PyNiryo Documentation

This documentation presents Ned’s PyPi package, which is a TCP API made with Python.

It offers a simple way for developers to create programs for robot and to control them via remote communication from their computers. Contrary to the Python ROS Wrapper, the user will not need to be connected on the robot through a terminal.

Note

This package is able to control Ned in simulation as well as the physical robot.

Ned

Version compatibility

Version compatibility table

PyNiryo version	ROS Stack version	Robot
<= 1.0.5	<= 3.2.0	Ned
1.1.0	4.0.0	Niryo One , Ned , Ned2
1.1.1	4.0.1	Niryo One , Ned , Ned2
1.1.2	>=4.1.1	Niryo One , Ned , Ned2

Before getting started

If you haven't already done so, make sure to learn about the ROS robot software by readingROS documentation.

This documentation also contains everything you need to know if you want to use Ned through simulation.

Sections organization

This document is organized in 4 main sections.

Setup

Install & Setup your environment in order to use Ned with PyNiryo.

Firstly, follow Installation instructions (index.html#document-source/setup/installation), then find your Robot IP address (index.html#document-source/setup/ip_address) to be ready.

Installation

The library uses Python, which must be installed and available in your working environment.
The version should be **equal or above**:

- 2.7 if you are using Python 2
- 3.6 if you are using Python 3

Hint

To check your Python version, use the command `python --version` if you are using Python 2 and `python3 --version` if you are using Python3.

The below sections explain how to install the library with **pip**, the package installer for Python.

Attention

If you have both Python 2 & Python 3 installed on your computer, the command `pip` will install packages in Python 2 version. You should use `pip3` instead in order to target Python 3.

Installation with pip

You need to install Numpy package beforehand:

```
pip install numpy
```

To install Ned's Python package via `pip`, simply execute:

```
pip install pyniryo
```

You can find more information about the PyPi packagehere (https://pypi.org/project/pyniryo/).

If you also want to use Vision functions to do your own Image Processing pipeline install OpenCV via the command:

```
pip install opencv-python
```

Uninstall

To uninstall the library use:

```
pip uninstall pyniryo
```

Find your Robot's IP address

In order to use your robot through TCP connection, you will first need to connect to it, which implies that you know its IP address.

The next sections explain how to find your robot IP according to your configuration:

- Hotspot mode
- Simulation or directly on the robot
- Direct ethernet connection
- Computer and robot connected on the same router
- Make IP permanent

Hotspot mode

If you are directly connected to your robot through its wi-fi, the IP address you will need to use is `10.10.10.10` .

Simulation or directly on the robot

In this situation, the robot is running on the same computer as the client, the IP address will be the localhost address `127.0.0.1` .

Direct ethernet connection

If you are directly connected to your robot with an ethernet cable, the static IP of your robot will be `169.254.200.200` .

The reader should note that he may need to change his wired settings to allow the connection. See how to Connect to Ned via Ethernet on Ubuntu (https://docs.niryo.com/applications/ned/source/tutorials/setup_connect_ned_ethernet.html).

Computer and robot connected on the same router

You will need to find the robot's address using `nmap` , or you can also use search button of Niryo Studio to see which robots are available.

For example for our network configuration, we have:

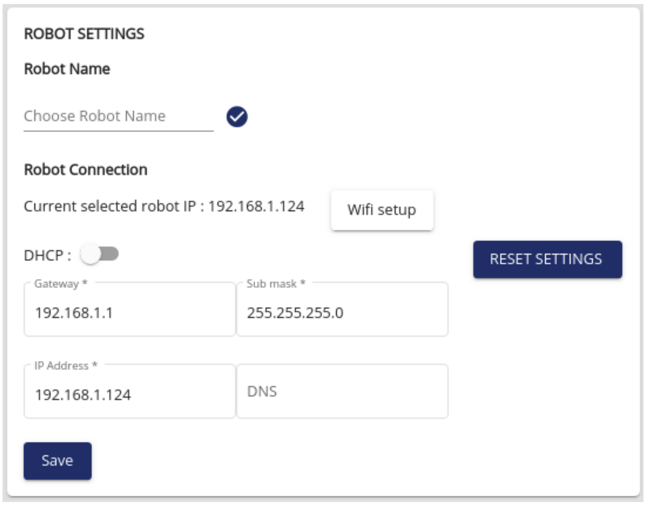
```
$ nmap -sP 192.168.1.0/24

Starting Nmap 7.60 ( https://nmap.org ) at 2022-01-10 17:10 CET
Nmap scan report for my_router.home (192.168.1.1)
Host is up (0.010s latency).
Nmap scan report for my_computer.home (192.168.1.10)
Host is up (0.010s latency).
Nmap scan report for niryo_pi4-11.home (192.168.1.107)
Host is up (0.0034s latency).
```

In our case the ip of our robot is **192.168.1.107**.

You can also Make IP permanent so that you will not have to search for it next time

Make IP permanent

Go on NiryoStudio in the network configuration section. For a dynamic IP address (recommended especially if you are a beginner), use the DHCP option. You will find the new IP address of the robot by running a scan of the robots on the network on Niryo Studio. Otherwise fill in the Gateway, Sub mask and IP address fields.


ROBOT SETTINGS

Robot Name

Choose Robot Name

Robot Connection

Current selected robot IP : 192.168.1.124

DHCP :

Gateway * 192.168.1.1

Sub mask * 255.255.255.0

IP Address * 192.168.1.124

DNS

Setting up a custom static IP on Niryo Studio

Verify your Setup and Get Started

In order to verify your computer’s setup, we are going to run a program from it, and see if the robot answers as expected.

