Ned ROS documentation

This documentation contains everything you need to understand Ned's functioning and how to control it through ROS.

It is made as well for users who are using the "physical" robot as for those who want to use a virtual version.

Preamble

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

Then, you should check the Getting Started (index.html#document-source/installation/getting_started) section to setup your environment and try out the stack by yourself. If you don't have a real robot at your disposal, you can still simulate it via the Use Niryo robot through 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 physical robot, or to use the simulation.

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

- Send commands via the terminal in order to call services, trigger actions, …
- Write an entire Python/C++ node to realize a full process.

See ROS (index.html#document-source/stack/high_level) 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 physical 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

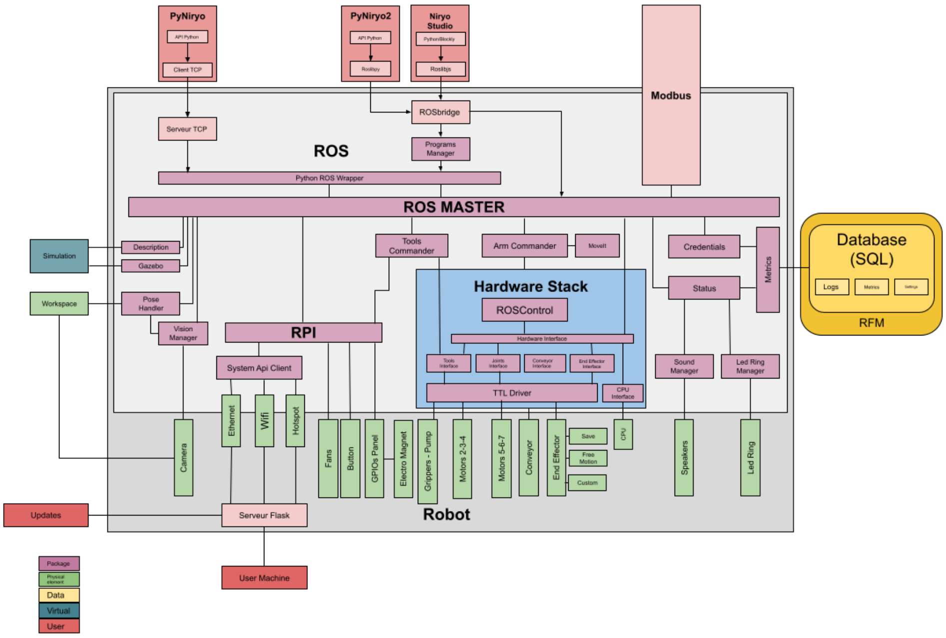
Other methods are available to control the robot allowing the user to code and run programs outside its terminal.

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 Niryo's robot software. It will help you understand where are placed each part of the software.



Niryo robot v3 software

Use your Niryo Robot

Every Niryo Robot is usable as it is when first switched on, with Niryo Studio for instance. However this robot can be used in many more ways if you want to go deeper into its understanding.

In this tutorial, we will explain how the robot is setup and the different options you have to control it.

Connecting to the Robot

You can connect to your robot in multiple ways (Ethernet direct, Wi-Fi Hotspot, LAN).

You can find more information on how to connect to your robothere (https://docs.niryo.com/product/niryo-studio/source/connection.html).

Once your robot is accessible from your computer, you can access it through three ways:

- Via Niryo Studio

  Niryo Studio provides you with all the tools you need to control the robot. Please refer to theNiryo Studio documentation (https://docs.niryo.com/product/niryo-studio/v3.2.1/en/index.html) for more information.

- Via ROS Multimachine.

  ROS implements a way to communicate between nodes launched on different machines. By indicating your computer where the Niryo Robot ROS Master Node is, you can communicate to any ROS Node as if you were directly connected on the robot. See the tutorial on the ROS wiki (https://wiki.ros.org/ROS/Tutorials/MultipleMachines) for more information.

- Via ssh (**for advanced users only**).

  Port 22 is open without restriction. The default user is “niryo” and its password is “robotics”.

Robot setup

To help you understand how the OS is setup in the robot, we provide you with some insights of it.

> **Attention**
>
> This document is not intended to explain how to completely install a robot from an empty SD card. It is only intended to give you clues on its architecture. Some of the installation steps are refered in Ubuntu 18 Installation (index.html#ubuntu-18-installation) in case you would like to reinstall some part of it (catkin_ws for example).

System setup

The robot is running on top of an Ubuntu server 18.04.5 for ARM customized to work on a Raspberry Pi 4B.

It comprises all the following elements :

- ROS melodic and its requirements

- Sound driver
- Led ring driver
- Robot System services (connectivity, databases, flask server)
- Basic development tools (compilation, editing tools)

We took care to update and upgrade the system before sending it to you

> **Attention**
>
> We can't ensure that the stability of the system will be kept if you try to update your system by yourself (using apt). We strongly advise you not to do so.

Home setup

The system has been configured with a default user "niryo". The core of the robot program is installed in the home directory of niryo user*/home/niryo*.

In this directory, you have:

- catkin_ws : contains the source code and the compiled binary for the Niryo ROS Stack
- firmware_updater : updater for the steppers and the End Effector
- niryo_robot_saved_files : set of files saved on the robot, used by Niryo Studio
- system_software : configuration files for system wide functions

Services and daemons

Two services are used on the robot:

- niryo_system_software : It launches the flask server for API communication with the robot
- niryo_robot_ros : It launches the stack via */opt/start_ros.sh* script at startup.

File */opt/start_ros.sh* on the ned2 robot :

```
source ~/.bashrc
source /home/niryo/catkin_ws/install/release/ned2/setup.bash && roslaunch niryo_robot_bringup niryo_ned2_robot.launch&
```

If you want to start, stop or disable one of those services, please refer to thededicated manual (https://manpages.ubuntu.com/manpages/bionic/man8/service.8.html).

Starting the robot manually (for advanced users only)

Before continuing, be sure you know what you are doing.

You will need to have a ssh access setup to continue.

Stopping the service

First you will need to stop the Niryo ROS Stack that is automatically started when the robot boots up. Use the system linux command to do so:

```
sudo service niryo_robot_ros stop
```

Starting the robot

To start the robot, launch the following commands in a ssh terminal:

For Ned

```
source /home/niryo/catkin_ws/install/release/ned/setup.bash
roslaunch niryo_robot_bringup niryo_ned_robot.launch
```

For Ned2

```
source /home/niryo/catkin_ws/install/release/ned2/setup.bash
roslaunch niryo_robot_bringup niryo_ned2_robot.launch
```

Robot launch options

Name	Default Value	Description
log_level	INFO	Log level to display for ROS loggers
ttl_enabled	true	Enable or disable the TTL bus usage. This feature is used for debug mainly and can lead to an unstable stack.
can_enabled	true	Enable or disable the CAN bus usage. This feature is used for debug mainly and can lead to an unstable stack.
debug	false	Launch in debug mode. For development and debug only.

Name	Default Value	Description
timed	true	To launch the node using timed_roslaunch. If enabled, will first launch sound and light nodes to have a better user experience. If disabled, the node is directly launched

Changing the log level

Before launching the robot, you can change the configuration file for the ROS Logger in order to change the log level displayed on the terminal. This file is located in */home/niryo/catkin_ws/src/niryo_robot_bringup/config/niryo_robot_trace.conf*.

It defines the logs levels for all cpp packages, based on log4cxx configuration file syntax. Please see documentation of rosconsole (http://wiki.ros.org/rosconsole) or log4cxx (https://logging.apache.org/log4cxx/latest_stable/index.html) for more information.

By default, the level is set to INFO, you can change this value if you want more logs.

```
# Set the default ros output to warning and higher
log4j.logger.ros=INFO
```

Attention

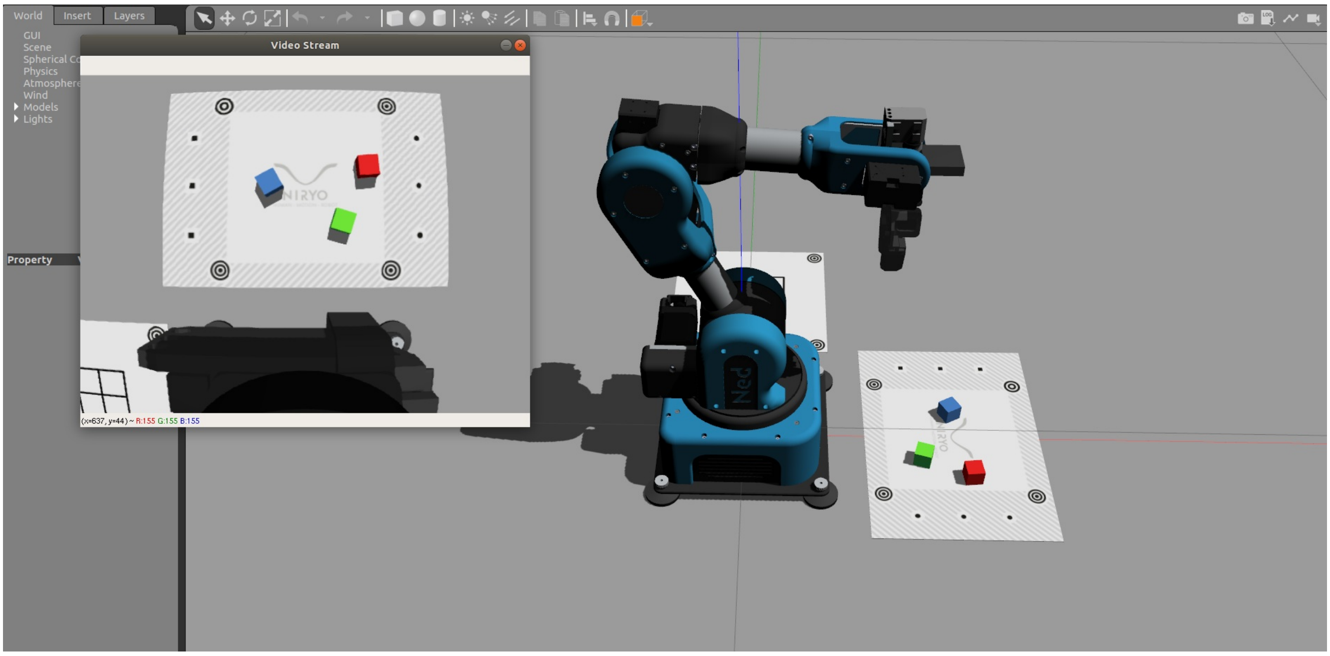
DEBUG level is very verbose, you can deteriorate the performances of your robot by doing so.

You can also choose to change only the log level of a specific cpp package by uncommenting one of the following lines and optionally change the associated log level.

```
#log4j.logger.ros.can_driver = DEBUG
log4j.logger.ros.common = DEBUG
log4j.logger.ros.conveyor_interface = ERROR
```

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 a robot simulation on a computer.

Note

You can use Niryo Studio with the simulation (https://docs.niryo.com/product/niryo-studio/source/connection.html#using-ned-in-simulation-with-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. Please follow the steps in Ubuntu 18 Installation (index.html#ubuntu-18-installation) to install a working environment.

Simulation usage

Important

- Hardware features are simulated as if you were using a real robot.
- The data returned by the faked drivers is arbitrary and immutable. Among this data, you will have : voltage, temperature, error state (always 0), ping (always true), end effector state (immutable)

The simulation is a powerful tool allowing to test new programs directly on your computer which prevents 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 - No visualization

This mode is mainly for simulation and tests purpose, bringing you in the closest state as possible to a real robot control. It is available for all currently supported architectures. You can access it by using the commands:

- One simulation:

```
roslaunch niryo_robot_bringup niryo_one_simulation.launch
```

- Ned simulation:

```
roslaunch niryo_robot_bringup niryo_ned_simulation.launch
```

- Ned2 simulation:

```
roslaunch niryo_robot_bringup niryo_ned2_simulation.launch
```

This mode is useful if your CPU capacity is limited or if you don't have X server available.

Options

This mode is the more flexible one, as it provides all the possible options to customize the simulation. For the other simulation modes (with RViz and Gazebo) we will just force some of these parameters to specific values.

Simulation without visualization Options

Name	Default Value	Description
log_level	INFO	Log level to display for ROS loggers
ttl_enabled	true	Enable or disable the TTL bus usage. This feature is used for debug mainly and can lead to an unstable stack.
can_enabled	true	Enable or disable the CAN bus usage. This feature is used for debug mainly and can lead to an unstable stack.
debug	false	Launch in debug mode. For development and debug only.
conf_location	version.txt	Location of the version.txt file. A path to the file is required.
simu_gripper	true	Simulate the presence of a gripper id 11 on the bus
simu_conveyor	true	Simulate the presence of a conveyor (CAN for One and Ned, TTL for Ned2, based on configuration files) on the bus
vision_enabled	true	Enable the Vision Kit
gazebo	false	Enable gazebo specific parameters (However it does not launch gazebo, use gazebo specific launch file for that)

Without physics - RViz Visualization

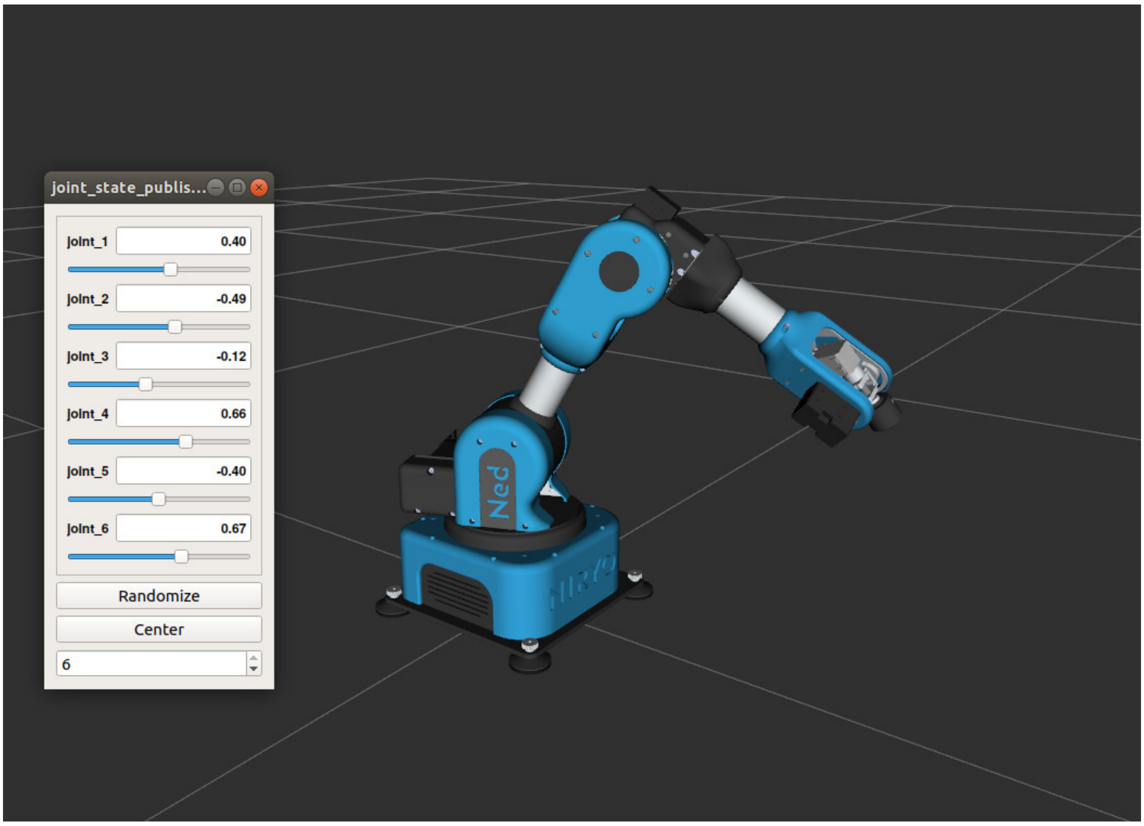
A simple visualization of the robot is possible via a tool called Rviz. This application will simulate the robot with its correct geometry and positions but without physics to avoid using too much CPU.

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.



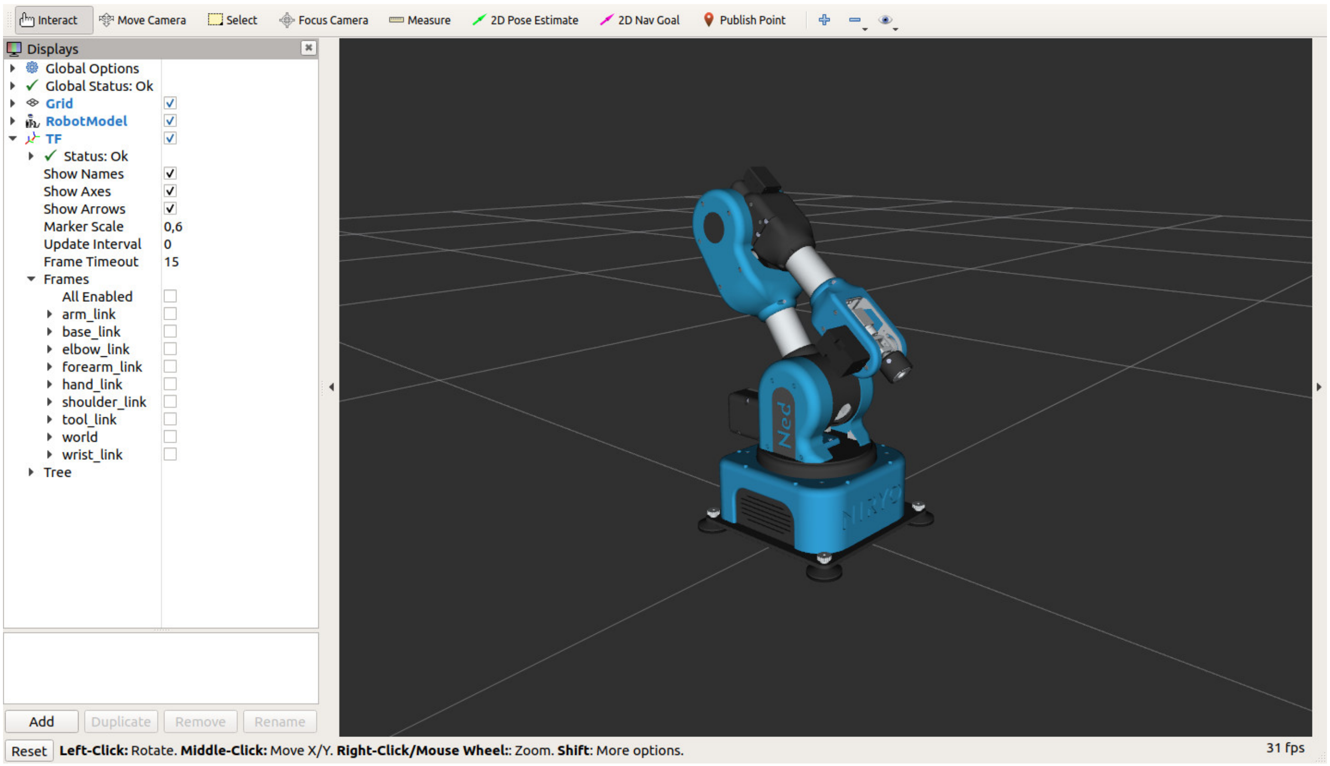
Example of trackbars use.

