl320","xl320"]'

Published topics - tools interface¶

Tools Interface's Published Topics

Name	Message Type	Description
/niryo_robot_hardware/tools/current_id	std_msgs/Int32	Current tool ID

Services - tools interface¶

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Tool Interface Package Services

Name	Message Type	Description
niryo_robot/tools/ping_and_set_dxl_tool	tools_interface/PingDxlTool	Scans and sets for a tool plugged
niryo_robot/tools/open_gripper	tools_interface/OpenGripper	Opens a gripper tool
niryo_robot/tools/close_gripper	tools_interface/CloseGripper	Closes a gripper tool
niryo_robot/tools/pull_air_vacuum_pump	tools_interface/PullAirVacuumPump	Pulls vacuum pump tool
niryo_robot/tools/push_air_vacuum_pump	tools_interface/PushAirVacuumPump	Pushes vacuum pump tool
niryo_robot/tools/reboot	std_srvs/Trigger	Reboots the motor of the equipped tool

Dependencies - tools interface¶

- std_msgs
- std_srvs
- Dynamixel_driver

Services & messages files - tools interface

PingDxlTool (Service)¶

```
---
int32 state
uint8 id
```

OpenGripper (Service)¶

```
uint8 id

int16 open_position
int16 open_speed
int16 open_hold_torque
---
uint8 state
```

CloseGripper (Service)¶

```
uint8 id

int16 close_position
int16 close_speed
int16 close_hold_torque
int16 close_max_torque
---
uint8 state
```

PullAirVacuumPump (Service)¶

```
uint8 id

int16 pull_air_position
int16 pull_air_hold_torque
---
uint8 state
```

PushAirVacuumPump (Service)¶

```
uint8 id

int16 push_air_position
---
uint8 state
```

CPU_interface

This package handles CPU states.

CPU interface node

The ROS Node is made to:

- monitor CPU temperature.

Parameters - CPU interface¶

CPU Interface's Parameters

Name	Description
read_rpi_diagnostics_frequency	Publishes rate for CPU temperature Default: '0.25'
temperature_warn_threshold	CPU temperature [celsius] threshold before a warn message Default: '75'
temperature_shutdown_threshold	CPU temperature [celsius] threshold before shutdown the robot Default: '85'

Fake_interface

This package provides fakes hardware interfaces when the robot is used in simulation.

Fake interface node

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

The ROS Node is made to simulate:

- tools interface
- Conveyor Belt interface
- joints interface

Published topics - fake interface¶

Fake Interface's Published Topics

Name	Message Type	Description
`/niryo_robot/learning_mode/state`	std_msgs/Bool	Learning mode state
`/niryo_robot_hardware/tools/current_id`	std_msgs/Int32	Current tool ID

Services - fake interface¶

Fake Interface Package Services

Name	Message Type	Description
`/niryo_robot/joints_interface/calibrate_motors`	SetInt	Starts motors calibration - value can be 1 for auto calibration, 2 for ma
`/niryo_robot/joints_interface/request_new_calibration`	Trigger	Unsets motors calibration
`niryo_robot/learning_mode/activate`	Trigger	Either activates or deactivates learning mode
`niryo_robot/tools/ping_and_set_dxl_tool`	tools_interface/PingDxlTool	Scans and sets for a tool plugged
`niryo_robot/tools/open_gripper`	tools_interface/OpenGripper	Opens a Gripper tool
`niryo_robot/tools/close_gripper`	tools_interface/CloseGripper	Closes a Gripper tool
`niryo_robot/tools/pull_air_vacuum_pump`	tools_interface/PullAirVacuumPump	Pulls Vacuum Pump tool
`niryo_robot/tools/push_air_vacuum_pump`	tools_interface/PushAirVacuumPump	Pushes Vacuum Pump tool
`/niryo_robot/conveyor/control_conveyor`	ControlConveyor	Sends a command to the desired Conveyor Belt
`/niryo_robot/conveyor/ping_and_set_conveyor`	SetConveyor	Scans and sets a new Conveyor Belt

Dependencies - fake interface¶

- std_msgs
- hardware_interface
- controller_manager
- Niryo_robot_msgs
- Tools_interface
- Joints_interface
- Conveyor_interface

Dynamixel_driver

This package handles dynamixel motors communication through dynamixel sdk.
It provides an interface to dynamixel_sdk.

Dynamixel Driver Node

The ROS Node is made to:

- Send commands to dynamixel motors
- Receive dynamixel motors data

Parameters - Dynamixel Driver¶

Dynamixel Driver's Parameters

Name	Description
`dxl_hardware_control_loop_frequency`	Controls loop frequency. Default: '100.0'
`dxl_hardware_write_frequency`	Writes frequency. Default: '50.0'
`dxl_hardware_read_data_frequency`	Reads frequency. Default: '15.0'
`dxl_hardware_read_status_frequency`	Reads dynamixels status frequency. Default: '0.5'
`dxl_hardware_check_connection_frequency`	Checks dynamixels connection frequency. Default: '2.0'
`dxl_motor_id_list`	List of dynamixels ID Default: '[2,3,6]'
`dxl_motor_type_list`	List of dynamixels type Default: '["xl430","xl430","xl320"]'

Services - Dynamixel Driver¶

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Dynamixel Driver Package Services

Name	Message Type	Description
niryo_robot/dynamixel_driver/set_dxl_leds	SetInt	Controls dynamixel LED
niryo_robot/dynamixel_driver/send_custom_dxl_value	dynamixel_driver/SendCustomDxlValue	Sends a custom dynamixel command

Dependencies - Dynamixel Driver¶

- dynamixel_sdk
- Niryo_robot_msgs

Services & Messages files - Dynamixel Driver

SendCustomDxlValue (Service)¶

```
# Check XL-320 and XL-430 reference doc for
# the complete register table

int8 motor_type # 3 (XL-320) or 2 (XL-430)
uint8 id
int32 value
int32 reg_address
int32 byte_number
---
int32 status
string message
```

DxlMotorHardwareStatus (Message)¶

```
niryo_robot_msgs/MotorHeader motor_identity

uint32 temperature
float64 voltage
uint32 error
string error_msg
```

DxlMotorCommand (Message)¶

```
uint8 cmd_type
uint8 CMD_TYPE_POSITION=1
uint8 CMD_TYPE_VELOCITY=2
uint8 CMD_TYPE_EFFORT=3
uint8 CMD_TYPE_TORQUE=4

uint8[] motors_id
uint32[] params
```

DxlArrayMotorHardwareStatus (Message)¶

```
std_msgs/Header header
dynamixel_driver/DxlMotorHardwareStatus[] motors_hw_status
```

Stepper_driver

This package handles stepper motors communication.

Stepper driver node

The ROS Node is made to:

- Send commands to stepper motors.
- Receive stepper motors data.

Parameters - stepper driver¶

Stepper Driver's Parameters

Name	Description
can_hardware_control_loop_frequency	Controls loop frequency. Default: '1500.0'
can_hw_write_frequency	Writes frequency. Default: '50.0'
can_hw_check_connection_frequency	Checks steppers connection frequency. Default: '2.0'
stepper_motor_id_list	List of steppers ID Default: '[1,2,3]'

Dependencies - stepper driver¶

- mcp_can_rpi
- Niryo_robot_msgs

Services & messages files - stepper driver

StepperMotorHardwareStatus (Message)¶

```
niryo_robot_msgs/MotorHeader motor_identity

string firmware_version
int32 temperature
int32 voltage
int32 error
```

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

StepperMotorCommand (Message)¶

```
uint8 cmd_type
uint8 CMD_TYPE_POSITION=1
uint8 CMD_TYPE_VELOCITY=2
uint8 CMD_TYPE_EFFORT=3
uint8 CMD_TYPE_TORQUE=4

uint8[] motors_id
int32[] params
```

StepperArrayMotorHardwareStatus (Message)¶

```
std_msgs/Header header
stepper_driver/StepperMotorHardwareStatus[] motors_hw_status
```

Dxl_debug_tools

This package offers scripts to change ping/scan DXL motors and changes register values of these motors.

Niryo robot - Send DXL custom value

This script can be launched via:

```
rosrun niryo_robot_debug send_custom_dxl_value.py
```

Parameters - Send DXL custom value¶

- **–type [Number]:** Motor type (3 for XL-320, 2 for XL-430)
- **–id [Number]:** Motor ID
- **–address [Number]:** Register address to modify
- **–value [Number]:** Value to store at the register address given
- **–size [Number]:** Size in bytes of the value given

Niryo robot - Dxl debug tools

This script can be launched via:

```
rosrun niryo_robot_debug dxl_debug_tools
```

Parameters - Dxl debug tools¶

- **–help / -h:** Print help message
- **–baudrate / -b [Number]:** Baudrate (1000000 by default)
- **–port / -p [Number]:** Set port
- **–id / -i [Number]:** Dxl motor ID (0 by default)
- **–scan:** Scan all Dxl motors on the bus
- **–ping:** Ping specific ID
- **–set-register [Number] [Number] [Number]:** Set a value to a register, parameters are in the order: register address / value / size (in bytes) of the value

Control with Python ROS Wrapper

Python Logo

In order to control Ned more easily than calling each topics & services one by one, a Python ROS Wrapper has been built on top of ROS.

For instance, a script realizing a moveJ via Python ROS Wrapper will look like:

```
niryo_robot = NiryoRosWrapper()
niryo_robot.move_joints(0.1, -0.2, 0.0, 1.1, -0.5, 0.2)
```

What this code is doing in a hidden way:

- It generates a RobotMove Action Goal and set it as a joint command with the corresponding joints value.
- Sends goal to the Commander Action Server.
- Waits for the Commander Action Server to set Action as finished.
- Checks if action finished with a success.

In this section, we will give some examples on how to use the Python ROS Wrapper to control Ned, as well as a complete documentation of the functions available in the Ned Python ROS Wrapper.

Hint

The Python ROS Wrapper forces the user to write his code directly in the robot, or, at least, copy the code on the robot via a terminal command. If you do not want that, and run code directly from your computer you can use the python Package PyNiryo (index.html#pyniryo).

Before running your programs

The variable PYTHONPATH

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

The Python interpreter needs to have all used packages in the environment variable **PYTHONPATH**, to do that, you need to have sourced your ROS environment:

- If you are coding directly on your robot, it is made directly in every terminal.
- If your are using simulation, be sure to have followed the setup from Setup Ned ROS environment.

Required piece of code

To run, your program will need some imports & initialization. We give you below the piece of code you must use to make Python ROS Wrapper work:

```
#!/usr/bin/env python

# Imports
from niryo_robot_python_ros_wrapper import *
import rospy

# Initializing ROS node
rospy.init_node('niryo_ned_example_python_ros_wrapper')

niryo_robot = NiryoRosWrapper()

# -- YOUR CODE HERE -- #
```

You have now everything you need to control the robot through its Python ROS Wrapper. To run a script, simply use the command `python my_script.py` .

Examples: Basics

In this file, two short programs are implemented & commented in order to help you understand the philosophy behind the Python ROS Wrapper.

Danger

If you are using the real robot, make sure the environment around is clear.

Your first move joint

The following example shows a first use case. It's a simple MoveJ.

```
#!/usr/bin/env python

# Imports
from niryo_robot_python_ros_wrapper import *
import rospy

# Initializing ROS node
rospy.init_node('niryo_ned_example_python_ros_wrapper')

# Connecting to the ROS Wrapper & calibrating if needed
niryo_robot = NiryoRosWrapper()
niryo_robot.calibrate_auto()

# Moving joint
niryo_robot.move_joints(0.1, -0.2, 0.0, 1.1, -0.5, 0.2)
```

Code details - First MoveJ¶
First of all, we indicate to the shell that we are running a Python Script:

```
#!/usr/bin/env python
```

Then, we import the API package to be able to access functions:

```
from niryo_robot_python_ros_wrapper import *
```

Then, we install a ROS Node in order to communicate with ROS master:

```
import rospy

# Initializing ROS node
rospy.init_node('niryo_ned_example_python_ros_wrapper')
```

We start a **`NiryoRosWrapper`** (index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper) instance:

```
niryo_robot = NiryoRosWrapper()
```

Once the connection is done, we calibrate the robot using its **`calibrate_auto()`** (index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper.calibrate_auto) function:

```
niryo_robot.calibrate_auto()
```

As the robot is now calibrated, we can do a Move Joints by giving the 6 axis positions in radians! To do so, we use **`move_joints()`** (index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper.move_joints):

```
niryo_robot.move_joints(0.1, -0.2, 0.0, 1.1, -0.5, 0.2)
```

Your first pick and place

For our second example, we are going to develop an algorithm of pick and place:

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
#!/usr/bin/env python

# Imports
from niryo_robot_python_ros_wrapper import *
import rospy

# Initializing ROS node
rospy.init_node('niryo_robot_example_python_ros_wrapper')

# Connecting to the ROS Wrapper & calibrating if needed
niryo_robot = NiryoRosWrapper()
niryo_robot.calibrate_auto()

# Updating tool
niryo_robot.update_tool()

# Opening Gripper/Pushing Air
niryo_robot.release_with_tool()
# Going to pick pose
niryo_robot.move_pose(0.2, 0.1, 0.14, 0.0, 1.57, 0)
# Picking
niryo_robot.grasp_with_tool()
# Moving to place pose
niryo_robot.move_pose(0.2, -0.1, 0.14, 0.0, 1.57, 0)
# Placing !
niryo_robot.release_with_tool()
```

Code details - first pick and place¶
First of all, we do the imports and start a ROS Node:

```
#!/usr/bin/env python

from niryo_robot_python_ros_wrapper import *
import rospy

rospy.init_node('niryo_robot_example_python_ros_wrapper')
```

Then, create a **`NiryoRosWrapper`** (index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper) instance & calibrate the robot:

```
niryo_robot = NiryoRosWrapper()
niryo_robot.calibrate_auto()
```

Then, we equip the tool with **`update_tool()`** (index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper.update_tool)

```
niryo_robot.update_tool()
```

Now that our initialization is done, we can open the Gripper (or push air from the Vacuum pump) with **`release_with_tool()`** (index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper.release_with_tool), go to the picking pose via **`move_pose()`** (index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper.move_pose) & then catch an object with **`grasp_with_tool()`** (index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper.grasp_with_tool)!

```
# Opening Gripper/Pushing Air
niryo_robot.release_with_tool()
# Going to pick pose
niryo_robot.move_pose(0.2, 0.1, 0.14, 0.0, 1.57, 0)
# Picking
niryo_robot.grasp_with_tool()
```

We now get to the place pose, and place the object.

```
# Moving to place pose
niryo_robot.move_pose(0.2, -0.1, 0.14, 0.0, 1.57, 0)
# Placing !
niryo_robot.release_with_tool()
```

Notes - Basics examples

You may not have fully understood how to move the robot and use tools of Ned and that is totally fine because you will find more details on another examples page!
The important thing to remember from this page is how to import the library & connect to the robot.

Examples: Movement

This document shows how to control Ned in order to make Move Joints & Move Pose.

If you want see more, you can look at API - Joints & Pose (index.html#joints-pose).

Danger

If you are using the real robot, make sure the environment around is clear.

Joints

To do a moveJ, you should pass 6 floats: (j1, j2, j3, j4, j5, j6) to the method **`move_joints()`** (index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper.move_joints):

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
#!/usr/bin/env python

# Imports
from niryo_robot_python_ros_wrapper import *
import rospy

# Initializing ROS node
rospy.init_node('niryo_ned_example_python_ros_wrapper')

niryo_robot = NiryoRosWrapper()
niryo_robot.move_joints(0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
```

To get joints, we use **`get_joints()`** (index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper.get_joints):

```
joints = niryo_robot.get_joints()
j1, j2, j3, j4, j5, j6 = joints
```

Pose

To do a moveP, you should pass 6 floats: (x, y, z, roll, pitch, yaw) to the method **`move_pose()`** (index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper.move_pose).

See on this example:

```
#!/usr/bin/env python

# Imports
from niryo_robot_python_ros_wrapper import *
import rospy

# Initializing ROS node
rospy.init_node('niryo_ned_example_python_ros_wrapper')

niryo_robot = NiryoRosWrapper()
niryo_robot.move_pose(0.25, 0.0, 0.25, 0.0, 0.0, 0.0)
```

To get pose, we use **`get_pose()`** (index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper.get_pose):

```
x, y, z, roll, pitch, yaw = niryo_robot.get_pose()
```

Examples: tool action

This page shows how to control Ned's tools via the Python ROS Wrapper.

If you want see more, you can look at API - Tools (index.html#tools).

> **Danger**
>
> If you are using the real robot, make sure the environment around it is clear.

Tool control

Equip tool¶
In order to use a tool, it should be mechanically plugged to the robot but also connected software wise.

To do that, we should use the function **`update_tool()`** (index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper.update_tool) which takes no argument. It will scan motor connections and set the new tool!

The line to equip a new tool is:

```
niryo_robot.update_tool()
```

Grasping¶
To grasp with any tool, you can use the function: **`grasp_with_tool()`** (index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper.grasp_with_tool). This action corresponds to:

- Close gripper for Grippers.
- Pull Air for Vacuum pump.
- Activate for Electromagnet.

The code to grasp is:

```
#!/usr/bin/env python

# Imports
from niryo_robot_python_ros_wrapper import *
import rospy

# Initializing ROS node
rospy.init_node('niryo_ned_example_python_ros_wrapper')

# Connecting to the ROS Wrapper & calibrating if needed
niryo_robot = NiryoRosWrapper()
niryo_robot.calibrate_auto()

# Updating tool
niryo_robot.update_tool()

# Grasping
niryo_robot.grasp_with_tool()
```

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

To grasp by specifying parameters:

```
#!/usr/bin/env python

# Imports
from niryo_robot_python_ros_wrapper import *
import rospy

# Initializing ROS node
rospy.init_node('niryo_ned_example_python_ros_wrapper')

# Connecting to the ROS Wrapper & calibrating if needed
niryo_robot = NiryoRosWrapper()
niryo_robot.calibrate_auto()

# Updating tool
tool_used = ToolID.XXX
niryo_robot.update_tool()

if tool_used in [ToolID.GRIPPER_1, ToolID.GRIPPER_2, ToolID.GRIPPER_3, ToolID.GRIPPER_4]:
    niryo_robot.close_gripper(speed=500)
elif tool_used == ToolID.ELECTROMAGNET_1:
    pin_electromagnet = PinID.XXX
    niryo_robot.setup_electromagnet(pin_electromagnet)
    niryo_robot.activate_electromagnet(pin_electromagnet)
elif tool_used == ToolID.VACUUM_PUMP_1:
    niryo_robot.pull_air_vacuum_pump()
```

Releasing¶
To release with any tool, you can use the function: **`release_with_tool()`** (index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper.release_with_tool).
This action correspond to:

- Open gripper for Grippers.
- Push Air for Vacuum pump.
- Deactivate for Electromagnet.

The line to release is:

```
niryo_robot.release_with_tool()
```

To release by specifying parameters:

```
#!/usr/bin/env python

# Imports
from niryo_robot_python_ros_wrapper import *
import rospy

# Initializing ROS node
rospy.init_node('niryo_ned_example_python_ros_wrapper')

# Connecting to the ROS Wrapper & calibrating if needed
niryo_robot = NiryoRosWrapper()
niryo_robot.calibrate_auto()

# Updating tool
tool_used = ToolID.XXX
niryo_robot.update_tool()

if tool_used in [ToolID.GRIPPER_1, ToolID.GRIPPER_2, ToolID.GRIPPER_3, ToolID.GRIPPER_4]:
    niryo_robot.open_gripper(speed=500)
elif tool_used == ToolID.ELECTROMAGNET_1:
    pin_electromagnet = PinID.XXX
    niryo_robot.setup_electromagnet(pin_electromagnet)
    niryo_robot.deactivate_electromagnet(pin_electromagnet)
elif tool_used == ToolID.VACUUM_PUMP_1:
    niryo_robot.push_air_vacuum_pump(tool_used)
```

Pick & place with tools

There are a plenty of ways to realize a pick and place with the ROS Wrapper. Methods will be presented from the lowest to highest level.

Code used will be:

```
# Imports
from niryo_robot_python_ros_wrapper import *

gripper_used = ToolID.XXX  # Tool used for picking

# The pick pose
pick_pose = (0.25, 0., 0.15, 0., 1.57, 0.0)
# The Place pose
place_pose = (0., -0.25, 0.1, 0., 1.57, -1.57)

def pick_n_place_version_x(niryo_ned):
    # -- SOME CODE -- #

if __name__ == '__main__':
    niryo_robot = NiryoRosWrapper()
    niryo_robot.calibrate_auto()
    pick_n_place_version_x(niryo_robot)
```

First solution: the heaviest¶
Everything is done by hand:

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
def pick_n_place_version_1(niryo_ned):
    height_offset = 0.05  # Offset according to Z-Axis to go over pick & place poses
    gripper_speed = 400

    # Going Over Object
    niryo_ned.move_pose(pick_pose[0], pick_pose[1], pick_pose[2] + height_offset,
                        pick_pose[3], pick_pose[4], pick_pose[5])
    # Opening Gripper
    niryo_ned.open_gripper(gripper_speed)
    # Going to picking place and closing gripper
    niryo_ned.move_pose(pick_pose[0], pick_pose[1], pick_pose[2],
                        pick_pose[3], pick_pose[4], pick_pose[5])
    niryo_ned.close_gripper(gripper_speed)

    # Raising
    niryo_ned.move_pose(pick_pose[0], pick_pose[1], pick_pose[2] + height_offset,
                        pick_pose[3], pick_pose[4], pick_pose[5])

    # Going Over Place pose
    niryo_ned.move_pose(place_pose[0], place_pose[1], place_pose[2] + height_offset,
                        place_pose[3], place_pose[4], place_pose[5])
    # Going to Place pose
    niryo_ned.move_pose(place_pose[0], place_pose[1], place_pose[2],
                        place_pose[3], place_pose[4], place_pose[5])
    # Opening Gripper
    niryo_ned.open_gripper(gripper_speed)
    # Raising
    niryo_ned.move_pose(place_pose[0], place_pose[1], place_pose[2] + height_offset,
                        place_pose[3], place_pose[4], place_pose[5])
```

Second solution: pick from pose & place from pose functions¶
We use predefined functions:

```
def pick_n_place_version_3(niryo_ned):
    # Pick
    niryo_ned.pick_from_pose(*pick_pose)
    # Place
    niryo_ned.place_from_pose(*place_pose)
```

Third solution: all in one¶
We use THE predifined function:

```
def pick_n_place_version_4(niryo_ned):
    # Pick & Place
    niryo_ned.pick_and_place(pick_pose, place_pose)
```

Examples: Conveyor Belt

This document shows how to use Ned's Conveyor Belt.

If you want see more about Ned's Conveyor Belt functions, you can look at API - Conveyor.

Note

Imports & initialisation are not mentioned, but you should not forget it!

Simple Conveyor Belt control

This short example shows how to connect a Conveyor Belt, activate the connection and launch its motor:

```
niryo_robot = NiryoRosWrapper()

# Activating connexion with conveyor and storing ID
conveyor_id = niryo_robot.set_conveyor()

# Running conveyor at 50% of its maximum speed, in Forward direction
niryo_robot.control_conveyor(conveyor_id, True, 100, ConveyorDirection.FORWARD)

# Stopping robot motor
niryo_robot.control_conveyor(conveyor_id, True, 0, ConveyorDirection.FORWARD)

# Deactivating connexion with conveyor
niryo_robot.unset_conveyor(conveyor_id)
```

Advanced Conveyor Belt control

This example shows how to do a certain amount of pick & place by using the Conveyor Belt with the infrared sensor:

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```python
def run_conveyor(robot, conveyor):
    robot.control_conveyor(conveyor, bool_control_on=True,
                           speed=50, direction=ConveyorDirection.FORWARD)

# -- Setting variables
sensor_pin_id = PinID.GPIO_1A

catch_nb = 5

# The pick pose
pick_pose = [0.25, 0., 0.15, 0., 1.57, 0.0]
# The Place pose
place_pose = [0.0, -0.25, 0.1, 0., 1.57, -1.57]

# -- MAIN PROGRAM

niryo_robot = NiryoRosWrapper()

# Activating connexion with conveyor
conveyor_id = niryo_robot.set_conveyor()

for i in range(catch_nb):
    run_conveyor(niryo_robot, conveyor_id)
    while niryo_robot.digital_read(sensor_pin_id) == PinState.LOW:
        niryo_robot.wait(0.1)

    # Stopping robot motor
    niryo_robot.control_conveyor(conveyor_id, True, 0, ConveyorDirection.FORWARD)
    # Making a pick & place
    niryo_robot.pick_and_place(pick_pose, place_pose)

# Deactivating connexion with conveyor
niryo_robot.unset_conveyor(conveyor_id)
```

Examples: Vision

This document shows how to use Ned's Vision Set.