Note

Before verifying your setup, be sure that your physical robot (or simulation) is turned on.

Firstly, go in the folder of your choice and create an empty file named “pyniryo_test.py”. This file will contain the checking code.

Edit this file and fill it with the following code:

```python
from pyniryo import *

robot_ip_address = "10.10.10.10"

# Connect to robot & calibrate
robot = NiryoRobot(robot_ip_address)
robot.calibrate_auto()
# Move joints
robot.move_joints(0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
# Turn learning mode ON
robot.set_learning_mode(True)
# Stop TCP connection
robot.close_connection()
```

Attention

Replace the third line with your Robot IP Address (index.html#document-source/setup/ip_address) if you are not using Hotspot Mode.

Still on your computer, open a terminal, and place your current directory in the same folder than your file. Then, run the command:

```
python pyniryo_test.py
```

Note

If you are using Python 3, you may need to change `python` to `python3` .

If your robot starts calibrating, then moves, and finally, goes to learning mode, your setup is validated, you can now start coding!

Examples

Learn how to use the PyNiryo package to implement various tasks.

Examples: Basics

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

Danger

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

Your first move joint

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

```python
from pyniryo import *

robot = NiryoRobot("10.10.10.10")

robot.calibrate_auto()

robot.move_joints(0.2, -0.3, 0.1, 0.0, 0.5, -0.8)

robot.close_connection()
```

Code Details - First Move J

First of all, we import the library to be able to access functions.

```
from pyniryo import *
```

Then, we instantiate the connection and link the variable robot to the robot at the IP Address 10.10.10.10 .

```
robot = NiryoRobot("10.10.10.10")
```

Once the connection is done, we calibrate the robot using its **calibrate_auto()** (index.html#api.tcp_client.NiryoRobot.calibrate_auto) function.

```
robot.calibrate_auto()
```

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

```
robot.move_joints(0.2, -0.3, 0.1, 0.0, 0.5, -0.8)
```

Our process is now over, we can close the connection with **quit()** .

```
robot.close_connection()
```

Your first pick and place

In the second example, we are going to develop a pick and place algorithm.

```
from pyniryo import *

robot = NiryoRobot("10.10.10.10")

robot.calibrate_auto()
robot.update_tool()

robot.release_with_tool()
robot.move_pose(0.2, -0.1, 0.25, 0.0, 1.57, 0.0)
robot.grasp_with_tool()

robot.move_pose(0.2, 0.1, 0.25, 0.0, 1.57, 0.0)
robot.release_with_tool()

robot.close_connection()
```

Code Details - First Pick And Place

First of all, we import the library and start the connection between our computer and the robot. We also calibrate the robot.

```
from pyniryo import *

robot = NiryoRobot("10.10.10.10")
robot.calibrate_auto()
```

Then, we equip the tool with **update_tool()** (index.html#api.tcp_client.NiryoRobot.update_tool).

```
robot.update_tool()
```

Now that our initialization is done, we can open the gripper (or push air from the vacuum) with **release_with_tool()** (index.html#api.tcp_client.NiryoRobot.release_with_tool), go to the picking pose via **move_pose()** (index.html#api.tcp_client.NiryoRobot.move_pose) & then catch an object with **grasp_with_tool()** (index.html#api.tcp_client.NiryoRobot.grasp_with_tool)!

```
robot.release_with_tool()
robot.move_pose(0.2, -0.1, 0.25, 0.0, 1.57, 0.0)
robot.grasp_with_tool()
```

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

```
robot.move_pose(0.2, 0.1, 0.25, 0.0, 1.57, 0.0)
robot.release_with_tool()
```

Our process is now over, we can close the connection.

```
robot.close_connection()
```

Notes

You may not have fully understood how to move the robot and use PyNiryo and that is totally fine because you will find more details on the next examples page!

The important thing to remember from this page is how to import the library, connect to the robot & call functions.

Examples: Movement

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

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

Important

In the following sections, you are supposed to be already connected to a calibrated robot. The robot's instance is saved in the variable `robot` . To know how to do so, go look at section Examples: Basics (index.html#document-source/examples/examples_basics).

Danger

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

Joints

Move Joints

To make a moveJ, you can either provide:

- 6 floats : `j1, j2, j3, j4, j5, j6`
- a list of 6 floats : `[j1, j2, j3, j4, j5, j6]`

It is possible to provide these parameters to the function **move_joints()** (index.html#api.tcp_client.NiryoRobot.move_joints) or via the `joints` setter, at your convenience:

```
# Moving Joints with function & 6 floats
robot.move_joints(0.0, 0.0, 0.0, 0.0, 0.0, 0.0)

# Moving Joints with function & a list of floats
robot.move_joints([-0.5, -0.6, 0.0, 0.3, 0.0, 0.0])

# Moving Joints with setter & 6 floats
robot.joints = 0.2, -0.4, 0.0, 0.0, 0.0, 0.0

# Moving Joints with setter & a list of floats
robot.joints = [-0.2, 0.3, 0.2, 0.3, -0.6, 0.0]
```

You should note that these 4 commands are doing exactly the same thing! In your future scripts, chose the one you prefer, but try to remain consistent to keep a good readability.

Get Joints

To get actual joint positions, you can use the function **get_joints()** (index.html#api.tcp_client.NiryoRobot.get_joints) or the `joints` getter. Both will return a list of the 6 joints position:

```
# Getting Joints with function
joints_read = robot.get_joints()

# Getting Joints with getter
joints_read = robot.joints
```

Hint

As we are developing in Python, we can unpack list very easily, which means that we can retrieve joints value in 6 variables by writing `j1, j2, j3, j4, j5, j6 = robot.get_joints()` .