Control with ROS

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

To run it:

```
roslaunch niryo_robot_bringup desktop_rviz_simulation.launch
```



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

RViz Visualization options

Table of RViz launch Options

Name	Default Value	Description
log_level	INFO	Log level to display for ROS loggers
hardware_version	ned	Use the parameters dedicated to this specific hardware_version. Possible values are “one”, “ned” and “ned2”
debug	false	Launch in debug mode. For development and debug only.
gui	true	Enable the gui visualization
conf_location	version.txt	Location of the version.txt file. A path to the file is required.
simu_gripper	false	Simulate the presence of a gripper id 11 on the bus (Visualisation of the tool will not be visible, whatever the value of this parameter)
simu_conveyor	false	Simulate the presence of a conveyor (Visualisation of the conveyor will not be visible, whatever the value of this parameter)

With physics - Gazebo Simulation

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

- Collision.
- World creation → A 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.

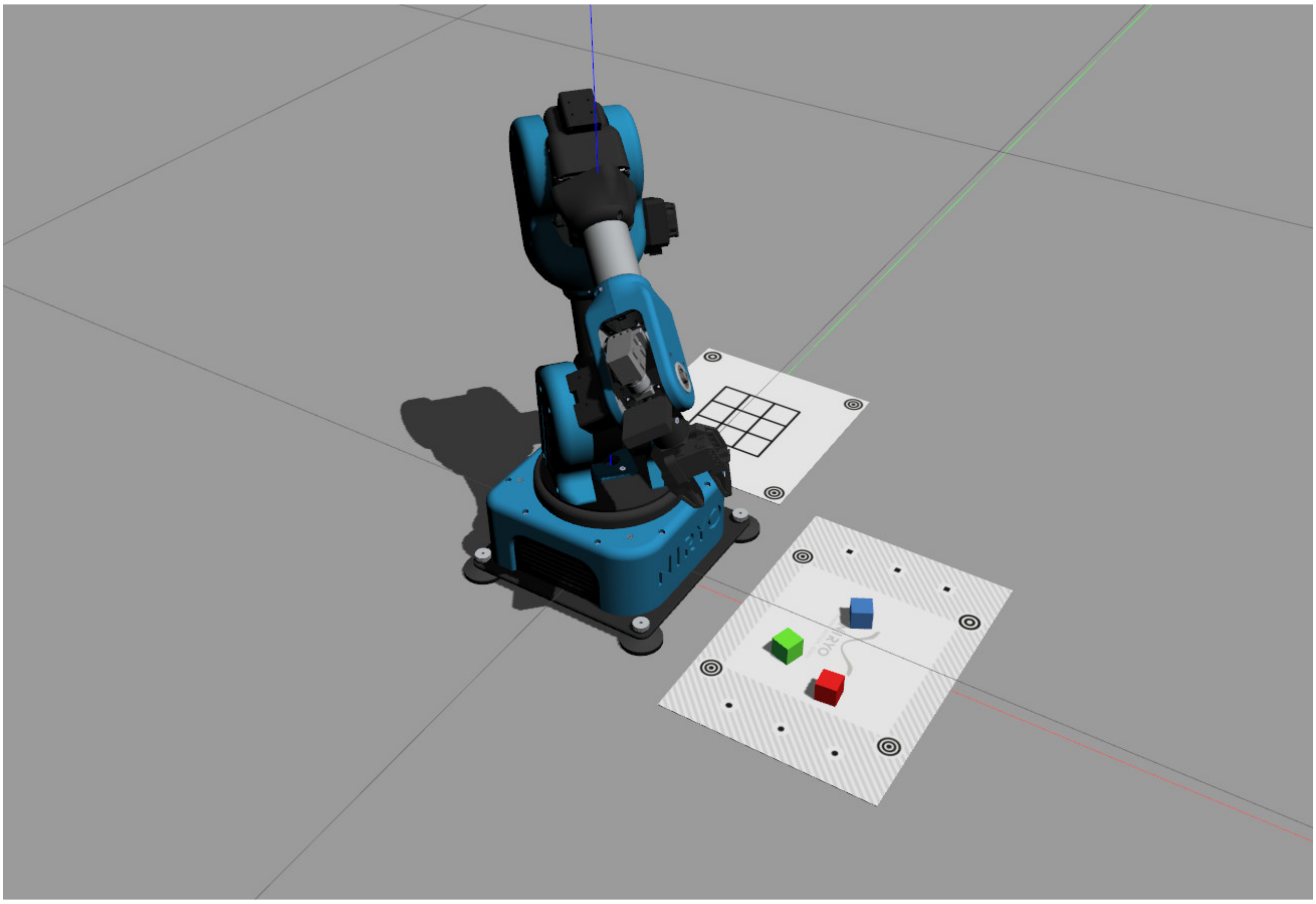
> **Note**
>
> Gazebo also generates camera distortion, which brings the simulation even closer from the reality!

Launch Gazebo 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
```



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.

Gazebo Simulation options

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.

Table of Gazebo launch Options

Name	Default Value	Description
log_level	INFO	Log level to display for ROS loggers
hardware_version	ned	Use the parameters dedicated to this specific hardware_version. Possible values are "one", "ned" and "ned2"
debug	false	Launch in debug mode. For development and debug only.
gui	true	Enable the gui visualization
conf_location	version.txt	Location of the version.txt file. A path to the file is required.
gripper_n_camera	true	Simulate the presence of a gripper id 11 and a camera on the bus
simu_conveyor	true	Simulate the presence of a conveyor (Visualisation of the conveyor will not be visible, whatever the value of this parameter)

Quick start

Welcome to the robot quick start. Here you will learn the essential features of the robot to get you started.

Robot connection

There are 4 ways to connect your computer to the robot:

Hotspot

- **Type:** Wi-Fi
- **Difficulty:** Easy
- **Description:** The robot provides its own Wi-Fi network. In this mode, you can connect to the robot like any other Wi-Fi network. The network name is in the format of **NiryoRobot xx-xx-xx** and the default password is **niryorobot**. To change the name of the robot, refer to the section: Robot Name .
- **More informations:** Wi-Fi settings, Using Ned in Hotspot mode .
- **Advantage:** Easy, no cable required.
- **Disadvantage:** Ethernet connection needed on the computer to have access to the internet. The robot has no access to the internet and cannot update itself.
- **IP address:** 10.10.10.10

Connected mode

- **Type:** Wi-Fi
- **Difficulty:** Medium
- **Description:** The robot is connected to an existing Wi-Fi network.
- **More informations:** Wi-Fi settings, Using Ned on your Wi-Fi network .
- **Advantage:** Ethernet connection needed on the computer to have access to the internet. The robot has no access to the internet and cannot update itself.
- **Disadvantage:** Stable Wi-Fi connection required.
- **IP address:** Dependant of your network.

Ethernet direct

- **Type:** Ethernet
- **Difficulty:** Medium
- **Description:** The robot is connected directly to the computer via an ethernet cable.
- **More informations:** Ethernet settings, Using Ned with an Ethernet cable .
- **Advantage:** The computer can have access to the internet through Wi-Fi. Safer and better communication with the robot.
- **Disadvantage:** Cable required. The robot has no access to the internet and cannot update itself.
- **IP address:** 169.254.200.200

Ethernet through network

- **Type:** Ethernet
- **Difficulty:** Medium
- **Description:** The robot is connected to the network via an ethernet cable, and the computer is connected to the network via an ethernet cable or via Wi-Fi.
- **More informations:** Ethernet settings .
- **Advantage:** The robot and the computer can have access to the internet. Better communication with the robot.
- **Disadvantage:** Cable required.
- **IP address:** Dependant of your network.

Robot programming

There are 6 ways to program Niryo's robots:

Ways to program the Niryo robots

Name	Language	Difficulty	Documentation	Description
Niryo Studio	Blockly	Beginner	Blockly	Simplified block programming. Program your use cases as quickly as possible.
PyNiryo	Python	Intermediate	Pyniryo	Program your robot remotely via a Python API 2.7 and 3.X .
Python ROS wrapper	Python	Intermediate	Python ROS wrapper	Program and run your Python code directly in the robot. No software or setup required except Niryo Studio or an ssh terminal.
ROS	Python, C++	Advanced	Niryo Ros	Program and run your ROS node directly on the robot, or remotely through ROS Multimachine.
MODBUS	Any	Advanced	MODBUS	Programs can communicate through network MODBUS with the robots in any language available.
TCP Server	Any	Advanced	TCP Server	Programs can communicate through network TCP with the robots in any language available.

Niryo One and Ned tips

Program your first move in 30 seconds

The fastest way to program the robot is via Blockly (https://docs.niryo.com/product/niryo-studio/source/blockly_api.html). When you are on the Blockly page and logged into the robot, switch to learning mode via the toggle. You can then press the button on top of the robot's base once to bring up a block with the robot's current position. Thus, move your robot by hand, press the button and connect the blocks. Congratulations you have programmed a robot at lightning speed!

At the top right of the Blockly window, you can choose to save the positions in either **Joints** or **Pose** mode.

Joints & Poses, what's the difference?

The joints are the different joints of the robot. In joint mode, you give the robot a command on each of the robot motors. The default unit used is the radian. 6.28318530718 radian is 2π and corresponds to 360°. On Niryo Studio you can switch to degrees for more simplicity.

The Pose corresponds to the x, y, z coordinates and the roll, pitch, yaw orientation (respecting the rotation around the x, y, z axes) of the extremity of the robot. The x-axis is directed to the front of the robot, and the y-axis to the left of the robot. A positive x-coordinate will move the robot forward. A positive y-coordinate will move the robot to the left, and negative y will move the robot to the right.

Sometimes there can be several axis configurations of the robot that correspond to the same coordinates. This is why it is recommended to use the **Joints** commands instead. The **Pose** is however easier and more intuitive to use to ask the robot to go for example 10cm higher, or 10 to the right.

Use a tool

To use a tool, remember to use the **scan** function to detect the connected tool. You can then use the grippers, the Vacuum Pump or the Electromagnet as you wish.

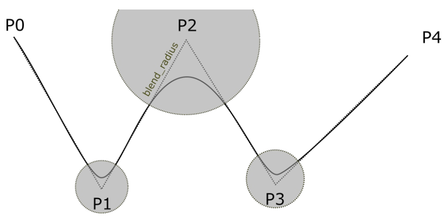
Remember to add the scan function at the beginning of each of your programs to avoid any surprises.

Our different tools are intelligent, so the robot will be able to adapt its movements according to the selected tool for a pick and place with vision. Also, you can program your movements with **Pose**. By activating the TCP (Tool Center Point) (https://docs.niryo.com/product/niryo-studio/source/settings.html#robot-tcp-tool-center-point) function, the TCP of the robot, and therefore the movements, will adapt to the tool equipped.

Standard, linear, waypointed moves, what's the difference?

There are many different types of movement possible for robot arms. The 3 most used are the following:

- **Standard movements:** Also called PTP (Point To Point). This is the simplest movement. In this type of movement, the duration of the movement is minimized, each joint reaches the final position at the same time. The robot draws a kind of arc of a circle according to the initial and final positions.
- **Linear movements:** The robot draws a straight line between the start and end position However, a linear movement is not always possible between two points depending on the constraints of the robot. Make sure that the movement is feasible. If not, the robot will return an error.
- **Smoothed movements by waypoints:** This is where we ask the robot to make a movement to an end point by passing through intermediate points. The robot draws linear paths, or PTP if linear motion is not possible, between each waypoint without stopping. It is also possible to record blend radius to smooth the movement and to draw curves between the points. This path is ideal for dodging obstacles.



Waypointed trajectory with blend radius (https://ros-planning.github.io/moveit_tutorials/doc/pilz_industrial_motion_planner/pilz_industrial_motion_planner.html#user-interface-sequence-capability)

Start, Pause, Cancel a program execution

You may not know it, but the button on the top of the base of the robot also allows you to start, pause and stop a program.

When a program is running:

- 1 press pauses the program
- 2 presses will pause the programme and activate the learning mode

When a program is paused:

- 1 press resumes the program
- 2 presses stop the program
- If there is no intervention for 30 seconds, the programme stops automatically

When the program is paused, the LED at the back flashes white.

When no program is running you can also start a program by pressing the same button once. To set it up, go to theProgram Autorun (https://docs.niryo.com/product/niryo-studio/source/programs.html#program-autorun).

Getting Started

The Niryo ROS Stack can be installed in multiple target OS:

- Raspberry Pi 3 (deprecated, not supported anymore)
- Raspberry Pi 4
- Desktop

As the stack is based on ROS Melodic or Kinetic (deprecated), you need to be on an Ubuntu based system.

We currently support the following versions:

- Ubuntu 18.04 Ubuntu 18 Installation
- Windows Subsystem for Linux 2 (WSL 2) - Ubuntu 18.04 Windows Subsystem for Linux installation (experimental)

Ubuntu 18 Installation

This guide will explain the steps needed to install the Niryo Robot Stack on an Ubuntu 18 OS. You can apply these steps to set up a working simulation environment on any development computer, or to set up a working robot stack on a Raspberry Pi.

Installation index:

- Prerequisites
- Install ROS dependencies
- Setup Ned ROS environment

Prerequisites

The Niryo ROS Stack runs on top of ROS Melodic or Kinetic (deprecated). This version of ROS is strongly dependent of Ubuntu 18.04 version, thus, this OS is currently the only official supported OS.

Be sure to have an up to date system before continuing

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get dist-upgrade
```

Ubuntu packages

The Niryo ROS Stack needs the following packages in order to run correctly:

- build-essential
- sqlite3
- ffmpeg

Note

These packages are mostly useful on a real robot, but as the code is identical between simulation and real functioning, a lack of these packages on a simulation can lead to unstabilities.

Python environment

The Python environment is installed using the requirements_ned2.txt file

```
pip2 install -r src/requirements_ned2.txt
```

Note

ROS Melodic is still using Python2 internally so are our Python scripts to keep version alignment. You thus need to install the requirements using Python2 pip2 tool

ROS set up

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 operations, 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 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 package, we have referenced all the dependencies in their respective *package.xml* file, which allows 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:

- 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 future terminals, you can add it to your **bashrc** file:

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

Installation is now finished!

Windows Subsystem for Linux installation (experimental)

Microsoft is developping since 2016 a compatibility layer for running Linux binary executables natively on Windows 10. With the version 2 issued in 2019, this "hidden Linux kernel" is now mature enough to run complex operations like the full ROS stack [2].

Thus you will be able to run simulations for the Ned, Niryo One or Ned2 robots on a Windows machine.

> **Note**
>
> You have to be running Windows 10 version 2004 (Build 19041) or higher for WSL2 to work.

> **Warning**
>
> The installation under WSL is not originally supported by Niryo, this guide is provided on an indicative basis only.

The following guide is mainly adapted from this blog post from Jack Kawell, feel free to refer to it for more complete information[1]

Install WSL2 [3]

1. Enable Windows Subsystem for Linux on your machine (in a powershell terminal)

```
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart
```

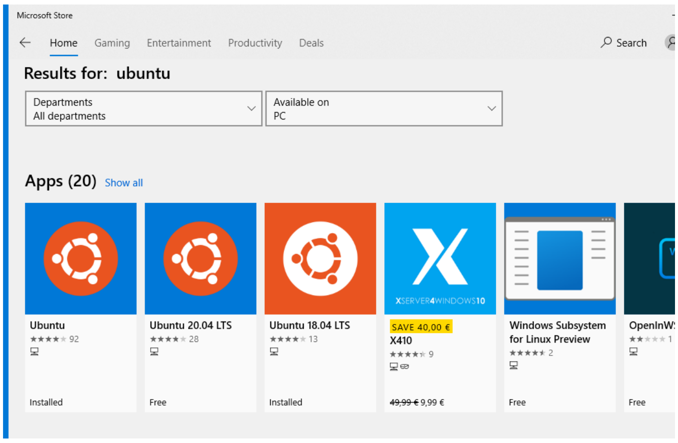
2. Update WSL to use version 2 (in a powershell terminal)

```
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart
```

3. You then need to restart your machine to finish the WSL installation and the upgrade to WSL2.
4. Set default version of WSL to 2 (in a powershell terminal)

```
wsl --set-default-version 2
```

5. Install an Ubuntu 18.04 distribution using the Windows Store



Ubuntu 18.04 in the Windows Store

6. Launch the app. The first time, it asks you to finish the initialization of the OS.

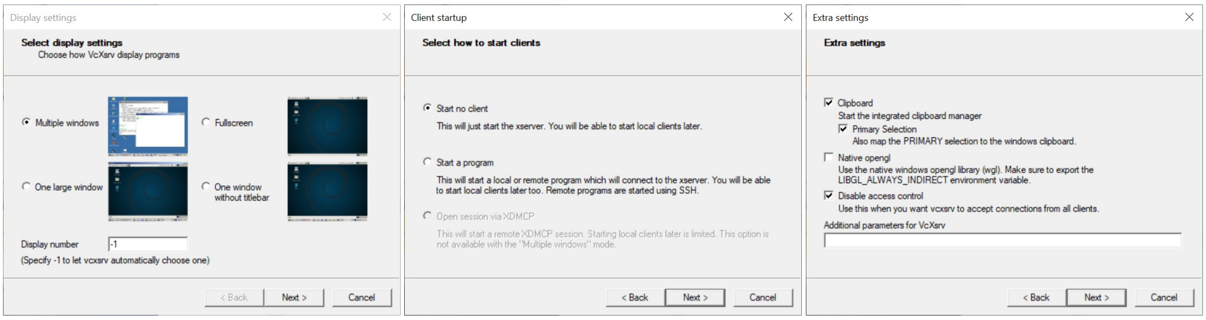
Your Ubuntu OS is now ready. You can continue the build of the stack using the tutorial.

Setting up GUI forwarding

WSL does not come with an X server. Thus, you will not be able to launch any graphical windows for now. But we can change this by using a Windows X server and forward the GUI to it using GUI forwarding.

Many X servers exist for Windows 10. We tested VcXsrc, and it correctly does the job. https://sourceforge.net/projects/vcxsrv/ (https://sourceforge.net/projects/vcxsrv/)

1. Launch VcXsrv. Be sure to have the following options : - Uncheck "Native OpenGL" - Check "Disable access control"



Note

You can directly use this configuration by using this configuration file (_downloads/818503a538e687731e85c64365865076/wsl_config.xlaunch)

2. You need to export the address of your Xserver in Ubuntu 18 to forward the GUI

```
export DISPLAY=$(cat /etc/resolv.conf | grep nameserver | awk '{print $2}'):0
```

You can add this to your bashrc file.

3. You can check that your forwarding works by using simple X11 apps for example:

```
sudo apt update
sudo apt install x11-apps
xcalc
```

4. Install ROS Melodic (see instructions here)
5. Try launching Rviz

```
roscore & rosrun rviz rviz
```

6. You should now be able to launch any simulation of the One, Ned or Ned2 using Rviz or Gazebo

Troubleshooting

Error: Can't open display: 192.168.1.44:0.0 Your DISPLAY variable does not match the address of your XServer.

Try :

- Check that you correctly launched your XServer with the required options (Disable access control is essential)
- Check that the IP you gave is correct (you need the address in /etc/resolv.conf to have it work)

OpenGL issues Some people have said that they run into issues with OpenGL applications like Rviz. If you do, try setting the environment variable LIBGL_ALWAYS_INDIRECT=0 in your WSL2 terminal (you can just add export LIBGL_ALWAYS_INDIRECT=0 to the end of your .bashrc file).