If you want see more about Ned's Vision functions, you can look at API - Vision (index.html#vision).

Beforehand

To realize the following examples, you need to have create a workspace.

As the examples start always the same, there is the code you need to add at the beginning of all of them:

```python
#!/usr/bin/env python

# Imports
from niryo_robot_python_ros_wrapper import *
import rospy

# Initializing ROS node
rospy.init_node('niryo_ned_example_python_ros_wrapper')

niryo_robot = NiryoRosWrapper()

# - Constants
workspace_name = "workspace_1"  # Robot's Workspace Name

# The observation pose
observation_pose = (0.18, 0., 0.35, 0., 1.57, -0.2)
# The Place pose
place_pose = (0., -0.25, 0.1, 0., 1.57, -1.57)

# - Main Program
# Calibrate robot if robot needs calibration
niryo_robot.calibrate_auto()
# Changing tool
niryo_robot.update_tool()
```

Simple Vision pick

This short example shows how to do your first vision pick:

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
niryo_robot.move_pose(*observation_pose)
# Trying to pick target using camera
ret = niryo_robot.vision_pick(workspace_name,
                              height_offset=0.0,
                              shape=ObjectShape.ANY,
                              color=ObjectColor.ANY)
obj_found, shape_ret, color_ret = ret
if obj_found:
    niryo_robot.place_from_pose(*place_pose)

niryo_robot.set_learning_mode(True)
```

Python ROS Wrapper documentation

This file presents the different Functions, Classes & Enums available with the API.

API functions

This class allows you to control the robot via internal API.
By controlling, we mean:

- Moving the robot.
- Using Vision.
- Controlling Conveyors Belt.
- Playing with hardware.

List of functions subsections:

Main purpose functions¶

class **NiryoRosWrapper**

calibrate_auto()

Call service to calibrate motors then wait for its end. If failed, raise NiryoRosWrapperException

Returns: status, message

Return type: (int, str)

calibrate_manual()

Call service to calibrate motors then wait for its end. If failed, raise NiryoRosWrapperException

Returns: status, message

Return type: (int, str)

get_learning_mode()

Use /niryo_robot/learning_mode/state topic subscriber to get learning mode status

Returns: `True` if activate else `False`

Return type: bool

set_learning_mode(*set_bool*)

Call service to set_learning_mode according to set_bool. If failed, raise NiryoRosWrapperException

Parameters: **set_bool** (*bool*) – `True` to activate, `False` to deactivate

Returns: status, message

Return type: (int, str)

set_arm_max_velocity(*percentage*)

Set relative max velocity (in %)

Parameters: **percentage** (*int*) – Percentage of max velocity

Returns: status, message

Return type: (int, str)

Joints & Pose¶

class **NiryoRosWrapper**

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

get_joints()

Use /joint_states topic to get joints status

Returns: list of joints value

Return type: list[float]

get_pose()

Use /niryo_robot/robot_state topic to get pose status

Returns: RobotState object (position.x/y/z && rpy.roll/pitch/yaw && orientation.x/y/z/w)

Return type: RobotState

get_pose_as_list()

Use /niryo_robot/robot_state topic to get pose status

Returns: list corresponding to [x, y, z, roll, pitch, yaw]

Return type: list[float]

move_joints(*j1*, *j2*, *j3*, *j4*, *j5*, *j6*)

Execute Move joints action

Parameters:
- **j1** (*float*) –
- **j2** (*float*) –
- **j3** (*float*) –
- **j4** (*float*) –
- **j5** (*float*) –
- **j6** (*float*) –

Returns: status, message

Return type: (int (https://docs.python.org/3/library/functions.html#int), str (https://docs.python.org/3/library/stdtypes.html#str))

move_to_sleep_pose()

Move to Sleep pose which allows the user to activate the learning mode without the risk of the robot hitting something because of gravity

Returns: status, message

Return type: (int, str)

move_pose(*x*, *y*, *z*, *roll*, *pitch*, *yaw*)

Move robot end effector pose to a (x, y, z, roll, pitch, yaw) pose.

Parameters:
- **x** (*float*) –
- **y** (*float*) –
- **z** (*float*) –
- **roll** (*float*) –
- **pitch** (*float*) –
- **yaw** (*float*) –

Returns: status, message

Return type: (int (https://docs.python.org/3/library/functions.html#int), str (https://docs.python.org/3/library/stdtypes.html#str))

shift_pose(*axis*, *value*)

Execute Shift pose action

Parameters:
- **axis** (*ShiftPose*) – Value of RobotAxis enum corresponding to where the shift happens
- **value** (*float*) – shift value

Returns: status, message

Return type: (int (https://docs.python.org/3/library/functions.html#int), str (https://docs.python.org/3/library/stdtypes.html#str))

shift_linear_pose(*axis*, *value*)

Execute Shift pose action with a linear trajectory

Parameters:
- **axis** (*ShiftPose*) – Value of RobotAxis enum corresponding to where the shift happens
- **value** (*float*) – shift value

Returns: status, message

Return type: (int (https://docs.python.org/3/library/functions.html#int), str (https://docs.python.org/3/library/stdtypes.html#str))

move_linear_pose(*x*, *y*, *z*, *roll*, *pitch*, *yaw*)

Move robot end effector pose to a (x, y, z, roll, pitch, yaw) pose, with a linear trajectory

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Parameters:
- **x** (*float*) –
- **y** (*float*) –
- **z** (*float*) –
- **roll** (*float*) –
- **pitch** (*float*) –
- **yaw** (*float*) –

Returns: status, message

Return type: (int (https://docs.python.org/3/library/functions.html#int), str (https://docs.python.org/3/library/stdtypes.html#str))

set_jog_use_state(*state*)

Turn jog controller On or Off

Parameters: **state** (*bool*) – `True` to turn on, else `False`

Returns: status, message

Return type: (int, str)

jog_joints_shift(*shift_values*)

Make a Jog on joints position

Parameters: **shift_values** (*list[float]*) – list corresponding to the shift to be applied to each joint

Returns: status, message

Return type: (int, str)

jog_pose_shift(*shift_values*)

Make a Jog on end-effector position

Parameters: **shift_values** (*list[float]*) – list corresponding to the shift to be applied to the position

Returns: status, message

Return type: (int, str)

forward_kinematics(*j1*, *j2*, *j3*, *j4*, *j5*, *j6*)

Compute forward kinematics

Parameters:
- **j1** (*float*) –
- **j2** (*float*) –
- **j3** (*float*) –
- **j4** (*float*) –
- **j5** (*float*) –
- **j6** (*float*) –

Returns: list corresponding to [x, y, z, roll, pitch, yaw]

Return type: list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]

inverse_kinematics(*x*, *y*, *z*, *roll*, *pitch*, *yaw*)

Compute inverse kinematics

Parameters:
- **x** (*float*) –
- **y** (*float*) –
- **z** (*float*) –
- **roll** (*float*) –
- **pitch** (*float*) –
- **yaw** (*float*) –

Returns: list of joints value

Return type: list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]

Saved poses¶

class **NiryoRosWrapper**

move_pose_saved(*pose_name*)

Move robot end effector pose to a pose saved

Parameters: **pose_name** (*str*) –

Returns: status, message

Return type: (int, str)

get_pose_saved(*pose_name*)

Get saved pose from robot intern storage Will raise error if position does not exist

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Parameters: **pose_name** (*str*) – Pose Name

Returns: x, y, z, roll, pitch, yaw

Return type: tuple[float]

save_pose(*name*, *x*, *y*, *z*, *roll*, *pitch*, *yaw*)

Save pose in robot's memory

Parameters:
- **name** (*str*) –
- **x** (*float*) –
- **y** (*float*) –
- **z** (*float*) –
- **roll** (*float*) –
- **pitch** (*float*) –
- **yaw** (*float*) –

Returns: status, message

Return type: (int (https://docs.python.org/3/library/functions.html#int), str (https://docs.python.org/3/library/stdtypes.html#str))

delete_pose(*name*)

Send delete command to the pose manager service

Parameters: **name** (*str*) –

Returns: status, message

Return type: (int, str)

get_saved_pose_list()

Ask the pose manager service which positions are available

Returns: list of positions name

Return type: list[str]

Pick & place¶

class **NiryoRosWrapper**

pick_from_pose(*x*, *y*, *z*, *roll*, *pitch*, *yaw*)

Execute a picking from a position. If an error happens during the movement, error will be raised. A picking is described as : - going over the object - going down until height = z - grasping with tool - going back over the object

Parameters:
- **x** (*float*) –
- **y** (*float*) –
- **z** (*float*) –
- **roll** (*float*) –
- **pitch** (*float*) –
- **yaw** (*float*) –

Returns: status, message

Return type: (int (https://docs.python.org/3/library/functions.html#int), str (https://docs.python.org/3/library/stdtypes.html#str))

place_from_pose(*x*, *y*, *z*, *roll*, *pitch*, *yaw*)

Execute a placing from a position. If an error happens during the movement, error will be raised. A placing is described as : - going over the place - going down until height = z - releasing the object with tool - going back over the place

Parameters:
- **x** (*float*) –
- **y** (*float*) –
- **z** (*float*) –
- **roll** (*float*) –
- **pitch** (*float*) –
- **yaw** (*float*) –

Returns: status, message

Return type: (int (https://docs.python.org/3/library/functions.html#int), str (https://docs.python.org/3/library/stdtypes.html#str))

pick_and_place(*pick_pose*, *place_pose*, *dist_smoothing=0.0*)

Execute a pick and place. If an error happens during the movement, error will be raised. -> Args param is for development purposes

Parameters:
- **pick_pose** (*list[float]*) –
- **place_pose** (*list[float]*) –
- **dist_smoothing** (*float*) – Distance from waypoints before smoothing trajectory

Returns: status, message

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Return type: (int (https://docs.python.org/3/library/functions.html#int), str (https://docs.python.org/3/library/stdtypes.html#str))

Trajectories¶

class **NiryoRosWrapper**

get_trajectory_saved(*trajectory_name*)

Get saved trajectory from robot intern storage Will raise error if position does not exist

Parameters: **trajectory_name** (*str*) –

Raises: **NiryoRosWrapperException** – If trajectory file doesn't exist

Returns: list of [x, y, z, qx, qy, qz, qw]

Return type: list[list[float]]

execute_trajectory_saved(*trajectory_name*)

Execute trajectory saved in Robot internal storage

Parameters: **trajectory_name** (*str*) –

Returns: status, message

Return type: (int, str)

execute_trajectory_from_poses(*list_poses_raw*, *dist_smoothing=0.0*)

Execute trajectory from a list of pose

Parameters:
- **list_poses_raw** (*list[list[float]]*) – list of [x, y, z, qx, qy, qz, qw] or list of [x, y, z, roll, pitch, yaw]
- **dist_smoothing** (*float*) – Distance from waypoints before smoothing trajectory

Returns: status, message

Return type: (int (https://docs.python.org/3/library/functions.html#int), str (https://docs.python.org/3/library/stdtypes.html#str))

execute_trajectory_from_poses_and_joints(*list_pose_joints*, *list_type=None*, *dist_smoothing=0.0*)

Execute trajectory from list of poses and joints

Parameters:
- **list_pose_joints** (*list[list[float]]*) – List of [x,y,z,qx,qy,qz,qw] or list of [x,y,z,roll,pitch,yaw] or a list of [j1,j2,j3,j4,j5,j6]
- **list_type** (*list[string]*) – List of string 'pose' or 'joint', or ['pose'] (if poses only) or ['joint'] (if joints only). If None, it is assumed there are only poses in the list.
- **dist_smoothing** (*float*) – Distance from waypoints before smoothing trajectory

Returns: status, message

Return type: (int (https://docs.python.org/3/library/functions.html#int), str (https://docs.python.org/3/library/stdtypes.html#str))

save_trajectory(*trajectory_name*, *list_poses_raw*)

Save trajectory object and send it to the trajectory manager service

Parameters:
- **trajectory_name** (*str*) – name which will have the trajectory
- **list_poses_raw** (*list[list[float]]*) – list of [x, y, z, qx, qy, qz, qw] or list of [x, y, z, roll, pitch, yaw]

Returns: status, message

Return type: (int (https://docs.python.org/3/library/functions.html#int), str (https://docs.python.org/3/library/stdtypes.html#str))

delete_trajectory(*trajectory_name*)

Send delete command to the trajectory manager service

Parameters: **trajectory_name** (*str*) – name

Returns: status, message

Return type: (int, str)

get_saved_trajectory_list()

Ask the pose trajectory service which trajectories are available

Returns: list of trajectory name

Return type: list[str]

Tools¶

class **NiryoRosWrapper**

get_current_tool_id()

Use /niryo_robot_hardware/tools/current_id topic to get current tool id

Returns: Tool Id

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Return type: ToolID

update_tool()

Call service niryo_robot_tools_commander/update_tool to update tool

Returns: status, message

Return type: (int, str)

grasp_with_tool(*pin_id=-1*)

Grasp with the tool linked to tool_id. This action correspond to - Close gripper for Grippers - Pull Air for Vacuum pump - Activate for Electromagnet

Parameters: **pin_id** (*PinID*) – [Only required for electromagnet] Pin ID of the electromagnet

Returns: status, message

Return type: (int, str)

release_with_tool(*pin_id=-1*)

Release with the tool associated to tool_id. This action correspond to - Open gripper for Grippers - Push Air for Vacuum pump - Deactivate for Electromagnet

Parameters: **pin_id** (*PinID*) – [Only required for electromagnet] Pin ID of the electromagnet

Returns: status, message

Return type: (int, str)

open_gripper(*speed=500*)

Open gripper with a speed 'speed'

Parameters: **speed** (*int*) – Default -> 500

Returns: status, message

Return type: (int, str)

close_gripper(*speed=500*)

Close gripper with a speed 'speed'

Parameters: **speed** (*int*) – Default -> 500

Returns: status, message

Return type: (int, str)

pull_air_vacuum_pump()

Pull air

Returns: status, message

Return type: (int, str)

push_air_vacuum_pump()

Pull air

Returns: status, message

Return type: (int, str)

setup_electromagnet(*pin_id*)

Setup electromagnet on pin

Parameters: **pin_id** (*PinID*) – Pin ID

Returns: status, message

Return type: (int, str)

activate_electromagnet(*pin_id*)

Activate electromagnet associated to electromagnet_id on pin_id

Parameters: **pin_id** (*PinID*) – Pin ID

Returns: status, message

Return type: (int, str)

deactivate_electromagnet(*pin_id*)

Deactivate electromagnet associated to electromagnet_id on pin_id

Parameters: **pin_id** (*PinID*) – Pin ID

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Returns: status, message

Return type: (int, str)

enable_tcp(*enable=True*)

Enables or disables the TCP function (Tool Center Point). If activation is requested, the last recorded TCP value will be applied. The default value depends on the gripper equipped. If deactivation is requested, the TCP will be coincident with the tool_link.

Parameters: **enable** (*Bool*) – True to enable, False otherwise.

Returns: status, message

Return type: (int, str)

set_tcp(*x*, *y*, *z*, *roll*, *pitch*, *yaw*)

Activates the TCP function (Tool Center Point) and defines the transformation between the tool_link frame and the TCP frame.

Parameters:
- **x** (*float*) –
- **y** (*float*) –
- **z** (*float*) –
- **roll** (*float*) –
- **pitch** (*float*) –
- **yaw** (*float*) –

Returns: status, message

Return type: (int (https://docs.python.org/3/library/functions.html#int), str (https://docs.python.org/3/library/stdtypes.html#str))

reset_tcp()

Reset the TCP (Tool Center Point) transformation. The TCP will be reset according to the tool equipped.

Returns: status, message

Return type: (int, str)

Hardware¶

class **NiryoRosWrapper**

set_pin_mode(*pin_id*, *pin_mode*)

Set pin number pin_id to mode pin_mode

Parameters:
- **pin_id** (*PinID*) –
- **pin_mode** (*PinMode*) –

Returns: status, message

Return type: (int (https://docs.python.org/3/library/functions.html#int), str (https://docs.python.org/3/library/stdtypes.html#str))

digital_write(*pin_id*, *digital_state*)

Set pin_id state to pin_state

Parameters:
- **pin_id** (*PinID*) –
- **digital_state** (*PinState*) –

Returns: status, message

Return type: (int (https://docs.python.org/3/library/functions.html#int), str (https://docs.python.org/3/library/stdtypes.html#str))

digital_read(*pin_id*)

Read pin number pin_id and return its state

Parameters: **pin_id** (*PinID*) –

Returns: state

Return type: PinState

get_hardware_status()

Get hardware status : Temperature, Hardware version, motors names & types ...

Returns: Infos contains in a HardwareStatus object (see niryo_robot_msgs)

Return type: HardwareStatus

get_digital_io_state()

Get Digital IO state : Names, modes, states

Returns: Infos contains in a DigitalIOState object (see niryo_robot_msgs)

Return type: DigitalIOState

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Conveyor Belt¶

class **NiryoRosWrapper**

set_conveyor()

Scan for conveyor on can bus. If conveyor detected, return the conveyor ID

Raises: **NiryoRosWrapperException** –

Returns: ID

Return type: ConveyorID

unset_conveyor(*conveyor_id*)

Remove specific conveyor.

Parameters: **conveyor_id** (*ConveyorID*) – Basically, ConveyorID.ONE or ConveyorID.TWO

Raises: **NiryoRosWrapperException** –

Returns: status, message

Return type: (int, str)

control_conveyor(*conveyor_id*, *bool_control_on*, *speed*, *direction*)

Control conveyor associated to conveyor_id. Then stops it if bool_control_on is False, else refreshes it speed and direction

Parameters:
- **conveyor_id** (*ConveyorID*) – ConveyorID.ID_1 or ConveyorID.ID_2
- **bool_control_on** (*bool*) – True for activate, False for deactivate
- **speed** (*int*) – target speed
- **direction** (*ConveyorDirection*) – Target direction

Returns: status, message

Return type: (int (https://docs.python.org/3/library/functions.html#int), str (https://docs.python.org/3/library/stdtypes.html#str))

Vision¶

class **NiryoRosWrapper**

get_compressed_image()

Get last stream image in a compressed format

Returns: string containing a JPEG compressed image

Return type: str

set_brightness(*brightness_factor*)

Modify image brightness

Parameters: **brightness_factor** (*float*) – How much to adjust the brightness. 0.5 will give a darkened image, 1 will give the original image while 2 will enhance the brightness by a factor of 2.

Returns: status, message

Return type: (int, str)

set_contrast(*contrast_factor*)

Modify image contrast

Parameters: **contrast_factor** (*float*) – While a factor of 1 gives original image. Making the factor towards 0 makes the image greyer, while factor>1 increases the contrast of the image.

Returns: status, message

Return type: (int, str)

set_saturation(*saturation_factor*)

Modify image saturation

Parameters: **saturation_factor** (*float*) – How much to adjust the saturation. 0 will give a black and white image, 1 will give the original image while 2 will enhance the saturation by a factor of 2.

Returns: status, message

Return type: (int, str)

get_target_pose_from_rel(*workspace_name*, *height_offset*, *x_rel*, *y_rel*, *yaw_rel*)

Given a pose (x_rel, y_rel, yaw_rel) relative to a workspace, this function returns the robot pose in which the current tool will be able to pick an object at this pose. The height_offset argument (in m) defines how high the tool will hover over the workspace. If height_offset = 0, the tool will nearly touch the workspace.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Parameters:
- **workspace_name** (*str*) – name of the workspace
- **height_offset** (*float*) – offset between the workspace and the target height
- **x_rel** (*float*) –
- **y_rel** (*float*) –
- **yaw_rel** (*float*) –

Returns: target_pose

Return type: RobotState

get_target_pose_from_cam(*workspace_name*, *height_offset*, *shape*, *color*)

First detects the specified object using the camera and then returns the robot pose in which the object can be picked with the current tool

Parameters:
- **workspace_name** (*str*) – name of the workspace
- **height_offset** (*float*) – offset between the workspace and the target height
- **shape** (*ObjectShape*) – shape of the target
- **color** (*ObjectColor*) – color of the target

Returns: object_found, object_pose, object_shape, object_color

Return type: (bool (https://docs.python.org/3/library/functions.html#bool), RobotState, str (https://docs.python.org/3/library/stdtypes.html#str), str (https://docs.python.org/3/library/stdtypes.html#str))

vision_pick_w_obs_joints(*workspace_name*, *height_offset*, *shape*, *color*, *observation_joints*)

Move Joints to observation_joints, then execute a vision pick

vision_pick_w_obs_pose(*workspace_name*, *height_offset*, *shape*, *color*, *observation_pose_list*)

Move Pose to observation_pose, then execute a vision pick

vision_pick(*workspace_name*, *height_offset*, *shape*, *color*)

Picks the specified object from the workspace. This function has multiple phases: 1. detect object using the camera 2. prepare the current tool for picking 3. approach the object 4. move down to the correct picking pose 5. actuate the current tool 6. lift the object

Parameters:
- **workspace_name** (*str*) – name of the workspace
- **height_offset** (*float*) – offset between the workspace and the target height
- **shape** (*ObjectShape*) – shape of the target
- **color** (*ObjectColor*) – color of the target

Returns: object_found, object_shape, object_color

Return type: (bool (https://docs.python.org/3/library/functions.html#bool), ObjectShape (index.html#niryo_robot_python_ros_wrapper.ros_wrapper_enums.ObjectShape), ObjectColor (index.html#niryo_robot_python_ros_wrapper.ros_wrapper_enums.ObjectColor))

move_to_object(*workspace*, *height_offset*, *shape*, *color*)

Same as *get_target_pose_from_cam* but directly moves to this position

Parameters:
- **workspace** (*str*) – name of the workspace
- **height_offset** (*float*) – offset between the workspace and the target height
- **shape** (*ObjectShape*) – shape of the target
- **color** (*ObjectColor*) – color of the target

Returns: object_found, object_shape, object_color

Return type: (bool (https://docs.python.org/3/library/functions.html#bool), ObjectShape (index.html#niryo_robot_python_ros_wrapper.ros_wrapper_enums.ObjectShape), ObjectColor (index.html#niryo_robot_python_ros_wrapper.ros_wrapper_enums.ObjectColor))

detect_object(*workspace_name*, *shape*, *color*)

Parameters:
- **workspace_name** (*str*) – name of the workspace
- **shape** (*ObjectShape*) – shape of the target
- **color** (*ObjectColor*) – color of the target

Returns: object_found, object_pose, object_shape, object_color

Return type: (bool (https://docs.python.org/3/library/functions.html#bool), RobotState, str (https://docs.python.org/3/library/stdtypes.html#str), str (https://docs.python.org/3/library/stdtypes.html#str))

get_camera_intrinsics()

Get calibration object: camera intrinsics, distortions coefficients

Returns: raw camera intrinsics, distortions coefficients

Return type: (list, list)

save_workspace_from_poses(*name*, *list_poses_raw*)

Save workspace by giving the poses of the robot to point its 4 corners with the calibration Tip. Corners should be in the good order

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Parameters:
- **name** (*str*) – workspace name, max 30 char.
- **list_poses_raw** (*list[list]*) – list of 4 corners pose

Returns: status, message

Return type: (int (https://docs.python.org/3/library/functions.html#int), str (https://docs.python.org/3/library/stdtypes.html#str))

save_workspace_from_points(*name*, *list_points_raw*)

Save workspace by giving the poses of its 4 corners in the good order

Parameters:
- **name** (*str*) – workspace name, max 30 char.
- **list_points_raw** (*list[list[float]]*) – list of 4 corners [x, y, z]

Returns: status, message

Return type: (int (https://docs.python.org/3/library/functions.html#int), str (https://docs.python.org/3/library/stdtypes.html#str))

delete_workspace(*name*)

Call workspace manager to delete a certain workspace

Parameters: **name** (*str*) – workspace name

Returns: status, message

Return type: (int, str)

get_workspace_poses(*name*)

Get the 4 workspace poses of the workspace called 'name'

Parameters: **name** (*str*) – workspace name

Returns: List of the 4 workspace poses

Return type: list[list]

get_workspace_ratio(*name*)

Give the length over width ratio of a certain workspace

Parameters: **name** (*str*) – workspace name

Returns: ratio

Return type: float

get_workspace_list()

Ask the workspace manager service names of the available workspace

Returns: list of workspaces name

Return type: list[str]

Enums

class **ShiftPose**

AXIS_X= *0*

AXIS_Y= *1*

AXIS_Z= *2*

ROT_ROLL= *3*

ROT_PITCH= *4*

ROT_YAW= *5*

class **ToolID**

Tools IDs (need to match tools ids in niryo_robot_tools_commander package)

NONE= *0*

GRIPPER_1= *11*

GRIPPER_2= *12*

GRIPPER_3= *13*

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

GRIPPER_4= *14*

ELECTROMAGNET_1= *30*

VACUUM_PUMP_1= *31*

class **PinMode**

Pin Mode is either OUTPUT or INPUT

OUTPUT= *0*

INPUT= *1*

class **PinState**

Pin State is either LOW or HIGH

LOW= *0*

HIGH= *1*

class **PinID**

Pins ID

GPIO_1A= *2*

GPIO_1B= *3*

GPIO_1C= *16*

GPIO_2A= *26*

GPIO_2B= *19*

GPIO_2C= *6*

SW_1= *12*

SW_2= *13*

class **ConveyorID**

NONE= *0*

ID_1= *12*

ID_2= *13*

class **ConveyorDirection**

FORWARD= *1*

BACKWARD= *-1*

class **ObjectColor**

RED= *'RED'*

GREEN= *'GREEN'*

BLUE= *'BLUE'*

ANY= *'ANY'*

class **ObjectShape**

CIRCLE= *'CIRCLE'*

SQUARE= *'SQUARE'*

ANY= *'ANY'*

Use the Modbus TCP server

todo: vu avec Etienne, supprimer la partie Modbus d'ici et mettre en place une redirection vers doc Modbus.

Ned is permanently running a Modbus TCP Server that enables Ned to communicate with a PLC, or another computer in the same network.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Connection - Modbus TCP server

The Modbus TCP server is running on port 5020 by default.

Description - Modbus TCP server

It has been built on top of the pymodbus library (https://pymodbus.readthedocs.io/en/latest/index.html).

All 4 Modbus datastores are implemented:

- Coil.
- Discrete Input.
- Holding Register.
- Input Register.

Each datastore has a different set of functionalities. Note that each datastore contains a completely different set of data.

Address tables start at 0.

Coil

Each address contains a 1bit value.

READ/WRITE (the stored values correspond to the last given command, not the current robot state).

Accepted Modbus functions:

- 0x01: READ_COILS
- 0x05: WRITE_SINGLE_COIL

This datastore can be used to set Digital I/O mode and state.

Digital I/O numbers used for Modbus:

- 0/100: 1A
- 1/101: 1B
- 2/102: 1C
- 3/103: 2A
- 4/104: 2B
- 5/105: 2C

Address	Description
0-5	Digital I/O mode (Input = 1, Output = 0)
100-105	Digital I/O state (High = 1, Low = 0)
200-299	Can be used to store your own variables

Discrete input

Each address contains a 1bit value.

READ-ONLY

Accepted Modbus functions:

- 0x02: READ_DISCRETE_INPUTS

Digital I/O numbers used for Modbus:

- 0/100: 1A
- 1/101: 1B
- 2/102: 1C
- 3/103: 2A
- 4/104: 2B
- 5/105: 2C

Address	Description
0-5	Digital I/O mode (Input = 1, Output = 0)
100-105	Digital I/O state (High = 1, Low = 0)

This datastore can be used to read Digital I/O mode and state of the robot. See Coil above for digital I/O number mapping.

Holding register

Each address contains a 16bits value.

READ/WRITE (the stored values correspond to the last given command, not the current robot state)

Accepted Modbus functions:

- 0x03: READ_HOLDING_REGISTERS
- 0x06: WRITE_SINGLE_REGISTER

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Address	Description
0-5	Joints (mrad)
10-12	Position x,y,z (mm)
13-15	Orientation roll, pitch, yaw (mrad)
100	Sends Joint Move command with stored joints
101	Sends Pose Move command with stored position and orientation
102	Sends Pose Linear Move command with stored position and orientation
110	Stops current command execution
150	Is executing command flag
151	Last command result (status of the last command)
152	Contains data retrieved from last cmd (depends of the cmd)
153 - 158	Vision - Target pose result
159	Vision - Shape of the object found (-1: ANY, 1: CIRCLE, 2: SQUARE, 3: TRIANGLE, 0: NONE)
160	Vision - Color of the object found (-1: ANY, 1: BLUE, 2: RED, 3: GREEN, 0: NONE)
200-299	Can be used to store your own variables
300	Learning Mode (On = 1, Off = 0)
301	Joystick Enabled (On = 1, Off = 0)
310	Requests new calibration
311	Starts auto calibration
312	Starts manual calibration
401	Gripper open speed (100-1000)
402	Gripper close speed (100-1000)
500	Updates the tool id according to the gripper plugged (gripper 1: 11, gripper 2: 12, gripper 3: 13, vaccum pump: 31)
501	Stores the tool id
510	Opens gripper previously updated
511	Closes gripper with given id
512	Pulls air vacuum pump from given id
513	Pushes air vacuum pump from given id
520	Enables a Conveyor Belt newly connected [on success: store its ID at 152]
521	Detaches / disables Conveyor Belt with the Conveyor Belt ID given at 525
522	Control Conveyor Belt with the Conveyor Belt ID given at 525
523 [related to 522]	Conveyor Belt direction (backward = -1 , forward = 1)
524 [related to 522]	Conveyor Belt speed (0-100)(%)
525 [related to 520/521/522/526]	Stores the Conveyor Belt ID for all related command
526	Stops Conveyor Belt with the Conveyor Belt ID given at 525
600	TCP - Enables or disables the TCP function (Tool Center Point).
601	Activates the TCP function (Tool Center Point) and defines the transformation between the tool_link frame and the TCP frame.
610	Vision - Gets target pose from relative pose, with stored relative pose and height_offset
611	Vision - Gets target pose from camera, with stored workspace name, height offset, shape and color
612	Vision - Vision pick, with stored workspace name, height offset, shape and color
613	Vision - Moves to object, with stored workspace name, height offset, shape and color
614	Vision - Detects object, with stored workspace name, shape and color
620	Vision - Stores workspace's height offset
621	Vision - Stores relative pose x_rel
622	Vision - Stores relative pose y_rel
623	Vision - Stores relative pose yaw_rel
624	Vision - Stores requested shape (-1: ANY, 1: CIRCLE, 2: SQUARE, 3: TRIANGLE)
625	Vision - Stores requested color (-1: ANY, 1: BLUE, 2: RED, 3: GREEN)
626 - max 641	Vision - Stores workspace's name, as a string encoded in 16 bits hex (see examples on how to store a workspace name from a client)

Input Register

Each address contains a 16bits value.

READ-ONLY

Accepted Modbus functions:

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

- 0x04: READ_INPUT_REGISTERS

Address	Description
0-5	Joints (mrad)
10-12	Position x,y,z (mm)
13-15	Orientation roll, pitch, yaw (mrad)
200	Selected tool ID (0 for no tool)
300	Learning Mode activated
400	Motors connection up (Ok = 1, Not ok = 0)
401	Calibration needed flag
402	Calibration in progress flag
403	Raspberry Pi temperature
404	Raspberry Pi available disk size
405	Raspberry Pi ROS log size
406	RPI software version n.1
407	RPI software version n.2
408	RPI software version n.3
409	Hardware version (1 or 2)
530	Conveyor 1 connection state (Connected = 1 , Not connected = 0)
531	Conveyor 1 control status (On = 0, Off = 1)
532	Conveyor 1 Speed (0-100 (%))
533	Conveyor 1 direction (Backward = -1, Forward = 1)
540	Conveyor 2 connection state (Connected = 1 , Not connected = 0)
541	Conveyor 2 control status (On = 0, Off = 1)
542	Conveyor 2 Speed (0-100 (%))
543	Conveyor 2 direction (Backward = -1, Forward = 1)

Dependencies - Modbus TCP Server

- pymodbus library
- Niryo_robot_msgs
- std_msgs

More ways to control Ned

There is even more ways to control Ned.

If you are a beginner, look at Blockly section to understand the programming fundamentals.
If you want to go further, maybe experience your own image processing, multi-robot, AI... You can go to PyNiryo for more information.

Blockly

Check out Niryo Studio.

PyNiryo

As explained in the page Use Ned's TCP server (index.html#use-ned-s-tcp-server), a TCP Server is running on Ned, which allows it to receive commands from any external device.

PyNiryo is a Python package available on Pip which allows to command the Niryo Robots with easy Python Binding.

Suggest a modification | Download as PDF

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Niryo Modbus Documentation

In this document, we will focus on the Modbus/TCP server.

The Modbus/TCP server is running on **port 5020** by default. It has been built on top of the pymodbus (https://pymodbus.readthedocs.io/en/latest/index.html) library. This enables you to make Ned communicates with a PLC, or another computer on the same network.

Modbus Python library installation

To use the Modbus Python library, your workspace must have a Python interpreter with **Python 3** (3.6 or greater) or **Python 2** (2.7 or greater).

Note

Download Python on the official Python website (https://www.python.org/) and find more information about the installation on this website (https://realpython.com/installing-python/).

This installation requires the use of pip (https://pypi.org/project/pip/), the package manager included in Python.

Start with the installation of numpy:

```
pip install numpy
```

To use the **Modbus API**, you also need to install Modbus python library pymodbus (https://pymodbus.readthedocs.io/en/latest/index.html):

```
pip install pymodbus
```

Attention

- Pip can require administrator authorizations to install packages. In this case, add

  ```
  sudo
  ```

  before your command lines on Linux.
- If pip is not automatically installed with Python, please visit the following website: pip installation (https://pypi.org/project/pip/).

Introduction

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

All 4 Modbus datastores are implemented: Coils, Discrete inputs, Holding registers, Input registers. Each datastore has a different set of functionalities. Note that each datastore contains a completely different set of data.

Discrete Input and Input register are **READ-ONLY** tables. Those have been used to keep the robot state.

Coil and Holding Register are **READ/WRITE** tables. Those have been used to give user commands to the robot. Hence, those 2 tables do not contain the robot state, but the last given command.

Address tables start at 0.

Coils

Each address contains a 1bit value.

READ/WRITE (the stored values correspond to the last given command, not the current robot state)

Accepted Modbus functions:

- 0x01: READ_COILS
- 0x05: WRITE_SINGLE_COIL

This datastore can be used to set Digital I/O mode and state. Digital I/O numbers used for Modbus:

- 0: 1A
- 1: 1B
- 2: 1C
- 3: 2A
- 4: 2B
- 5: 2C

Address	Description
0-5	Digital I/O mode (Input = 1, Output = 0)
100-105	Digital I/O state (High = 1, Low = 0)
200-299	Can be used to store your own variables

Discrete inputs

Each address contains a 1bit value.

READ-ONLY

Accepted Modbus functions:

- 0x02: READ_DISCRETE_INPUTS

This datastore can be used to read Digital I/O mode and state. See the Coils section above for digital I/O number mapping.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Address	Description
0-5	Digital I/O mode (Input = 1, Output = 0)
100-105	Digital I/O state (High = 1, Low = 0)

Holding registers

Each address contains a 16bit value.

READ/WRITE (the stored values correspond to the last given command, not the current robot state)

Accepted Modbus functions:

- 0x03: READ_HOLDING_REGISTERS
- 0x06: WRITE_SINGLE_REGISTER

Address	Description
0-5	Joints (mrad)
10-12	Position x,y,z (mm)
13-15	Orientation roll, pitch, yaw (mrad)
100	Sends Joint Move command with stored joints
101	Sends Pose Move command with stored position and orientation
102	Sends Linear Pose Move command with stored position and orientation
110	Stops current command execution
150	Is executing command flag
151	Last command result*
152	Last command data result (if not vision related)
153 - 158	Vision - Target pose result
159	Vision - Shape of the object found (-1: ANY, 1: CIRCLE, 2: SQUARE, 3: TRIANGLE, 0: NONE)
160	Vision - Color of the object found (-1: ANY, 1: BLUE, 2: RED, 3: GREEN, 0: NONE)
200-299	Can be used to store your own variables
300	Learning Mode (On = 1, Off = 0)
301	Joystick Enabled (On = 1, Off = 0)
310	Requests new calibration
311	Starts auto calibration
312	Starts manual calibration
401	Gripper open speed (100-1000)
402	Gripper close speed (100-1000)
500	Updates the tool id according to the gripper plugged (gripper 1: 11, gripper 2: 12, gripper 3: 13, vaccum pump: 31)
501	Stores the tool id
510	Opens gripper previously updated

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Address	Description
511	Closes gripper previously updated
512	Pulls air vacuum pump with id 31
513	Pushes air vacuum pump with id 31
520	Updates the conveyor id and enable it
521	Detaches or disables the conveyor previously enabled and updated
522	Starts the conveyor previously enabled and updated
523	Sets the conveyor direction (backward = number_to_raw_data(-1), forward = 1)
524	Sets the conveyor speed (0-100)(%)
525	Stores the conveyor id
526	Stops conveyor previously enabled and updated
600	TCP - Enables or disables the TCP function (Tool Center Point).
601	Activates the TCP function (Tool Center Point) and defines the transformation between the tool_link frame and the TCP frame.
610	Vision - Gets target pose from relative pose, with stored relative pose and height_offset
611	Vision - Gets target pose from camera, with stored workspace name, height offset, shape and color
612	Vision - Vision pick, with stored workspace name, height offset, shape and color
613	Vision - Moves to object, with stored workspace name, height offset, shape and color
614	Vision - Detects object, with stored workspace name, shape and color
620	Vision - Stores workspace's height offset
621	Vision - Stores relative pose x_rel
622	Vision - Stores relative pose y_rel
623	Vision - Stores relative pose yaw_rel
624	Vision - Stores requested shape (-1: ANY, 1: CIRCLE, 2: SQUARE, 3: TRIANGLE)
625	Vision - Stores requested color (-1: ANY, 1: BLUE, 2: RED, 3: GREEN)
626 - max 641	Vision - Stores workspace's name, as a string encoded in 16 bits hex (see examples on how to store a workspace name from a client)

'*' The "Last command result" gives you more information about the last executed command:

- 0: no result yet
- 1: success
- 2: command was rejected (invalid params, ...)
- 3: command was aborted
- 4: command was canceled
- 5: command had an unexpected error
- 6: command timeout
- 7: internal error

Input registers

Each address contains a 16bit value.

READ-ONLY.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Accepted Modbus functions:

- 0x04: READ_INPUT_REGISTERS

Address	Description
0-5	Joints (mrad)
10-12	Position x,y,z (mm)
13-15	Orientation roll, pitch, yaw (mrad)
200	Selected tool ID (0 for no tool)
300	Learning Mode activated
400	Motors connection up (Ok = 1, Not ok = 0)
401	Calibration needed flag
402	Calibration in progress flag
403	Raspberry Pi temperature
404	Raspberry Pi available disk size
405	Raspberry Pi ROS log size
406	Ned RPI image version n.1
407	Ned RPI image version n.2
408	Ned RPI image version n.3
409	Hardware version (1 or 2)
530	Conveyor 1 connection state (Connected = 1 , Not connected = 0)
531	Conveyor 1 control status (On = 0, Off = 1)
532	Conveyor 1 Speed (0-100 (%))
533	Conveyor 1 direction (Backward = -1, Forward = 1)
540	Conveyor 2 connection state (Connected = 1 , Not connected = 0)
541	Conveyor 2 control status (On = 0, Off = 1)
542	Conveyor 2 Speed (0-100 (%))
543	Conveyor 2 direction (Backward = -1, Forward = 1)

Repository

Examples of Modbus python lib can be found here Python Modbus examples (https://github.com/NiryoRobotics/ned_ros/tree/master/niryo_robot_modbus/examples/).

Examples

In the examples folder, you can find several example scripts that control Ned. These scripts are commented to help you understand every step.

Client Modbus Test

Calls several functions on the IO of Ned.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Client Move Command

This script shows the calibration and Ned's moves.

Client Modbus Conveyor Example

This script shows how to activate the Conveyor Belt through the Modbus Python API, set a direction, a speed, and start and stop the device.

Client Modbus Vision Example

This script shows how to use the vision pick method from a Modbus Client, through the Modbus Python API. Ned picks a red object seen in its workspace and releases it on its left. Note that we use the **string_to_register** method to convert a string into an object storable in registers.

```
#!/usr/bin/env python

from pymodbus.client.sync import ModbusTcpClient
from pymodbus.payload import BinaryPayloadBuilder, BinaryPayloadDecoder
import time
from enum import Enum, unique

# Enums for shape and color. Those enums are the one used by the modbus server to receive requests
@unique
class ColorEnum(Enum):
    ANY = -1
    BLUE = 1
    RED = 2
    GREEN = 3
    NONE = 0

@unique
class ShapeEnum(Enum):
    ANY = -1
    CIRCLE = 1
    SQUARE = 2
    TRIANGLE = 3
    NONE = 0

# Functions to convert variables for/from registers

# Positive number : 0 - 32767
# Negative number : 32768 - 65535
def number_to_raw_data(val):
    if val < 0:
        val = (1 << 15) - val
    return val


def raw_data_to_number(val):
    if (val >> 15) == 1:
        val = - (val & 0x7FFF)
    return val

def string_to_register(string):
    # code a string to 16 bits hex value to store in register
    builder = BinaryPayloadBuilder()
    builder.add_string(string)
    payload = builder.to_registers()
    return payload

# ---------- Modbus server related function

def back_to_observation():
    # To change
    # joint_real = [0.057, 0.604, -0.576, -0.078, -1.384,0.253]
    joint_simu = [0, -0.092, 0, 0, -1.744, 0]

    joint_to_send = list(map(lambda j: int(number_to_raw_data(j * 1000)), joint_simu))
    client.write_registers(0, joint_to_send)
    client.write_register(100, 1)

    while client.read_holding_registers(150, count=1).registers[0] == 1:
```

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
        time.sleep(0.01)

def register_workspace_name(ws_name):
    workspace_request_register = string_to_register(ws_name)
    client.write_registers(626, workspace_request_register)

def register_height_offset(height_offset):
    client.write_registers(620, int(number_to_raw_data(height_offset * 1000)))

def auto_calibration():
    print "Calibrate Robot if needed ..."
    client.write_register(311, 1)
    # Wait for end of calibration
    while client.read_input_registers(402, 1).registers[0] == 1:
        time.sleep(0.05)

def get_current_tool_id():
    return client.read_input_registers(200, count=1).registers[0]

def open_tool():
    tool_id = get_current_tool_id()
    if tool_id == 31:
        client.write_register(513, 1)
    else:
        client.write_register(510, 1)
    while client.read_holding_registers(150, count=1).registers[0] == 1:
        time.sleep(0.05)


# Function to call Modbus Server vision pick function
def vision_pick(workspace_str, height_offset, shape_int, color_int):
    register_workspace_name(workspace_str)
    register_height_offset(height_offset)

    client.write_registers(624, number_to_raw_data(shape_int))
    client.write_registers(625, number_to_raw_data(color_int))

    # launch vision pick function
    client.write_registers(612, 1)

    # Wait for end of function
    while client.read_holding_registers(150, count=1).registers[0] == 1:
        time.sleep(0.01)

    # - Check result : SHAPE AND COLOR
    result_shape_int = raw_data_to_number(client.read_holding_registers(159).registers[0])
    result_color_int = raw_data_to_number(client.read_holding_registers(160).registers[0])

    return result_shape_int, result_color_int

# ---------- Main programm

if __name__ == '__main__':
    print "--- START"

    client = ModbusTcpClient('localhost', port=5020)

    # -------- Variable definition
    # To change
    workspace_name = 'gazebo_1'
    height_offset = 0.0

    # connect to modbus server
    client.connect()
    print "Connected to modbus server"

    # launch auto calibration then go to obs. pose
    auto_calibration()
    back_to_observation()

    # update tool
    client.write_registers(500, 1)

    print 'VISION PICK - pick a red pawn, lift it and release it'
    shape = ShapeEnum.ANY.value
    color = ColorEnum.RED.value
    shape_picked, color_picked = vision_pick(workspace_name, height_offset, shape, color)

    # ---- Go to release pose
    joints = [0.866, -0.24, -0.511, 0.249, -0.568, -0.016]
    joints_to_send = list(map(lambda j: int(number_to_raw_data(j * 1000)), joints))
```