Pose

Move Pose

To perform a moveP, you can provide:

- 6 floats : x, y, z, roll, pitch, yaw
- a list of 6 floats : [x, y, z, roll, pitch, yaw]
- a **PoseObject** (index.html#api.objects.PoseObject)

As for MoveJ, it is possible to provide these parameters to the function **move_pose()** (index.html#api.tcp_client.NiryoRobot.move_pose) or the `pose` setter, at your convenience:

```
pose_target = [0.2, 0.0, 0.2, 0.0, 0.0, 0.0]
pose_target_obj = PoseObject(0.2, 0.0, 0.2, 0.0, 0.0, 0.0)

# Moving Pose with function
robot.move_pose(0.2, 0.0, 0.2, 0.0, 0.0, 0.0)
robot.move_pose(pose_target)
robot.move_pose(pose_target_obj)

# Moving Pose with setter
robot.pose = (0.2, 0.0, 0.2, 0.0, 0.0, 0.0)
robot.pose = pose_target
robot.pose = pose_target_obj
```

Each of these 6 commands are doing the same thing.

Get Pose

To get end effector actual pose, you can use the function **get_pose()** (index.html#api.tcp_client.NiryoRobot.get_pose) or the pose getter. Both will return a **PoseObject** (index.html#api.objects.PoseObject):

```
# Getting Joints with function
pose_read = robot.get_pose()

# Getting Joints with getter
pose_read = robot.pose
```

How to use the PoseObject

The **PoseObject** (index.html#api.objects.PoseObject) is a Python object which allows to store all poses' 6 coordinates (x, y, z, roll, pitch, yaw) in one single instance. It can be converted into a list if needed with the method **to_list()** (index.html#api.objects.PoseObject.to_list).

It also allows to create new **PoseObject** (index.html#api.objects.PoseObject) with some offset, much easier than copying list and editing only 1 or 2 values. For instance, imagine that we want to shift the place pose by 5 centimeters at each iteration of a loop, you can use the **copy_with_offsets()** (index.html#api.objects.PoseObject.copy_with_offsets) method:

```
pick_pose = PoseObject(
x=0.30, y=0.0, z=0.15,
roll=0, pitch=1.57, yaw=0.0
)
first_place_pose = PoseObject(
    x=0.0, y=0.2, z=0.15,
    roll=0, pitch=1.57, yaw=0.0
)
for i in range(5):
    robot.move_pose(pick_pose)
    new_place_pose = first_place_pose.copy_with_offsets(x_offset=0.05 * i)
    robot.move_pose(new_place_pose)
```

Examples: Tool action

This page shows how to control Ned's tools.

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

Important

In this section, you are already supposed to be connected to a calibrated robot. The robot instance is saved in the variable robot .

Danger

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

Tool control

Equip Tool

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

To do that, we should use the function **update_tool()** (index.html#api.tcp_client.NiryoRobot.update_tool) which takes no argument. It will scan motor's connections and set the new tool!

The line to equip a new tool is:

```
robot.update_tool()
```

Note

For the Grasping and Releasing sections, this command should be added in your codes! If you want to use a specific tool, you need to store the **ToolID** you are using in a variable named tool_used .

Grasping

To grasp with any tool, you can use the function **grasp_with_tool()** (index.html#api.tcp_client.NiryoRobot.grasp_with_tool). This action corresponds to:

- Close gripper for Grippers
- Pull Air for Vacuum Pump
- Activate for Electromagnet

The line to grasp is:

```
robot.grasp_with_tool()
```

To grasp an object by specifying the tool:

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

Releasing

To release with any tool, you can use the function **release_with_tool()** (index.html#api.tcp_client.NiryoRobot.release_with_tool). This action corresponds to:

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

To release an object by specifying parameters:

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

Pick & Place with tools

A Pick & Place is an action which consists in going to a certain pose in order to pick an object and then, going to another pose to place it.

This operation can be proceed as follows:

1. Going over the object with a certain offset to avoid collision;
2. Going down to the object's height;
3. Grasping with tool;
4. Going back to step 1's pose;
5. Going over the place pose with a certain offset to avoid collision;
6. Going down to place's height;
7. Releasing the object with tool;
8. Going back to step 5's pose.

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

Code Baseline

For the sake of brevity, every piece of code beside the Pick & Place function will not be rewritten for every method. So that, you will need to use the code and implement the Pick & Place function to it.

```python
# Imports
from pyniryo import *

tool_used = ToolID.XXX  # Tool used for picking
robot_ip_address = "x.x.x.x" # Robot address

# The pick pose
pick_pose = PoseObject(
    x=0.25, y=0., z=0.15,
    roll=-0.0, pitch=1.57, yaw=0.0,
)
# The Place pose
place_pose = PoseObject(
    x=0.0, y=-0.25, z=0.1,
    roll=0.0, pitch=1.57, yaw=-1.57)

def pick_n_place_version_x(robot):
    # -- -------------- -- #
    # -- CODE GOES HERE -- #
    # -- -------------- -- #

if __name__ == '__main__':
    # Connect to robot
    client = NiryoRobot(robot_ip_address)
    # Calibrate robot if robot needs calibration
    client.calibrate_auto()
    # Changing tool
    client.update_tool()

    pick_n_place_version_x(client)

    # Releasing connection
    client.close_connection()
```

First Solution: the heaviest

For this first function, every steps are done by hand, as well as poses computing.

Note

In this example, the tool used is a Gripper. If you want to use another tool than a gripper, do not forget to adapt grasp & release functions!