[1] https://jack-kawell.com/2020/06/12/ros-wsl2/

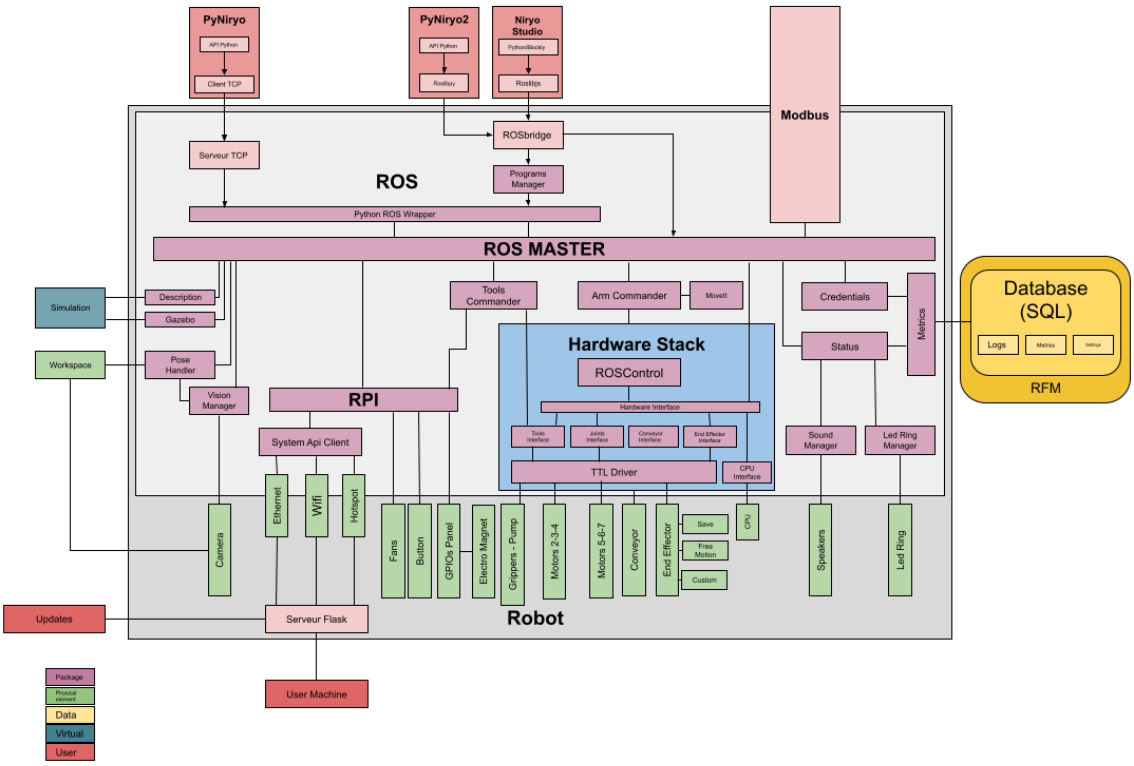
[2] https://docs.microsoft.com/en-us/windows/wsl/compare-versions

[3] https://docs.microsoft.com/en-us/windows/wsl/install-win10

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.



Niryo robot v3 software

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 Stack is split into two parts:

- The High Level Packages (motion planner, vision, …), developed in Python to give good readability
- The Low Level Packages (drivers, hardware management, …), developed in C++ to ensure real time capabilities.

Note

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

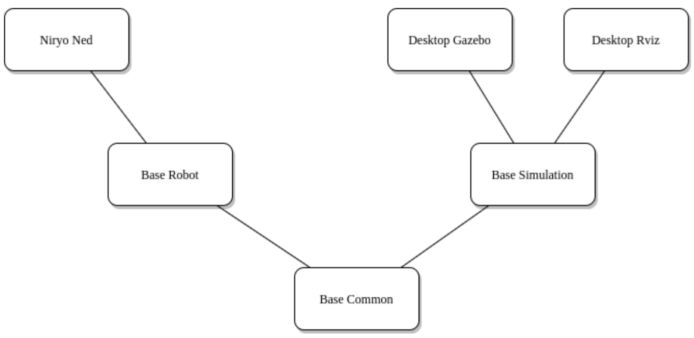
High Level Packages

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

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.



Bring Up Launch Files' organization

On RaspberryPI

One

The file **niryo_one_robot.launch** allows to launch ROS on a Raspberry Pi 3.
This file is automatically launched when Niryo One boots (Niryo One RPi3B image).

Command to launch Niryo One's ROS Stack:

```
roslaunch niryo_robot_bringup niryo_none_robot.launch
```

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
```

Ned2

The file **niryo_ned2_robot.launch** allows to launch ROS on a Raspberry Pi 4.
This file is automatically launched when Ned2 boots (Ned2 RPi4B image).

Command to launch Ned2's ROS Stack:

```
roslaunch niryo_robot_bringup niryo_ned2_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
```

To run it with a specific hardware version, use the command:

```
roslaunch niryo_robot_bringup desktop_gazebo_simulation.launch hardware_version:=ned  # one, ned2
```

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
```

To run it with a specific hardware version, use the command:

```
roslaunch niryo_robot_bringup desktop_rviz_simulation.launch hardware_version:=ned  # one, ned2
```

Notes - Ned Bringup

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

The following files are used to configure the robot logs:

- *desktop_gazebo_simulation_trace.conf*
- *desktop_rviz_simulation_trace.conf*
- *niryo_robot_trace.conf*

Niryo_robot_arm_commander

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

It is composed of only one node, which runs 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.

It belongs to the ROS namespace: `/niryo_robot_arm_commander/` .

Parameters - Commander

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
- python-numpy
- ros_controllers
- rosbridge_server
- sensor_msgs

- std_msgs
- tf2_web_republisher
- trajectory_msgs

Action files - Commander

RobotMove

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

Services files - Commander

GetFK

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

GetIK

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

JogShift

```
int32 JOINTS_SHIFT = 1
int32 POSE_SHIFT = 2

int32 cmd

float32[] shift_values

---
int32 status
string message
```

Messages files - Commander

ArmMoveCommand

```
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 DRAW_CIRCLE = 10
int32 EXECUTE_FULL_TRAJ = 11
int32 EXECUTE_RAW_TRAJ = 12

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

trajectory_msgs/JointTrajectory trajectory

float64[] args
```

PausePlanExecution

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

float64 PAUSE_TIMEOUT = 30.0

int8 state
```

ShiftPose

```
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
```

To display other Niryo robots on Rviz:

```
roslaunch niryo_robot_description display.launch hardware_version:=ned2  # one, ned
```

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

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 hardware_version:=ned  # one, ned2
```

Niryo_robot_msgs

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

Niryo messages

Ned Messages

Name	Description
BusState	TTL bus state
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 information
SoftwareVersion	several software version

Niryo services

Ned Services

Name	Description
GetBool	Return a bool
GetInt	Return a integer
GetNameDescriptionList	Return a name list and a description list
GetString	Return a string
GetStringList	Return a list of string
Ping	Used to ping APIs

Name	Description
SetBool	Set a bool and return status
SetFloat	Set a float 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

BusState

```
std_msgs/Header header
bool connection_status
uint8[] motor_id_connected
string error
```

CommandStatus

```
int32 val

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

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

int32 BAD_HARDWARE_VERSION = -10
int32 ROS_ERROR = -20

int32 FILE_ALREADY_EXISTS = -30
int32 FILE_NOT_FOUND = -31


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 TTL_WRITE_ERROR = -118
int32 TTL_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 COLLISION = -226

int32 PAUSE_TIMEOUT= -227
int32 CANCEL_PAUSE= -228

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 DYNAMIC_FRAME_EDIT_FAILED = -305
int32 DYNAMIC_FRAME_CREATION_FAILED = -306
int32 CONVERT_FAILED = -307

int32 WORKSPACE_CREATION_FAILED = -308

# - Trajectory Handler
```

```
# - Trajectory Handler
int32 TRAJECTORY_HANDLER_CREATION_FAILED = -310
int32 TRAJECTORY_HANDLER_REMOVAL_FAILED = -311
int32 TRAJECTORY_HANDLER_RENAME_FAILURE = -312
int32 TRAJECTORY_HANDLER_EXECUTE_REGISTERED_FAILURE = -313
int32 TRAJECTORY_HANDLER_EXECUTE_FAILURE = -314
int32 TRAJECTORY_HANDLER_GET_TRAJECTORY_FAILURE = -315
int32 TRAJECTORY_HANDLER_GET_TRAJECTORY_LIST_FAILURE = -316

# - 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


# - Credentials
int32 CREDENTIALS_FILE_ERROR = -400
int32 CREDENTIALS_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
int32 SYSTEM_API_CLIENT_UNKNOWN_COMMAND = -443
int32 SYSTEM_API_CLIENT_COMMAND_FAILED = -444

# - Database
int32 DATABASE_DB_ERROR = -460
int32 DATABASE_SETTINGS_UNKNOWN = -461
int32 DATABASE_SETTINGS_TYPE_MISMATCH = -462
int32 DATABASE_FILE_PATH_UNKNOWN = -463


# - Reports
int32 REPORTS_UNABLE_TO_SEND = -470
int32 REPORTS_SENDING_FAIL = -471
int32 REPORTS_FETCHING_FAIL = -472
int32 REPORTS_SERVICE_UNREACHABLE = -473

# - Sound interface
int32 SOUND_FILE_NOT_FOUND = -500
int32 PROTECTED_SOUND_NAME = -501
int32 INVALID_SOUND_NAME = -502
int32 INVALID_SOUND_FORMAT = -503
int32 SOUND_TIMEOUT = -504

# - I2C interface
int32 MISSING_I2C = -510
```

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

geometry_msgs/Twist twist
float64 tcp_speed
```

RPY

```
# roll, pitch and yaw

float64 roll
float64 pitch
float64 yaw
```

HardwareStatus

```
std_msgs/Header header

# Raspberry Pi board
int32 rpi_temperature

# Ned, One, ....
string hardware_version

# Hardware State
int8 ERROR = -1
int8 NORMAL = 0
int8 DEBUG = 1
int8 REBOOT = 2

int8 hardware_state

# 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 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
```

GetString

```
---
string value
```

GetStringList

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

Ping

```
string name
bool state
---
```

SetBool

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

SetFloat

```
float32 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.

It belongs to the ROS namespace: `/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

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 introduced 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. The camera detects objects 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
`dynamic_frame_dir`	Path to the dynamic frames 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

Name	Message Type	Description
get_workspace_list	GetNameDescriptionList	Get list of workspaces name & description
get_workspace_poses	GetWorkspaceRobotPoses	Get workspace's robot poses
get_workspace_points	GetWorkspacePoints	Get workspace's robot points
get_workspace_matrix_poses	GetWorkspaceMatrixPoses	Get workspace's robot matrix 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_dynamic_frame	ManageDynamicFrame	Save/Edit/Delete a dynamic frame
get_dynamic_frame_list	GetNameDescriptionList	Get list of dynamic frame
get_dynamic_frame	GetDynamicFrame	Get dynamic frame
get_transform_pose	GetTransformPose	Get transform between two frames

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

GetDynamicFrame (Service)

```
string name
---
int32 status
string message
niryo_robot_poses_handlers/DynamicFrame dynamic_frame
```

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
```

GetTransformPose (Service)

```
string source_frame
string local_frame

geometry_msgs/Point position
niryo_robot_msgs/RPY rpy
---
int32 status
string message
geometry_msgs/Point position
niryo_robot_msgs/RPY rpy
```

GetWorkspaceMatrixPoses (Service)

```
string name
---
int32 status
string message
geometry_msgs/Point[] matrix_position
geometry_msgs/Quaternion[] matrix_orientation
```

GetWorkspacePoints (Service)

```
string name
---
int32 status
string message
geometry_msgs/Point[] points
```

GetWorkspaceRatio (Service)

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

GetWorkspaceRobotPoses (Service)

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

ManageDynamicFrame (Service)

```
int32 SAVE = 1
int32 SAVE_WITH_POINTS = 2
int32 EDIT = 3
int32 DELETE = -1

int32 cmd

niryo_robot_poses_handlers/DynamicFrame dynamic_frame

---
int32 status
string message
```

ManagePose (Service)

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

niryo_robot_poses_handlers/NiryoPose pose
---
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
```

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 is made to run its programs with the Python ROS Wrapper.

It belongs to the ROS namespace: `/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

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 files - Programs manager

ExecuteProgram

```
bool execute_from_string

string name
string code_string

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

GetProgram

```
string name

niryo_robot_programs_manager/ProgramLanguage language
---
int32 status
string message

string code
string description
```

GetProgramAutorunInfos

```
---
int32 status
string message

niryo_robot_programs_manager/ProgramLanguage language
string name

# Mode
int8 ONE_SHOT = 1
int8 LOOP = 2

int8 mode
```

GetProgramList

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

ManageProgram

```
# 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

```
# 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
```

Messages files - Programs manager

ProgramIsRunning

```
bool program_is_running

int8 EXECUTION_ERROR = -2
int8 FILE_ERROR = -1
int8 NONE = 0
int8 PREEMPTED = 1
int8 SUCCESS = 2

int8 last_execution_status
string last_execution_msg
```

ProgramLanguage

```
int8 NONE = -1

int8 ALL = 0

# Runnable
int8 PYTHON2 = 1
int8 PYTHON3 = 2

# Not Runnable
int8 BLOCKLY = 66

int8 used
```

ProgramLanguageList

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

ProgramList

```
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.
- Analog I/O panel: gets commands and sends the current state of analog I/Os.
- End Effector I/O panel: gets commands and sends the current state of the digital I/Os of the end effector panel on Ned2. Also controls tools like the Electromagnet.
- Robot fans.
- Led: sets the LED color.
- Shutdown Manager: shutdown or reboot the Raspberry.

- ROS log: can remove all previous logs on start_up to prevent a lack of disk space in the long run (SD cards do not have infinite storage).

It belongs to the ROS namespace: `/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	Publishes the current execution state launched when button is pressed
`/niryo_robot/blockly/save_current_point`	std_msgs/Int32	Publishes current point when user is in Blockly page to save block by pressing button
`/niryo_robot/rpi/is_button_pressed`	std_msgs/Bool	Publishes the button state (true if pressed)
`digital_io_state`	DigitalIOState	Publishes the digital I/Os state by giving for each it's pin / name / mode / state
`analog_io_state`	AnalogIOState	Publishes the analog I/Os state by giving for each it's pin / name / mode / state
`/niryo_robot/rpi/led_state`	std_msgs/Int8	Publishes the current LED color
`ros_log_status`	LogStatus	Publishes 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	Shutdowns the Raspberry Pi
`/niryo_robot/rpi/change_button_mode`	SetInt	Changes top button mode (autorun program, blockly, nothing, …)
`get_analog_io`	GetAnalogIO	Gets analog IO state list
`get_digital_io`	GetDigitalIO	Gets digital IO state list
`set_analog_io`	SetAnalogIO	Sets an analog IO to the given value
`set_digital_io`	SetDigitalIO	Sets a digital IO to the given value
`set_digital_io_mode`	SetDigitalIO	Sets a digital IO to the given mode
`set_led_state`	std_msgs/SetInt	Sets LED state
`set_led_custom_blinker`	LedBlinker	Sets the LED in blink mode with the color given
`purge_ros_logs`	SetInt	Purges ROS log
`set_purge_ros_log_on_startup`	SetInt	Modifies the permanent settings that tell if the robot should purge its ROS log at each boot

Dependencies - Raspberry Pi

- std_msgs
- actionlib_msgs
- sensor_msgs
- Niryo_robot_msgs
- Niryo_robot_arm_commander
- Adafruit-GPIO==1.0.3
- Adafruit-PureIO==1.0.1
- Adafruit-BBIO==1.0.9
- Adafruit-ADS1x15==1.0.2
- board==1.0
- smbus==1.1.post2
- smbus2==0.4.1
- spidev==3.5

Services files - Raspberry Pi

ChangeMotorConfig (Service)

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

GetAnalogIO (Service)

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

float64 value
```

GetDigitalIO (Service)

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

bool value
```

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)

```
string name
bool value

---
int32 status
string message
```

SetAnalogIO (Service)

```
string name
float64 value
---
int32 status
string message
```

SetIOMode (Service)

```
string name

int8 OUTPUT = 0
int8 INPUT = 1
int8 mode

---
int32 status
string message
```

SetPullup (Service)

```
string name
bool enable

---
int32 status
string message
```

Messages files - Raspberry Pi

AnalogIO

```
string name
float64 value
```

AnalogIOState (Topic)

```
niryo_robot_rpi/AnalogIO[] analog_inputs
niryo_robot_rpi/AnalogIO[] analog_outputs
```

DigitalIO

```
string name
bool value
```

DigitalIOState (Topic)

```
niryo_robot_rpi/DigitalIO[] digital_inputs
niryo_robot_rpi/DigitalIO[] digital_outputs
```

LogStatus (Topic)

```
std_msgs/Header header

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

Niryo_robot_sound

This package deals with the sound of the robot.

Sound Node

The ROS Node is made of services to play, stop, import and delete a sound on the robot. It is also possible to set the volume of the robot.

It belongs to the ROS namespace: `/niryo_robot_sound/` .

Parameters - Sound

Here is a list of the different parameters that allow you to adjust the default settings of the robot and the system sounds.

Parameters of the volume Sound component

Name	Description	Default value
`default_volume`	Default volume on the real robot	100
`default_volume_simulation`	Default volume in simulation	10
`min_volume`	Minimum volume of the robot	0
`max_volume`	Maximum volume of the robot	200
`volume_file_path`	File where the volume of the real robot set by the user is stored	"~/niryo_robot_saved_files/robot_sound_volume.txt"
`volume_file_path_simulation`	File where the volume in simulation set by the user is stored	"~/.niryo/simulation/robot_sound_volume.txt"

Parameters of the Sound component

Name	Description	Default value
`path_user_sound`	Default volume on the real robot	"~/niryo_robot_saved_files/niryo_robot_user_sounds"
`path_user_sound_simulation`	Default volume in simulation	"~/.niryo/simulation/niryo_robot_user_sounds"
`path_robot_sound`	Minimum volume of the robot	"niryo_robot_state_sounds"
`robot_sounds/error_sound`	Sound played when an error occurs	error.wav
`robot_sounds/turn_on_sound`	Sound played at the start-up of the robot	booting.wav
`robot_sounds/turn_off_sound`	Sound played at shutdown	stop.wav
`robot_sounds/connection_sound`	Sound played an Niryo Studio connection	connected.wav
`robot_sounds/robot_ready_sound`	Sound played when the robot is ready	ready.wav
`robot_sounds/calibration_sound`	Sound played at start of calibration	calibration.wav

State sounds

State	Description	Sound
Booting	Sound played while booting	Your browser does not support the `audio` element.
Ready	Sound played when the robot is ready after booting	Your browser does not support the `audio` element.
Calibration	Sound played at start of calibration	Your browser does not support the `audio` element.
Connected	Notify of a connection to Niryo Studio	Your browser does not support the `audio` element.
Reboot	Sound played at start of a motor reboot	Your browser does not support the `audio` element.
Warn	Sound played when a warning occurs	Your browser does not support the `audio` element.
Error	Sound played when a robot/motor/raspberry/program/overheating error occurs	Your browser does not support the `audio` element.
Shutdown	Sound played at shutdown	Your browser does not support the `audio` element.