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
client.write_registers(0, joints_to_send)
client.write_register(100, 1)

# Wait for end of Move command
while client.read_holding_registers(150, count=1).registers[0] == 1:
    time.sleep(0.01)

open_tool()

back_to_observation()

# Activate learning mode and close connexion
client.write_register(300, 1)
client.close()
print "Close connection to modbus server"
print "--- END"
```

Suggest a modification Download as PDF

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

PyNiryo Documentation

This documentation presents Ned's PyPi package, which is a TCP API made with Python.

It offers a simple way for developers to create programs for robot and to control them via remote communication from their computers. Contrary to the Python ROS Wrapper, the user will not need to be connected on the robot through a terminal.

Note

This package is able to control Ned in simulation as well as the physical robot.

Ned

Before getting started

If you haven't already done so, make sure to learn about the ROS robot software by readingROS documentation.

This documentation also contains everything you need to know if you want to use Ned through simulation.

Sections organization

This document is organized in 4 main sections.

Setup

Install & Setup your environment in order to use Ned with PyNiryo.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Firstly, follow Installation instructions (index.html#document-source/setup/installation), then find your Robot IP address (index.html#document-source/setup/ip_address) to be ready.

Installation

The library uses Python, which must be installed and available in your working environment.
The version should be **equal or above**:

- 2.7 if you are using Python 2
- 3.6 if you are using Python 3

> **Hint**
>
> To check your Python version, use the command `python --version` if you are using Python 2 and `python3 --version` if you are using Python3.

The below sections explain how to install the library with **pip**, the package installer for Python.

> **Attention**
>
> If you have both Python 2 & Python 3 installed on your computer, the command `pip` will install packages in Python 2 version. You should use `pip3` instead in order to target Python 3.

Installation with pip

You need to install Numpy package beforehand:

```
pip install numpy
```

To install Ned's Python package via `pip` , simply execute:

```
pip install pyniryo
```

You can find more information about the PyPi packagehere (https://pypi.org/project/pyniryo/).

If you also want to use Vision functions to do your own Image Processing pipeline install OpenCV via the command:

```
pip install opencv-python
```

Uninstall

To uninstall the library use:

```
pip uninstall pyniryo
```

Find your Robot's IP address

In order to use your robot through TCP connection, you will first need to connect to it, which implies that you know its IP address.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

The next sections explain how to find your robot IP according to your configuration:

Hotspot mode

If you are directly connected to your robot through its wi-fi, the IP address you will need to use is `10.10.10.10` .

Simulation or directly on the robot

In this situation, the robot is running on the same computer as the client, the IP address will be the localhost address `127.0.0.1` .

Direct ethernet connection

If you are directly connected to your robot with an ethernet cable, the static IP of your robot will be `169.254.200.200` .

The reader should note that he may need to change his wired settings to allow the connection. See how to Connect to Ned via Ethernet on Ubuntu (https://docs.niryo.com/applications/ned/source/tutorials/setup_connect_ned_ethernet.html).

Computer and robot connected on the same router

You will need to find the robot's address using `nmap` , or you can also use search button of Niryo Studio to see which robots are available.

You can also Make IP permanent so that you will not have to search for it next time

Make IP permanent

Step 1

Firstly, you need to be connected to your robot via SSH.

On Ubuntu, use the command line:

```
ssh niryo@<robot_ip_address>
```

The password is `robotics` .

On Windows, you can use Putty (https://www.putty.org/). Robot username is `niryo` and password is `robotics` .

Step 2

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Find your proxy key:

```
ifconfig
```

Your proxy key is written on the first line and should look something like `eth0` .

Step 3

Select arbitrarily a number between 50 & 255. It will be your IP address' last number.

Then, edit the file `/etc/network/interfaces` :

```
sudo nano /etc/network/interfaces
```

And add to its end:

```
auto <robot_proxy_key>
iface <robot_proxy_key> inet static
   address 192.168.1.<your_ip_address_last_number>
   broadcast 192.168.1.255
   netmask 255.255.255.0
   gateway 192.168.1.1
   dns-nameservers 192.168.1.1
```

From its next reboot, the robot will appear under the IP `192.168.1.<your_ip_address_last_number>` .

Verify your Setup and Get Started

In order to verify your computer's setup, we are going to run a program from it, and see if the robot answers as expected.

Note

Before verifying your setup, be sure that your physical robot (or simulation) is turned on.

Firstly, go in the folder of your choice and create an empty file named "pyniryo_test.py". This file will contain the checking code.

Edit this file and fill it with the following code:

```
from pyniryo import *

robot_ip_address = "10.10.10.10"

# Connect to robot & calibrate
robot = NiryoRobot(robot_ip_address)
robot.calibrate_auto()
# Move joints
robot.move_joints(0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
# Turn learning mode ON
robot.set_learning_mode(True)
# Stop TCP connection
robot.close_connection()
```

Attention

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Replace the third line with your Robot IP Address (index.html#document-source/setup/ip_address) if you are not using Hotspot Mode.

Still on your computer, open a terminal, and place your current directory in the same folder than your file. Then, run the command:

```
python pyniryo_test.py
```

Note

If you are using Python 3, you may need to change `python` to `python3` .

If your robot starts calibrating, then moves, and finally, goes to learning mode, your setup is validated, you can now start coding!

Examples

Learn how to use the PyNiryo package to implement various tasks.

Examples: Basics

In this file, two short programs are implemented & commented in order to help you understand the philosophy behind the PyNiryo package.

Danger

If you are using the real robot, make sure the environment around it is clear.

Your first move joint

The following example shows a first use case. It's a simple MoveJ.

```
from pyniryo import *

robot = NiryoRobot("10.10.10.10")

robot.calibrate_auto()

robot.move_joints(0.2, -0.3, 0.1, 0.0, 0.5, -0.8)

robot.close_connection()
```

Code Details - First Move J

First of all, we import the library to be able to access functions.

```
from pyniryo import *
```

Then, we instantiate the connection and link the variable `robot` to the robot at the IP Address `10.10.10.10` .

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
robot = NiryoRobot("10.10.10.10")
```

Once the connection is done, we calibrate the robot using its **calibrate_auto()** (index.html#api.tcp_client.NiryoRobot.calibrate_auto) function.

```
robot.calibrate_auto()
```

As the robot is now calibrated, we can do a Move Joints by giving the 6 axis positions in radians! To do so, we use **move_joints()** (index.html#api.tcp_client.NiryoRobot.move_joints).

```
robot.move_joints(0.2, -0.3, 0.1, 0.0, 0.5, -0.8)
```

Our process is now over, we can close the connection with **quit()** .

```
robot.close_connection()
```

Your first pick and place

In the second example, we are going to develop a pick and place algorithm.

```
from pyniryo import *

robot = NiryoRobot("10.10.10.10")

robot.calibrate_auto()
robot.update_tool()

robot.release_with_tool()
robot.move_pose(0.2, -0.1, 0.25, 0.0, 1.57, 0.0)
robot.grasp_with_tool()

robot.move_pose(0.2, 0.1, 0.25, 0.0, 1.57, 0.0)
robot.release_with_tool()

robot.close_connection()
```

Code Details - First Pick And Place

First of all, we import the library and start the connection between our computer and the robot. We also calibrate the robot.

```
from pyniryo import *

robot = NiryoRobot("10.10.10.10")
robot.calibrate_auto()
```

Then, we equip the tool with **update_tool()** (index.html#api.tcp_client.NiryoRobot.update_tool).

```
robot.update_tool()
```

Now that our initialization is done, we can open the gripper (or push air from the vacuum) with **release_with_tool()** (index.html#api.tcp_client.NiryoRobot.release_with_tool), go to the picking pose via **move_pose()** (index.html#api.tcp_client.NiryoRobot.move_pose) & then catch an object with

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

grasp_with_tool() (index.html#api.tcp_client.NiryoRobot.grasp_with_tool)!

```
robot.release_with_tool()
robot.move_pose(0.2, -0.1, 0.25, 0.0, 1.57, 0.0)
robot.grasp_with_tool()
```

We now get to the place pose, and place the object.

```
robot.move_pose(0.2, 0.1, 0.25, 0.0, 1.57, 0.0)
robot.release_with_tool()
```

Our process is now over, we can close the connection.

```
robot.close_connection()
```

Notes

You may not have fully understood how to move the robot and use PyNiryo and that is totally fine because you will find more details on the next examples page!

The important thing to remember from this page is how to import the library, connect to the robot & call functions.

Examples: Movement

This document shows how to control Ned in order to make Move Joints & Move Pose.

If you want to see more, you can look at PyNiryo - Joints & Pose (index.html#joints-pose)

Important

In the following sections, you are supposed to be already connected to a calibrated robot. The robot's instance is saved in the variable robot . To know how to do so, go look at section Examples: Basics (index.html#document-source/examples/examples_basics).

Danger

If you are using the real robot, make sure the environment around it is clear.

Joints

Move Joints

To make a moveJ, you can either provide:

- 6 floats : j1, j2, j3, j4, j5, j6
- a list of 6 floats : [j1, j2, j3, j4, j5, j6]

It is possible to provide these parameters to the function **move_joints()** (index.html#api.tcp_client.NiryoRobot.move_joints) or via the joints setter, at your convenience:

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
# Moving Joints with function & 6 floats
robot.move_joints(0.0, 0.0, 0.0, 0.0, 0.0, 0.0)

# Moving Joints with function & a list of floats
robot.move_joints([-0.5, -0.6, 0.0, 0.3, 0.0, 0.0])

# Moving Joints with setter & 6 floats
robot.joints = 0.2, -0.4, 0.0, 0.0, 0.0, 0.0

# Moving Joints with setter & a list of floats
robot.joints = [-0.2, 0.3, 0.2, 0.3, -0.6, 0.0]
```

You should note that these 4 commands are doing exactly the same thing! In your future scripts, chose the one you prefer, but try to remain consistent to keep a good readability.

Get Joints

To get actual joint positions, you can use the function **get_joints()** (index.html#api.tcp_client.NiryoRobot.get_joints) or the `joints` getter. Both will return a list of the 6 joints position:

```
# Getting Joints with function
joints_read = robot.get_joints()

# Getting Joints with getter
joints_read = robot.joints
```

Hint

As we are developing in Python, we can unpack list very easily, which means that we can retrieve joints value in 6 variables by writing `j1, j2, j3, j4, j5, j6 = robot.get_joints()` .

Pose

Move Pose

To perform a moveP, you can provide:

- 6 floats : x, y, z, roll, pitch, yaw
- a list of 6 floats : [x, y, z, roll, pitch, yaw]
- a **PoseObject** (index.html#api.objects.PoseObject)

As for MoveJ, it is possible to provide these parameters to the function **move_pose()** (index.html#api.tcp_client.NiryoRobot.move_pose) or the `pose` setter, at your convenience:

```
pose_target = [0.2, 0.0, 0.2, 0.0, 0.0, 0.0]
pose_target_obj = PoseObject(0.2, 0.0, 0.2, 0.0, 0.0, 0.0)

# Moving Pose with function
robot.move_pose(0.2, 0.0, 0.2, 0.0, 0.0, 0.0)
robot.move_pose(pose_target)
robot.move_pose(pose_target_obj)

# Moving Pose with setter
robot.pose = (0.2, 0.0, 0.2, 0.0, 0.0, 0.0)
robot.pose = pose_target
robot.pose = pose_target_obj
```

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Each of these 6 commands are doing the same thing.

Get Pose

To get end effector actual pose, you can use the function **get_pose()** (index.html#api.tcp_client.NiryoRobot.get_pose) or the pose getter. Both will return a **PoseObject** (index.html#api.objects.PoseObject):

```
# Getting Joints with function
pose_read = robot.get_pose()

# Getting Joints with getter
pose_read = robot.pose
```

How to use the PoseObject

The **PoseObject** (index.html#api.objects.PoseObject) is a Python object which allows to store all poses' 6 coordinates (x, y, z, roll, pitch, yaw) in one single instance. It can be converted into a list if needed with the method **to_list()** (index.html#api.objects.PoseObject.to_list).

It also allows to create new **PoseObject** (index.html#api.objects.PoseObject) with some offset, much easier than copying list and editing only 1 or 2 values. For instance, imagine that we want to shift the place pose by 5 centimeters at each iteration of a loop, you can use the **copy_with_offsets()** (index.html#api.objects.PoseObject.copy_with_offsets) method:

```
pick_pose = PoseObject(
x=0.30, y=0.0, z=0.15,
roll=0, pitch=1.57, yaw=0.0
)
first_place_pose = PoseObject(
   x=0.0, y=0.2, z=0.15,
   roll=0, pitch=1.57, yaw=0.0
)
for i in range(5):
   robot.move_pose(pick_pose)
   new_place_pose = first_place_pose.copy_with_offsets(x_offset=0.05 * i)
   robot.move_pose(new_place_pose)
```

Examples: Tool action

This page shows how to control Ned's tools.

If you want to see more, you can look at PyNiryo - Tools (index.html#tools)

Important

In this section, you are already supposed to be connected to a calibrated robot. The robot instance is saved in the variable robot .

Danger

If you are using the real robot, make sure the environment around it is clear.

Tool control

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Equip Tool

In order to use a tool, it should be plugged mechanically to the robot but also connected to the software wise.

To do that, we should use the function **update_tool()** (index.html#api.tcp_client.NiryoRobot.update_tool) which takes no argument. It will scan motor's connections and set the new tool!

The line to equip a new tool is:

```
robot.update_tool()
```

Note

For the Grasping and Releasing sections, this command should be added in your codes! If you want to use a specific tool, you need to store the **ToolID** you are using in a variable named tool_used .

Grasping

To grasp with any tool, you can use the function **grasp_with_tool()** (index.html#api.tcp_client.NiryoRobot.grasp_with_tool). This action corresponds to:

- Close gripper for Grippers
- Pull Air for Vacuum Pump
- Activate for Electromagnet

The line to grasp is:

```
robot.grasp_with_tool()
```

To grasp an object by specifying the tool:

```
if tool_used in [ToolID.GRIPPER_1, ToolID.GRIPPER_2, ToolID.GRIPPER_3]:
    robot.close_gripper(speed=500)
elif tool_used == ToolID.ELECTROMAGNET_1:
    pin_electromagnet = PinID.XXX
    robot.setup_electromagnet(pin_electromagnet)
    robot.activate_electromagnet(pin_electromagnet)
elif tool_used == ToolID.VACUUM_PUMP_1:
    robot.pull_air_vacuum_pump()
```

Releasing

To release with any tool, you can use the function **release_with_tool()** (index.html#api.tcp_client.NiryoRobot.release_with_tool). This action corresponds to:

- Open gripper for Grippers
- Push Air for Vacuum pump
- Deactivate for Electromagnet

To release an object by specifying parameters:

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
if tool_used in [ToolID.GRIPPER_1, ToolID.GRIPPER_2, ToolID.GRIPPER_3]:
    robot.open_gripper(speed=500)
elif tool_used == ToolID.ELECTROMAGNET_1:
    pin_electromagnet = PinID.XXX
    robot.setup_electromagnet(pin_electromagnet)
    robot.deactivate_electromagnet(pin_electromagnet)
elif tool_used == ToolID.VACUUM_PUMP_1:
    robot.push_air_vacuum_pump()
```

Pick & Place with tools

A Pick & Place is an action which consists in going to a certain pose in order to pick an object and then, going to another pose to place it.

This operation can be proceed as follows:

1. Going over the object with a certain offset to avoid collision;
2. Going down to the object's height;
3. Grasping with tool;
4. Going back to step 1's pose;
5. Going over the place pose with a certain offset to avoid collision;
6. Going down to place's height;
7. Releasing the object with tool;
8. Going back to step 5's pose.

There are plenty of ways to perform a pick and place with PyNiryo. Methods will be presented from the lowest to highest level.

Code Baseline

For the sake of brevity, every piece of code beside the Pick & Place function will not be rewritten for every method. So that, you will need to use the code and implement the Pick & Place function to it.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
# Imports
from pyniryo import *

tool_used = ToolID.XXX  # Tool used for picking
robot_ip_address = "x.x.x.x" # Robot address

# The pick pose
pick_pose = PoseObject(
    x=0.25, y=0., z=0.15,
    roll=-0.0, pitch=1.57, yaw=0.0,
)
# The Place pose
place_pose = PoseObject(
    x=0.0, y=-0.25, z=0.1,
    roll=0.0, pitch=1.57, yaw=-1.57)

def pick_n_place_version_x(robot):
    # -- ---------- -- #
    # -- CODE GOES HERE -- #
    # -- ---------- -- #

if __name__ == '__main__':
    # Connect to robot
    client = NiryoRobot(robot_ip_address)
    # Calibrate robot if robot needs calibration
    client.calibrate_auto()
    # Changing tool
    client.update_tool()

    pick_n_place_version_x(client)

    # Releasing connection
    client.close_connection()
```

First Solution: the heaviest

For this first function, every steps are done by hand, as well as poses computing.

Note

In this example, the tool used is a Gripper. If you want to use another tool than a gripper, do not forget to adapt grasp & release functions!

```
def pick_n_place_version_1(robot):
    height_offset = 0.05  # Offset according to Z-Axis to go over pick & place poses
    gripper_speed = 400

    # Going Over Object
    robot.move_pose(pick_pose.x, pick_pose.y, pick_pose.z + height_offset,
                    pick_pose.roll, pick_pose.pitch, pick_pose.yaw)
    # Opening Gripper
    robot.open_gripper(gripper_speed)
    # Going to picking place and closing gripper
    robot.move_pose(pick_pose)
    robot.close_gripper(gripper_speed)

    # Raising
    robot.move_pose(pick_pose.x, pick_pose.y, pick_pose.z + height_offset,
                    pick_pose.roll, pick_pose.pitch, pick_pose.yaw)

    # Going Over Place pose
    robot.move_pose(place_pose.x, place_pose.y, place_pose.z + height_offset,
                    place_pose.roll, place_pose.pitch, place_pose.yaw)
    # Going to Place pose
    robot.move_pose(place_pose)
    # Opening Gripper
    robot.open_gripper(gripper_speed)
    # Raising
    robot.move_pose(place_pose.x, place_pose.y, place_pose.z + height_offset,
                    place_pose.roll, place_pose.pitch, place_pose.yaw)
```

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Second Solution: Use of PoseObject

For the second solution, we use a **PoseObject** (index.html#api.objects.PoseObject) in order to calculate approach poses more easily.

> **Note**
>
> To see more about **PoseObject** (index.html#api.objects.PoseObject), go look at PoseObject dedicated section (index.html#how-to-use-the-poseobject)

```
def pick_n_place_version_2(robot):
    height_offset = 0.05  # Offset according to Z-Axis to go over pick & place poses

    pick_pose_high = pick_pose.copy_with_offsets(z_offset=height_offset)
    place_pose_high = place_pose.copy_with_offsets(z_offset=height_offset)

    # Going Over Object
    robot.move_pose(pick_pose_high)
    # Opening Gripper
    robot.release_with_tool()
    # Going to picking place and closing gripper
    robot.move_pose(pick_pose)
    robot.grasp_with_tool()
    # Raising
    robot.move_pose(pick_pose_high)

    # Going Over Place pose
    robot.move_pose(place_pose_high)
    # Going to Place pose
    robot.move_pose(place_pose)
    # Opening Gripper
    robot.release_with_tool(gripper_speed)
    # Raising
    robot.move_pose(place_pose_high)
```

Third Solution: Pick from pose & Place from pose functions

For those who have already seen the API Documentation, you may have seen pick & place dedicated functions!

In this example, we use **pick_from_pose()** (index.html#api.tcp_client.NiryoRobot.pick_from_pose) and **place_from_pose()** (index.html#api.tcp_client.NiryoRobot.place_from_pose) in order to split our function in only 2 commands!

```
def pick_n_place_version_3(robot):
    # Pick
    robot.pick_from_pose(pick_pose)
    # Place
    robot.place_from_pose(place_pose)
```

Fourth Solution: All in one

The example exposed in the previous section could be useful if you want to do an action between the pick & the place phases.

For those who want to do everything in one command, you can use the **pick_and_place()** (index.html#api.tcp_client.NiryoRobot.pick_and_place) function!

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
def pick_n_place_version_4(robot):
    # Pick & Place
    robot.pick_and_place(pick_pose, place_pose)
```

Examples: Conveyor Belt

This document shows how to use Ned's Conveyor Belt.

If you want to see more about Ned's Conveyor Belt functions, you can look at PyNiryo - Conveyor (index.html#conveyor)

Danger

If you are using the real robot, make sure the environment around it is clear.

Simple Conveyor control

This short example shows how to connect a conveyor and launch its motor (control it by setting its speed and direction):

```
from pyniryo import *

# Connecting to robot
robot = NiryoRobot(<robot_ip_address>)

# Activating connexion with Conveyor Belt
conveyor_id = robot.set_conveyor()

# Running the Conveyor Belt at 50% of its maximum speed, in forward direction
robot.run_conveyor(conveyor_id, speed=50, direction=ConveyorDirection.FORWARD)

# Waiting 3 seconds
robot.wait(3)

# Stopping robot's motor
robot.stop_conveyor(conveyor_id)

# Deactivating connexion with the Conveyor Belt
robot.unset_conveyor(conveyor_id)
```

Advanced Conveyor Belt control

This example shows how to do a certain amount of pick & place by using the Conveyor Belt with the infrared sensor:

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
from pyniryo import *

# -- Setting variables
sensor_pin_id = PinID.GPIO_1A

catch_nb = 5

# The pick pose
pick_pose = PoseObject(
   x=0.25, y=0., z=0.15,
   roll=-0., pitch=1.57, yaw=0.0,
)
# The Place pose
place_pose = PoseObject(
   x=0., y=-0.25, z=0.1,
   roll=0., pitch=1.57, yaw=-1.57)

# -- MAIN PROGRAM

# Connecting to the robot
robot = NiryoRobot(<robot_ip_address>)

# Activating connexion with the Conveyor Belt
conveyor_id = robot.set_conveyor()

for i in range(catch_nb):
   robot.run_conveyor(conveyor_id)
   while robot.digital_read(sensor_pin_id) == PinState.LOW:
      robot.wait(0.1)

   # Stopping robot's motor
   robot.stop_conveyor(conveyor_id)
   # Making a pick & place
   robot.pick_and_place(pick_pose, place_pose)

# Deactivating connexion with the Conveyor Belt
robot.unset_conveyor(conveyor_id)
```

Examples: Vision

This page shows how to use Ned's Vision Set.

If you want to see more about Ned's Vision functions, you can look at PyNiryo - Vision
If you want to see how to do Image Processing, go check out theImage Processing section.

Note

Even if you do not own a Vision Set, you can still realize these examples with the Gazebo simulation version.

Danger

If you are using the real robot, make sure the environment around it is clear.

Needed piece of code

Important

In order to achieve the following examples, you need to create a vision workspace. In this page, the workspace used is named workspace_1 . To create it, the user should go on Niryo Studio!

As the examples start always the same, add the following lines at the beginning of codes:

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
# Imports
from pyniryo import *

# - Constants
workspace_name = "workspace_1"  # Robot's Workspace Name
robot_ip_address = "x.x.x.x"

# The pose from where the image processing happens
observation_pose = PoseObject(
    x=0.16, y=0.0, z=0.35,
    roll=0.0, pitch=1.57, yaw=0.0,
)
# Place pose
place_pose = PoseObject(
    x=0.0, y=-0.2, z=0.12,
    roll=0.0, pitch=1.57, yaw=-1.57
)

# - Initialization

# Connect to robot
robot = NiryoRobot(robot_ip_address)
# Calibrate robot if the robot needs calibration
robot.calibrate_auto()
# Updating tool
robot.update_tool()

# --- -------------- --- #
# --- CODE GOES HERE --- #
# --- -------------- --- #

robot.close_connection()
```

Hint

All the following examples are only a part of what can be made with the API in terms of Vision. We advise you to look at API - Vision (index.html#vision) to understand more deeply

Simple Vision Pick & Place

The goal of a Vision Pick & Place is the same as a classical Pick & Place, with a close difference: the camera detects where the robot has to go in order to pick!