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

    # Going Over Object
    robot.move_pose(pick_pose.x, pick_pose.y, pick_pose.z + height_offset,
                    pick_pose.roll, pick_pose.pitch, pick_pose.yaw)
    # Opening Gripper
    robot.open_gripper(gripper_speed)
    # Going to picking place and closing gripper
    robot.move_pose(pick_pose)
    robot.close_gripper(gripper_speed)

    # Raising
    robot.move_pose(pick_pose.x, pick_pose.y, pick_pose.z + height_offset,
                    pick_pose.roll, pick_pose.pitch, pick_pose.yaw)

    # Going Over Place pose
    robot.move_pose(place_pose.x, place_pose.y, place_pose.z + height_offset,
                    place_pose.roll, place_pose.pitch, place_pose.yaw)
    # Going to Place pose
    robot.move_pose(place_pose)
    # Opening Gripper
    robot.open_gripper(gripper_speed)
    # Raising
    robot.move_pose(place_pose.x, place_pose.y, place_pose.z + height_offset,
                    place_pose.roll, place_pose.pitch, place_pose.yaw)
```

Second Solution: Use of PoseObject

For the second solution, we use a **PoseObject** (index.html#api.objects.PoseObject) in order to calculate approach poses more easily.

Note

To see more about **PoseObject** (index.html#api.objects.PoseObject), go look at PoseObject dedicated section (index.html#how-to-use-the-poseobject)

```
def pick_n_place_version_2(robot):
    height_offset = 0.05  # Offset according to Z-Axis to go over pick & place poses

    pick_pose_high = pick_pose.copy_with_offsets(z_offset=height_offset)
    place_pose_high = place_pose.copy_with_offsets(z_offset=height_offset)

    # Going Over Object
    robot.move_pose(pick_pose_high)
    # Opening Gripper
    robot.release_with_tool()
    # Going to picking place and closing gripper
    robot.move_pose(pick_pose)
    robot.grasp_with_tool()
    # Raising
    robot.move_pose(pick_pose_high)

    # Going Over Place pose
    robot.move_pose(place_pose_high)
    # Going to Place pose
    robot.move_pose(place_pose)
    # Opening Gripper
    robot.release_with_tool(gripper_speed)
    # Raising
    robot.move_pose(place_pose_high)
```

Third Solution: Pick from pose & Place from pose functions

For those who have already seen the API Documentation, you may have seen pick & place dedicated functions!

In this example, we use **pick_from_pose()** (index.html#api.tcp_client.NiryoRobot.pick_from_pose) and **place_from_pose()** (index.html#api.tcp_client.NiryoRobot.place_from_pose) in order to split our function in only 2 commands!

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

Fourth Solution: All in one

The example exposed in the previous section could be useful if you want to do an action between the pick & the place phases.

For those who want to do everything in one command, you can use the **pick_and_place()** (index.html#api.tcp_client.NiryoRobot.pick_and_place) function!

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

Examples : Conveyor Belt

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

If you want to see more about Ned's Conveyor Belt functions, you can look at PyNiryo - Conveyor (index.html#conveyor)

Danger

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

Simple Conveyor control

This short example shows how to connect a conveyor and launch its motor (control it by setting its speed and direction):

```python
from pyniryo import *

# Connecting to robot
robot = NiryoRobot(<robot_ip_address>)

# Activating connexion with Conveyor Belt
conveyor_id = robot.set_conveyor()

# Running the Conveyor Belt at 50% of its maximum speed, in forward direction
robot.run_conveyor(conveyor_id, speed=50, direction=ConveyorDirection.FORWARD)

# Waiting 3 seconds
robot.wait(3)

# Stopping robot's motor
robot.stop_conveyor(conveyor_id)

# Deactivating connexion with the Conveyor Belt
robot.unset_conveyor(conveyor_id)
```

Advanced Conveyor Belt control

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

```python
from pyniryo import *

# -- Setting variables
sensor_pin_id = PinID.GPIO_1A

catch_nb = 5

# The pick pose
pick_pose = PoseObject(
    x=0.25, y=0., z=0.15,
    roll=-0., pitch=1.57, yaw=0.0,
)
# The Place pose
place_pose = PoseObject(
    x=0., y=-0.25, z=0.1,
    roll=0., pitch=1.57, yaw=-1.57)

# -- MAIN PROGRAM

# Connecting to the robot
robot = NiryoRobot(<robot_ip_address>)

# Activating connexion with the Conveyor Belt
conveyor_id = robot.set_conveyor()

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

    # Stopping robot's motor
    robot.stop_conveyor(conveyor_id)
    # Making a pick & place
    robot.pick_and_place(pick_pose, place_pose)

# Deactivating connexion with the Conveyor Belt
robot.unset_conveyor(conveyor_id)
```

Examples: Vision

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

If you want to see more about Ned's Vision functions, you can look at PyNiryo - Vision
If you want to see how to do Image Processing, go check out theImage Processing section.

Note

Even if you do not own a Vision Set, you can still realize these examples with the Gazebo simulation version.

Danger

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

Needed piece of code

Important

In order to achieve the following examples, you need to create a vision workspace. In this page, the workspace used is named `workspace_1` . To create it, the user should go on Niryo Studio!

As the examples start always the same, add the following lines at the beginning of codes:

```python
# Imports
from pyniryo import *

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

# The pose from where the image processing happens
observation_pose = PoseObject(
    x=0.16, y=0.0, z=0.35,
    roll=0.0, pitch=1.57, yaw=0.0,
)
# Place pose
place_pose = PoseObject(
    x=0.0, y=-0.2, z=0.12,
    roll=0.0, pitch=1.57, yaw=-1.57
)