Publisher - Sound

Sound Package's Publishers

Name	Message Type	Description
`/niryo_robot_sound/sound`	std_msgs/String	Publisesh the sound being played
`/niryo_robot_sound/volume`	std_msgs/UInt8	Publishes the volume of the robot
`/niryo_robot_sound/sound_database`	SoundList	Publishes the sounds (and their duration) on the robot

Services - Sound

Sound Services

Name	Message Type	Description
`/niryo_robot_sound/play`	PlaySound	Plays a sound from the robot database
`/niryo_robot_sound/stop`	Trigger	Stops the sound being played
`/niryo_robot_sound/set_volume`	SetInt	Sets the volume percentage between 0 and 200%
`/niryo_robot_sound/text_to_speech`	TextToSpeech	Pronouncses a sentence via GTTS
`/niryo_robot_sound/manage`	ManageSound	Stops a sound being played

Subscribers - Sound

Sound Package subscribers

Topic name	Message type	Description
`/niryo_robot_status/robot_status`	RobotStatus	Retrieves the current robot status, and controls the sound accordingly (see Niryo_robot_status section)
`/niryo_studio_connection`	std_msgs/Empty	Catches Niryo Studio's connection to make a sound.

Dependencies - Sound

- std_msgs
- niryo_robot_msgs
- niryo_robot_status

Services & Messages files - Sound

SoundList (Message)

```
niryo_robot_sound/SoundObject[] sounds
```

SoundObject (Message)

```
string name
float64 duration
```

ManageSound (Service)

```
string sound_name

int8 ADD = 1
int8 DELETE = 2
int8 action

# Data to add a new sound
string sound_data

---
int32 status
string message
```

PlaySound (Service)

```
string sound_name

float64 start_time_sec
float64 end_time_sec  #if 0 or if end_time_sec>sound_duration the entire sound will be played

bool wait_end

---
int32 status
string message
```

TextToSpeech (Service)

```
string text # < 100 char


int8 ENGLISH = 0
int8 FRENCH = 1
int8 SPANISH = 3
int8 MANDARIN = 4
int8 PORTUGUESE = 5

int8 language
---
bool success
string message
```

Sound API functions

In order to control the robot 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 playing sound via Python ROS Wrapper will look like:

```
from niryo_robot_led_ring.api import SoundRosWrapper

sound = SoundRosWrapper()
sound.play(sound.sounds[0])
```

This class allows you to control the sound of the robot via the internal API.

List of functions subsections:

- Play sound
- Sound database

Play sound

class **SoundRosWrapper**(*hardware_version='ned2'*, *service_timeout=1*)

play(*sound_name*, *wait_end=True*, *start_time_sec=0*, *end_time_sec=0*)

Play a sound from the robot If failed, raise NiryoRosWrapperException

Parameters:
- **sound_name** (*str*) – Name of the sound to play
- **start_time_sec** (*float*) – start the sound from this value in seconds
- **end_time_sec** (*float*) – end the sound at this value in seconds
- **wait_end** (*bool*) – wait for the end of the sound before exiting the function

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_volume(*sound_volume*)

Set the volume percentage of the robot. If failed, raise NiryoRosWrapperException

Parameters: **sound_volume** (*int*) – volume percentage of the sound (0: no sound, 100: max sound)

Returns: status, message

Return type: (int, str)

stop()

Stop a sound being played. If failed, raise NiryoRosWrapperException

Returns: status, message

Return type: (int, str)

say(*text*, *language=0*)

Use gtts (Google Text To Speech) to interpret a string as sound Languages available are: - English: 0 - French: 1 - Spanish: 2 - Mandarin: 3 - Portuguese: 4

Parameters:
- **text** (*string*) – text to speek < 100 char
- **language** (*int*) – language of the text

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))

Sound database

class **SoundRosWrapper**(*hardware_version='ned2'*, *service_timeout=1*)

sounds

Get sound name list

Returns: list of the sounds of the robot

Return type: list[string]

delete_sound(*sound_name*)

Delete a sound on the RaspberryPi of the robot. If failed, raise NiryoRosWrapperException

Parameters: **sound_name** (*str*) – name of the sound which needs to be deleted

Returns: status, message

Return type: (int, str)

import_sound(*sound_name*, *sound_data*)

Delete a sound on the RaspberryPi of the robot. If failed, raise NiryoRosWrapperException

Parameters:
- **sound_name** (*str*) – name of the sound which needs to be deleted
- **sound_data** (*str*) – String containing the encoded data of the sound file (wav or mp3)

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))

get_sound_duration(*sound_name*)

Returns the duration in seconds of a sound stored in the robot database raise SoundRosWrapperException if the sound doesn't exists

Parameters: **sound_name** (*str*) – name of sound

Returns: sound duration in seconds

Return type: float

Niryo_robot_status

Robot status Node

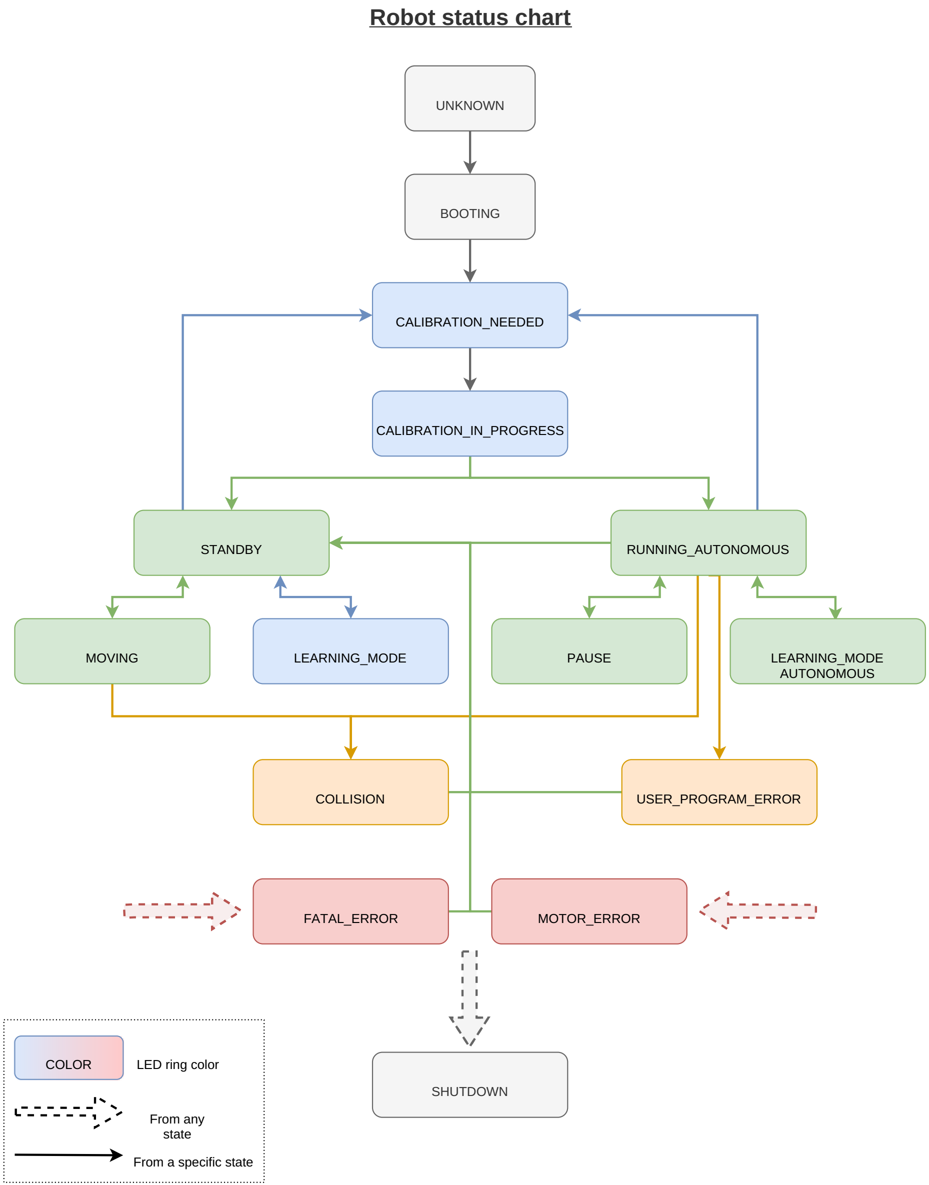
The ROS Node is listening to the topics of the robot to deduce the current state of the robot. It manages the status of the robot, the status of the logs and informs about the overheating of the Raspberry PI and the out of limit joints.

It belongs to the ROS namespace: `/niryo_robot_status/` .

Niryo Robot Status Table

Name	Description	Troubleshoot
`SHUTDOWN`	The robot is being shut down	
`FATAL_ERROR`	ROS crash	Please restart the robot
`MOTOR_ERROR`	Motor voltage error, overheating, overload	Check the error code on Niryo Studio. Restart the robot and check the wiring. If the problem persists, contact customer service
`COLLISION`	Arm collision detected	Restart your movement or switch to learning mode to remove this error.
`USER_PROGRAM_ERROR`	User program error	Launch a movement or switch to learning mode to remove this error.
`UNKNOWN`	Node not initialized	
`BOOTING`	ROS a and the Raspberry are booting up	If the startup seems to timeout, restart the robot electrically. If the problem persists, update the robot with ssh, change the SD card or contact customer service.
`UPDATE`	Robot update in progress	Just wait and be patient :)
`CALIBRATION_NEEDED`	New calibration requested	Run a new calibration before processing any movement.
`CALIBRATION_IN_PROGRESS`	Calibration in progress	If the calibration fails or takes longer than 30 seconds. The status will return to `CALIBRATION_NEED.`
`LEARNING_MODE`	Free motion enabled, the torques are disabled	
`STANDBY`	Free motion disabled, the torques are enabled and no user program is running	
`MOVING`	A single motion or jog is being processed and no user program is running	
`RUNNING_AUTONOMOUS`	A user program is running and the torques are enabled	

Name	Description	Troubleshoot
RUNNING_DEBUG	A debug procedure is running	A short press on the top button cancels it.
PAUSE	User program error	A short press on the top button resumes the program, a long press (on Ned2) or a double press (on Ned and One) cancels the program execution. After 30 seconds, the program stops automatically.
LEARNING_MODE_AUTONOMOUS	A user program is running and the torques are disabled	



Niryo Robot Status Diagram

Publisher - Robot Status

Robot Status Package's Publishers

Name	Message Type	Latch Mode	Description
/niryo_robot_status/robot_status	RobotStatus	True	Publish the robot, log, overheating and out of bounds status.

Services - Robot Status

Robot Status Services

Name	Message Type	Description
/niryo_robot_status/advertise_shutdown	Trigger	Notify of a shutdown request

Subscribers - Robot Status

Robot Status Package subscribers

Topic name	Message type	Description
/niryo_robot_hardware_interface/hardware_status	HardwareStatus	Detection of a motor or end effector panel error, raspberry overheating
niryo_robot_rpi/pause_state	PausePlanExecution	Detection of the pause state
/niryo_robot_arm_commander/is_active	std_msgs/Bool	Detection of a motion
/niryo_robot_arm_commander/is_debug_motor_active	std_msgs/Bool	Detection of a debug procedure
/niryo_robot/jog_interface/is_enabled	std_msgs/Bool	Detection of a jog motion
/niryo_robot_programs_manager/program_is_running	ProgramIsRunning	Detection of a user program
/niryo_robot_user_interface/is_client_connected	std_msgs/Bool	Detection of a pyniryo user
/niryo_robot/learning_mode/state	std_msgs/Bool	Detection of the free motion mode
/niryo_robot_arm_commander/collision_detected	std_msgs/Bool	Detection of collision
/joint_states	sensor_msgs/JointState	Get the joint state in order to detect an out of bounds
/ping_pyniryo	std_msgs/Bool	Detection of a pyniryo2 user

Dependencies - Robot Status

- std_msgs
- sensor_msgs
- niryo_robot_msgs
- niryo_robot_programs_manager
- niryo_robot_arm_commander

Messages files - Robot Status

RobotStatus

```
int8 UPDATE=-7
int8 REBOOT=-6
int8 SHUTDOWN=-5
int8 FATAL_ERROR=-4     # Node crash
int8 MOTOR_ERROR=-3     # Electrical/overload or disconnected motor error
int8 COLLISION=-2
int8 USER_PROGRAM_ERROR=-1
int8 UNKNOWN=0
int8 BOOTING=1          # Robot is booting
int8 REBOOT_MOTOR=2
int8 CALIBRATION_NEEDED=3
int8 CALIBRATION_IN_PROGRESS=4
int8 LEARNING_MODE=5
int8 STANDBY=6          # Torque ON
int8 MOVING=7           # Moving with NiryoStudio interface or ros topics without user program
int8 RUNNING_AUTONOMOUS=8   # User program is running
int8 RUNNING_DEBUG=9    # Debug program is running
int8 PAUSE=10           # User program paused
int8 LEARNING_MODE_AUTONOMOUS=11    # User program is running + Learning mode activated
int8 LEARNING_TRAJECTORY = 12
int8 REBOOT_MOTOR=13

int8 robot_status
string robot_status_str
string robot_message

int8 FATAL=-3
int8 ERROR=-2
int8 WARN=-1
int8 NONE=0

int8 logs_status
string logs_status_str
string logs_message

bool out_of_bounds
bool rpi_overheating
```

Niryo_robot_system_api_client

This packages handle the flask server requests to manage:

- Robot name
- Wifi settings
- Ethernet settings

Publisher - System API Client

System API Client Package's Publishers

Name	Message Type	Description
/niryo_robot/wifi/status	WifiStatus	Publish the current wifi status

Services - System API Client

System API Client Services

Name	Message Type	Description
`/niryo_robot/wifi/set_robot_name`	SetString	Change the robot name
`/niryo_robot/wifi/manage`	ManageWifi	Change the wifi hotspot mode
`/niryo_robot/ethernet/manage`	ManageEthernet	Change the ethernet setup (ip address, netmask, gateway, dhcp) based on nmcli interface.

Services files - System API Client

ManageEthernet (Service)

```
int8 STATIC = 1
int8 AUTO = 2
int8 CUSTOM = 3

int8 profile

# Only for the custom profile
string ip       # ex: '192.168.1.73'
string mask     # ex: '255.255.255.0'
string gateway  # ex: '192.168.1.1'
# Optional
string dns      # ex: '8.8.8.8 4.4.4.4' separated by spaces

---
int32 status
string message
```

ManageWifi (Service)

```
int8 HOTSPOT = 0
int8 RESTART = 1
int8 DEACTIVATE = 2
int8 RECONNECT = 3

int8 cmd
---
int32 status
string message
```

Messages files - System API Client

WifiStatus (Message)

```
int8 UNKNOWN = 0
int8 HOTSPOT = 1
int8 DISABLED = 2
int8 CONNECTED = 3
int8 DISCONNECTED = 4

int8 status
```

Niryo robot tools commander

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.

It belongs to the ROS namespace: `/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

Tools Package Publishers

Name	Message Type	Description
`current_id`	std_msgs/Int32	Publishes the current tool ID
`tcp`	TCP	Publishes 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	Pings/scans for a dxl motor flashed with an ID corresponding to a tool and equip it (if found)
`equip_electromagnet`	SetInt	Equips the electromagnet with the motor ID given as parameter
`enable_tcp`	SetBool	Enables or disablse 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	Activates the TCP (Tool Center Point) functionality and defines a new TCP transformation.
`reset_tcp`	std_srvs/Trigger	Resets 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
int8 OPEN_GRIPPER = 1
int8 CLOSE_GRIPPER = 2

# Vacuump pump
int8 PULL_AIR_VACUUM_PUMP = 10
int8 PUSH_AIR_VACUUM_PUMP = 11

# Tools controlled by digital I/Os
int8 SETUP_DIGITAL_IO = 20
int8 ACTIVATE_DIGITAL_IO = 21
int8 DEACTIVATE_DIGITAL_IO = 22

uint8 cmd_type

# Gripper1= 11, Gripper2=12, Gripper3=13, VaccuumPump=31, Electromagnet=30
int8 tool_id

# if gripper Ned1/One
uint16 speed

# if gripper Ned2
uint8 max_torque_percentage
uint8 hold_torque_percentage

# if vacuum pump or electromagnet grove
bool activate

# if tool is set by digital outputs (electromagnet)
string 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_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 and 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

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*

`GET_TRAJECTORY_SAVED`= *80*

`GET_SAVED_TRAJECTORY_LIST`= *81*

`EXECUTE_REGISTERED_TRAJECTORY`= *82*

`EXECUTE_TRAJECTORY_FROM_POSES`= *83*

`EXECUTE_TRAJECTORY_FROM_POSES_AND_JOINTS`= *84*

`SAVE_TRAJECTORY`= *85*

`SAVE_LAST_LEARNED_TRAJECTORY`= *86*

`UPDATE_TRAJECTORY_INFOS`= *87*

`DELETE_TRAJECTORY`= *88*

`CLEAN_TRAJECTORY_MEMORY`= *89*

`GET_SAVED_DYNAMIC_FRAME_LIST`= *95*

`GET_SAVED_DYNAMIC_FRAME`= *96*

`SAVE_DYNAMIC_FRAME_FROM_POSES`= *97*

`SAVE_DYNAMIC_FRAME_FROM_POINTS`= *98*

`EDIT_DYNAMIC_FRAME`= *99*

`DELETE_DYNAMIC_FRAME`= *100*

`MOVE_RELATIVE`= *101*

`MOVE_LINEAR_RELATIVE`= *102*

`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*

`ANALOG_WRITE`= *155*

`ANALOG_READ`= *156*

`GET_ANALOG_IO_STATE`= *157*

`CUSTOM_BUTTON_STATE`= *158*

`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*

`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*

`PLAY_SOUND`= *240*

`SET_VOLUME`= *241*

`STOP_SOUND`= *242*

`DELETE_SOUND`= *243*

`IMPORT_SOUND`= *244*

`GET_SOUNDS`= *245*

`GET_SOUND_DURATION`= *246*

`SAY`= *247*

`LED_RING_SOLID`= *250*

`LED_RING_TURN_OFF`= *251*

`LED_RING_FLASH`= *252*

`LED_RING_ALTERNATE`= *253*

`LED_RING_CHASE`= *254*

`LED_RING_WIPE`= *255*

`LED_RING_RAINBOW`= *256*

`LED_RING_RAINBOW_CYCLE`= *257*

`LED_RING_RAINBOW_CHASE`= *258*

`LED_RING_GO_UP`= *259*

> **LED_RING_GO_UP_DOWN**= *260*
>
> **LED_RING_BREATH**= *261*
>
> **LED_RING_SNAKE**= *262*
>
> **LED_RING_CUSTOM**= *263*
>
> **LED_RING_SET_LED**= *264*

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*

It belongs to the ROS namespace: `/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

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
`visualization`	Visualization	Add visuals markers of objects detected by the vision kit to rviz

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
```

Visualization (Service)