This short example shows how to do your first Vision pick using the **vision_pick()** (index.html#api.tcp_client.NiryoRobot.vision_pick) function:

```
robot.move_pose(observation_pose)
# Trying to pick target using camera
obj_found, shape_ret, color_ret = robot.vision_pick(workspace_name)
if obj_found:
    robot.place_from_pose(place_pose)

robot.set_learning_mode(True)
```

Code Details - Simple Vision Pick and Place

To execute a Vision pick, we firstly need to go to a place where the robot will be able to see the workspace:

```
robot.move_pose(observation_pose)
```

Then, we try to perform a Vision pick in the workspace with the **vision_pick()** (index.html#api.tcp_client.NiryoRobot.vision_pick) function:

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
obj_found, shape_ret, color_ret = robot.vision_pick(workspace_name)
```

Variables shape_ret and color_ret are respectively of type **ObjectShape** and **ObjectColor**, and store the shape and the color of the detected object! We will not use them for this first example.

The obj_found variable is a boolean which indicates whereas an object has been found and picked, or not. Thus, if the pick worked, we can place the object at the place pose.

```
if obj_found:
    robot.place_from_pose(place_pose)
```

Finally, we turn learning mode on:

```
robot.set_learning_mode(True)
```

Note

If your obj_found variable indicates False, check that:

- Nothing obstructs the camera field of view
- Workspace's 4 markers are visible
- At least 1 object is placed fully inside the workspace

First conditioning via Vision

In most of use cases, the robot will need to perform more than one Pick & Place. In this example, we will see how to condition multiple objects according to a straight line:

```
# Initializing variables
offset_size = 0.05
max_catch_count = 4

# Loop until enough objects have been caught
catch_count = 0
while catch_count < max_catch_count:
    # Moving to observation pose
    robot.move_pose(observation_pose)

    # Trying to get object via Vision Pick
    obj_found, shape, color = robot.vision_pick(workspace_name)
    if not obj_found:
        robot.wait(0.1)
        continue

    # Calculate place pose and going to place the object
    next_place_pose = place_pose.copy_with_offsets(x_offset=catch_count * offset_size)
    robot.place_from_pose(next_place_pose)

    catch_count += 1

robot.go_to_sleep()
```

Code Details - First Conditioning via Vision

We want to catch max_catch_count objects, and space each of them by offset_size meter:

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
offset_size = 0.05
max_catch_count = 4
```

We start a loop until the robot has caught `max_catch_count` objects:

```
catch_count = 0
while catch_count < max_catch_count:
```

For each iteration, we firstly go to the observation pose and then, try to make a Vision pick in the workspace:

```
robot.move_pose(observation_pose)

obj_found, shape, color = robot.vision_pick(workspace_name)
```

If the Vision pick failed, we wait 0.1 second and then, start a new iteration:

```
if not obj_found:
    robot.wait(0.1)
    continue
```

Else, we compute the new place position according to the number of catches, and then, go placing the object at that place:

```
next_place_pose = place_pose.copy_with_offsets(x_offset=catch_count * offset_size)
robot.place_from_pose(next_place_pose)
```

We also increment the `catch_count` variable:

```
catch_count += 1
```

Once the target catch number is achieved, we go to sleep:

```
robot.go_to_sleep()
```

Multi Reference Conditioning

During a conditioning task, objects may not always be placed as the same place according to their type. In this example, we will see how to align object according to their color, using the color element **`ObjectColor`** returned by **`vision_pick()`** (index.html#api.tcp_client.NiryoRobot.vision_pick) function:

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
# Distance between elements
offset_size = 0.05
max_failure_count = 3

# Dict to write catch history
count_dict = {
    ObjectColor.BLUE: 0,
    ObjectColor.RED: 0,
    ObjectColor.GREEN: 0,
}

try_without_success = 0
# Loop until too much failures
while try_without_success < max_failure_count:
    # Moving to observation pose
    robot.move_pose(observation_pose)
    # Trying to get object via Vision Pick
    obj_found, shape, color = robot.vision_pick(workspace_name)
    if not obj_found:
        try_without_success += 1
        robot.wait(0.1)
        continue

    # Choose X position according to how the color line is filled
    offset_x_ind = count_dict[color]

    # Choose Y position according to ObjectColor
    if color == ObjectColor.BLUE:
        offset_y_ind = -1
    elif color == ObjectColor.RED:
        offset_y_ind = 0
    else:
        offset_y_ind = 1

    # Going to place the object
    next_place_pose = place_pose.copy_with_offsets(x_offset=offset_x_ind * offset_size,
                                                   y_offset=offset_y_ind * offset_size)
    robot.place_from_pose(next_place_pose)

    # Increment count
    count_dict[color] += 1
    try_without_success = 0

robot.go_to_sleep()
```

Code Details - Multi Reference Conditioning

We want to catch objects until Vision Pick failed `max_failure_count` times. Each of the object will be put on a specific column according to its color. The number of catches for each color will be stored on a dictionary `count_dict` .

```
# Distance between elements
offset_size = 0.05
max_failure_count = 3

# Dict to write catch history
count_dict = {
    ObjectColor.BLUE: 0,
    ObjectColor.RED: 0,
    ObjectColor.GREEN: 0,
}

try_without_success = 0
# Loop until too much failures
while try_without_success < max_failure_count:
```

For each iteration, we firstly go to the observation pose and then, try to make a Vision pick in the workspace:

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
robot.move_pose(observation_pose)

obj_found, shape, color = robot.vision_pick(workspace_name)
```

If the Vision pick failed, we wait 0.1 second and then, start a new iteration, without forgetting to increment the failure counter:

```
if not obj_found:
    try_without_success += 1
    robot.wait(0.1)
    continue
```

Else, we compute the new place position according to the number of catches, and then, go place the object at that place:

```
# Choose X position according to how the color line is filled
offset_x_ind = count_dict[color]

# Choose Y position according to ObjectColor
if color == ObjectColor.BLUE:
    offset_y_ind = -1
elif color == ObjectColor.RED:
    offset_y_ind = 0
else:
    offset_y_ind = 1

# Going to place the object
next_place_pose = place_pose.copy_with_offsets(x_offset=offset_x_ind * offset_size,
                                               y_offset=offset_y_ind * offset_size)
robot.place_from_pose(next_place_pose)
```

We increment the `count_dict` dictionary and reset `try_without_success` :

```
count_dict[color] += 1
try_without_success = 0
```

Once the target catch number is achieved, we go to sleep:

```
robot.go_to_sleep()
```

Sorting Pick with Conveyor

An interesting way to bring objects to the robot, is the use of a Conveyor Belt. In this examples, we will see how to catch only a certain type of object by stopping the conveyor as soon as the object is detected on the workspace.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
# Initializing variables
offset_size = 0.05
max_catch_count = 4
shape_expected = ObjectShape.CIRCLE
color_expected = ObjectColor.RED

conveyor_id = robot.set_conveyor()

catch_count = 0
while catch_count < max_catch_count:
    # Turning conveyor on
    robot.run_conveyor(conveyor_id)
    # Moving to observation pose
    robot.move_pose(observation_pose)
    # Check if object is in the workspace
    obj_found, pos_array, shape, color = robot.detect_object(workspace_name,
                                            shape=shape_expected,
                                            color=color_expected)
    if not obj_found:
        robot.wait(0.5)  # Wait to let the conveyor turn a bit
        continue
    # Stopping conveyor
    robot.stop_conveyor(conveyor_id)
    # Making a vision pick
    obj_found, shape, color = robot.vision_pick(workspace_name,
                                    shape=shape_expected,
                                    color=color_expected)
    if not obj_found:  # If visual pick did not work
        continue

    # Calculate place pose and going to place the object
    next_place_pose = place_pose.copy_with_offsets(x_offset=catch_count * offset_size)
    robot.place_from_pose(next_place_pose)

    catch_count += 1

# Stopping & unsetting conveyor
robot.stop_conveyor(conveyor_id)
robot.unset_conveyor(conveyor_id)

robot.go_to_sleep()
```

Code Details - Sort Picking

Firstly, we initialize your process: we want the robot to catch 4 red circles. To do so, we set variables `shape_expected` and `color_expected` with **ObjectShape.CIRCLE** and **ObjectColor.RED** .

```
offset_size = 0.05
max_catch_count = 4
shape_expected = ObjectShape.CIRCLE
color_expected = ObjectColor.RED
```

We activate the connection with the Conveyor Belt and start a loop until the robot has caught `max_catch_count` objects:

```
conveyor_id = robot.set_conveyor()

catch_count = 0
while catch_count < max_catch_count:
```

For each iteration, we firstly run the Conveyor Belt (if the latter is already running, nothing will happen), then go to the observation pose:

```
# Turning the Conveyor Belt on
robot.run_conveyor(conveyor_id)
# Moving to observation pose
robot.move_pose(observation_pose)
```

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

We then check if an object corresponding to our criteria is in the workspace. If not, we wait 0.5 second and then, start a new iteration:

```
obj_found, pos_array, shape, color = robot.detect_object(workspace_name,
                                          shape=shape_expected,
                                          color=color_expected)
if not obj_found:
    robot.wait(0.5)  # Wait to let the conveyor turn a bit
    continue
```

Else, stop the Conveyor Belt and try to make a Vision pick:

```
# Stopping Conveyor Belt
robot.stop_conveyor(conveyor_id)
# Making a Vision pick
obj_found, shape, color = robot.vision_pick(workspace_name,
                                shape=shape_expected,
                                color=color_expected)
if not obj_found:  # If visual pick did not work
    continue
```

If Vision Pick succeed, compute new place pose, and place the object:

```
# Calculate place pose and going to place the object
next_place_pose = place_pose.copy_with_offsets(x_offset=catch_count * offset_size)
robot.place_from_pose(next_place_pose)

catch_count += 1
```

Once the target catch number is achieved, we stop the Conveyor Belt and go to sleep:

```
# Stopping & unsetting Conveyor Belt
robot.stop_conveyor(conveyor_id)
robot.unset_conveyor(conveyor_id)

robot.go_to_sleep()
```

Code templates

As code structures are always the same, we wrote down few templates for you to start your code file with a good form.

The short template

Very simple, straightforward:

```
from pyniryo import *

# Connect to robot & calibrate
robot = NiryoRobot(<robot_ip_address>)
robot.calibrate_auto()

# --- --------- --- #
# --- YOUR CODE --- #
# --- --------- --- #

# Releasing connection
robot.close_connection()
```

Advanced template

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

This template let the user define his own process but it handles connection, calibration, tool equipping, and makes the robot go to sleep at the end:

```
from pyniryo import *

local_mode = False # Or True
tool_used = ToolID.GRIPPER_1
# Set robot address
robot_ip_address_rpi = "x.x.x.x"
robot_ip_address_local = "127.0.0.1"

robot_ip_address = robot_ip_address_local if local_mode else robot_ip_address_rpi


def process(niryo_edu):
    # --- --------- --- #
    # --- YOUR CODE --- #
    # --- --------- --- #

if __name__ == '__main__':
    # Connect to robot
    robot = NiryoRobot(robot_ip_address)
    # Calibrate robot if robot needs calibration
    robot.calibrate_auto()
    # Equip tool
    robot.update_tool()
    # Launching main process
    process(client)
    # Ending
    robot.go_to_sleep()
    # Releasing connection
    robot.close_connection()
```

Advanced template for Conveyor Belt

Same as Advanced template but with a Conveyor Belt

```
from pyniryo import *

# Set robot address
robot_ip_address = "x.x.x.x"


def process(robot, conveyor_id):
    robot.run_conveyor(conveyor_id)

    # --- --------- --- #
    # --- YOUR CODE --- #
    # --- --------- --- #

    robot.stop_conveyor()


if __name__ == '__main__':
    # Connect to robot
    robot = NiryoRobot(robot_ip_address)
    # Calibrate robot if robot needs calibration
    robot.calibrate_auto()
    # Equip tool
    robot.update_tool()
    # Activating connexion with conveyor
    conveyor_id = robot.set_conveyor()
    # Launching main process
    process(robot, conveyor_id)
    # Ending
    robot.go_to_sleep()
    # Deactivating connexion with conveyor
    robot.unset_conveyor(conveyor_id)
    # Releasing connection
    robot.close_connection()
```

Advanced template for Vision

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Huge template for Vision users!

```
from pyniryo import *

local_mode = False # Or True
workspace_name = "workspace_1"  # Robot's Workspace Name
# Set robot address
robot_ip_address_rpi = "x.x.x.x"
robot_ip_address_local = "127.0.0.1"

robot_ip_address = robot_ip_address_local if local_mode else robot_ip_address_rpi

# The pose from where the image processing happens
observation_pose = PoseObject(
    x=0.18, y=0.0, z=0.35,
    roll=0.0, pitch=1.57, yaw=-0.2,
)

# Center of the conditioning area
place_pose = PoseObject(
    x=0.0, y=-0.23, z=0.12,
    roll=0.0, pitch=1.57, yaw=-1.57
)

def process(robot):
    robot.move_pose(observation_pose)
    catch_count = 0
    while catch_count < 3:
        ret = robot.get_target_pose_from_cam(workspace_name,
                                             height_offset=0.0,
                                             shape=ObjectShape.ANY,
                                             color=ObjectColor.ANY)
        obj_found, obj_pose, shape, color = ret
        if not obj_found:
            continue
        catch_count += 1
        # --- --------- --- #
        # --- YOUR CODE --- #
        # --- --------- --- #
        robot.place_from_pose(place_pose)

if __name__ == '__main__':
    # Connect to robot
    robot = NiryoRobot(robot_ip_address)
    # Calibrate robot if robot needs calibration
    robot.calibrate_auto()
    # Equip tool
    robot.update_tool()
    # Launching main process
    process(client)
    # Ending
    robot.go_to_sleep()
    # Releasing connection
    robot.close_connection()
```

API Documentation

Master controls with PyNiryo with full the detailed functions here (index.html#document-source/api_doc/api).

Discover also Vision Functions (index.html#document-source/vision/image_processing_overview) to create your own image processing pipelines!

PyNiryo API Documentation

This file presents the different Command functions, Enums & Python object classes available with the API.

- Command functions are used to deal directly with the robot. It could be **move_joints()** , **get_hardware_status()** **vision_pick()** , or also **run_conveyor()**

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

- Enums are used to pass specific arguments to functions. For instance **PinState** , **ConveyorDirection** , ...
- Python object classes, as **PoseObject** , ease some operations

Command functions

This section references all existing functions to control your robot, which includes:

- Moving the robot
- Using Vision
- Controlling Conveyor Belts
- Playing with Hardware

All functions to control the robot are accessible via an instance of the class **NiryoRobot**

```
robot = NiryoRobot(<robot_ip_address>)
```

See examples on Examples: Basics (index.html#examples-basics)

List of functions subsections:

- TCP Connection
- Main purpose functions
- Joints & Pose
- Saved Poses
- Pick & Place
- Trajectories
- Tools
- Hardware
- Conveyor
- Vision

TCP Connection

NiryoRobot.connect(*ip_address*)

Connect to the TCP Server

Parameters:
ip_address (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – IP Address

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.close_connection()

Close connection with robot

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Main purpose functions

NiryoRobot.calibrate(*calibrate_mode*)

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Calibrate (manually or automatically) motors. Automatic calibration will do nothing if motors are already calibrated

Parameters:
calibrate_mode (*CalibrateMode*) – Auto or Manual

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.calibrate_auto()

Start a automatic motors calibration if motors are not calibrated yet

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.need_calibration()

Return a bool indicating whereas the robot motors need to be calibrate

Return type:
bool (https://docs.python.org/3/library/functions.html#bool)

NiryoRobot.get_learning_mode()

Get learning mode state

Returns:
True if learning mode is on

Return type:
bool (https://docs.python.org/3/library/functions.html#bool)

NiryoRobot.set_learning_mode(*enabled*)

Set learning mode if param is **True** , else turn it off

Parameters:
enabled (*bool* (https://docs.python.org/3/library/functions.html#bool)) – **True** or **False**

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.set_arm_max_velocity(*percentage_speed*)

Limit arm max velocity to a percentage of its maximum velocity

Parameters:
percentage_speed (*int* (https://docs.python.org/3/library/functions.html#int)) – Should be between 1 & 100

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.set_jog_control(*enabled*)

Set jog control mode if param is True, else turn it off

Parameters:
enabled (*bool* (https://docs.python.org/3/library/functions.html#bool)) – **True** or **False**

Return type:

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

None (https://docs.python.org/3/library/constants.html#None)

static **NiryoRobot.wait(*duration*)**

Wait for a certain time

Parameters:
duration (*float* (https://docs.python.org/3/library/functions.html#float)) – duration in seconds

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Joints & Pose

NiryoRobot.get_joints()

Get joints value in radians You can also use a getter

```
joints = robot.get_joints()
joints = robot.joints
```

Returns:
List of joints value

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]

NiryoRobot.get_pose()

Get end effector link pose as [x, y, z, roll, pitch, yaw]. x, y & z are expressed in meters / roll, pitch & yaw are expressed in radians You can also use a getter

```
pose = robot.get_pose()
pose = robot.pose
```

Return type:
PoseObject (index.html#api.objects.PoseObject)

NiryoRobot.get_pose_quat()

Get end effector link pose in Quaternion coordinates

Returns:
Position and quaternion coordinates concatenated in a list : [x, y, z, qx, qy, qz, qw]

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]

NiryoRobot.move_joints(args*)**

Move robot joints. Joints are expressed in radians.

All lines of the next example realize the same operation:

```
robot.joints = [0.2, 0.1, 0.3, 0.0, 0.5, 0.0]
robot.move_joints([0.2, 0.1, 0.3, 0.0, 0.5, 0.0])
robot.move_joints(0.2, 0.1, 0.3, 0.0, 0.5, 0.0)
```

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Parameters:
args (*Union[list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)], tuple (https://docs.python.org/3/library/stdtypes.html#tuple)[float (https://docs.python.org/3/library/functions.html#float)]]*) – either 6 args (1 for each joints) or a list of 6 joints

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.move_pose(**args*)

Move robot end effector pose to a (x, y, z, roll, pitch, yaw) pose. x, y & z are expressed in meters / roll, pitch & yaw are expressed in radians

All lines of the next example realize the same operation:

```
robot.pose = [0.2, 0.1, 0.3, 0.0, 0.5, 0.0]
robot.move_pose([0.2, 0.1, 0.3, 0.0, 0.5, 0.0])
robot.move_pose(0.2, 0.1, 0.3, 0.0, 0.5, 0.0)
robot.move_pose(PoseObject(0.2, 0.1, 0.3, 0.0, 0.5, 0.0))
```

Parameters:
args (*Union[tuple (https://docs.python.org/3/library/stdtypes.html#tuple)[float (https://docs.python.org/3/library/functions.html#float)], list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)], PoseObject (index.html#api.objects.PoseObject)]*) – either 6 args (1 for each coordinates) or a list of 6 coordinates or a **PoseObject**

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.move_linear_pose(**args*)

Move robot end effector pose to a (x, y, z, roll, pitch, yaw) pose with a linear trajectory

Parameters:
args (*Union[tuple (https://docs.python.org/3/library/stdtypes.html#tuple)[float (https://docs.python.org/3/library/functions.html#float)], list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)], PoseObject (index.html#api.objects.PoseObject)]*) – either 6 args (1 for each coordinates) or a list of 6 coordinates or a PoseObject

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.shift_pose(*axis*, *shift_value*)

Shift robot end effector pose along one axis

Parameters:
- **axis** (*RobotAxis*) – Axis along which the robot is shifted
- **shift_value** (*float* (https://docs.python.org/3/library/functions.html#float)) – In meter for X/Y/Z and radians for roll/pitch/yaw

Return type:
None (https://docs.python.org/3/library/constants.html#None)

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

NiryoRobot.shift_linear_pose(*axis*, *shift_value*)

Shift robot end effector pose along one axis, with a linear trajectory

Parameters:

- **axis** (*RobotAxis*) – Axis along which the robot is shifted
- **shift_value** (*float* (https://docs.python.org/3/library/functions.html#float)) – In meter for X/Y/Z and radians for roll/pitch/yaw

Return type:

None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.jog_joints(**args*)

Jog robot joints'. Jog corresponds to a shift without motion planning. Values are expressed in radians.

Parameters:

args (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*]]*) – either 6 args (1 for each joints) or a list of 6 joints offset

Return type:

None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.jog_pose(**args*)

Jog robot end effector pose Jog corresponds to a shift without motion planning Arguments are [dx, dy, dz, d_roll, d_pitch, d_yaw] dx, dy & dz are expressed in meters / d_roll, d_pitch & d_yaw are expressed in radians

Parameters:

args (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*]]*) – either 6 args (1 for each coordinates) or a list of 6 offset

Return type:

None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.move_to_home_pose()

Move to a position where the forearm lays on shoulder

Return type:

None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.go_to_sleep()

Go to home pose and activate learning mode

Return type:

None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.forward_kinematics(**args*)

Compute forward kinematics of a given joints configuration and give the associated spatial pose

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Parameters:
args (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*]]*) – either 6 args (1 for each joints) or a list of 6 joints

Return type:
PoseObject (index.html#api.objects.PoseObject)

NiryoRobot.inverse_kinematics(**args*)

Compute inverse kinematics

Parameters:
args (*Union[tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*], list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], PoseObject* (index.html#api.objects.PoseObject)*]*) – either 6 args (1 for each coordinates) or a list of 6 coordinates or a **PoseObject**

Returns:
List of joints value

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]

Saved Poses

NiryoRobot.get_pose_saved(*pose_name*)

Get pose saved in from Ned's memory

Parameters:
pose_name (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – Pose name in robot's memory

Returns:
Pose associated to pose_name

Return type:
PoseObject (index.html#api.objects.PoseObject)

NiryoRobot.save_pose(*pose_name*, **args*)

Save pose in robot's memory

Parameters:
args (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*], PoseObject* (index.html#api.objects.PoseObject)*]*) – either 6 args (1 for each coordinates) or a list of 6 coordinates or a PoseObject

Return type:
None (https://docs.python.org/3/library/constants.html#None)

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

NiryoRobot.delete_pose(*pose_name*)

Delete pose from robot's memory

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.get_saved_pose_list()

Get list of poses' name saved in robot memory

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[str (https://docs.python.org/3/library/stdtypes.html#str)]

Pick & Place

NiryoRobot.pick_from_pose(args*)**

Execute a picking from a pose.

A picking is described as :

* going over the object
* going down until height = z
* grasping with tool
* going back over the object

Parameters:
args (*Union[list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)], tuple (https://docs.python.org/3/library/stdtypes.html#tuple)[float (https://docs.python.org/3/library/functions.html#float)], PoseObject (index.html#api.objects.PoseObject)]*) – either 6 args (1 for each coordinates) or a list of 6 coordinates or a PoseObject

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.place_from_pose(args*)**

Execute a placing from a position.

A placing is described as :

* going over the place
* going down until height = z
* releasing the object with tool
* going back over the place

Parameters:
args (*Union[list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)], tuple (https://docs.python.org/3/library/stdtypes.html#tuple)[float (https://docs.python.org/3/library/functions.html#float)], PoseObject (index.html#api.objects.PoseObject)]*) – either 6 args (1 for each coordinates) or a list of 6 coordinates or a PoseObject

Return type:

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.pick_and_place(*pick_pose*, *place_pos*, *dist_smoothing=0.0*)

Execute a pick then a place

Parameters:

- **pick_pose** (*Union[list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)], PoseObject (index.html#api.objects.PoseObject)]*) – Pick Pose : [x, y, z, roll, pitch, yaw] or PoseObject
- **place_pos** (*Union[list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)], PoseObject (index.html#api.objects.PoseObject)]*) – Place Pose : [x, y, z, roll, pitch, yaw] or PoseObject
- **dist_smoothing** (*float (https://docs.python.org/3/library/functions.html#float)*) – Distance from waypoints before smoothing trajectory

Return type:

None (https://docs.python.org/3/library/constants.html#None)

Trajectories

NiryoRobot.get_trajectory_saved(*trajectory_name*)

Get trajectory saved in Ned's memory

Returns:

Trajectory

Return type:

list (https://docs.python.org/3/library/stdtypes.html#list)[list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]]

NiryoRobot.execute_trajectory_from_poses(*list_poses*, *dist_smoothing=0.0*)

Execute trajectory from list of poses

Parameters:

- **list_poses** (*list (https://docs.python.org/3/library/stdtypes.html#list)[list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]]*) – List of [x,y,z,qx,qy,qz,qw] or list of [x,y,z,roll,pitch,yaw]
- **dist_smoothing** (*float (https://docs.python.org/3/library/functions.html#float)*) – Distance from waypoints before smoothing trajectory

Return type:

None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.execute_trajectory_from_poses_and_joints(*list_pose_joints*, *list_type=None*, *dist_smoothing=0.0*)

Execute trajectory from list of poses and joints

Parameters:

- **list_pose_joints** (*list (https://docs.python.org/3/library/stdtypes.html#list)[list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]]*) – List of [x,y,z,qx,qy,qz,qw] or list of [x,y,z,roll,pitch,yaw] or a list of [j1,j2,j3,j4,j5,j6]

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

- **list_type** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[string]*) – List of string 'pose' or 'joint', or ['pose'] (if poses only) or ['joint'] (if joints only). If None, it is assumed there are only poses in the list.
- **dist_smoothing** (*float* (https://docs.python.org/3/library/functions.html#float)) – Distance from waypoints before smoothing trajectory

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.execute_trajectory_saved(*trajectory_name*)

Execute trajectory from Ned's memory

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.save_trajectory(*trajectory_name*, *list_poses*)

Save trajectory in robot memory

Parameters:
list_poses (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]]*) – List of [x,y,z,qx,qy,qz,qw] or list of [x,y,z,roll,pitch,yaw]

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.delete_trajectory(*trajectory_name*)

Delete trajectory from robot's memory

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.get_saved_trajectory_list()

Get list of trajectories' name saved in robot memory

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[str (https://docs.python.org/3/library/stdtypes.html#str)]

Tools

NiryoRobot.get_current_tool_id()

Get equipped tool Id

Return type:
ToolID

NiryoRobot.update_tool()

Update equipped tool

Return type:
None (https://docs.python.org/3/library/constants.html#None)

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

NiryoRobot.grasp_with_tool()

Grasp with tool | This action correspond to | - Close gripper for Grippers | - Pull Air for Vacuum pump | - Activate for Electromagnet

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.release_with_tool()

Release with tool | This action correspond to | - Open gripper for Grippers | - Push Air for Vacuum pump | - Deactivate for Electromagnet

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.open_gripper(*speed=500*)

Open gripper associated to 'gripper_id' with a speed 'speed'

Parameters:
speed (*int* (https://docs.python.org/3/library/functions.html#int)) – Between 100 & 1000

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.close_gripper(*speed=500*)

Close gripper associated to 'gripper_id' with a speed 'speed'

Parameters:
speed (*int* (https://docs.python.org/3/library/functions.html#int)) – Between 100 & 1000

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.pull_air_vacuum_pump()

Pull air of vacuum pump

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.push_air_vacuum_pump()

Push air of vacuum pump

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.setup_electromagnet(*pin_id*)

Setup electromagnet on pin

Parameters:
pin_id (*PinID*) –

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.activate_electromagnet(*pin_id*)

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Activate electromagnet associated to electromagnet_id on pin_id

Parameters:
pin_id (*PinID*) –

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.deactivate_electromagnet(*pin_id*)

Deactivate electromagnet associated to electromagnet_id on pin_id

Parameters:
pin_id (*PinID*) –

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.enable_tcp(*enable=True*)

Enables or disables the TCP function (Tool Center Point). If activation is requested, the last recorded TCP value will be applied. The default value depends on the gripper equipped. If deactivation is requested, the TCP will be coincident with the tool_link.

Parameters:
enable (*Bool*) – True to enable, False otherwise.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.set_tcp(**args*)

Activates the TCP function (Tool Center Point) and defines the transformation between the tool_link frame and the TCP frame.

Parameters:
args (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*], PoseObject* (index.html#api.objects.PoseObject)*]*) – either 6 args (1 for each coordinates) or a list of 6 coordinates or a PoseObject

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.reset_tcp()

Reset the TCP (Tool Center Point) transformation. The TCP will be reset according to the tool equipped.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.tool_reboot()

Reboot the motor of the tool equipped. Useful when an Overload error occurs. (cf HardwareStatus)

Return type:
None (https://docs.python.org/3/library/constants.html#None)

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Hardware

NiryoRobot.set_pin_mode(*pin_id*, *pin_mode*)

Set pin number pin_id to mode pin_mode

Parameters:
- **pin_id** (*PinID*) –
- **pin_mode** (*PinMode*) –

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.digital_write(*pin_id*, *digital_state*)

Set pin_id state to digital_state

Parameters:
- **pin_id** (*PinID*) –
- **digital_state** (*PinState*) –

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.digital_read(*pin_id*)

Read pin number pin_id and return its state

Parameters:
pin_id (*PinID*) –

Return type:
PinState

NiryoRobot.get_hardware_status()

Get hardware status : Temperature, Hardware version, motors names & types ...

Returns:
Infos contains in a HardwareStatusObject

Return type:
HardwareStatusObject (index.html#api.objects.HardwareStatusObject)

NiryoRobot.get_digital_io_state()

Get Digital IO state : Names, modes, states

Returns:
List of DigitalPinObject instance

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[DigitalPinObject (index.html#api.objects.DigitalPinObject)]

Conveyor

NiryoRobot.set_conveyor()

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Activate a new conveyor and return its ID

Returns:
New conveyor ID

Return type:
ConveyorID

NiryoRobot.unset_conveyor(*conveyor_id*)

Remove specific conveyor.

Parameters:
conveyor_id (*ConveyorID*) – Basically, ConveyorID.ONE or ConveyorID.TWO

NiryoRobot.run_conveyor(*conveyor_id*, *speed=50*, *direction=<ConveyorDirection.FORWARD: 1>*)

Run conveyor at id 'conveyor_id'

Parameters:
- **conveyor_id** (*ConveyorID*) –
- **speed** (*int* (https://docs.python.org/3/library/functions.html#int)) –
- **direction** (*ConveyorDirection*) –

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.stop_conveyor(*conveyor_id*)

Stop conveyor at id 'conveyor_id'

Parameters:
conveyor_id (*ConveyorID*) –

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.control_conveyor(*conveyor_id*, *control_on*, *speed*, *direction*)

Control conveyor at id 'conveyor_id'

Parameters:
- **conveyor_id** (*ConveyorID*) –
- **control_on** (*bool* (https://docs.python.org/3/library/functions.html#bool)) –
- **speed** (*int* (https://docs.python.org/3/library/functions.html#int)) – New speed which is a percentage of maximum speed
- **direction** (*ConveyorDirection*) – Conveyor direction

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.get_connected_conveyors_id()

Returns:
List of the connected conveyors' ID

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[ConveyorID]

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Vision

NiryoRobot.get_img_compressed()

Get image from video stream in a compressed format. Use **uncompress_image** from the vision package to uncompress it

Returns:
string containing a JPEG compressed image

Return type:
str (https://docs.python.org/3/library/stdtypes.html#str)

NiryoRobot.set_brightness(*brightness_factor*)

Modify video stream brightness

Parameters:
brightness_factor (*float* (https://docs.python.org/3/library/functions.html#float)) – How much to adjust the brightness. 0.5 will give a darkened image, 1 will give the original image while 2 will enhance the brightness by a factor of 2.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.set_contrast(*contrast_factor*)

Modify video stream contrast

Parameters:
contrast_factor (*float* (https://docs.python.org/3/library/functions.html#float)) – While a factor of 1 gives original image. Making the factor towards 0 makes the image greyer, while factor>1 increases the contrast of the image.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.set_saturation(*saturation_factor*)

Modify video stream saturation

Parameters:
saturation_factor (*float* (https://docs.python.org/3/library/functions.html#float)) – How much to adjust the saturation. 0 will give a black and white image, 1 will give the original image while 2 will enhance the saturation by a factor of 2.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.get_image_parameters()

Get last stream image parameters: Brightness factor, Contrast factor, Saturation factor.

Brightness factor: How much to adjust the brightness. 0.5 will give a darkened image, 1 will give the original image while 2 will enhance the brightness by a factor of 2.

Contrast factor: A factor of 1 gives original image. Making the factor towards 0 makes the image greyer, while factor>1 increases the contrast of the image.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Saturation factor: 0 will give a black and white image, 1 will give the original image while 2 will enhance the saturation by a factor of 2.

Returns:
Brightness factor, Contrast factor, Saturation factor

Return type:
float (https://docs.python.org/3/library/functions.html#float), float (https://docs.python.org/3/library/functions.html#float), float (https://docs.python.org/3/library/functions.html#float)

NiryoRobot.get_target_pose_from_rel(*workspace_name*, *height_offset*, *x_rel*, *y_rel*, *yaw_rel*)

Given a pose (x_rel, y_rel, yaw_rel) relative to a workspace, this function returns the robot pose in which the current tool will be able to pick an object at this pose.

The height_offset argument (in m) defines how high the tool will hover over the workspace. If height_offset = 0, the tool will nearly touch the workspace.

Parameters:
- **workspace_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of the workspace
- **height_offset** (*float* (https://docs.python.org/3/library/functions.html#float)) – offset between the workspace and the target height
- **x_rel** (*float* (https://docs.python.org/3/library/functions.html#float)) – x relative pose (between 0 and 1)
- **y_rel** (*float* (https://docs.python.org/3/library/functions.html#float)) – y relative pose (between 0 and 1)
- **yaw_rel** (*float* (https://docs.python.org/3/library/functions.html#float)) – Angle in radians

Returns:
target_pose

Return type:
PoseObject (index.html#api.objects.PoseObject)

NiryoRobot.get_target_pose_from_cam(*workspace_name*, *height_offset=0.0*, *shape=<ObjectShape.ANY: 'ANY'>*, *color=<ObjectColor.ANY: 'ANY'>*)

First detects the specified object using the camera and then returns the robot pose in which the object can be picked with the current tool

Parameters:
- **workspace_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of the workspace
- **height_offset** (*float* (https://docs.python.org/3/library/functions.html#float)) – offset between the workspace and the target height
- **shape** (*ObjectShape*) – shape of the target
- **color** (*ObjectColor*) – color of the target

Returns:
object_found, object_pose, object_shape, object_color

Return type:
(bool (https://docs.python.org/3/library/functions.html#bool), PoseObject (index.html#api.objects.PoseObject), ObjectShape, ObjectColor)

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

NiryoRobot.vision_pick(*workspace_name, height_offset=0.0, shape=<ObjectShape.ANY: 'ANY'>, color=<ObjectColor.ANY: 'ANY'>*)

Picks the specified object from the workspace. This function has multiple phases:

1. detect object using the camera
2. prepare the current tool for picking
3. approach the object
4. move down to the correct picking pose
5. actuate the current tool
6. lift the object

Example:

```
robot = NiryoRobot(ip_address="x.x.x.x")
robot.calibrate_auto()
robot.move_pose(<observation_pose>)
obj_found, shape_ret, color_ret = robot.vision_pick(<workspace_name>,
                                    height_offset=0.0,
                                    shape=ObjectShape.ANY,
                                    color=ObjectColor.ANY)
```

Parameters:

- **workspace_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of the workspace
- **height_offset** (*float* (https://docs.python.org/3/library/functions.html#float)) – offset between the workspace and the target height
- **shape** (*ObjectShape*) – shape of the target
- **color** (*ObjectColor*) – color of the target

Returns:
object_found, object_shape, object_color

Return type:
(bool (https://docs.python.org/3/library/functions.html#bool), ObjectShape, ObjectColor)

NiryoRobot.move_to_object(*workspace_name, height_offset, shape, color*)

Same as *get_target_pose_from_cam* but directly moves to this position

Parameters:

- **workspace_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of the workspace
- **height_offset** (*float* (https://docs.python.org/3/library/functions.html#float)) – offset between the workspace and the target height
- **shape** (*ObjectShape*) – shape of the target
- **color** (*ObjectColor*) – color of the target

Returns:
object_found, object_shape, object_color

Return type:
(bool (https://docs.python.org/3/library/functions.html#bool), ObjectShape, ObjectColor)

NiryoRobot.detect_object(*workspace_name, shape=<ObjectShape.ANY: 'ANY'>, color=<ObjectColor.ANY: 'ANY'>*)

Detect object in workspace and return its pose and characteristics

Parameters:

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

- **workspace_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of the workspace
- **shape** (*ObjectShape*) – shape of the target
- **color** (*ObjectColor*) – color of the target

Returns:
object_found, object_pose, object_shape, object_color

Return type:
(bool (https://docs.python.org/3/library/functions.html#bool), PoseObject (index.html#api.objects.PoseObject), str (https://docs.python.org/3/library/stdtypes.html#str), str (https://docs.python.org/3/library/stdtypes.html#str))

NiryoRobot.get_camera_intrinsics()

Get calibration object: camera intrinsics, distortions coefficients

Returns:
camera intrinsics, distortions coefficients

Return type:
(list (https://docs.python.org/3/library/stdtypes.html#list)[list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]] , list (https://docs.python.org/3/library/stdtypes.html#list)[list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]])

NiryoRobot.save_workspace_from_robot_poses(*workspace_name*, *pose_origin*, *pose_2*, *pose_3*, *pose_4*)

Save workspace by giving the poses of the robot to point its 4 corners with the calibration Tip. Corners should be in the good order. Markers' pose will be deduced from these poses

Poses should be either a list [x, y, z, roll, pitch, yaw] or a PoseObject

Parameters:
- **workspace_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – workspace name, maximum lenght 30 char.
- **pose_origin** (*Union[list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)], PoseObject (index.html#api.objects.PoseObject)]*) –
- **pose_2** (*Union[list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)], PoseObject (index.html#api.objects.PoseObject)]*) –
- **pose_3** (*Union[list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)], PoseObject (index.html#api.objects.PoseObject)]*) –
- **pose_4** (*Union[list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)], PoseObject (index.html#api.objects.PoseObject)]*) –

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.save_workspace_from_points(*workspace_name*, *point_origin*, *point_2*, *point_3*, *point_4*)

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Save workspace by giving the points of worskpace's 4 corners. Points are written as [x, y, z] Corners should be in the good order.

Parameters:

- **workspace_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – workspace name, maximum lenght 30 char.
- **point_origin** (*list* (https://docs.python.org/3/library/stdtypes.html#list)[*float* (https://docs.python.org/3/library/functions.html#float)]) –
- **point_2** (*list* (https://docs.python.org/3/library/stdtypes.html#list)[*float* (https://docs.python.org/3/library/functions.html#float)]) –
- **point_3** (*list* (https://docs.python.org/3/library/stdtypes.html#list)[*float* (https://docs.python.org/3/library/functions.html#float)]) –
- **point_4** (*list* (https://docs.python.org/3/library/stdtypes.html#list)[*float* (https://docs.python.org/3/library/functions.html#float)]) –

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.delete_workspace(*workspace_name*)

Delete workspace from robot's memory

Parameters:
workspace_name (*str* (https://docs.python.org/3/library/stdtypes.html#str)) –

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.get_workspace_ratio(*workspace_name*)

Get workspace ratio from robot's memory

Parameters:
workspace_name (*str* (https://docs.python.org/3/library/stdtypes.html#str)) –

Return type:
float (https://docs.python.org/3/library/functions.html#float)

NiryoRobot.get_workspace_list()

Get list of workspaces' name store in robot's memory

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[str (https://docs.python.org/3/library/stdtypes.html#str)]

Enums

Enums are used to pass specific parameters to functions.

For instance, **shift_pose()** will need a parameter from **RobotAxis** enum

```
robot.shift_pose(RobotAxis.Y, 0.15)
```

List of enums:

- **CalibrateMode**

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

- **RobotAxis**
- **ToolID**
- **PinMode**
- **PinState**
- **PinID**
- **ConveyorID**
- **ConveyorDirection**
- **ObjectColor**
- **ObjectShape**

class **CalibrateMode**(*value*)

Enumeration of Calibration Modes

AUTO= *0*

MANUAL= *1*

class **RobotAxis**(*value*)

Enumeration of Robot Axis : it used for Shift command

X= *0*

Y= *1*

Z= *2*

ROLL= *3*

PITCH= *4*

YAW= *5*

class **ToolID**(*value*)

Enumeration of Tools IDs

NONE= *0*

GRIPPER_1= *11*

GRIPPER_2= *12*

GRIPPER_3= *13*

ELECTROMAGNET_1= *30*

VACUUM_PUMP_1= *31*

class **PinMode**(*value*)

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Enumeration of Pin Modes

INPUT= *0*

OUTPUT= *1*

class **PinState**(*value*)

Pin State is either LOW or HIGH

LOW= *0*

HIGH= *1*

class **PinID**(*value*)

Enumeration of Robot Pins

GPIO_1A= *0*

GPIO_1B= *1*

GPIO_1C= *2*

GPIO_2A= *3*

GPIO_2B= *4*

GPIO_2C= *5*

class **ConveyorID**(*value*)

Enumeration of Conveyor IDs used for Conveyor control

NONE= *0*

ID_1= *12*

ID_2= *13*

class **ConveyorDirection**(*value*)

Enumeration of Conveyor Directions used for Conveyor control

FORWARD= *1*

BACKWARD= *-1*

class **ObjectColor**(*value*)

Enumeration of Colors available for image processing

RED= *'RED'*

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

BLUE= *'BLUE'*

GREEN= *'GREEN'*

ANY= *'ANY'*

class **ObjectShape**(*value*)

Enumeration of Shapes available for image processing

SQUARE= *'SQUARE'*

CIRCLE= *'CIRCLE'*

ANY= *'ANY'*

Python object classes

Special objects

class **PoseObject**(*x*, *y*, *z*, *roll*, *pitch*, *yaw*)

Pose object which stores x, y, z, roll, pitch & yaw parameters

copy_with_offsets(*x_offset=0.0*, *y_offset=0.0*, *z_offset=0.0*, *roll_offset=0.0*, *pitch_offset=0.0*, *yaw_offset=0.0*)

Create a new pose from copying from copying actual pose with offsets

Return type:
PoseObject (index.html#api.objects.PoseObject)

to_list()

Return a list [x, y, z, roll, pitch, yaw] corresponding to the pose's parameters

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]

class **HardwareStatusObject**(*rpi_temperature*, *hardware_version*, *connection_up*, *error_message*, *calibration_needed*, *calibration_in_progress*, *motor_names*, *motor_types*, *motors_temperature*, *motors_voltage*, *hardware_errors*)

Object used to store every hardware information

class **DigitalPinObject**(*pin_id*, *name*, *mode*, *state*)

Object used to store information on digital pins

Start with Image Processing

Discover how to create your own image processing pipelines!

Overview & examples

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

This file illustrates few image processing pipeline using vision module from niryo_edu package. This module is based in OpenCV (https://opencv.org/) and its functions are detailed in Functions documentation (index.html#document-source/vision/image_processing_api).

The package niryo_edu comes up with the **vision** module which contains image processing functions including thresholding, blob detection, ...
To use it, add to your imports `from pyniryo.vision import *` .

> **Note**
>
> It is also possible to merge both import lines by using `from pyniryo import *` .

Play with Robot video stream

We are firstly going to take a look at robot's functions which can be find at API - Vision (index.html#vision)

Get & display image from stream

Ned can share its video stream through TCP. As sending raw images will lead to heavy packets which can saturate the network, it sends compressed images. You access it through the robot's function: **get_img_compressed()** (index.html#api.tcp_client.NiryoRobot.get_img_compressed). Once your image is received, you firstly need to uncompress via **uncompress_image()** (index.html#vision.image_functions.uncompress_image) and you can then display it with **show_img_and_wait_close()** (index.html#vision.image_functions.show_img_and_wait_close).