# - Initialization

# Connect to robot
robot = NiryoRobot(robot_ip_address)
# Calibrate robot if the robot needs calibration
robot.calibrate_auto()
# Updating tool
robot.update_tool()

# --- -------------- --- #
# --- CODE GOES HERE --- #
# --- -------------- --- #

robot.close_connection()
```

Hint

All the following examples are only a part of what can be made with the API in terms of Vision. We advise you to look at API - Vision (index.html#vision) to understand more deeply

Simple Vision Pick & Place

The goal of a Vision Pick & Place is the same as a classical Pick & Place, with a close difference: the camera detects where the robot has to go in order to pick!

This short example shows how to do your first Vision pick using the **vision_pick()** (index.html#api.tcp_client.NiryoRobot.vision_pick) function:

```python
robot.move_pose(observation_pose)
# Trying to pick target using camera
obj_found, shape_ret, color_ret = robot.vision_pick(workspace_name)
if obj_found:
    robot.place_from_pose(place_pose)

robot.set_learning_mode(True)
```

Code Details - Simple Vision Pick and Place

To execute a Vision pick, we firstly need to go to a place where the robot will be able to see the workspace:

```python
robot.move_pose(observation_pose)
```

Then, we try to perform a Vision pick in the workspace with the **vision_pick()** (index.html#api.tcp_client.NiryoRobot.vision_pick) function:

```
obj_found, shape_ret, color_ret = robot.vision_pick(workspace_name)
```

Variables shape_ret and color_ret are respectively of type **ObjectShape** and **ObjectColor**, and store the shape and the color of the detected object! We will not use them for this first example.

The obj_found variable is a boolean which indicates whereas an object has been found and picked, or not. Thus, if the pick worked, we can place the object at the place pose.

```
if obj_found:
    robot.place_from_pose(place_pose)
```

Finally, we turn learning mode on:

```
robot.set_learning_mode(True)
```

Note

If your obj_found variable indicates False, check that:

- Nothing obstructs the camera field of view
- Workspace's 4 markers are visible
- At least 1 object is placed fully inside the workspace

First conditioning via Vision

In most of use cases, the robot will need to perform more than one Pick & Place. In this example, we will see how to condition multiple objects according to a straight line:

```
# Initializing variables
offset_size = 0.05
max_catch_count = 4

# Loop until enough objects have been caught
catch_count = 0
while catch_count < max_catch_count:
    # Moving to observation pose
    robot.move_pose(observation_pose)

    # Trying to get object via Vision Pick
    obj_found, shape, color = robot.vision_pick(workspace_name)
    if not obj_found:
        robot.wait(0.1)
        continue

    # Calculate place pose and going to place the object
    next_place_pose = place_pose.copy_with_offsets(x_offset=catch_count * offset_size)
    robot.place_from_pose(next_place_pose)

    catch_count += 1

robot.go_to_sleep()
```

Code Details - First Conditioning via Vision

We want to catch max_catch_count objects, and space each of them by offset_size meter:

```
offset_size = 0.05
max_catch_count = 4
```

We start a loop until the robot has caught max_catch_count objects:

```
catch_count = 0
while catch_count < max_catch_count:
```

For each iteration, we firstly go to the observation pose and then, try to make a Vision pick in the workspace:

```
robot.move_pose(observation_pose)

obj_found, shape, color = robot.vision_pick(workspace_name)
```

If the Vision pick failed, we wait 0.1 second and then, start a new iteration:

```
if not obj_found:
    robot.wait(0.1)
    continue
```

Else, we compute the new place position according to the number of catches, and then, go placing the object at that place:

```
next_place_pose = place_pose.copy_with_offsets(x_offset=catch_count * offset_size)
robot.place_from_pose(next_place_pose)
```

We also increment the catch_count variable:

```
catch_count += 1
```

Once the target catch number is achieved, we go to sleep:

```
robot.go_to_sleep()
```

Multi Reference Conditioning

During a conditioning task, objects may not always be placed as the same place according to their type. In this example, we will see how to align object according to their color, using the color element **ObjectColor** returned by **vision_pick()** (index.html#api.tcp_client.NiryoRobot.vision_pick) function:

```
# Distance between elements
offset_size = 0.05
max_failure_count = 3

# Dict to write catch history
count_dict = {
    ObjectColor.BLUE: 0,
    ObjectColor.RED: 0,
    ObjectColor.GREEN: 0,
}

try_without_success = 0
# Loop until too much failures
while try_without_success < max_failure_count:
    # Moving to observation pose
    robot.move_pose(observation_pose)
    # Trying to get object via Vision Pick
    obj_found, shape, color = robot.vision_pick(workspace_name)
    if not obj_found:
        try_without_success += 1
        robot.wait(0.1)
        continue

    # Choose X position according to how the color line is filled
    offset_x_ind = count_dict[color]

    # Choose Y position according to ObjectColor
    if color == ObjectColor.BLUE:
        offset_y_ind = -1
    elif color == ObjectColor.RED:
        offset_y_ind = 0
    else:
        offset_y_ind = 1

    # Going to place the object
    next_place_pose = place_pose.copy_with_offsets(x_offset=offset_x_ind * offset_size,
                                                   y_offset=offset_y_ind * offset_size)
    robot.place_from_pose(next_place_pose)

    # Increment count
    count_dict[color] += 1
    try_without_success = 0

robot.go_to_sleep()
```

Code Details - Multi Reference Conditioning

We want to catch objects until Vision Pick failed `max_failure_count` times. Each of the object will be put on a specific column according to its color. The number of catches for each color will be stored on a dictionary `count_dict` .