```
string workspace
bool clearing
---
int32 status
```

Niryo_robot_led_ring

This package is the one managing the LED Ring of Ned2.

It is composed of one node, receiving commands and the current robot status, and publishing LED Ring states.

The LED Ring is composed of 30 WS2811 RGB LEDs, controlled by the package with the rpi_ws281x library.

LED Ring node

The ROS Node is made to manage the LED Ring state, and to publish its currents status and state on ROS topics. It uses a class implementing several animation (11 for now), allowing to control the LED Ring or to display the current robot status. The LED Ring is also implemented in Rviz.

The LED Ring can either be:

- in **ROBOT STATUS** mode: the LED is displaying the status of the robot.
- **in USER mode: the user can control the LED Ring with the several methods implemented, through**

  Blockly , Pyniryo or Python ROS Wrapper .

Robot status mode

When displaying the **robot status**, the LED Ring has several states which represent different modes and error status. Refer to the following table. The node subscribes to the ROS topic `/niryo_robot_status/robot_status` , published by the package RobotStatus (index.html#robotstatus).

Animation and color	Description	Troubleshooting
White Breath	Robot is booting	N/A
Blue Chase	Calibration is needed	Press the *Custom* button, or launch a calibration
Blue Snake	Calibration in progress	N/A
Blue Breath	Free Motion enabled	N/A
3 Yellow Flashing	Calibration start	N/A
Green Breath	Free Motion disabled, torque enabled	N/A
Solid Green	Program in progress	N/A
Green Chase	Program paused	Long press on the TOP button to cancel the program, short press to resume
Orange Breath	Program execution error	Launch a new action to clear this state
Flashing Orange	Collision	Launch a new action to clear this state
Solid Orange	Joint out of bounds	Switch to Free Motion mode to bring the joints within limits.
1 Purple Flashing	New connection form Niryo Studio	N/A
2 Purple Flashing	Save a robot positions from the 'Save' button	N/A
Flashing Red	Motor error / Raspberry overheating	Please check the error on Niryo Studio.
Solid Red	ROS Crash	Please restart the robot.

User mode

Several animations are implemented to allow the user different ways to control the LED Ring. Refer to the following table. The node receives commands through the service `/niryo_robot_led_ring/set_user_animation` (see the service section)

> **Important**
>
> Ned must be in autonomous mode in order to allow the user to control the LED Ring.

Animation	Appearance	Gif
None	LEDs are turned off	
Solid	Set the whole LED Ring to the same color at once	
Flashing	Flashes a color according to a frequency	

Animation	Appearance	Gif
Alternate	The different colors are alternated one after the other.	
Chase	Movie theater light style chase animation.	
Color Wipe	Wipe a color across the LED Ring. Similar to go_up, but LEDs are not turned off at the end.	
Rainbow	Draws rainbow that fades across all LEDs at once.	
Rainbow cycle	Draw rainbow that uniformly distributes itself across all LEDs.	
Rainbow chase	Rainbow chase animation.	

Animation	Appearance	Gif
Go up	LEDs turn on like a loading circle until lighting up the whole LED Ring. and are then all turned off at the same time.	
Go up and down	Like go_up, but LEDs are turned off the same way they are turned on.	
Breath	Variation of light intensity to imitate breathing.	
Snake	Luminous snake that turns around the LED Ring.	

> **Note**
>
> When displaying the robot status, the LED Ring commander uses those methods, with the default parameters defined below.

It belongs to the ROS namespace: `/niryo_robot_led_ring/` .

Parameters - LED Ring

Firstly, the LED Ring component, controlled with the rpi_ws281x library (https://github.com/rpi-ws281x/rpi-ws281x-python), through the Python class PixelStrip, is parameterizable. Default parameters are set in the *led_strim_params.yaml* file of the */config* folder of the package

Parameters of the Led Ring component

Name	Description	Default value
`led_count`	Number of LED pixels in the LED Ring	30
`led_pin`	Raspberry Pi GPIO pin connected to the pixels It must support PWM.	13
`led_freq_hs`	LED signal frequency in Hertz	800khz
`led_dma`	DMA channel to use for generating signal	10
`led_brightness`	LEDs brightness. Set to 0 for darkest and 255 for brightest	255
`led_invert`	True to invert the signal (when using NPN transistor level shift)	True
`led_channel`	the PWM channel to use	0

Another configuration file, the *led_ring_params.yaml*, sets the default parameters of LED Ring animations.

Parameters of the LED Ring animations

Name	Description	Default value
`default_flashing_period`	Default Flashing animation period in seconds	0.25
`default_alternate_period`	Default Alternate animation period in seconds	1
`default_chase_period`	Default Chase animation period in seconds	4
`default_colorwipe_period`	Default Wipe animation period in seconds	5
`default_rainbow_period`	Default Rainbow animation period in seconds	5
`default_rainbowcycle_period`	Default Rainbow cycle animation period in seconds	5
`default_rainbowchase_period`	Default Rainbow chase animation period in seconds	5
`default_goup_period`	Default Go up animation period in seconds	5
`default_goupanddown_period`	Default Go up and down animation period in seconds	5
`default_breath_period`	Default Breath animation period in seconds	4
`default_snake_period`	Default Snake animation period in seconds	1.5
`led_offset`	Offset ID between the LED with the ID 0 and the ID of the LED at the back of the robot.	8
`simulation_led_ring_markers_publish_rate`	Rviz LED ring markers publishinf rate in simulation mode	20
`led_ring_markers_publish_rate`	Rviz LED ring markers publishing rate on a real robot	5

Services - LED Ring

The ROS node implements one service, designed for the user to control the LED Ring.

LED Ring Package services

Name	Message type	Description
`set_user_animation`	LedUser	Allows user to control the LED Ring, with implemented animations. A new request will **interrupt** the previous one, if still playing. Depending on the `wait` boolean field and the `iterations` field of the request, the service will either **answer immediately** after launching the animation, or **wait for the animation to finish** to answer.
`set_led_color`	SetLedColor	Lights up a LED identified by an ID

Publishers - LED Ring

LED Ring Package publishers

Name	Message type	Description
`led_ring_status`	LedRingStatus	Publishes the **status** of the LED Ring, providing information on the **current mode** (displaying robot status or controlled by user if the robot works in AUTONOMOUS mode), the **current animation played** and the animation color (except for rainbow methods, where the animation color is not defined). Publishes every time at least **one field changed**.
`visualization_marker_array`	visualization_msgs/MarkerArray	Publishes shapes representing LEDs when Ned is used in simulation with **Rviz**, as a list of 30 visualization_msgs/Marker of size 30.

Subscribers - LED Ring

LED Ring Package subscribers

Topic name	Message type	Description
`/niryo_robot_status/robot_status`	RobotStatus	Retrieves the current robot status, and control LED accordingly (see Niryo_robot_status section)
`/niryo_robot/blockly/save_current_point`	std_msgs/Int32	Catches the 'Save Point' action to make the LED ring blink.
`/niryo_studio_connection`	std_msgs/Empty	Catches the Niryo Studio connection to make the LED ring blink.

Dependencies - LED Ring

- niryo_robot_msgs
- std_msgs
- visualization_msgs
- rpi_ws281x==4.3.0

Services files - LED Ring

LedUser (Service)

```
niryo_robot_led_ring/LedRingAnimation animation_mode

std_msgs/ColorRGBA[] colors
float64 period         # Time of 1 iteration in seconds
int16 iterations

# The service either wait for the iterations to finish to answer,
# or answer immediatly a Success after launching the function of Led Ring control.
# if iterations is 0, answer immediatly in any case, because the function never
# stops.
bool wait_end

---
int32 status
string message
```

SetLedColor (Service)

```
int8 led_id
std_msgs/ColorRGBA color

---
int32 status
string message
```

Messages files - LED Ring

LedRingAnimation (Message)

```
int32 NONE = -1
int32 SOLID = 1
int32 FLASHING = 2
int32 ALTERNATE = 3
int32 CHASE = 4
int32 COLOR_WIPE = 5
int32 RAINBOW = 6
int32 RAINBOW_CYLE = 7
int32 RAINBOW_CHASE = 8
int32 GO_UP = 9
int32 GO_UP_AND_DOWN = 10
int32 BREATH = 11
int32 SNAKE = 12
int32 CUSTOM = 13

int32 animation
```

LedRingCurrentState (Message)

```
Header header
std_msgs/ColorRGBA[] led_ring_colors
```

LedRingStatus (Message)

```
int32 ROBOT_STATUS = 1
int32 USER = 2

int32 led_mode

niryo_robot_led_ring/LedRingAnimation animation_mode

std_msgs/ColorRGBA animation_color # except for rainbow related animation
```

LED Ring API functions

In order to control the robot 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 turning on the LED Ring via Python ROS Wrapper will look like:

```python
from niryo_robot_led_ring.api import LedRingRosWrapper

led_ring = LedRingRosWrapper()
led_ring.solid(color=[255, 255, 255])
```

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

List of functions subsections:

- Custom animations functions
- Pre-made animations functions

Custom animations functions

class **LedRingRosWrapper**(*hardware_version='ned2', service_timeout=1*)

set_led_color(*led_id, color*)

Lights up an LED in one colour. RGB colour between 0 and 255.

Example:

```
from std_msgs.msg import ColorRGBA

led_ring.set_led_color(5, [15, 50, 255])
led_ring.set_led_color(5, ColorRGBA(r=15, g=50, b=255))
```

Parameters:
- **led_id** (*int*) – Id of the led: between 0 and 29
- **color** (*list[float] or ColorRGBA*) – Led color in a list of size 3[R, G, B] or in an ColorRGBA object. RGB channels from 0 to 255.

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))

custom(*led_colors*)

Sends a colour command to all LEDs of the LED ring. The function expects a list of colours for the 30 LEDs of the robot.

Example:

```
led_list = [[i / 30. * 255 , 0, 255 - i / 30.] for i in range(30)]
led_ring.custom(led_list)
```

Parameters: **led_colors** (*list[list[float] or ColorRGBA]*) – List of size 30 of led color in a list of size 3[R, G, B] or in an ColorRGBA object. RGB channels from 0 to 255.

Returns: status, message

Return type: (int, str)

Pre-made animations functions

class **LedRingRosWrapper**(*hardware_version='ned2', service_timeout=1*)

solid(*color, wait=False*)

Sets the whole Led Ring to a fixed color.

Example:

```
from std_msgs.msg import ColorRGBA

led_ring.solid([15, 50, 255])
led_ring.solid(ColorRGBA(r=15, g=50, b=255), True)
```

Parameters:
- **color** (*list[float] or ColorRGBA*) – Led color in a list of size 3[R, G, B] or in an ColorRGBA object. RGB channels from 0 to 255.
- **wait** (*bool*) – The service wait for the animation to finish or not to answer. For this method, the action is quickly done, so waiting doesn't take a lot of time.

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))

turn_off(*wait=False*)

Turns off all Leds

Example:

```
led_ring.turn_off()
```

Parameters: **wait** (*bool*) – The service wait for the animation to finish or not to answer. For this method, the action is quickly done, so waiting doesn't take a lot of time.

Returns: status, message

Return type: (int, str)

flashing(*color, period=0, iterations=0, wait=False*)

Flashes a color according to a frequency. The frequency is equal to 1 / period.

Examples:

```
from std_msgs.msg import ColorRGBA

led_ring.flashing([15, 50, 255])
led_ring.flashing([15, 50, 255], 1, 100, True)
led_ring.flashing([15, 50, 255], iterations=20, wait=True)

frequency = 20  # Hz
total_duration = 10 # seconds
led_ring.flashing(ColorRGBA(r=15, g=50, b=255), 1./frequency, total_duration * frequency , True)
```

Parameters:
- **color** (*list[float] or ColorRGBA*) – Led color in a list of size 3[R, G, B] or in an ColorRGBA object. RGB channels from 0 to 255.
- **period** (*float*) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **iterations** (*int*) – Number of consecutive flashes. If 0, the Led Ring flashes endlessly.
- **wait** (*bool*) – The service wait for the animation to finish all iterations or not to answer. If iterations is 0, the service answers immediately.

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))

alternate(*color_list, period=0, iterations=0, wait=False*)

Several colors are alternated one after the other.

Examples:

```
from std_msgs.msg import ColorRGBA

color_list = [
    ColorRGBA(r=15, g=50, b=255),
    [255, 0, 0],
    [0, 255, 0],
]

led_ring.alternate(color_list)
led_ring.alternate(color_list, 1, 100, True)
led_ring.alternate(color_list, iterations=20, wait=True)
```

Parameters:
- **color_list** (*list[list[float] or ColorRGBA]*) – Led color list of lists of size 3[R, G, B] or ColorRGBA objects. RGB channels from 0 to 255.
- **period** (*float*) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **iterations** (*int*) – Number of consecutive alternations. If 0, the Led Ring alternates endlessly.
- **wait** (*bool*) – The service wait for the animation to finish all iterations or not to answer. If iterations is 0, the service answers immediately.

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))

chase(*color, period=0, iterations=0, wait=False*)

Movie theater light style chaser animation.

Examples:

```
from std_msgs.msg import ColorRGBA

led_ring.chase(ColorRGBA(r=15, g=50, b=255))
led_ring.chase([15, 50, 255], 1, 100, True)
led_ring.chase(ColorRGBA(r=15, g=50, b=255), iterations=20, wait=True)
```

Parameters:
- **color** (*list or ColorRGBA*) – Led color in a list of size 3[R, G, B] or in an ColorRGBA object. RGB channels from 0 to 255.
- **period** (*float*) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **iterations** (*int*) – Number of consecutive chase. If 0, the animation continues endlessly. One chase just lights one Led every 3 Leds.
- **wait** (*bool*) – The service wait for the animation to finish all iterations or not to answer. If iterations is 0, the service answers immediately.

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))

wipe(*color, period=0, wait=False*)

Wipes a color across the LED Ring, light a LED at a time.

Examples:

```
from std_msgs.msg import ColorRGBA

led_ring.wipe(ColorRGBA(r=15, g=50, b=255))
led_ring.wipe([15, 50, 255], 1, True)
led_ring.wipe(ColorRGBA(r=15, g=50, b=255), wait=True)
```

Parameters:
- **color** (*list[float] or ColorRGBA*) – Led color in a list of size 3[R, G, B] or in an ColorRGBA object. RGB channels from 0 to 255.
- **period** (*float*) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **wait** (*bool*) – The service wait for the animation to finish or not to answer.

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))

rainbow(*period=0, iterations=0, wait=False*)

Draws rainbow that fades across all LEDs at once.

Examples:

```
led_ring.rainbow()
led_ring.rainbow(5, 2, True)
led_ring.rainbow(wait=True)
```

Parameters:
- **period** (*float*) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **iterations** (*int*) – Number of consecutive rainbows. If 0, the animation continues endlessly.
- **wait** (*bool*) – The service wait for the animation to finish or not to answer. If iterations is 0, the service answers immediately.

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))

rainbow_cycle(*period=0*, *iterations=0*, *wait=False*)

Draws rainbow that uniformly distributes itself across all LEDs.

Examples:

```
led_ring.rainbow_cycle()
led_ring.rainbow_cycle(5, 2, True)
led_ring.rainbow_cycle(wait=True)
```

Parameters:
- **period** (*float*) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **iterations** (*int*) – Number of consecutive rainbow cycles. If 0, the animation continues endlessly.
- **wait** (*bool*) – The service wait for the animation to finish or not to answer. If iterations is 0, the service answers immediately.

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))

rainbow_chase(*period=0*, *iterations=0*, *wait=False*)

Rainbow chase animation, like the led_ring_chase method.

Examples:

```
led_ring.rainbow_chase()
led_ring.rainbow_chase(5, 2, True)
led_ring.rainbow_chase(wait=True)
```

Parameters:
- **period** (*float*) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **iterations** (*int*) – Number of consecutive rainbow cycles. If 0, the animation continues endlessly.
- **wait** (*bool*) – The service wait for the animation to finish or not to answer. If iterations is 0, the service answers immediately.

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))

go_up(*color*, *period=0*, *iterations=0*, *wait=False*)

LEDs turn on like a loading circle, and are then all turned off at once.

Examples:

```
from std_msgs.msg import ColorRGBA

led_ring.go_up(ColorRGBA(r=15, g=50, b=255))
led_ring.go_up([15, 50, 255], 1, 100, True)
led_ring.go_up(ColorRGBA(r=15, g=50, b=255), iterations=20, wait=True)
```

Parameters:
- **color** (*list[float] or ColorRGBA*) – Led color in a list of size 3[R, G, B] or in an ColorRGBA object. RGB channels from 0 to 255.
- **period** (*float*) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **iterations** (*int*) – Number of consecutive turns around the Led Ring. If 0, the animation continues endlessly.
- **wait** (*bool*) – The service wait for the animation to finish or not to answer. If iterations is 0, the service answers immediately.

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))

go_up_down(*color*, *period=0*, *iterations=0*, *wait=False*)

LEDs turn on like a loading circle, and are turned off the same way.

Examples:

```
from std_msgs.msg import ColorRGBA

led_ring.go_up_down(ColorRGBA(r=15, g=50, b=255))
led_ring.go_up_down([15, 50, 255], 1, 100, True)
led_ring.go_up_down(ColorRGBA(r=15, g=50, b=255), iterations=20, wait=True)
```

Parameters:
- **color** (*list[float] or ColorRGBA*) – Led color in a list of size 3[R, G, B] or in an ColorRGBA object. RGB channels from 0 to 255.
- **period** (*float*) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **iterations** (*int*) – Number of consecutive turns around the Led Ring. If 0, the animation continues endlessly.
- **wait** (*bool*) – The service wait for the animation to finish or not to answer. If iterations is 0, the service answers immediately.

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))

breath(*color*, *period=0*, *iterations=0*, *wait=False*)

Variation of the light intensity of the LED ring, similar to human breathing.

Examples:

```
from std_msgs.msg import ColorRGBA

led_ring.breath(ColorRGBA(r=15, g=50, b=255))
led_ring.breath([15, 50, 255], 1, 100, True)
led_ring.breath(ColorRGBA(r=15, g=50, b=255), iterations=20, wait=True)
```

Parameters:
- **color** (*list[float] or ColorRGBA*) – Led color in a list of size 3[R, G, B] or in an ColorRGBA object. RGB channels from 0 to 255.
- **period** (*float*) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **iterations** (*int*) – Number of consecutive turns around the Led Ring. If 0, the animation continues endlessly.
- **wait** (*bool*) – The service wait for the animation to finish or not to answer. If iterations is 0, the service answers immediately.

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))

snake(*color*, *period=0*, *iterations=0*, *wait=False*)

A small coloured snake (certainly a python :D) runs around the LED ring.

Examples:

```
from std_msgs.msg import ColorRGBA

led_ring.snake(ColorRGBA(r=15, g=50, b=255))
led_ring.snake([15, 50, 255], 1, 100, True)
led_ring.snake(ColorRGBA(r=15, g=50, b=255), iterations=20, wait=True)
```

Parameters:
- **color** (*list[float] or ColorRGBA*) – Led color in a list of size 3[R, G, B] or in an ColorRGBA object. RGB channels from 0 to 255.
- **period** (*float*) – Execution time for a pattern in seconds. If 0, the default duration will be used.
- **iterations** (*int*) – Number of consecutive turns around the Led Ring. If 0, the animation continues endlessly.
- **wait** (*bool*) – The service wait for the animation to finish or not to answer. If iterations is 0, the service answers immediately.

Returns: status, message

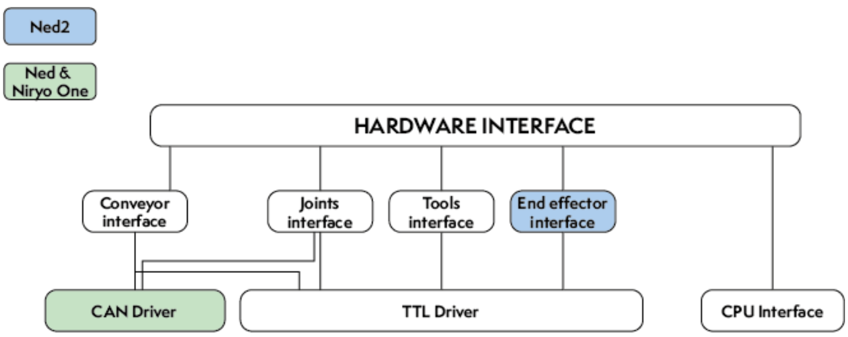
Return type: (int (https://docs.python.org/3/library/functions.html#int), str (https://docs.python.org/3/library/stdtypes.html#str))

Low Level Packages

In this section, you will have access to all information about each Niryo robot's ROS hardware stack packages, dedicated to low-level interfaces

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

This node has been conceived to instantiate all the interfaces we need to have a fully functional robot.

Among those interfaces we have:

- Conveyor Interface
- Joints Interface
- Tools Interface
- Cpu Interface
- End Effector Interface (Ned2 only)
- Can Driver (Ned and One Only)
- Ttl Driver

It belongs to the ROS namespace: `/niryo_robot_hardware_interface/` .

Parameters

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'

Dependencies

- Tools Interface
- Joints Interface
- Conveyor Interface
- CPU Interface
- Niryo_robot_msgs

Services, Topics and Messages

Published topics

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	Software version of the Raspberry PI and every hardware components (motors, end effector, conveyors and tools)

Services

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

Joints Interface

This package handles packages related to the robot's joints controller.

It provides an interface to ros_control.

Joints interface node

It is instantiated in Niryo Robot Hardware Interface (index.html#document-source/stack/low_level/niryo_robot_hardware_interface) package.

It has been conceived to:

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

It belongs to the ROS namespace: `/joints_interface/` .

Parameters

Joints Interface's default Parameters

default.yaml file

Name	Description	Default value	Unit
`ros_control_loop_frequency`	Controls loop frequency.	100	Hz

Joints Interface's hardware specific Parameters

These parameters are specific to the hardware version (Ned, Niryo One or Ned2). This file comes in a different version for each hardware version. They are located in a directory of the hardware version name.

joints_params.yaml file

Name	Description	Supported Hardware versions
`joint_N/id`	Joint N (1, 2, 3, 4, 5 or 6) id Default: -1 (invalid id)	All versions
`joint_N/type`	Joint N (1, 2, 3, 4, 5 or 6) motor type among: "stepper", "xl320", "xl430", "fakeStepper" or "fakeDxl" Default: ""	All versions
`joint_N/bus`	Joint N (1, 2, 3, 4, 5 or 6) bus ("ttl" or "can") Default: ""	All versions

calibration_params.yaml file

Name	Description	Default value	Unit	Supported Hardware versions
`calibration_timeout`	Waiting time between 2 commands during the calibration process.	30	seconds	All versions
`calibration_file`	File path where is saved motors calibration value.	*/home/niryo/niryo_robot_saved_files /stepper_motor_calibration_offsets.txt*	N.A.	All versions
`stepper_N/id`	Stepper N (1, 2 or 3) id	-1 (invalid id)	N.A.	All versions
`stepper_N/v_start`	Stepper N (1, 2 or 3) starting velocity for the acceleration profile	1	0.01 RPM	Ned 2 only
`stepper_N/a_1`	Stepper N (1, 2 or 3) first acceleration for the acceleration profile	0	RPM^2	Ned 2 only
`stepper_N/v_1`	Stepper N (1, 2 or 3) first velocity for the acceleration profile	0	0.01 RPM	Ned 2 only
`stepper_N/a_max`	Stepper N (1, 2 or 3) max acceleration for the acceleration profile	6000	RPM^2	Ned 2 only
`stepper_N/v_max`	Stepper N (1, 2 or 3) max velocity for the acceleration profile	6	0.01 RPM	Ned 2 only
`stepper_N/d_max`	Stepper N (1, 2 or 3) max deceleration for the acceleration profile	6000	RPM^2	Ned 2 only
`stepper_N/d_1`	Stepper N (1, 2 or 3) last deceleration for the acceleration profile	0	RPM^2	Ned 2 only
`stepper_N/v_stop`	Stepper N (1, 2 or 3) stop velocity for the acceleration profile	2	0.01 RPM	Ned 2 only
`stepper_N/stall_threshold`	Stepper N (1, 2 or 3) stall threshold for which we detect the end of the joint course for the calibration process	0	N.A.	Ned 2 only
`stepper_N/direction`	Stepper N (1, 2 or 3) direction for the calibration (1 = same as motor direction, -1 = against motor direction)	1	N.A.	All versions
`stepper_N/delay`	Stepper N (1, 2 or 3) delay	0	milliseconds	All versions

dynamixels_params.yaml file

Name	Description	Unit	Supported Hardware versions
`dxl_N/offset_position`	Dynamixel N (1, 2 or 3) offset position for the zero position Default: '0.0'	Rad	All versions
`dxl_N/home_position`	Dynamixel N (1, 2 or 3) home position Default: '0.0'	Rad	All versions
`dxl_N/direction`	Dynamixel N (1, 2 or 3) direction (1 = ClockWise, -1 = Counter ClockWise) Default: '1'	N.A.	All versions

Name	Description	Unit	Supported Hardware versions
`dxl_N/limit_position_max`	Dynamixel N (1, 2 or 3) maximal position allowed Default: '0.0'	Rad	All versions
`dxl_N/limit_position_min`	Dynamixel N (1, 2 or 3) minimal position allowed Default: '0.0'	Rad	All versions
`dxl_N/position_P_gain`	Dynamixel N (1, 2 or 3) Proportional gain of the position PID controller Default: '0.0'	N.A.	All versions
`dxl_N/position_I_gain`	Dynamixel N (1, 2 or 3) Integral gain of the position PID controller Default: '0.0'	N.A.	All versions
`dxl_N/position_D_gain`	Dynamixel N (1, 2 or 3) Derivative gain of the position PID controller Default: '0.0'	N.A.	All versions
`dxl_N/velocity_P_gain`	Dynamixel N (1, 2 or 3) Proportional gain of the velocity PID controller Default: '0.0'	N.A.	All versions
`dxl_N/velocity_I_gain`	Dynamixel N (1, 2 or 3) Integral gain of the velocity PID controller Default: '0.0'	N.A.	All versions
`dxl_N/FF1_gain`	Dynamixel N (1, 2 or 3) Feed Forward velocity Gain Default: '0.0'	N.A.	All versions
`dxl_N/FF2_gain`	Dynamixel N (1, 2 or 3) Feed Forward acceleration Gain Default: '0.0'	N.A.	All versions
`dxl_N/acceleration_profile`	Dynamixel N (1, 2 or 3) acceleration profile parameter [*] Default: '0.0'	RPM²	All versions
`dxl_N/velocity_profile`	Dynamixel N (1, 2 or 3) velocity profile parameter Default: '0.0'	RPM	All versions

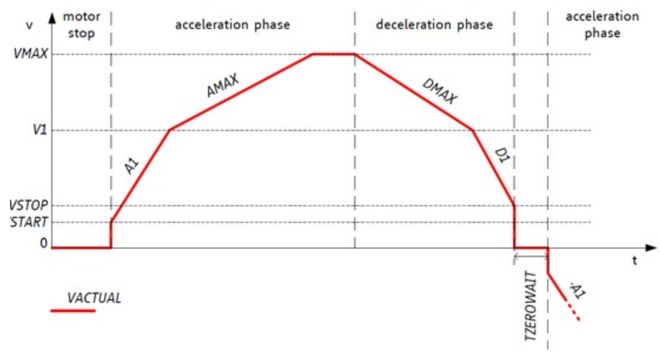
[*] refers to the dedicated motor reference documentation.

steppers_params.yaml file

Name	Description	Unit	Supported Hardware versions
`stepper_N/id`	Stepper N (1, 2 or 3) id Default: -1 (invalid id)	N.A.	All versions
`stepper_N/gear_ratio`	Stepper N (1, 2 or 3) gear ratio Default: 1	N.A.	Ned and One only
`stepper_N/max_effort`	Stepper N (1, 2 or 3) max effort Default: 0	N.A.	Ned and One only
`stepper_N/motor_ratio`	Stepper N (1, 2 or 3) motor ratio for conversion into radian Default: 1	N.A.	Ned 2 only
`stepper_N/offset_position`	Stepper N (1, 2 or 3) offset position to position limit min Default: 0	Rad	All versions
`stepper_N/home_position`	Stepper N (1, 2 or 3) Home position of the motor Default: 0	Rad	All versions
`stepper_N/limit_position_min`	Stepper N (1, 2 or 3) position limit min of the motor Default: 0	Rad	All versions
`stepper_N/limit_position_max`	Stepper N (1, 2 or 3) position limit max of the motor Default: 0	Rad	All versions
`stepper_N/direction`	Stepper N (1, 2 or 3) assembly direction of the motor (1 = CW, -1 = CCW) Default: 1	N.A.	All versions
`stepper_N/v_start`	Stepper N (1, 2 or 3) starting velocity for the acceleration profile Default: 1	RPM	Ned 2 only
`stepper_N/a_1`	Stepper N (1, 2 or 3) first acceleration for the acceleration profile Default: 0	RPM²	Ned 2 only
`stepper_N/v_1`	Stepper N (1, 2 or 3) first velocity for the acceleration profile Default: 0	RPM	Ned 2 only
`stepper_N/a_max`	Stepper N (1, 2 or 3) max acceleration for the acceleration profile Default: 6000	RPM²	Ned 2 only
`stepper_N/v_max`	Stepper N (1, 2 or 3) max velocity for the acceleration profile Default: 6	RPM	Ned 2 only
`stepper_N/d_max`	Stepper N (1, 2 or 3) max deceleration for the acceleration profile Default: 6000	RPM²	Ned 2 only

Name	Description	Unit	Supported Hardware versions
`stepper_N/d_1`	Stepper N (1, 2 or 3) last deceleration for the acceleration profile Default: 0	RPM²	Ned 2 only
`stepper_N/v_stop`	Stepper N (1, 2 or 3) stop velocity for the acceleration profile Default: 2	RPM	Ned 2 only
`stepper_N/stall_threshold`	Stepper N (1, 2 or 3) stall threshold for which we detect the end of the joint course Default:	N.A.	Ned 2 only

The velocity profiles for the Stepper motors (in *calibration_params.yaml* and *steppers_params.yaml* files) can be defined for TTL bus only (thus for Ned2 only). They are defined according to the following graph:



Dependencies

- hardware_interface
- controller_manager
- TTL Driver
- CAN Driver
- Niryo_robot_msgs
- control_msgs

Services, Topics and Messages

Subscribed topics

Joints Interface's Published Topics

Name	Message Type	Description
`niryo_robot_follow_joint_trajectory_controller/follow_joint_trajectory/result`	:control_actions:`control_msgs/FollowJointTrajectory Action<FollowJointTrajectory>`	Trajectory results from controller

Published topics

Joints Interface's Published Topics

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

Services

Joints 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 manual
`/niryo_robot/joints_interface/request_new_calibration`	Trigger	Resets motor calibration state to "uncalibrated". This will allow the user to ask a new calibration.
`niryo_robot/learning_mode/activate`	Trigger	Changes learning mode (Free Motion) state. When learning mode is activated, torques are disabled and the joints can move freely.
`niryo_robot/joints_interface/steppers_reset_controller`	Trigger	Resets the controller

Errors and warning messages

List of Errors and warning messages

Type	Message	Description
Error	JointHardwareInterface::init - Fail to add joint, return :	The joint is not correctly initialized
Error	JointHardwareInterface::init - stepper state init failed	The stepper state parameters are not correctly retrieved

Type	Message	Description
Error	JointHardwareInterface::init - dxl state init failed	The dynamixel state parameters are not correctly retrieved
Error	JointHardwareInterface::init - Dynamixel motors are not available on CAN Bus	The robot wrongly tries to initialize a dynamixel motor for the CAN bus (works only on TTL)
Error	JointHardwareInterface::init - Fail to reboot motor id	The motor failed to reboot. Try rebooting it again
WARNING	JointHardwareInterface::init - initialize stepper joint failure, return %d. Retrying	Failed to initialize a stepper. Will try again up to 3 times
WARNING	JointHardwareInterface::init - add stepper joint failure, return %d. Retrying	Failed to add a stepper joint. Will try again up to 3 times
WARNING	JointHardwareInterface::init - init dxl joint failure, return : %d. Retrying	Failed to initialize a dynamixel joint. Will try again up to 3 times
WARNING	JointHardwareInterface::init - add dxl joint failure, return : %d. Retrying	Failed to add a dynamixel joint. Will try again up to 3 times

Conveyor Interface

This package handles Niryo's Conveyors.

It allows you to control up to two Conveyors at the same time.

Two version of the conveyor exist: The Conveyor Belt, communicating via a CAN bus, and the Conveyor Belt (V2), communicating via a TTL bus. Both of them are directly compatible for the Ned and One. For Ned2, you will need to change the stepper card of the CAN Conveyor Belt to be able to use it on a TTL port (there is no CAN port on Ned2).

Conveyor Interface node (For development and debugging purpose only)

This ROS Node has been conceived to:

- Use the correct low level driver according to the hardware version of the robot.
- Initialize the Conveyor Interface.

Conveyor Interface core

It is instantiated in Niryo Robot Hardware Interface (index.html#document-source/stack/low_level/niryo_robot_hardware_interface) package.

It has been conceived to:

- Interface itself with low level drivers (CAN or TTL for Ned and Niryo One, TTL only for Ned2)
- Initialize conveyor motors parameters.
- Handle the requests from services to set, control or remove the conveyors.
- Publish conveyor states.

It belongs to the ROS namespace: `/niryo_robot/conveyor/` .

Parameters

Conveyor Interface's Parameters

Name	Description
`publish_frequency`	Publishing rate for conveyors state. Default: '2.0'
`type`	Type of the motor used. Default: 'Stepper'
`protocol`	Protocol of the communication. It can be 'CAN' (for Ned or One) or 'TTL' (for Ned or One or Ned 2)
`default_id`	Default id of the conveyor before the connection.
`Pool_id_list`	Id of the conveyor after the connection.
`Direction`	Direction of the conveyor.
`max_effort` (CAN Only)	Max effort used by the steppers Default: '90'
`micro_steps` (CAN only)	Micro steps used by the Steppers Default: '8'

Published topics - Conveyor interface

Conveyor Interface's Published Topics

Name	Message Type	Description
`feedback`	ConveyorFeedbackArray	Conveyors states

Services

Conveyor Interface Package Services

Name	Message Type	Description
`control_conveyor`	ControlConveyor	Sends a command to the desired Conveyor
`ping_and_set_conveyor`	SetConveyor	Scans and sets a new Conveyor or removes a connected Conveyor

Dependencies - Conveyor interface

- std_msgs
- CAN Driver
- TTL Driver

Services & messages files - Conveyor 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 (For Development and Debugging)

The ROS Node is made to:

- Initialize Tool Interface with configuration parameters.
- Start ROS stuffs like services, topics.

Tools Interface Core

It is instantiated in Niryo Robot Hardware Interface (index.html#document-source/stack/low_level/niryo_robot_hardware_interface) package.

It has been conceived to:

- Initialize the Tool Interface.
- Provide services for setting and controlling tools.
- Publish tool connection state.

It belongs to the ROS namespace: `/tools_interface/`.

Tool Interface's default Parameters

default.yaml

Name	Description

Name	Description
`check_tool_connection_frequency`	The frequency where tool interface check and publish the state of the tool connected, or remove tool if it is disconnected. Default: '2.0'

Tool Interface's hardware specific Parameters

These parameters are specific to the hardware version (Ned, One or Ned2). This file comes in a different version for each hardware version, located in a directory of the hardware version name.

tools_params.yaml

Name	Description	Supported Hardware versions
`id_list`	List of default IDs of each tool supported by Niryo Default: '[11,12,13,30,31]'	All Versions
`type_list`	List of motor tools type Default: 'xl320' for NED and ONE Default: 'xl330' for NED2 Default: 'fakeDxl' for simulation	All Versions
`name_list`	List of tools's name corresponds to ID list and type list above Default: '["Standard Gripper", "Large Gripper", "Adaptive Gripper", "Vacuum Pump", "Electromagnet"]'	All Versions

Dependencies

- std_msgs
- std_srvs
- TTL Driver
- Common

Services, Topics and Messages

Published topics

Tools Interface's Published Topics

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

Services

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 the gripper
`niryo_robot/tools/close_gripper`	tools_interface/OpenGripper	Closes the gripper
`niryo_robot/tools/pull_air_vacuum_pump`	tools_interface/OpenGripper	Pulls air with the vacuum pump
`niryo_robot/tools/push_air_vacuum_pump`	tools_interface/OpenGripper	Pushes air with the vacuum pump
`niryo_robot/tools/reboot`	std_srvs/Trigger	Reboots the motor of the equipped tool

PingDxlTool (Service)

```
---
int8 state
tools_interface/Tool tool
```

ToolCommand (Service)

```
uint8 id

uint16 position
uint16 speed
int16 hold_torque
int16 max_torque
---
uint8 state
```

End Effector Interface

This package handles the End Effector Panel of a robot, it is supported from Ned 2.
It provides services and topics specific to the End Effector Panel in order to be used by a final user.

However, it does not deal with the low level bus communication with the components: this is done in theTTL Driver package.

End Effector Interface node (For development and debug)

The ROS Node in End Effector Interface Package is used to:

- Instantiate a TTL Driver manager to communicate with hardware.
- Initialize End Effector Interface.

End Effector Interface Core

It is instantiated in Niryo Robot Hardware Interface (index.html#document-source/stack/low_level/niryo_robot_hardware_interface) package.

It has been conceived to:

- Interface with TTL Driver.
- Initialize End Effector parameters.
- Retrieve End Effector data from TTL driver.
- Publish the status of buttons.
- Publish the collision detection status.
- Start service on IO State.

It belongs to the ROS namespace: `/end_effector_interface/` .

Parameters - End Effector Interface

end_effector_interface's Parameters

Name	Description
`end_effector_id`	Id of the End Effector in TTL bus Default: 0
`check_end_effector_status_frequency`	Frequency to get the End Effector from driver Default: 40.0
`button_2__type`	Button used to activate the FreeMotion mode Default: free_drive
`button_1__type`	Button used to save the actual position of the robot Default: save_position
`button_0__type`	Custom Button used by users to do something Default: custom
`hardware_type`	Type of the End Effector. It can be end_effector or fake_end_effector Default: end_effector

Published topics - End Effector Interface

end_effector_interface Package Published Topics

Name	Message Type	Description
/niryo_robot_hardware_interface/end_effector_interface/_free_drive_button_state_publisher	EEButtonStatus	Publishes state of Free Motion Button
/niryo_robot_hardware_interface/end_effector_interface/_save_button_state_publisher	EEButtonStatus	Publishes state of Save Position Button
/niryo_robot_hardware_interface/end_effector_interface/_custom_button_state_publisher	EEButtonStatus	Publishes state of Custom Button
/niryo_robot_hardware_interface/end_effector_interface/_digital_out_publisher	EEIOState	Publishes state of IO Digital

Services - End Effector Interface

end_effector_interface Package Services

Name	Service Type	Description
set_ee_io_state	SetEEDigitalOut	Set up digital output on End Effector

Dependencies - End Effector Interface

- std_msgs
- TTL Driver
- Common

Services & Messages files - End Effector Interface

SetEEDigitalOut (Service)