```
from pyniryo import *

# Connecting to robot
robot = NiryoRobot("10.10.10.10")

# Getting image
img_compressed = robot.get_img_compressed()
# Uncompressing image
img = uncompress_image(img_compressed)

# Displaying
show_img_and_wait_close("img_stream", img)
```

> **Note**
>
> **show_img_and_wait_close()** (index.html#vision.image_functions.show_img_and_wait_close) will wait for the user to press either *Q* or *Esc* key, before closing the window.

Undistort and display video stream

In this section, we are going to display the raw video stream & the undistorted video stream.

As Ned's camera is passing raw images to the robot, these images are distorted due to the camera lens. In order to undistort them, we need to use Ned's camera intrinsics.

To undistort the raw image, we use **undistort_image()** (index.html#vision.image_functions.undistort_image) which needs to be called with the parameters given by Ned through **get_camera_intrinsics()** (index.html#api.tcp_client.NiryoRobot.get_camera_intrinsics).

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Once, we have both raw & undistorted images, we can concatenate them in order to display them in once with **concat_imgs()** (index.html#vision.image_functions.concat_imgs). Finally, we display the image **show_img()** (index.html#vision.image_functions.show_img).

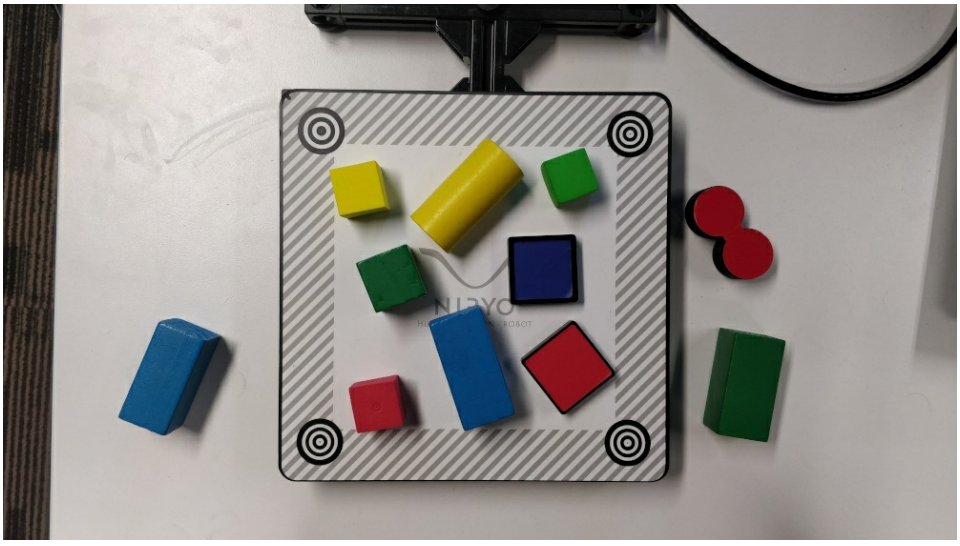
```
from pyniryo import *

observation_pose = PoseObject(
    x=0.18, y=0.0, z=0.35,
    roll=0.0, pitch=1.57, yaw=-0.2,
)

# Connecting to robot
robot = NiryoRobot("10.10.10.10")
robot.calibrate_auto()

# Getting calibration param
mtx, dist = robot.get_camera_intrinsics()
# Moving to observation pose
robot.move_pose(observation_pose)

while "User do not press Escape neither Q":
    # Getting image
    img_compressed = robot.get_img_compressed()
    # Uncompressing image
    img_raw = uncompress_image(img_compressed)
    # Undistorting
    img_undistort = undistort_image(img_raw, mtx, dist)

    # - Display
    # Concatenating raw image and undistorted image
    concat_ims = concat_imgs((img_raw, img_undistort))

    # Showing images
    key = show_img("Images raw & undistorted", concat_ims, wait_ms=30)
    if key in [27, ord("q")]:  # Will break loop if the user press Escape or Q
        break
```

Note

To see more about camera distortion/undistortion, go on OpenCV Documentation about Camera Calibration (https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html).

Pure image processing functions

In order to illustrate functions, we are going to use the following image.



Attention

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

In this section it is supposed that:

- You have imported pyniryo.vision
- The variable img is containing the image on which image processing is applied

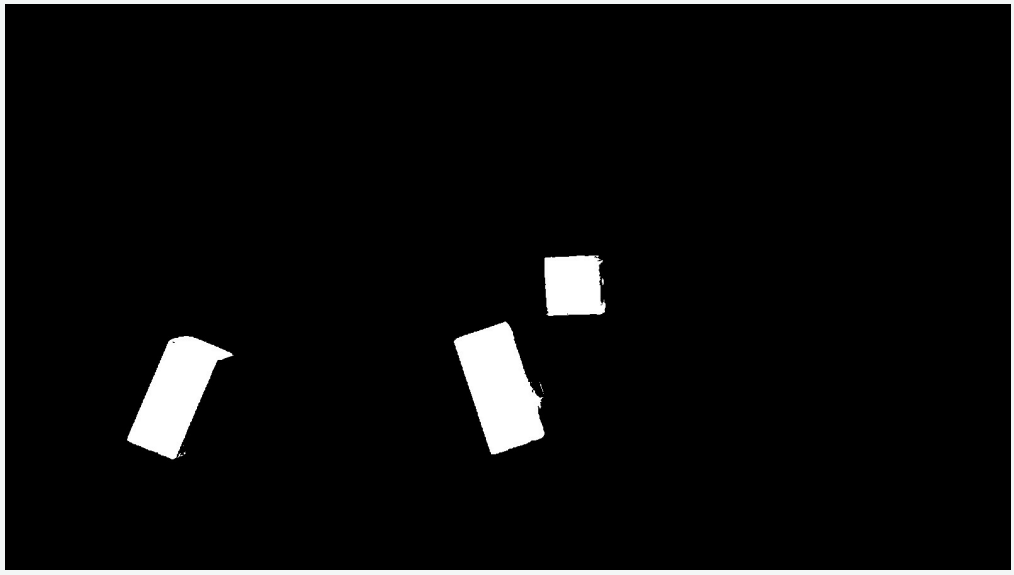
Color thresholding

Color thresholding is very useful in order to detect object with an uniform color. The implemented function to realize this operation is **threshold_hsv()** (index.html#vision.image_functions.threshold_hsv).

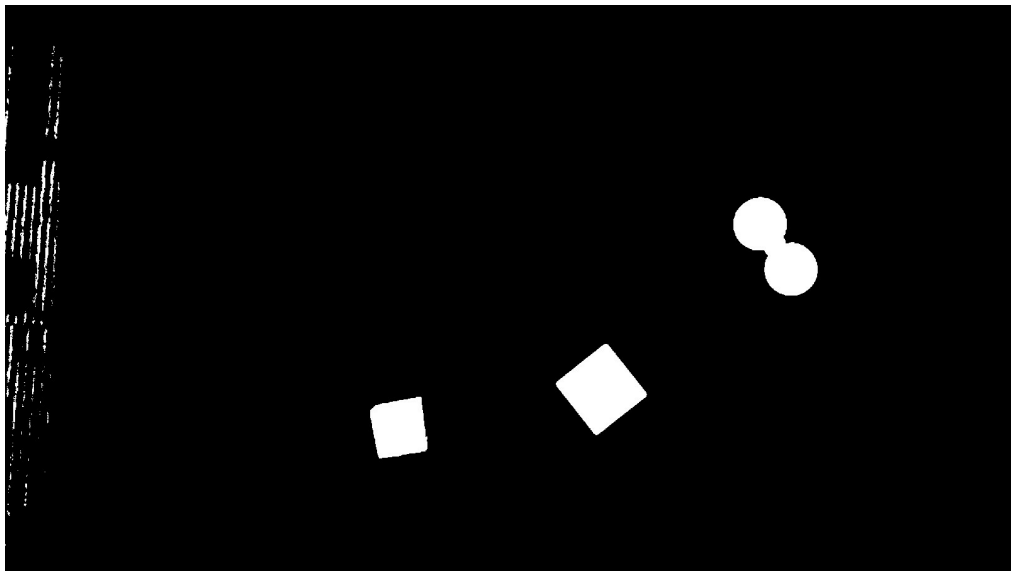
The following code is using parameters from **ColorHSV** (index.html#vision.enums.ColorHSV) enum in order to threshold Red features & *hand made* parameters to extract Blue:

```
img_threshold_red = threshold_hsv(img_test, *ColorHSV.RED.value)

blue_min_hsv = [90, 85, 70]
blue_max_hsv = [125, 255, 255]

img_threshold_blue = threshold_hsv(img_test, list_min_hsv=blue_min_hsv,
                                   list_max_hsv=blue_max_hsv, reverse_hue=False)

show_img("img_threshold_red", img_threshold_red)

show_img_and_wait_close("img_threshold_blue", img_threshold_blue)
```

Images result

Thresh color	Image result
Blue	

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Thresh color	Image result
Red	

Morphological transformations

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. It needs two inputs, one is our original image, second one is called structuring element or kernel which decides the nature of operation. Two basic morphological operators are Erosion (https://en.wikipedia.org/wiki/Mathematical_morphology#Erosion) and Dilation (https://en.wikipedia.org/wiki/Mathematical_morphology#Dilation).

Then its variant forms like Opening (https://en.wikipedia.org/wiki/Mathematical_morphology#Opening), Closing (https://en.wikipedia.org/wiki/Mathematical_morphology#Closing) also comes into play. Learn more on Wikipedia page (https://en.wikipedia.org/wiki/Mathematical_morphology).

The implemented function to realize these operations is **morphological_transformations()** (index.html#vision.image_functions.morphological_transformations). It uses **MorphoType** (index.html#vision.enums.MorphoType) and **KernelType** (index.html#vision.enums.KernelType) to determine which operation should be applied on the image.

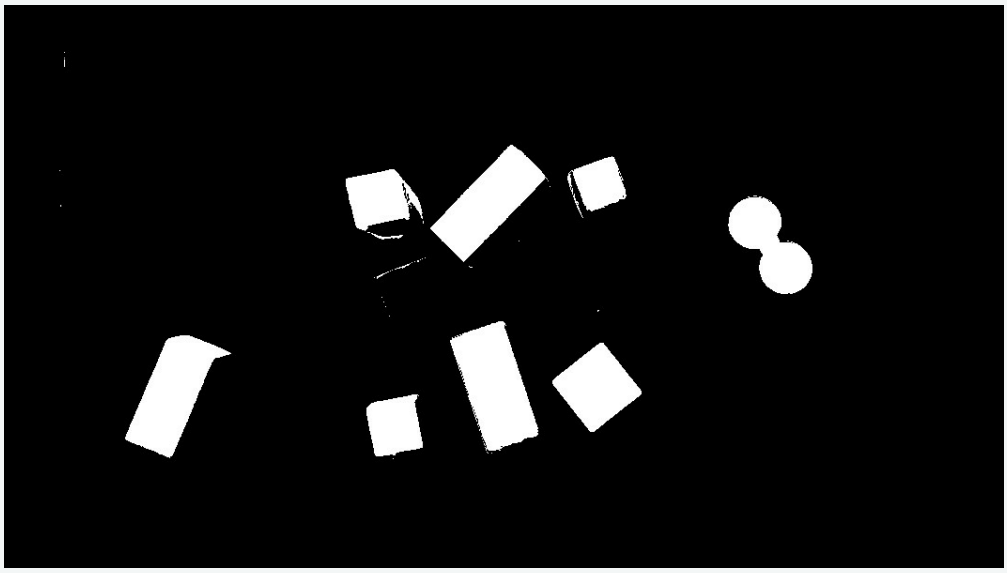
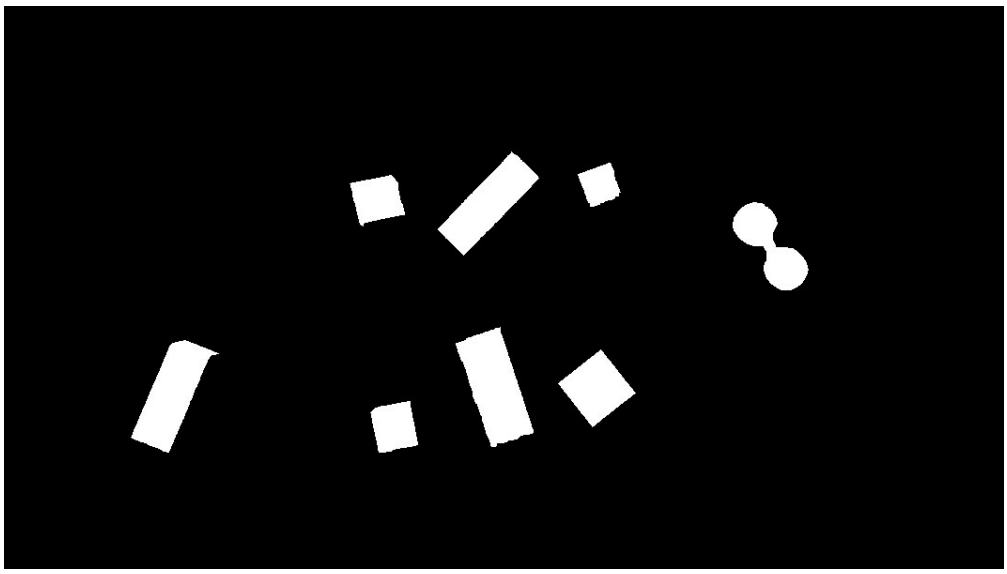
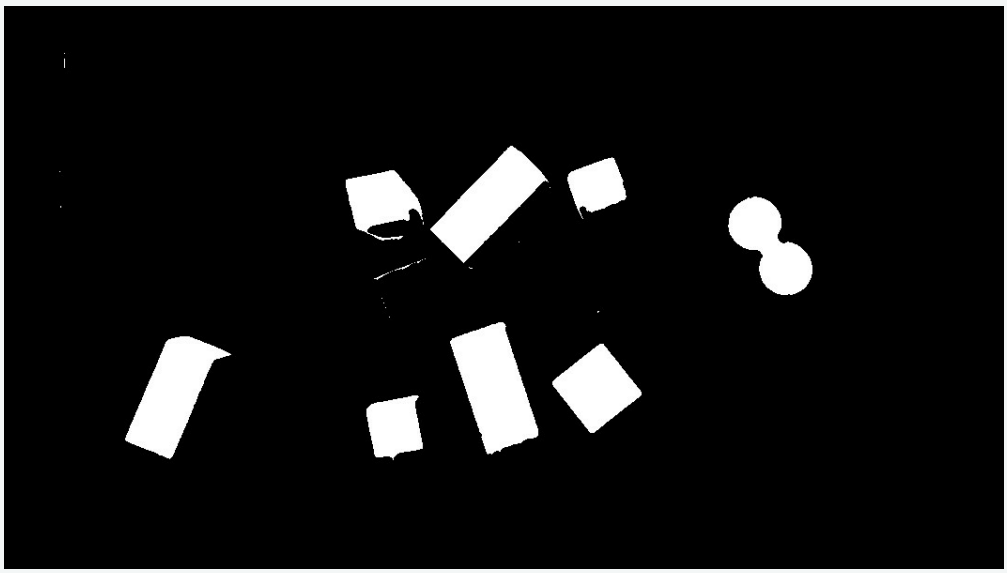
The code shows how to do a Closing & an Erosion:

```
img_threshold = threshold_hsv(img_test, *ColorHSV.ANY.value)

img_close = morphological_transformations(img_threshold, morpho_type=MorphoType.CLOSE,
                                          kernel_shape=(11, 11), kernel_type=KernelType.ELLIPSE)

img_erode = morphological_transformations(img_threshold, morpho_type=MorphoType.ERODE,
                                          kernel_shape=(9, 9), kernel_type=KernelType.RECT)

show_img("img_threshold", img_threshold)
show_img("img_erode", img_erode)
show_img_and_wait_close("img_close", img_close)
```

Images result

Morpho type	Image result

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Morpho type	Image result
None	
Erode	
Close	

Contours finder

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition. See more on OpenCV Documentation

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

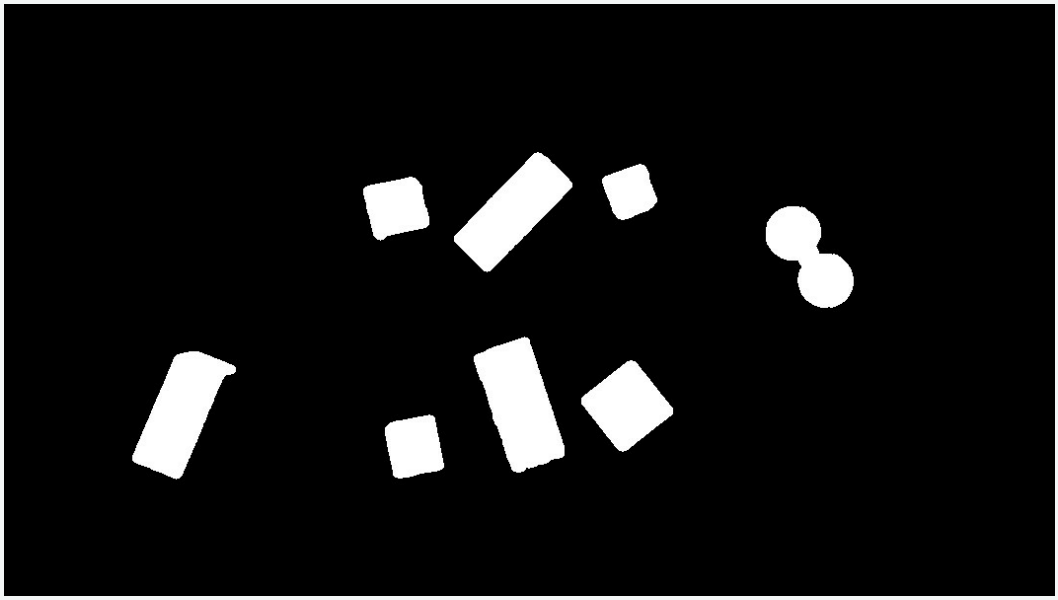
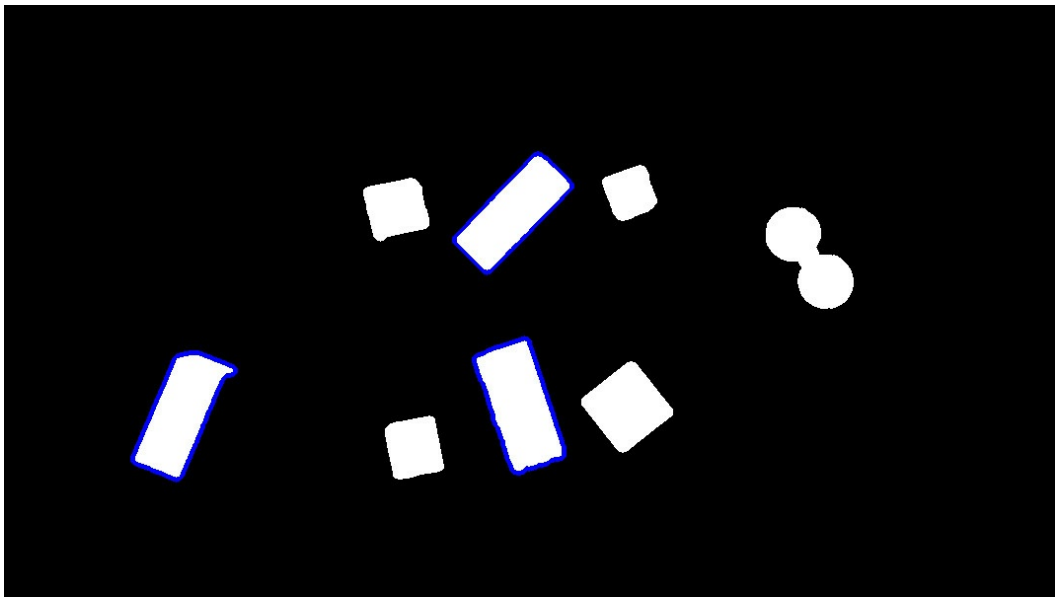
(https://docs.opencv.org/3.4/d3/d05/tutorial_py_table_of_contents_contours.html).

The implemented function to realize these operations is **biggest_contours_finder()** (index.html#vision.image_functions.biggest_contours_finder) which takes a Black & White image, and extracts the biggest (in term of area) contours from it.

The code to extract and draw the 3 biggest contours from an image is the following:

```
img_threshold = threshold_hsv(img_test, *ColorHSV.ANY.value)
img_threshold = morphological_transformations(img_threshold, morpho_type=MorphoType.OPEN,
                                              kernel_shape=(11, 11), kernel_type=KernelType.ELLIPSE)

cnts = biggest_contours_finder(img_threshold, 3)

img_contours = draw_contours(img_threshold, cnts)

show_img("init", img_threshold)
show_img_and_wait_close("img with contours", img_contours)
```

Images result

Thresh + Opening	
3 contours	

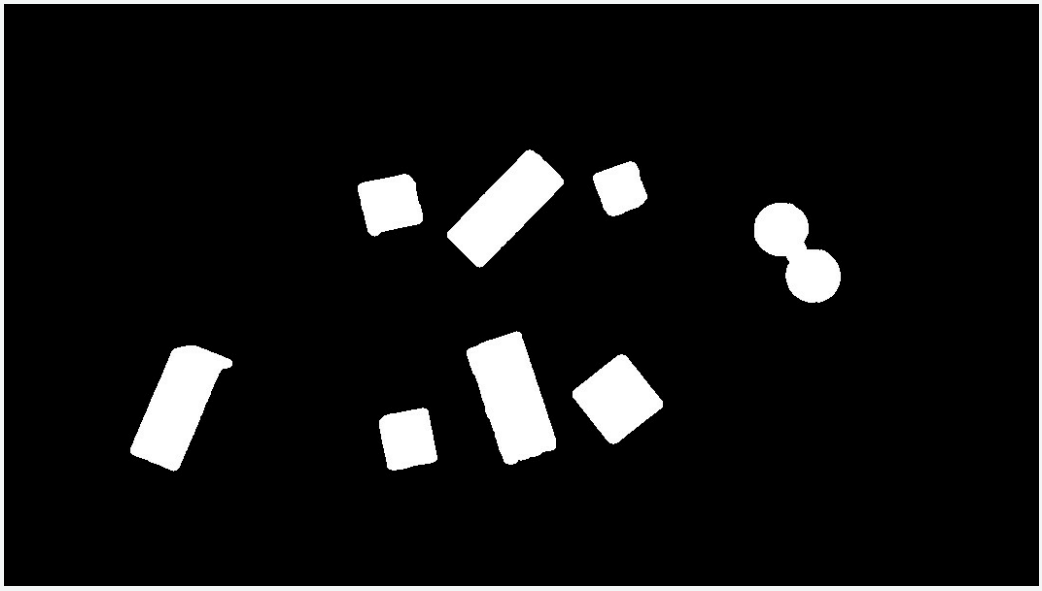
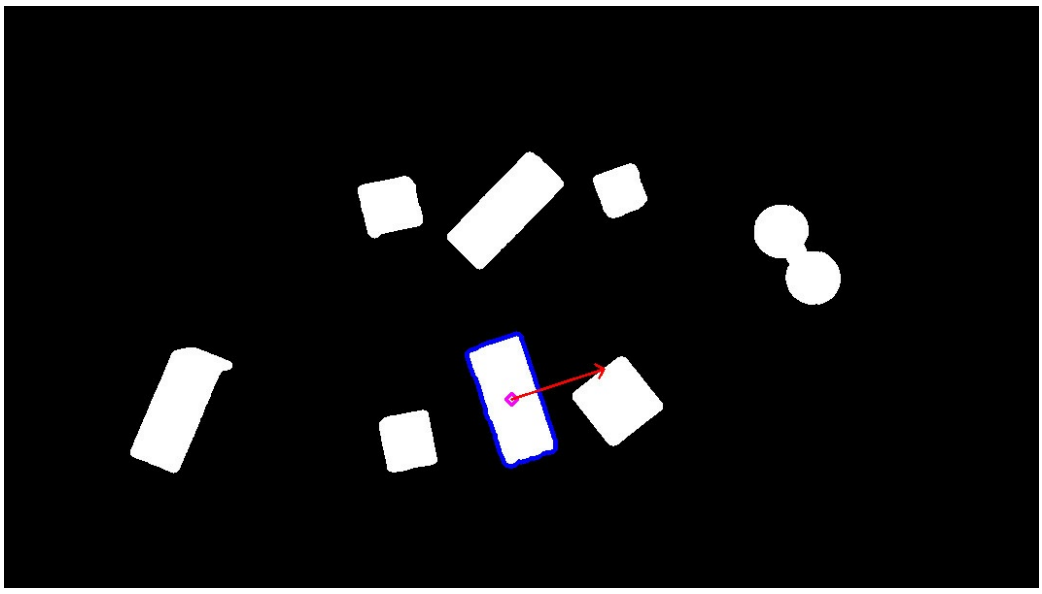
Find object center position

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

In order to catch an object, we need to find a pose from where the end effector can grasp the object. The following method uses contours which have been found in the previous section and finds their barycenter and orientation via the functions **get_contour_barycenter()** (index.html#vision.image_functions.get_contour_barycenter) & **get_contour_angle()** (index.html#vision.image_functions.get_contour_angle).