```
# Distance between elements
offset_size = 0.05
max_failure_count = 3

# Dict to write catch history
count_dict = {
    ObjectColor.BLUE: 0,
    ObjectColor.RED: 0,
    ObjectColor.GREEN: 0,
}

try_without_success = 0
# Loop until too much failures
while try_without_success < max_failure_count:
```

For each iteration, we firstly go to the observation pose and then, try to make a Vision pick in the workspace:

```
robot.move_pose(observation_pose)

obj_found, shape, color = robot.vision_pick(workspace_name)
```

If the Vision pick failed, we wait 0.1 second and then, start a new iteration, without forgetting to increment the failure counter:

```
if not obj_found:
    try_without_success += 1
    robot.wait(0.1)
    continue
```

Else, we compute the new place position according to the number of catches, and then, go place the object at that place:

```
# Choose X position according to how the color line is filled
offset_x_ind = count_dict[color]

# Choose Y position according to ObjectColor
if color == ObjectColor.BLUE:
    offset_y_ind = -1
elif color == ObjectColor.RED:
    offset_y_ind = 0
else:
    offset_y_ind = 1

# Going to place the object
next_place_pose = place_pose.copy_with_offsets(x_offset=offset_x_ind * offset_size,
                                               y_offset=offset_y_ind * offset_size)
robot.place_from_pose(next_place_pose)
```

We increment the `count_dict` dictionary and reset `try_without_success` :

```
count_dict[color] += 1
try_without_success = 0
```

Once the target catch number is achieved, we go to sleep:

```
robot.go_to_sleep()
```

Sorting Pick with Conveyor

An interesting way to bring objects to the robot, is the use of a Conveyor Belt. In this examples, we will see how to catch only a certain type of object by stopping the conveyor as soon as the object is detected on the workspace.

```
# Initializing variables
offset_size = 0.05
max_catch_count = 4
shape_expected = ObjectShape.CIRCLE
color_expected = ObjectColor.RED

conveyor_id = robot.set_conveyor()

catch_count = 0
while catch_count < max_catch_count:
    # Turning conveyor on
    robot.run_conveyor(conveyor_id)
    # Moving to observation pose
    robot.move_pose(observation_pose)
    # Check if object is in the workspace
    obj_found, pos_array, shape, color = robot.detect_object(workspace_name,
                                                             shape=shape_expected,
                                                             color=color_expected)
    if not obj_found:
        robot.wait(0.5)  # Wait to let the conveyor turn a bit
        continue
    # Stopping conveyor
    robot.stop_conveyor(conveyor_id)
    # Making a vision pick
    obj_found, shape, color = robot.vision_pick(workspace_name,
                                                shape=shape_expected,
                                                color=color_expected)
    if not obj_found:  # If visual pick did not work
        continue

    # Calculate place pose and going to place the object
    next_place_pose = place_pose.copy_with_offsets(x_offset=catch_count * offset_size)
    robot.place_from_pose(next_place_pose)

    catch_count += 1

# Stopping & unsetting conveyor
robot.stop_conveyor(conveyor_id)
robot.unset_conveyor(conveyor_id)

robot.go_to_sleep()
```

Code Details - Sort Picking

Firstly, we initialize your process: we want the robot to catch 4 red circles. To do so, we set variables `shape_expected` and `color_expected` with **ObjectShape.CIRCLE** and **ObjectColor.RED** .

```
offset_size = 0.05
max_catch_count = 4
shape_expected = ObjectShape.CIRCLE
color_expected = ObjectColor.RED
```

We activate the connection with the Conveyor Belt and start a loop until the robot has caught `max_catch_count` objects:

```
conveyor_id = robot.set_conveyor()

catch_count = 0
while catch_count < max_catch_count:
```

For each iteration, we firstly run the Conveyor Belt (if the latter is already running, nothing will happen), then go to the observation pose:

```
# Turning the Conveyor Belt on
robot.run_conveyor(conveyor_id)
# Moving to observation pose
robot.move_pose(observation_pose)
```

We then check if an object corresponding to our criteria is in the workspace. If not, we wait 0.5 second and then, start a new iteration:

```
obj_found, pos_array, shape, color = robot.detect_object(workspace_name,
                                                         shape=shape_expected,
                                                         color=color_expected)
if not obj_found:
    robot.wait(0.5)  # Wait to let the conveyor turn a bit
    continue
```

Else, stop the Conveyor Belt and try to make a Vision pick:

```
# Stopping Conveyor Belt
robot.stop_conveyor(conveyor_id)
# Making a Vision pick
obj_found, shape, color = robot.vision_pick(workspace_name,
                                            shape=shape_expected,
                                            color=color_expected)
if not obj_found:  # If visual pick did not work
    continue
```

If Vision Pick succeed, compute new place pose, and place the object:

```
# Calculate place pose and going to place the object
next_place_pose = place_pose.copy_with_offsets(x_offset=catch_count * offset_size)
robot.place_from_pose(next_place_pose)

catch_count += 1
```

Once the target catch number is achieved, we stop the Conveyor Belt and go to sleep:

```
# Stopping & unsetting Conveyor Belt
robot.stop_conveyor(conveyor_id)
robot.unset_conveyor(conveyor_id)

robot.go_to_sleep()
```

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 PyNiryo - 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:

```python
from pyniryo import *

robot_ip_address = "192.168.1.91"
gripper_speed = 400

if __name__ == '__main__':
    robot = NiryoRobot(robot_ip_address)

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

    robot.save_dynamic_frame_from_points("dynamic_frame", "description", point_o, point_x, point_y)

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

    # Pick
    robot.open_gripper(gripper_speed)
    # Move to the frame
    initial_pose = PoseObject(0, 0, 0, 0, 1.57, 0)
    robot.move_pose(initial_pose, "dynamic_frame")
    robot.close_gripper(gripper_speed)

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

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

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

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

Code templates

As code structures are always the same, we wrote down few templates for you to start your code file with a good form.