```
bool data
---
bool state
```

EEButtonStatus (Message)

```
uint8 HANDLE_HELD_ACTION=0
uint8 LONG_PUSH_ACTION=1
uint8 SINGLE_PUSH_ACTION=2
uint8 DOUBLE_PUSH_ACTION=3
uint8 NO_ACTION=100

uint8 action
```

EEIOState (Message)

```
bool digital_input
bool digital_output
```

CPU Interface

This package provides an interface for CPU temperature monitoring.

CPU Interface Node (For development and debugging purpose only)

This ROS Node has been conceived to launch the CPU interface in an isolated way.

CPU Interface Core

It is instantiated in Niryo Robot Hardware Interface (index.html#document-source/stack/low_level/niryo_robot_hardware_interface) package.

It has been made to monitor CPU temperature of the Raspberry Pi and automatically shutdown the Raspberry Pi if it reaches a critical threshold. Two thresholds can be defined via parameters: a warning threshold and a shutdown threshold.

The CPU temperature is read from the Ubuntu system file */sys/class/thermal/thermal_zone0/temp*.

In simulation, the CPU temperature of the computer running the simulation is used, but the threshold are deactivated (no shutdown in case of high temperature).

It belongs to the ROS namespace: `/cpu_interface/` .

Parameters

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'

Dependencies

- Common

Services, Topics and Messages

None

Common

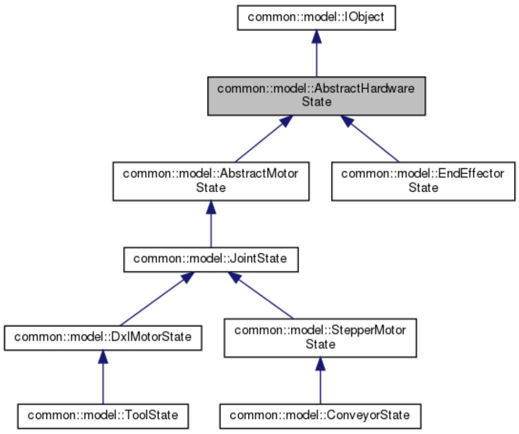
The Common package defines the common software components of the low level stack. It is split into a model part and a utility part:
- The 'model' subpackage defines the model tree needed to keep a virtual state of the robot up to date at any time.
- The 'util' subpackage defines cpp interfaces and useful functions

Model

The model subpackage is comprised of:

States

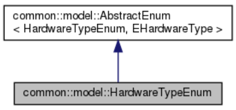
Classes, to represent the virtual state of each hardware component at any moment. The hierarchy allows powerful polymorphism so that we can interpret each component differently based on the information we need to obtain.



Abstract Hardware State inheritance graph

Enums

Enhanced enums, to keep trace of various enumeration and be able to have useful utilities attached to them (like conversion in string).

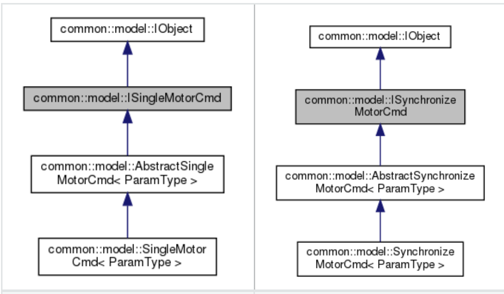


Hardware Type Enum inheritance graph

Commands

Classes representing single and synchronize commands, for steppers and dynamixels. They are needed in queues in the ttl_driver and can_driver packages.

Commands graphs



Single Cmd | Sync Cmd

Each type of command is an alias to specified versions of two base template classes: AbstractSynchronizeMotorCmd and AbstractSingleMotorCmd

Util

The util subpackage is comprised of:

- Cpp interfaces, used globally in the stack for polymorphism for instance
- Utility functions usable globally in the stack

Dependencies

This package does not depend on any package. This package is a dependency of the following packages:

- can_driver
- conveyor_interface
- cpu_interface
- end_effector_interface
- joints_interface
- niryo_robot_hardware_interface
- tools_interface
- ttl_driver

TTL Driver

This package handles motors which communicate via the protocol TTL.

This package is based on the DXL SDK. It provides an interface todynamixel_sdk.

TTL Driver Node (For only the development and debugging propose)

The ROS Node is made to:

- Initialize TTL Interface.
- Get configuration of motors and add to TTL Interface.

TTL Interface Core

It is instantiated in Niryo Robot Hardware Interface (index.html#document-source/stack/low_level/niryo_robot_hardware_interface) package.

It has been conceived to:

- Initialize the TTL Interface (Interface used by other packages) and physical bus with the configurations.
- Add, remove and monitor devices.
- Start getting data and sending data on the physical bus.
- Start ROS stuffs like services, topics.

It belongs to the ROS namespace: `/niryo_robot/ttl_driver/` .

Parameters - TTL Driver

> **Note**
>
> These configuration parameters are chosen and tested many times to work correctly. Please make sure that you understand what you do before editing these files.

TTL Driver's Parameters

Name	Description
`ttl_hardware_control_loop_frequency`	Frequency of the bus control loop. Default: '240.0'
`ttl_hardware_write_frequency`	Writes frequency on the bus. Default: '120.0'
`ttl_hardware_read_data_frequency`	Reads frequency on the bus. Default: '120.0'
`ttl_hardware_read_status_frequency`	Reads frequency for device status on the bus. Default: '0.7'
`ttl_hardware_read_end_effector_frequency`	Read frequency for End Effector's status. Default: '13.0'
`bus_params/Baudrate`	Baudrates of TTL bus Default: '1000000'
`bus_params/uart_device_name`	Name of UART port using Default: '/dev/ttyAMA0'

Dependencies - TTL Driver

- dynamixel_sdk
- Niryo_robot_msgs
- Common
- std_msgs

Services - TTL Driver

TTL Driver Package Services

Name	Message Type	Description
`niryo_robot/ttl_driver/set_dxl_leds`	SetInt	Controls dynamixel LED
`niryo_robot/ttl_driver/send_custom_value`	SendCustomValue	Writes data at a custom register address of a given TTL device
`niryo_robot/ttl_driver/read_custom_value`	ReadCustomValue	Reads data at a custom register address of a given TTL device
`niryo_robot/ttl_driver/read_pid_value`	ReadPIDValue	Reads the PID of dxl motors
`niryo_robot/ttl_driver/write_pid_value`	WritePIDValue	Writes the PID for dxl motors
`niryo_robot/ttl_driver/read_velocity_profile`	ReadVelocityProfile	Reads velocity Profile for steppers
`niryo_robot/ttl_driver/write_velocity_profile`	WriteVelocityProfile	Writes velocity Profile for steppers

Services & Messages files - TTL Driver

SendCustomValue (Service)

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

uint8 id
int32 value
int32 reg_address
int32 byte_number
---
int32 status
string message
```

ReadCustomValue (Service)

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

uint8 id
int32 reg_address
int32 byte_number
---
int32 value
int32 status
string message
```

ReadPIDValue (Service)

```
# Check XL-XXX motors reference doc for
# the complete register table

uint8 id
---
uint16 pos_p_gain
uint16 pos_i_gain
uint16 pos_d_gain

uint16 vel_p_gain
uint16 vel_i_gain

uint16 ff1_gain
uint16 ff2_gain

int32 status
string message
```

WritePIDValue (Service)

```
# Check XL-XXX motors reference doc for
# the complete register table

uint8 id
---
uint16 pos_p_gain
uint16 pos_i_gain
uint16 pos_d_gain

uint16 vel_p_gain
uint16 vel_i_gain

uint16 ff1_gain
uint16 ff2_gain

int32 status
string message
```

ReadVelocityProfile (Service)

```
# Check stepper ttl reference doc for
# the complete register table

uint8 id
---
float64 v_start

float64 a_1
float64 v_1

float64 a_max
float64 v_max
float64 d_max

float64 d_1

float64 v_stop

int32 status
string message
```

WriteVelocityProfile (Service)

```
# Check stepper ttl reference doc for
# the complete register table

uint8 id

float64 v_start

float64 a_1
float64 v_1

float64 a_max
float64 v_max
float64 d_max

float64 d_1

float64 v_stop
---
int32 status
string message
```

MotorHardwareStatus (Message)

```
niryo_robot_msgs/MotorHeader motor_identity

string firmware_version
uint32 temperature
float64 voltage
uint32 error
string error_msg
```

MotorCommand (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
```

ArrayMotorHardwareStatus (Message)

```
std_msgs/Header header
ttl_driver/MotorHardwareStatus[] motors_hw_status
```

CAN Driver

This package provides an interface between high level ROS packages and handler of CAN Bus. It uses the mcp_can_rpi for CAN bus communication.

It is used by only Ned and the Niryo One.

CAN Driver Node (For only the development and debugging propose)

The ROS Node is made to:

- Initialize CAN Interface.

CAN Interface Core

It is instantiated in Niryo Robot Hardware Interface (index.html#document-source/stack/low_level/niryo_robot_hardware_interface) package.

It has been conceived to:

- Initialize the CAN Interface and physical bus with the configurations.
- Add, remove and monitor devices on bus.
- Start control loop to get and send data from/to motors.
- Start ROS stuffs like services, topics if they exist.

It belongs to the ROS namespace: `/can_driver/`.

Parameters

Note

These configuration parameters are set to work with Niryo's robot. Do not edit them.

CAN Driver's Parameters

Name	Description
`can_hardware_control_loop_frequency`	Control loop frequency. Default: '1500.0'
`can_hw_write_frequency`	Write frequency. Default: '200.0'

Name	Description
`can_hw_read_frequency`	Read frequency. Default: '50.0'
`bus_params/spi_channel`	spi channel. Default: '0'
`bus_params/spi_baudrate`	Baudrate. Default: '1000000'
`bus_params/gpio_can_interrupt`	GPIO Interrupt. Default: '25'

Dependencies

- MCP CAN rpi
- Niryo_robot_msgs
- Common
- std_msgs

Services, Topics and Messages

StepperCmd (Service)

```
uint8 cmd_type
uint8 CMD_TYPE_SYNCHRONIZE=5
uint8 CMD_TYPE_RELATIVE_MOVE=6
uint8 CMD_TYPE_MAX_EFFORT=7
uint8 CMD_TYPE_MICRO_STEPS=8
uint8 CMD_TYPE_POSITION_OFFSET=9
uint8 CMD_TYPE_CALIBRATION=10

uint8[] motors_id
int32[] params
---
bool result
```

StepperMotorHardwareStatus (Message)

```
niryo_robot_msgs/MotorHeader motor_identity

string firmware_version
int32 temperature
int32 voltage
int32 error
```

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
can_driver/StepperMotorHardwareStatus[] motors_hw_status
```

TTL Debug Tools

This package is a debugging package to setup and access directly to all hardware components on the TTL bus. It provides main functions like ping, scan device and read/write/syncRead/syncWrite operations on devices.

There are two ways to use this package: directly with the compiled binary, or via TTL Driver (index.html#document-source/stack/low_level/ttl_driver) services called in dedicated scripts.

Ttl debug tool binary

The compiled binary (located in *install/lib/ttl_debug_tools/ttl_debug_tools*) directly accesses the TTL bus using Dynamixel SDK (index.html#document-source/stack/third_parties/dynamixel_sdk) third party library. Thus, it cannot be used if the Niryo ROS Stack is already running and you should first stop the robot stack (sudo service niryo_robot_ros stop)

This tool can be launched via:

```
rosrun ttl_debug_tools ttl_debug_tools
```

or

```
roslaunch ttl_debug_tools ttl_debug_tools
```

Parameters - Ttl debug tools

- **–help / -h:** Prints help message
- **–baudrate / -b [Baudrate]:** Baudrates (1000000 by default)
- **–port / -p [Port]:** Sets port
- **–id / -i [ID]:** Devices ID (-1 by default)
- **–ids [IDs]:** Lists of devices IDs
- **–scan:** Scans all devices on the bus
- **–ping:** Pings specific ID
- **–get-register [Addr]:** Gets a value from a register, parameters is: register address
- **–get-registers [Addr]:** Gets list of values from multiple devices at a register address, parameters is: register address
- **–get-size [Size]:** Size of data to be read with get-register or get-registers, parameters is: size of data in bytes
- **–set-register [Addr] [Value] [Size]:** Sets a value to a register, parameters are in the order: register address / value / size (in bytes) of the data
- **–set-registers [Addr] [Values] [Size]:** Sets values to a register on multiple devices, parameters are in the order: register address / list of values / size (in bytes) of the data
- **–calibrate:** Calibrates all steppers on the bus. It is used in Ned2 only

Scripts

In order to use Ttl debug tools to debug an already running ROS stack, it was necessary to develop another tool. To do so, two python scripts have been developped. They ensure access to the data on the TTL bus via two services implemented in the package TTL Driver (index.html#document-source/stack/low_level/ttl_driver):

- read_custom_dxl_value.py : uses service ReadCustomValue to read values from the TTL bus
- send_custom_dxl_value : uses service SendCustomValue to write values to the TTL bus

Niryo robot - Send DXL custom value

It uses a ttl_driver service to send data to a register of a device on the TTL bus when the ROS stack is running. This script can be launched via:

```
rosrun ttl_debug_tools send_custom_dxl_value.py
```

Parameters - Send custom value

- **–id [ID]:** Device ID
- **–address [Addr]:** Registers address to modify
- **–value [Value]:** Value to store at the register address given
- **–size [Size]:** Size in bytes of the data to write

Niryo robot - Read DXL custom value

It uses a service to read data from a register a device on the TTL bus when the ROS stack is running. This script can be launched via:

```
rosrun ttl_debug_tools read_custom_dxl_value.py
```

Parameters - Read custom value

- **–id [ID]:** Device ID
- **–address [Addr]:** Register address to modify
- **–size [Size]:** Size in bytes of the data to read

CAN Debug Tools

This package offers scripts to debug with Hardware and setup CAN devices. It provides some main functions like setting up the CAN bus and dumping data on bus.

Niryo robot - CAN debug tools

It provides service to dump data on CAN bus. This script can be launched via:

```
rosrun can_debug_tools can_debug_tools
```

Parameters - CAN debug tools

- **–help / -h:** Prints help message
- **–baudrate / -b [Baudrate]:** Baudrates (1000000 by default)
- **–channel / -c [Channel]:** Sets channel SPI (0 by default)
- **–gpio / -g:** GPIO Interrupts for CAN (25 by default)
- **–freq / -f:** frequency of control loop to check data (100Hz by default)
- **–dump:** runs dump service to dump and shows all data found on bus

When you dump data on CAN bus, the result is a table including:

- Number of data's package
- Status of package
- Control byte
- Data in 8 bytes

Third Parties Packages

In this section, you will have access to all information about each Niryo robot's ROS hardware stack packages, dedicated to low-level interfaces

Dynamixel SDK

This package has been forked from the official [dynamixel_sdk] package (https://github.com/ROBOTIS-GIT/DynamixelSDK/).

It has been adapted to work on Ned's custom Raspberry Pi 4B shield, using the [wiringPi] library (http://wiringpi.com/).

MCP CAN rpi

Raspberry Pi library for MCP2515 module (CAN bus interface) through SPI GPIOs

Forked from [MCP_CAN] library (https://github.com/coryjfowler/MCP_CAN_lib).

The MCP2515 module is a SPI-CAN interface. The MCP_CAN library is using the SPI protocol on Arduino to program and use this module. It has been adapted here to work with the Raspberry Pi 4 GPIOs, using the SPI functions of the using the [wiringPi] library (http://wiringpi.com/).

—

One of the main difference is that we don't handle SPI Chip Select PIN. This is already done by the wiringPi library, and all PINs for SPI are already predefined (spi channel 0 or 1).

To poll the MCP2515 module (to see if there is any data to read), the _digitalRead_ function of wiringPi is used.

Third Parties ROS packages

- ros_core
- moveit
- ros_control
- roscpp
- rosdoc_lite
- roslint
- rostest

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

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 Ubuntu 18 Installation.

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:

```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()

# -- 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.

```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')

# 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:

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

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

```python
from niryo_robot_python_ros_wrapper import *
```

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

```python
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:

```python
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:

```python
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):

```python
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:

```
#!/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):

```
#!/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:

```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')

# 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()
```

To grasp by specifying parameters:

```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')

# 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 corresponds to:

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

The line to release is:

```python
niryo_robot.release_with_tool()
```

To release by specifying parameters:

```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')

# 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:

```python
# 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:

```python
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:

```python
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:

```python
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 & initialization are not mentionned, 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:

```python
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:

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

```
#!/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:

```
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)
```

Examples: Dynamic frames

This document shows how to use dynamic frames.

If you want to see more about dynamic frames functions, you can look at API - Dynamic frames (index.html#dynamic-frames)

Danger

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

Simple dynamic frame control

This example shows how to create a frame and do a small pick and place in this frame:

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

# Imports
from niryo_robot_python_ros_wrapper import *
import rospy

gripper_speed = 400

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

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

# Create frame
point_o = [0.15, 0.15, 0]
point_x = [0.25, 0.2, 0]
point_y = [0.2, 0.25, 0]

niryo_robot.save_dynamic_frame_from_points("dynamic_frame", "description", [point_o, point_x, point_y])

# Get list of frames
print(niryo_robot.get_saved_dynamic_frame_list())
# Check creation of the frame
info = niryo_robot.get_saved_dynamic_frame("dynamic_frame")
print(info)

# Pick
#niryo_robot.open_gripper(gripper_speed)
# Move to the frame
niryo_robot.move_pose(0, 0, 0, 0, 1.57, 0, "dynamic_frame")
#niryo_robot.close_gripper(gripper_speed)

# Move in frame
niryo_robot.move_linear_relative([0, 0, 0.1, 0, 0, 0], "dynamic_frame")
niryo_robot.move_relative([0.1, 0, 0, 0, 0, 0], "dynamic_frame")
niryo_robot.move_linear_relative([0, 0, -0.1, 0, 0, 0], "dynamic_frame")

# Place
#niryo_robot.open_gripper(gripper_speed)
niryo_robot.move_linear_relative([0, 0, 0.1, 0, 0, 0], "dynamic_frame")

# Home
niryo_robot.move_joints(0, 0.5, -1.25, 0, 0, 0)

# Delete frame
niryo_robot.delete_dynamic_frame("dynamic_frame")
```

Python ROS Wrapper documentation

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

- API functions
- Enums

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
- Joints & Pose

- Saved poses
- Pick & place
- Trajectories
- Dynamic frames
- Tools
- Hardware
- Conveyor Belt
- Vision
- Sound
- Led Ring
- Custom Button

Main purpose functions

class **NiryoRosWrapper**

calibrate_auto()

Calls service to calibrate motors then waits for its end. If failed, raises NiryoRosWrapperException

Returns: status, message

Return type: (int, str)

calibrate_manual()