```
img_threshold = threshold_hsv(img_test, *ColorHSV.ANY.value)
img_threshold = morphological_transformations(img_threshold, morpho_type=MorphoType.OPEN,
                                              kernel_shape=(11, 11), kernel_type=KernelType.ELLIPSE)

cnt = biggest_contour_finder(img_threshold)

cnt_barycenter = get_contour_barycenter(cnt)
cnt_angle = get_contour_angle(cnt)
```

Images result

Thresh + Opening	
Barycenter + Angle	

Note

The drawn vector is normal to the contour's length because we want Ned to catch the object by the width rather than the length. Indeed, it leads to least cases where the gripper cannot open enough.

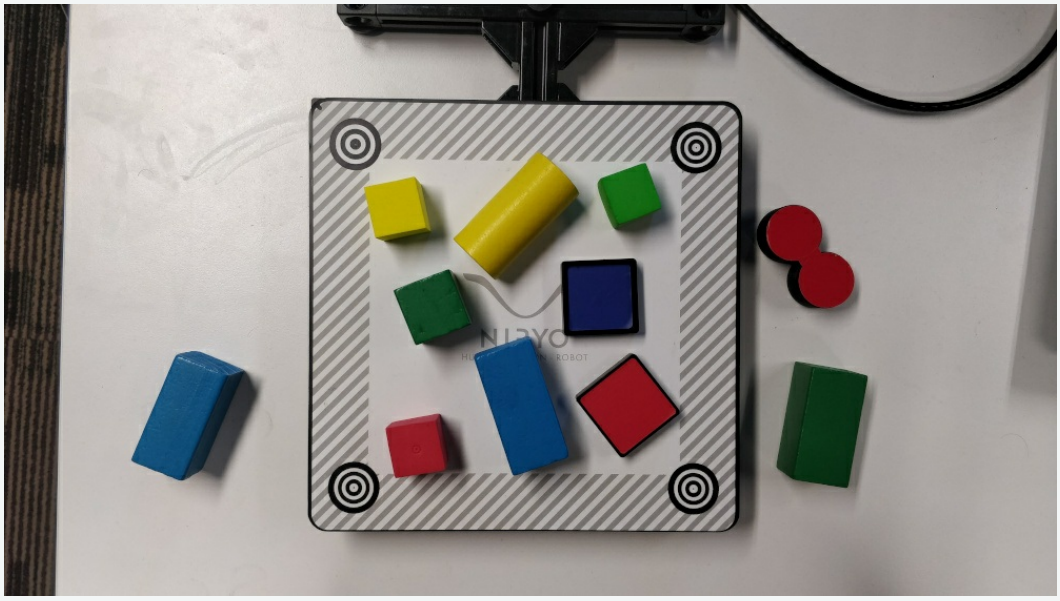
Markers extraction

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

As image processing happens in a workspace, it is important to extract the workspace beforehand! To do so, you can use the function **extract_img_workspace()** (index.html#vision.image_functions.extract_img_workspace).

```
status, im_work = extract_img_workspace(img, workspace_ratio=1.0)
show_img("init", img_test)
show_img_and_wait_close("img_workspace", img_workspace)
```

Images result

Original	
Extracted	

Debug mode

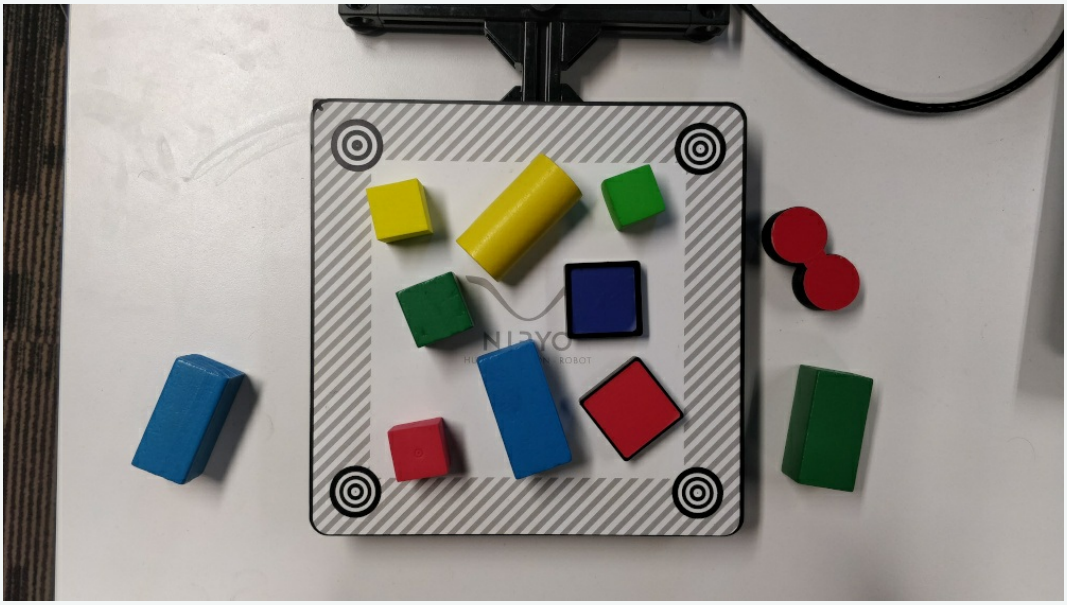
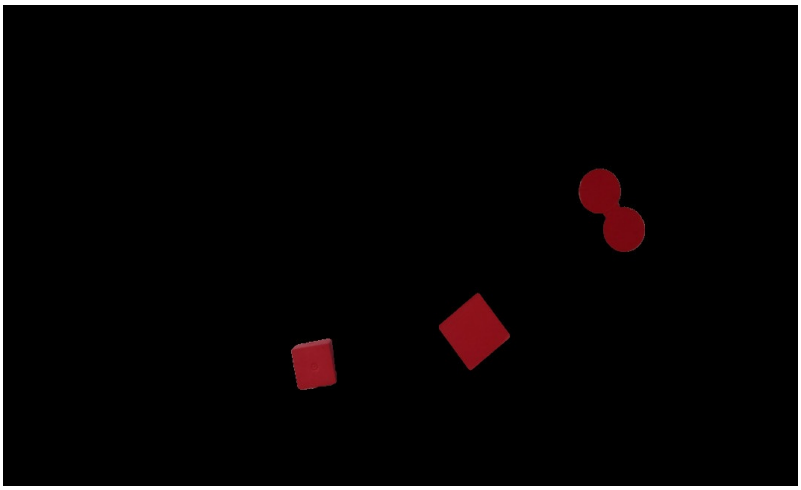
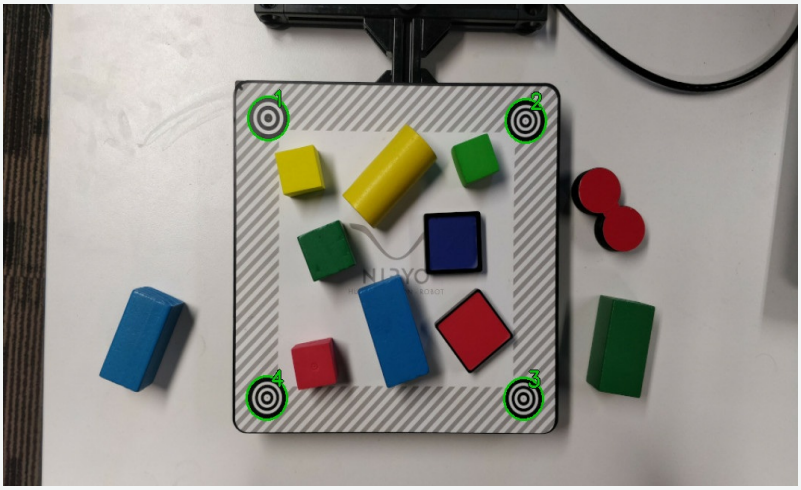
If Ned's functions are failing, you can use Debug functions which are **debug_threshold_color()** (index.html#vision.image_functions.debug_threshold_color) & **debug_markers()** (index.html#vision.image_functions.debug_markers) in order to display what the robot sees.

You can use the functions as follow:

```
debug_color = debug_threshold_color(img_test, ColorHSV.RED)
_status, debug_markers_im = debug_markers(img_test, workspace_ratio=1.0)

show_img("init", img_test)
show_img("debug_color", debug_color)
show_img_and_wait_close("debug_markers", debug_markers_im)
```

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Images result

Original	
Debug red	
Debug Markers	

Do your own image processing!

Now that you are a master in image processing, let's look at full examples.

Display video stream with extracted workspace

In the current state, the following code will display the video stream and the extracted workspace image. You can add your own image processing functions maybe to apply additional process.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```python
from pyniryo import *

# Connecting to robot
robot = NiryoRobot("10.10.10.10")
robot.calibrate_auto()

# Getting calibration param
mtx, dist = robot.get_camera_intrinsics()
# Moving to observation pose
robot.move_pose(*observation_pose.to_list())

while "User do not press Escape neither Q":
    # Getting image
    img_compressed = robot.get_img_compressed()
    # Uncompressing image
    img_raw = uncompress_image(img_compressed)
    # Undistorting
    img_undistort = undistort_image(img_raw, mtx, dist)
    # Trying to find markers
    workspace_found, res_img_markers = debug_markers(img_undistort)
    # Trying to extract workspace if possible
    if workspace_found:
        img_workspace = extract_img_workspace(img_undistort, workspace_ratio=1.0)
    else:
        img_workspace = None

    # --- --------- --- #
    # --- YOUR CODE --- #
    # --- --------- --- #

    # - Display
    # Concatenating raw image and undistorted image
    concat_ims = concat_imgs((img_raw, img_undistort))
    # Concatenating extracted workspace with markers annotation
    if img_workspace is not None:
        resized_img_workspace = resize_img(img_workspace, height=res_img_markers.shape[0])
        res_img_markers = concat_imgs((res_img_markers, resized_img_workspace))
    # Showing images
    show_img("Images raw & undistorted", concat_ims)
    key = show_img("Markers", res_img_markers, wait_ms=30)
    if key in [27, ord("q")]:  # Will break loop if the user press Escape or Q
        break
```

Vision pick via your image processing pipeline

You may want to send coordinate to Ned in order to pick the object of your choice! To do that, use the function **get_target_pose_from_rel()** (index.html#api.tcp_client.NiryoRobot.get_target_pose_from_rel) which converts a relative pose in the workspace into a pose in the robot's world!

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

```
# Initializing variables
obj_pose = None
try_without_success = 0
count = 0
color_hsv_setting = ColorHSV.ANY.value


mtx, dist = robot.get_camera_intrinsics()
# Loop
while try_without_success < 5:
    # Moving to observation pose
    robot.move_pose(observation_pose)

    img_compressed = robot.get_img_compressed()
    img = uncompress_image(img_compressed)
    img = undistort_image(img, mtx, dist)
    # extracting working area
    im_work = extract_img_workspace(img, workspace_ratio=1.0)
    if im_work is None:
        print("Unable to find markers")
        try_without_success += 1
        if display_stream:
            cv2.imshow("Last image saw", img)
            cv2.waitKey(25)
        continue

    # Applying Threshold on ObjectColor
    img_thresh = threshold_hsv(im_work, *color_hsv_setting)

    if display_stream:
        show_img("Last image saw", img, wait_ms=0)
        show_img("Image thresh", img_thresh, wait_ms=30)
    # Getting biggest contour/blob from threshold image
    contour = biggest_contour_finder(img_thresh)
    if contour is None or len(contour) == 0:
        print("No blob found")
        obj_found = False
    else:
        img_thresh_rgb = cv2.cvtColor(img_thresh, cv2.COLOR_GRAY2BGR)
        draw_contours(img_thresh_rgb, [contour])
        show_img("Image thresh", img_thresh, wait_ms=30)

        # Getting contour/blob center and angle
        cx, cy = get_contour_barycenter(contour)
        cx_rel, cy_rel = relative_pos_from_pixels(im_work, cx, cy)
        angle = get_contour_angle(contour)

        # Getting object world pose from relative pose
        obj_pose = robot.get_target_pose_from_rel(workspace_name,
                                                  height_offset=0.0,
                                                  x_rel=cx_rel, y_rel=cy_rel,
                                                  yaw_rel=angle)
        obj_found = True

    if not obj_found:
        try_without_success += 1
        continue

    # Everything is good, so we going to object
    robot.pick_from_pose(obj_pose)

    # Computing new place pose
    offset_x = count % grid_dimension[0] - grid_dimension[0] // 2
    offset_y = (count // grid_dimension[1]) % 3 - grid_dimension[1] // 2
    offset_z = count // (grid_dimension[0] * grid_dimension[1])
    place_pose = center_conditioning_pose.copy_with_offsets(0.05 * offset_x, 0.05 * offset_y, 0.025 * offset_z)

    # Placing
    robot.place_from_pose(place_pose)

    try_without_success = 0
    count += 1
```

Functions documentation

This file presents the different functions and Enums Image Processing available for image processing.

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

These functions are divided in subsections:

- Pure image processing are used to deal with the thresholding, contours detection, ..
- Workspaces wise section contains functions to extract workspace and deal with the relative position in the workspace
- The section Show allows to display images
- Image Editing contains lot of function which can compress images, add text to image, ...

Pure image processing

image_functions.threshold_hsv(*list_min_hsv*, *list_max_hsv*, *reverse_hue=False*, *use_s_prime=False*)

Take BGR image (OpenCV imread result) and return thresholded image according to values on HSV (Hue, Saturation, Value) Pixel will worth 1 if a pixel has a value between min_v and max_v for all channels

Parameters:

- **img** (*numpy.array*) – image BGR if rgb_space = False
- **list_min_hsv** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[int* (https://docs.python.org/3/library/functions.html#int)*]*) – list corresponding to [min_value_H,min_value_S,min_value_V]
- **list_max_hsv** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[int* (https://docs.python.org/3/library/functions.html#int)*]*) – list corresponding to [max_value_H,max_value_S,max_value_V]
- **use_s_prime** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – True if you want to use S channel as S' = S x V else classic
- **reverse_hue** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – Useful for Red color cause it is at both extremum

Returns:
threshold image

Return type:
numpy.array

image_functions.debug_threshold_color(*color_hsv*)

Return masked image to see the effect of color threshold

Parameters:

- **img** (*numpy.array*) – OpenCV image
- **color_hsv** (*ColorHSV* (index.html#vision.enums.ColorHSV)) – Color used for debug

Returns:
Masked image

Return type:
numpy.array

image_functions.morphological_transformations(*morpho_type=<MorphoType.CLOSE: 3>*, *kernel_shape=(5, 5)*, *kernel_type=<KernelType.ELLIPSE: 2>*)

Take black & white image and apply morphological transformation

Parameters:

- **im_thresh** (*numpy.array*) – Black & White Image

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

- **morpho_type** (*MorphoType* (index.html#vision.enums.MorphoType)) – CLOSE/OPEN/ERODE/DILATE => See on OpenCV/Google if you do not know these words
- **kernel_shape** (*tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)[*float* (https://docs.python.org/3/library/functions.html#float)]) – tuple corresponding to the size of the kernel
- **kernel_type** (*KernelType* (index.html#vision.enums.KernelType)) – RECT/ELLIPSE/CROSS => see on OpenCV

Returns:
image after processing

Return type:
numpy.array

image_functions.get_contour_barycenter()

Return barycenter of an OpenCV Contour

Parameters:
contour (*OpenCV Contour*) –

Returns:
Barycenter X, Barycenter Y

Return type:
int (https://docs.python.org/3/library/functions.html#int), int (https://docs.python.org/3/library/functions.html#int)

image_functions.get_contour_angle()

Return orientation of a contour according to the smallest side in order to be well oriented for gripper

Parameters:
contour (*OpenCV Contour*) – contour

Returns:
Angle in radians

Return type:
float (https://docs.python.org/3/library/functions.html#float)

image_functions.biggest_contour_finder()

image_functions.biggest_contours_finder(*nb_contours_max=3*)

Function to find the biggest contour in an binary image

Parameters:
- **img** (*numpy.array*) – Binary Image
- **nb_contours_max** (*int* (https://docs.python.org/3/library/functions.html#int)) – maximal number of contours which will be returned

Returns:
biggest contours found

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[OpenCV Contour]

image_functions.draw_contours(*contours*)

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Draw a list of contour on an image and return the drawing image

Parameters:
- **img** (*numpy.array*) – Image
- **contours** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[OpenCV Contour]*) – contours list

Returns:
Image with drawing

Return type:
numpy.array

Workspaces wise

image_functions.extract_img_workspace(*workspace_ratio=1.0*)

Extract working area from an image thanks to 4 Niryo's markers

Parameters:
- **img** (*numpy.array*) – OpenCV image which contain 4 Niryo's markers
- **workspace_ratio** (*float* (https://docs.python.org/3/library/functions.html#float)) – Ratio between the width and the height of the area represented by the markers

Returns:
extracted and warped working area image

Return type:
numpy.array

image_functions.debug_markers(*workspace_ratio=1.0*)

Display detected markers on an image

Parameters:
- **img** (*numpy.array*) – OpenCV image which contain Niryo's markers
- **workspace_ratio** (*float* (https://docs.python.org/3/library/functions.html#float)) – Ratio between the width and the height of the area represented by the markers

Returns:
(status, annotated image)

Return type:
numpy.array

image_functions.relative_pos_from_pixels(*x_pixels*, *y_pixels*)

Transform a pixels position to a relative position

Parameters:
- **img** (*numpy.array*) – Image where the object is detected
- **x_pixels** (*int* (https://docs.python.org/3/library/functions.html#int)) – coordinate X
- **y_pixels** (*int* (https://docs.python.org/3/library/functions.html#int)) – coordinate Y

Returns:
X relative, Y relative

Return type:

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

float (https://docs.python.org/3/library/functions.html#float), float (https://docs.python.org/3/library/functions.html#float)

Show

image_functions.show_img_and_check_close(*img*)

Display an image and check whether the user want to close

Parameters:
- **window_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – window's name
- **img** (*numpy.array*) – Image

Returns:
boolean indicating if the user wanted to leave

Return type:
bool (https://docs.python.org/3/library/functions.html#bool)

image_functions.show_img(*img*, *wait_ms=1*)

Display an image during a certain time

Parameters:
- **window_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – window's name
- **img** (*numpy.array*) – Image
- **wait_ms** (*int* (https://docs.python.org/3/library/functions.html#int)) – Wait time in milliseconds

Returns:
value of the key pressed during the display

Return type:
int (https://docs.python.org/3/library/functions.html#int)

image_functions.show_img_and_wait_close(*img*)

Display an image and wait that the user close it

Parameters:
- **window_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – window's name
- **img** (*numpy.array*) – Image

Returns:
None

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Image Editing

image_functions.compress_image(*quality=90*)

Compress OpenCV image

Parameters:
- **img** (*numpy.array*) – OpenCV Image
- **quality** (*int* (https://docs.python.org/3/library/functions.html#int)) – integer between 1 - 100. The higher it is, the less information will be lost, but the heavier the compressed image will be

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Returns:
status & string representing compressed image

Return type:
bool (https://docs.python.org/3/library/functions.html#bool), str (https://docs.python.org/3/library/stdtypes.html#str)

image_functions.uncompress_image()

Take a compressed img and return an OpenCV image

Parameters:
compressed_image (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – compressed image

Returns:
OpenCV image

Return type:
numpy.array

image_functions.add_annotation_to_image(*text*, *write_on_top=True*)

Add Annotation to an image

Parameters:
- **img** (*numpy.array*) – Image
- **text** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – text string
- **write_on_top** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – if you write the text on top

Returns:
img with text written on it

Return type:
numpy.array

image_functions.undistort_image(*mtx*, *dist*)

Use camera intrinsics to undistort raw image

Parameters:
- **img** (*numpy.array*) – Raw Image
- **mtx** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]]*) – Camera Intrinsics matrix
- **dist** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]]*) – Distortion Coefficient

Returns:
Undistorted image

Return type:
numpy.array

image_functions.resize_img(*width=None*, *height=None*, *inter=3*)

Resize an image. The user should precise only width or height if he wants to keep image's ratio

Parameters:

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

- **img** (*numpy.array*) – OpenCV Image
- **width** (*int* (https://docs.python.org/3/library/functions.html#int)) – Target Width
- **height** (*int* (https://docs.python.org/3/library/functions.html#int)) – Target Height
- **inter** (*long*) – OpenCV interpolation flag

Returns:
resized image

Return type:
numpy.array

image_functions.concat_imgs(*axis=1*)

Concat multiple images along 1 axis

Parameters:
- **tuple_imgs** (*tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[numpy.array]*) – tuple of images
- **axis** (*int* (https://docs.python.org/3/library/functions.html#int)) – 0 means vertically and 1 means horizontally

Returns:
Concat image

Return type:
numpy.array

Enums Image Processing

Enums are used to pass specific parameters to functions.

List of enums:

- **ColorHSV**
- **ColorHSVPrime**
- **ObjectType**
- **MorphoType**
- **KernelType**

class **ColorHSV(*value*)**

MIN HSV, MAX HSV, Invert Hue (bool)

BLUE= *([90, 50, 85], [125, 255, 255], False)*

RED= *([15, 80, 75], [170, 255, 255], True)*

GREEN= *([40, 60, 75], [85, 255, 255], False)*

ANY= *([0, 50, 100], [179, 255, 255], False)*

class **ColorHSVPrime(*value*)**

MIN HSV, MAX HSV, Invert Hue (bool)

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

BLUE= *([90, 70, 100], [115, 255, 255], False)*

RED= *([15, 70, 100], [170, 255, 255], True)*

GREEN= *([40, 70, 100], [85, 255, 255], False)*

ANY= *([0, 70, 140], [179, 255, 255], False)*

class **ObjectType**(*value*)

An enumeration.

SQUARE= *4*

TRIANGLE= *3*

CIRCLE= *-1*

ANY= *0*

class **MorphoType**(*value*)

An enumeration.

ERODE= *0*

DILATE= *1*

OPEN= *2*

CLOSE= *3*

class **KernelType**(*value*)

An enumeration.

RECT= *0*

ELLIPSE= *2*

CROSS= *1*

Indices and tables

- Index (genindex.html)
- Module Index (py-modindex.html)
- Search Page (search.html)

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS

Suggest a modification Download as PDF

2021, Niryo All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Niryo SAS