The short template

Very simple, straightforward:

```python
from pyniryo import *

# Connect to robot & calibrate
robot = NiryoRobot(<robot_ip_address>)
robot.calibrate_auto()

# --- --------- --- #
# --- YOUR CODE --- #
# --- --------- --- #

# Releasing connection
robot.close_connection()
```

Advanced template

This template let the user define his own process but it handles connection, calibration, tool equipping, and makes the robot go to sleep at the end:

```
from pyniryo import *

local_mode = False # Or True
tool_used = ToolID.GRIPPER_1
# Set robot address
robot_ip_address_rpi = "x.x.x.x"
robot_ip_address_local = "127.0.0.1"

robot_ip_address = robot_ip_address_local if local_mode else robot_ip_address_rpi


def process(niryo_edu):
    # --- --------- --- #
    # --- YOUR CODE --- #
    # --- --------- --- #

if __name__ == '__main__':
    # Connect to robot
    robot = NiryoRobot(robot_ip_address)
    # Calibrate robot if robot needs calibration
    robot.calibrate_auto()
    # Equip tool
    robot.update_tool()
    # Launching main process
    process(client)
    # Ending
    robot.go_to_sleep()
    # Releasing connection
    robot.close_connection()
```

Advanced template for Conveyor Belt

Same as Advanced template but with a Conveyor Belt

```
from pyniryo import *

# Set robot address
robot_ip_address = "x.x.x.x"


def process(robot, conveyor_id):
    robot.run_conveyor(conveyor_id)

    # --- --------- --- #
    # --- YOUR CODE --- #
    # --- --------- --- #

    robot.stop_conveyor()


if __name__ == '__main__':
    # Connect to robot
    robot = NiryoRobot(robot_ip_address)
    # Calibrate robot if robot needs calibration
    robot.calibrate_auto()
    # Equip tool
    robot.update_tool()
    # Activating connexion with conveyor
    conveyor_id = robot.set_conveyor()
    # Launching main process
    process(robot, conveyor_id)
    # Ending
    robot.go_to_sleep()
    # Deactivating connexion with conveyor
    robot.unset_conveyor(conveyor_id)
    # Releasing connection
    robot.close_connection()
```

Advanced template for Vision

Huge template for Vision users!

```python
from pyniryo import *

local_mode = False # Or True
workspace_name = "workspace_1"  # Robot's Workspace Name
# Set robot address
robot_ip_address_rpi = "x.x.x.x"
robot_ip_address_local = "127.0.0.1"

robot_ip_address = robot_ip_address_local if local_mode else robot_ip_address_rpi

# The pose from where the image processing happens
observation_pose = PoseObject(
    x=0.18, y=0.0, z=0.35,
    roll=0.0, pitch=1.57, yaw=-0.2,
)

# Center of the conditioning area
place_pose = PoseObject(
    x=0.0, y=-0.23, z=0.12,
    roll=0.0, pitch=1.57, yaw=-1.57
)

def process(robot):
    robot.move_pose(observation_pose)
    catch_count = 0
    while catch_count < 3:
        ret = robot.get_target_pose_from_cam(workspace_name,
                                             height_offset=0.0,
                                             shape=ObjectShape.ANY,
                                             color=ObjectColor.ANY)
        obj_found, obj_pose, shape, color = ret
        if not obj_found:
            continue
        catch_count += 1
        # --- --------- --- #
        # --- YOUR CODE --- #
        # --- --------- --- #
        robot.place_from_pose(place_pose)

if __name__ == '__main__':
    # Connect to robot
    robot = NiryoRobot(robot_ip_address)
    # Calibrate robot if robot needs calibration
    robot.calibrate_auto()
    # Equip tool
    robot.update_tool()
    # Launching main process
    process(client)
    # Ending
    robot.go_to_sleep()
    # Releasing connection
    robot.close_connection()
```

API Documentation

Master controls with PyNiryo with full the detailed functions here (index.html#document-source/api_doc/api).

Discover also Vision Functions (index.html#document-source/vision/image_processing_overview) to create your own image processing pipelines!

PyNiryo API Documentation

This file presents the different Command functions, Enums & Python object classes available with the API.

- Command functions are used to deal directly with the robot. It could be **move_joints()** , **get_hardware_status()** **vision_pick()** , or also **run_conveyor()**
- Enums are used to pass specific arguments to functions. For instance **PinState** , **ConveyorDirection** , ...
- Python object classes, as **PoseObject** , ease some operations

Command functions

This section references all existing functions to control your robot, which includes:

- Moving the robot
- Using Vision
- Controlling Conveyor Belts
- Playing with Hardware

All functions to control the robot are accessible via an instance of the class **NiryoRobot**

```
robot = NiryoRobot(<robot_ip_address>)
```

See examples on Examples: Basics (index.html#examples-basics)

List of functions subsections:

- TCP Connection
- Main purpose functions
- Joints & Pose
- Saved Poses
- Pick & Place
- Trajectories
- Dynamic frames
- Tools
- Hardware
- Conveyor
- Vision
- Led Ring
- Sound

TCP Connection

NiryoRobot.connect(*ip_address*)

Connect to the TCP Server

Parameters:
ip_address (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – IP Address

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.close_connection()