Calls service to calibrate motors then waits for its end. If failed, raises NiryoRosWrapperException

Returns: status, message

Return type: (int, str)

get_learning_mode()

Uses /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*)

Calsl service to set_learning_mode according to set_bool. If failed, raises NiryoRosWrapperException

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

Returns: status, message

Return type: (int, str)

set_arm_max_velocity(*percentage*)

Sets relative max velocity (in %)

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

Returns: status, message

Return type: (int, str)

Joints & Pose

class **NiryoRosWrapper**

get_joints()

Uses /joint_states topic to get joints status

Returns: list of joints value

Return type: list[float]

get_pose()

Uses /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()

Uses /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*)

Executes 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()

Moves 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*, *frame=''*)

Moves robot end effector pose to a (x, y, z, roll, pitch, yaw) pose, in a particular frame if defined

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

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*)

Executes 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*)

Executes 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*, *frame=''*)

Moves robot end effector pose to a (x, y, z, roll, pitch, yaw) pose, with a linear trajectory, in a particular frame if defined

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

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*)

Turns 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*)

Makes 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*)

Makes 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*)

Computes 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*)

Computes 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*)

Moves robot end effector pose to a pose saved

Parameters: **pose_name** (*str*) –

Returns: status, message

Return type: (int, str)

get_pose_saved(*pose_name*)

Gets saved pose from robot intern storage Will raise error if position does not exist

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*)

Saves 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*)

Sends delete command to the pose manager service

Parameters: **name** (*str*) –

Returns: status, message

Return type: (int, str)

`get_saved_pose_list`(*with_desc=False*)

Asks the pose manager service which positions are available

Parameters: **with_desc** (*bool*) – If True it returns the poses descriptions

Returns: list of positions name

Return type: list[str]

Pick & place

class `NiryoRosWrapper`

`pick_from_pose`(*x, y, z, roll, pitch, yaw*)

Executes 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*)

Executes 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*)

Executes 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

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*)

Gets 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]]

`get_saved_trajectory_list`()

Asks the pose trajectory service which trajectories are available

Returns: list of trajectory name

Return type: list[str]

`execute_trajectory_from_poses`(*list_poses_raw*, *dist_smoothing=0.0*)

Executes 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*)

Executes 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_points*, *trajectory_name*, *trajectory_description*)

Saves trajectory object and sends it to the trajectory manager service

Parameters:
- **trajectory_name** (*str*) – name which will have the trajectory
- **trajectory_points** (*list[trajectory_msgsJointTrajectorypoint]*) – list of trajectory_msgs/JointTrajectoryPoint

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*)

Sends delete command to the trajectory manager service

Parameters: **trajectory_name** (*str*) – name

Returns: status, message

Return type: (int, str)

Dynamic frames

class `NiryoRosWrapper`

`save_dynamic_frame_from_poses`(*frame_name*, *description*, *list_robot_poses*, *belong_to_workspace=False*)

Create a dynamic frame with 3 poses (origin, x, y)

Parameters:
- **frame_name** (*str*) – name of the frame
- **description** (*str*) – description of the frame
- **list_robot_poses** (*list[list[float]]*) – 3 poses needed to create the frame
- **belong_to_workspace** (*boolean*) – indicate if the frame belong to a workspace

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_dynamic_frame_from_points(*frame_name*, *description*, *list_points*, *belong_to_workspace=False*)

Create a dynamic frame with 3 points (origin, x, y)

Parameters:
- **frame_name** (*str*) – name of the frame
- **description** (*str*) – description of the frame
- **list_points** (*list[list[float]]*) – 3 points needed to create the frame
- **belong_to_workspace** (*boolean*) – indicate if the frame belong to a workspace

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))

edit_dynamic_frame(*frame_name*, *new_frame_name*, *new_description*)

Modify a dynamic frame

Parameters:
- **frame_name** (*str*) – name of the frame
- **new_frame_name** (*str*) – new name of the frame
- **new_description** (*str*) – new description of the frame

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_dynamic_frame(*frame_name*, *belong_to_workspace=False*)

Delete a dynamic frame

Parameters:
- **frame_name** (*str*) – name of the frame to remove
- **belong_to_workspace** (*boolean*) – indicate if the frame belong to a workspace

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))

get_saved_dynamic_frame(*frame_name*)

Get name, description and pose of a dynamic frame

Parameters: **frame_name** (*str*) – name of the frame

Returns: name, description, position and orientation of a frame

Return type: list[str, str, list[float]]

get_saved_dynamic_frame_list()

Get list of saved dynamic frames

Returns: list of dynamic frames name, list of description of dynamic frames

Return type: list[str], list[str]

move_relative(*offset*, *frame='world'*)

Move robot end of a offset in a frame

Parameters:
- **offset** (*list[float]*) – list which contains offset of x, y, z, roll, pitch, yaw
- **frame** (*str*) – name of local frame

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_relative(*offset*, *frame='world'*)

Move robot end of a offset by a linear movement in a frame

Parameters:
- **offset** (*list[float]*) – list which contains offset of x, y, z, roll, pitch, yaw
- **frame** (*str*) – name of local frame

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))

Tools

class **NiryoRosWrapper**

get_current_tool_id()

Uses /niryo_robot_tools_commander/current_id topic to get current tool id

Returns: Tool Id

Return type: ToolID

update_tool()

Calls service niryo_robot_tools_commander/update_tool to update tool

Returns: status, message

Return type: (int, str)

grasp_with_tool(*pin_id=''*)

Grasps with the tool linked to tool_id This action corresponds 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=''*)

Releases with the tool associated to tool_id This action corresponds 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*, *max_torque_percentage=100*, *hold_torque_percentage=20*)

Opens gripper with a speed 'speed'

Parameters:
- **speed** (*int*) – Default -> 500
- **max_torque_percentage** (*int*) – Default -> 100
- **hold_torque_percentage** (*int*) – Default -> 20

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))

close_gripper(*speed=500*, *max_torque_percentage=100*, *hold_torque_percentage=50*)

Closes gripper with a speed 'speed'

Parameters:
- **speed** (*int*) – Default -> 500
- **max_torque_percentage** (*int*) – Default -> 100
- **hold_torque_percentage** (*int*) – Default -> 20

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))

pull_air_vacuum_pump()

Pulls air

Returns: status, message

Return type: (int, str)

push_air_vacuum_pump()

Pulls air

Returns: status, message

Return type: (int, str)

setup_electromagnet(*pin_id*)

Setups electromagnet on pin

Parameters: **pin_id** (*PinID*) – Pin ID

Returns: status, message

Return type: (int, str)

activate_electromagnet(*pin_id*)

Activates 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*)

Deactivates electromagnet associated to electromagnet_id on pin_id

Parameters: **pin_id** (*PinID*) – Pin ID

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()

Resets 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*)

Sets 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*)

Sets pin_id state to pin_state

Parameters:

- **pin_id** (*Union[PinID, str]*) – The name of the pin
- **digital_state** (*Union[PinState, bool]*) –

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*)

Reads pin number pin_id and returns its state

Parameters: **pin_id** (*Union[PinID, str]*) – The name of the pin

Returns: state

Return type: PinState

get_digital_io_state()

Gets Digital IO state : Names, modes, states

Returns: Infos contains in a IOsState object (see niryo_robot_msgs)

Return type: IOsState

get_hardware_status()

Gets hardware status : Temperature, Hardware version, motors names & types …

Returns: Infos contains in a HardwareStatus object (see niryo_robot_msgs)

Return type: HardwareStatus

Conveyor Belt

class **NiryoRosWrapper**

set_conveyor()

Scans for conveyor on can bus. If conveyor detected, returns the conveyor ID

Raises: **NiryoRosWrapperException** –

Returns: ID

Return type: ConveyorID

unset_conveyor(*conveyor_id*)

Removes 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*)

Controls 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(*with_seq=False*)

Gets last stream image in a compressed format

Returns: string containing a JPEG compressed image

Return type: str

set_brightness(*brightness_factor*)

Modifies 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*)

Modifies 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*)

Modifies 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.

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 executes a vision pick

`vision_pick_w_obs_pose`(*workspace_name*, *height_offset*, *shape*, *color*, *observation_pose_list*)

Move Pose to observation_pose, then executes a vision pick

`vision_pick`(*workspace_name*, *height_offset*, *shape*, *color*)

Picks the specified object from the workspace. This function has multiple phases: 1. detects object using the camera 2. prepares the current tool for picking 3. approaches the object 4. moves down to the correct picking pose 5. actuates the current tool 6. lifts 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`()

Gets calibration object: camera intrinsics, distortions coefficients

Returns: raw camera intrinsics, distortions coefficients

Return type: (list, list)

`save_workspace_from_poses`(*name*, *list_poses_raw*)

Saves workspace by giving the poses of the robot to point its 4 corners with the calibration Tip. Corners should be in the good order

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*)

Saves 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*)

Calls workspace manager to delete a certain workspace

Parameters: **name** (*str*) – workspace name

Returns: status, message

Return type: (int, str)

`get_workspace_poses`(*name*)

Gets 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*)

Gives the length over width ratio of a certain workspace

Parameters: **name** (*str*) – workspace name

Returns: ratio

Return type: float

get_workspace_list(*with_desc=False*)

Asks the workspace manager service names of the available workspace

Returns: list of workspaces name

Return type: list[str]

Sound

For more function, please refer to: Sound API functions (index.html#sound-api-functions)

class **NiryoRosWrapper**

sound

Manages sound

Example:

```
from niryo_robot_python_ros_wrapper.ros_wrapper import *

robot = NiryoRosWrapper()
robot.sound.play(sound.sounds[0])
```

Returns: SoundRosWrapper API instance

Return type: SoundRosWrapper

Led Ring

For more function, please refer to: Led Ring API functions (index.html#led-ring-api-functions)

class **NiryoRosWrapper**

led_ring

Manages the LED ring

Example:

```
from niryo_robot_python_ros_wrapper.ros_wrapper import *

robot = NiryoRosWrapper()
robot.led_ring.solid(color=[255, 255, 255])
```

Returns: LedRingRosWrapper API instance

Return type: LedRingRosWrapper

Custom Button

class **NiryoRosWrapper**

custom_button

Manages the custom button

Example:

```
from niryo_robot_python_ros_wrapper.ros_wrapper import *

robot = NiryoRosWrapper()
print(robot.custom_button.state)
```

Returns: CustomButtonRosWrapper API instance

Return type: CustomButtonRosWrapper

class **CustomButtonRosWrapper**(*hardware_version='ned2'*)

state

Get the button state from the ButtonAction class

Returns: int value from the ButtonAction class

Return type: int

is_pressed()

Button press state

Return type: bool

wait_for_action(*action, timeout=0*)

Waits until a specific action occurs and returns true. Returns false if the timeout is reached.

Parameters: **action** (*int*) – int value from the ButtonAction class

Returns: True if the action has occurred, false otherwise

Return type: bool

`wait_for_any_action`(*timeout=0*)

Returns the detected action. Returns ButtonAction.NO_ACTION if the timeout is reached without action.

Returns: Returns the detected action, or ButtonAction.NO_ACTION if the timeout is reached without any action.

Return type: int

`get_and_wait_press_duration`(*timeout=0*)

Waits for the button to be pressed and returns the press time. Returns 0 if no press is detected after the timeout duration.

Return type: float

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*

`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`= *False*

`HIGH`= *True*

class `PinID`

Pins ID

`GPIO_1A`= *'1A'*

`GPIO_1B`= *'1B'*

GPIO_1C= *'1C'*

GPIO_2A= *'2A'*

GPIO_2B= *'2B'*

GPIO_2C= *'2C'*

SW_1= *'SW1'*

SW_2= *'SW2'*

DO1= *'DO1'*

DO2= *'DO2'*

DO3= *'DO3'*

DO4= *'DO4'*

DI1= *'DI1'*

DI2= *'DI2'*

DI3= *'DI3'*

DI4= *'DI4'*

DI5= *'DI5'*

AI1= *'AI1'*

AI2= *'AI2'*

AO1= *'AO1'*

AO2= *'AO2'*

class **ConveyorID**

ConveyorID to be able to have CAN (id 12 and 13) and TTL (id 9 and 10) conveyor in any possible combination

ID_1 = 12 # One, Ned ID_2 = 13 # One, Ned ID_3 = 9 # Ned2 ID_4 = 10 # Ned2

NONE= *0*

ID_1= *-1*

ID_2= *-2*

class **ConveyorCan**

ConveyorID to control conveyors with CAN interface

NONE= *0*

ID_1= *12*

ID_2= *13*

class **ConveyorTTL**

ConveyorID to control conveyors with TTL interface

NONE= *0*

ID_1= *9*

ID_2= *10*

class **ConveyorDirection**

FORWARD= *1*

BACKWARD= *-1*

class **ObjectColor**

RED= *'RED'*

GREEN= *'GREEN'*

BLUE= *'BLUE'*

ANY= *'ANY'*

class **ObjectShape**

CIRCLE= *'CIRCLE'*

SQUARE= *'SQUARE'*

ANY= *'ANY'*

class **ProgramLanguage**

NONE= *-1*

ALL= *0*

PYTHON2= *1*

PYTHON3= *2*

BLOCKLY= *66*

class **AutorunMode**

DISABLE= *0*

ONE_SHOT= *1*

LOOP= *2*

class **ButtonAction**

HANDLE_HELD_ACTION= *0*

LONG_PUSH_ACTION= *1*

SINGLE_PUSH_ACTION= *2*

DOUBLE_PUSH_ACTION= *3*

NO_ACTION= *100*

Modbus

In this document, we will focus on the Modbus/TCP server.

Ned is permanently running a Modbus TCP Server that enables Ned to communicate with a PLC, or another computer in the same network.

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/).

Use the Modbus TCP server

In this document, we will focus on the Modbus/TCP server.

Ned is permanently running a Modbus TCP Server that enables Ned to communicate with a PLC, or another computer in the same network.

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.

Introduction

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:

Digital IO addresses offset table

Address offset	Niryo One / Ned digital IO	Ned2 digital IO
0	1A	DI1
1	1B	DI2
2	1C	DI3
3	2A	DI4
4	2B	DI5
5	2C	D01

Address offset	Niryo One / Ned digital IO	Ned2 digital IO
6	SW1	D02
7	SW2	D03
8		D04

Address	Description
0-8	Digital I/O mode (Input = 1, Output = 0)
100-108	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.

Address	Description
0-8	Digital I/O mode (Input = 1, Output = 0)
100-108	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

Address	Description
510	Opens gripper previously updated
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)
650	Set Analog IO - Arg: [Analog IO number, voltage 0V- 5000mV]

'*' 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.

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

Address	Description
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)
600 - 604	Analog IO mode
610 - 614	Analog IO state in mV

Analog IO addresses offset table

Address offset	Niryo One / Ned analog IO	Ned2 analog IO
0	/	AI1
1	/	AI2
2	/	AO1
3	/	AO2

Dependencies - Modbus TCP Server

- pymodbus library
- Niryo_robot_msgs
- std_msgs

Modbus Examples

Examples of Modbus python lib can be found here Python Modbus examples (https://github.com/NiryoRobotics/ned_ros/tree/master/niryo_robot_modbus/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.

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.

```python
#!/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
```

```python
    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:
        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))

    client.write_registers(0, joints_to_send)
```

```
    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"
```

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 toPyNiryo 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.

January 2022 release - Niryo One & Ned compatibility - Hardware Stack refinement

Requirements

Ubuntu packages

- sqlite3
- ffmpeg
- build-essential
- catkin
- python-catkin-pkg
- python-pymodbus
- python-rosdistro
- python-rospkg
- python-rosdep-modules
- python-rosinstall python-rosinstall-generator
- python-wstool
- ros-melodic-moveit
- ros-melodic-control
- ros-melodic-controllers
- ros-melodic-tf2-web-republisher
- ros-melodic-rosbridge-server
- ros-melodic-joint-state-publisher-gui

Python libraries

See *src/requirements_ned2.txt* file

Packages

New packages

- niryo_robot_database
- niryo_robot_led_ring
- niryo_robot_metrics
- niryo_robot_reports
- niryo_robot_sound
- niryo_robot_status
- niryo_robot_hardware_stack/can_debug_tools
- niryo_robot_hardware_stack/common
- niryo_robot_hardware_stack/end_effector_interface
- niryo_robot_hardware_stack/serial

Renamed packages

- niryo_ned_moveit_config_standalone becomes niryo_moveit_config_standalone
- niryo_ned_moveit_config_w_gripper1 becomes niryo_moveit_config_w_gripper1
- niryo_robot_hardware_stack/stepper_driver becomes niryo_robot_hardware_stack/can_driver
- niryo_robot_hardware_stack/dynamixel_driver becomes niryo_robot_hardware_stack/ttl_driver

- niryo_robot_hardware_stack/niryo_robot_debug becomes niryo_robot_hardware_stack/ttl_debug_tools

Removed packages

- niryo_robot_serial_number
- niryo_robot_unit_tests
- niryo_robot_hardware_stack/fake_interface

Cleaning and Refactoring

- roslint compliant
- catkin lint compliant for most part
- add xsd validation for launch files and package.xml files
- updated packages format to version 3
- updated c++ version to c++14
- clang and clazy compliance improvement
- rosdoc_lite set up in all packages
- catkin_tools compliant
- install space working
- sphinx_doc restructuration
- add hardware_version discrimination between ned, one and ned2
- add ned2 configuration files in all packages
- niryo_robot_arm_commmander refactoring
- niryo_robot_python_ros_wrapper refactoring

Features (for Ned and One only)

- add VERSION file at root
- add CHANGELOG.rst in every package (using catkin_generate_changelog tool)
- update PID values for Dynamixels
- Replace fake interface by mock drivers for steppers and Dynamixels
- Add compatibility for TTL conveyor belts (upcoming)
- Add Ned2 features (upcoming)
- niryo_robot_bringup refactoring
- improve control loops for ttl_driver and joints interface

Know issues (for Ned and One only)

Can't scan 2 conveyors at the same time. Please scan the conveyors one by one.

Limitations

- Calibration deactivated on Simulated Ned and One
- Not officially supporting Ned2 hardware version
- Hotspot mode is always on by default on reboot for the Niryo One

Niryo Studio

New features

- Network settings (DHCP / Static IP)
- Hardware detection One / Ned / Ned2
- Display TCP Speed
- Blockly - Dynamic blocks (Saved pose, workspace)

Bugs fix

- Blockly - Conversion RAD / DEG in block

September release - New features batch

Features

Tool commander package

- TCP service settings

  TCP.msg

  SetTCP.srv

Arm commander package

- New movements available in ArmMoveCommand.msg

  linear pose

  shift linear pose

  trajectory

Python ROS Wrapper package

- New movement functions available

  move linear pose

  linear pose

  jog pose shift

  jog joints shift

  shift linear pose

  execute trajectory from pose
- New TCP functions available

  set_tcp

  enable_tcp

  reset_tcp
- New camera settings functions available

  set_brightness

  set_contrast

  set_saturation

Improvements

- Refactoring Tool Commander and Robot Commander packages.

  Remove Robot Commander package

  Reorder Robot Commander package between Tool Commander and Arm Commander packages.
- Self collision detection

  Add self-collision detection via MoveIt.
- Collision detection

  Collision detection improvement on each joints.

  Learning mode activation in case of a collision.

Suggest a modification Download as PDF