Close connection with robot

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Main purpose functions

NiryoRobot.calibrate(*calibrate_mode*)

Calibrate (manually or automatically) motors. Automatic calibration will do nothing if motors are already calibrated

Parameters:
calibrate_mode (*CalibrateMode*) – Auto or Manual

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.calibrate_auto()

Start a automatic motors calibration if motors are not calibrated yet

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.need_calibration()

Return a bool indicating whereas the robot motors need to be calibrate

Return type:
bool (https://docs.python.org/3/library/functions.html#bool)

NiryoRobot.get_learning_mode()

Get learning mode state

Returns:
True if learning mode is on

Return type:
bool (https://docs.python.org/3/library/functions.html#bool)

NiryoRobot.set_learning_mode(*enabled*)

Set learning mode if param is True , else turn it off

Parameters:
enabled (*bool* (https://docs.python.org/3/library/functions.html#bool)) – True or False

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.set_arm_max_velocity(*percentage_speed*)

Limit arm max velocity to a percentage of its maximum velocity

Parameters:
percentage_speed (*int* (https://docs.python.org/3/library/functions.html#int)) – Should be between 1 & 100

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.set_jog_control(*enabled*)

Set jog control mode if param is True, else turn it off

Parameters:
enabled (*bool* (https://docs.python.org/3/library/functions.html#bool)) – True or False

Return type:
None (https://docs.python.org/3/library/constants.html#None)

static **NiryoRobot.wait**(*duration*)

Wait for a certain time

Parameters:
duration (*float* (https://docs.python.org/3/library/functions.html#float)) – duration in seconds

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Joints & Pose

NiryoRobot.get_joints()

Get joints value in radians You can also use a getter

```
joints = robot.get_joints()
joints = robot.joints
```

Returns:
List of joints value

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]

NiryoRobot.get_pose()

Get end effector link pose as [x, y, z, roll, pitch, yaw]. x, y & z are expressed in meters / roll, pitch & yaw are expressed in radians You can also use a getter

```
pose = robot.get_pose()
pose = robot.pose
```

Return type:
PoseObject (index.html#api.objects.PoseObject)

NiryoRobot.get_pose_quat()

Get end effector link pose in Quaternion coordinates

Returns:
Position and quaternion coordinates concatenated in a list : [x, y, z, qx, qy, qz, qw]

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]

NiryoRobot.move_joints(**args*)

Move robot joints. Joints are expressed in radians.

All lines of the next example realize the same operation:

```
robot.joints = [0.2, 0.1, 0.3, 0.0, 0.5, 0.0]
robot.move_joints([0.2, 0.1, 0.3, 0.0, 0.5, 0.0])
robot.move_joints(0.2, 0.1, 0.3, 0.0, 0.5, 0.0)
```

Parameters:

args (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*]]*) – either 6 args (1 for each joints) or a list of 6 joints

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.move_pose(**args*)

Move robot end effector pose to a (x, y, z, roll, pitch, yaw, frame_name) pose in a particular frame (frame_name) if defined. x, y & z are expressed in meters / roll, pitch & yaw are expressed in radians

All lines of the next example realize the same operation:

```
robot.pose = [0.2, 0.1, 0.3, 0.0, 0.5, 0.0]
robot.move_pose([0.2, 0.1, 0.3, 0.0, 0.5, 0.0])
robot.move_pose(0.2, 0.1, 0.3, 0.0, 0.5, 0.0)
robot.move_pose(0.2, 0.1, 0.3, 0.0, 0.5, 0.0)
robot.move_pose(PoseObject(0.2, 0.1, 0.3, 0.0, 0.5, 0.0))
robot.move_pose([0.2, 0.1, 0.3, 0.0, 0.5, 0.0], "frame")
robot.move_pose(PoseObject(0.2, 0.1, 0.3, 0.0, 0.5, 0.0), "frame")
```

Parameters:
args (*Union[tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*], list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], PoseObject* (index.html#api.objects.PoseObject)*, [tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*], str* (https://docs.python.org/3/library/stdtypes.html#str)*], [list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], str* (https://docs.python.org/3/library/stdtypes.html#str)*], [PoseObject* (index.html#api.objects.PoseObject)*, str* (https://docs.python.org/3/library/stdtypes.html#str)*]]*) – either 7 args (1 for each coordinates and 1 for the name of the frame) or a list of 6 coordinates or a PoseObject and 1 for the frame name

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.move_linear_pose(**args*)

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

Parameters:
args (*Union[tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*], list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], PoseObject* (index.html#api.objects.PoseObject)*, [tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*], str* (https://docs.python.org/3/library/stdtypes.html#str)*], [list* (https://docs.python.org/3/library/stdtypes.html#list)*[float*

(https://docs.python.org/3/library/functions.html#float)*]*, *str* (https://docs.python.org/3/library/stdtypes.html#str)*]*, *[PoseObject* (index.html#api.objects.PoseObject)*, str* (https://docs.python.org/3/library/stdtypes.html#str)*]]*) – either 7 args (1 for each coordinates and 1 for the name of the frame) or a list of 6 coordinates or a PoseObject and 1 for the frame name

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.shift_pose(*axis*, *shift_value*)

Shift robot end effector pose along one axis

Parameters:
- **axis** (*RobotAxis*) – Axis along which the robot is shifted
- **shift_value** (*float* (https://docs.python.org/3/library/functions.html#float)) – In meter for X/Y/Z and radians for roll/pitch/yaw

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.shift_linear_pose(*axis*, *shift_value*)

Shift robot end effector pose along one axis, with a linear trajectory

Parameters:
- **axis** (*RobotAxis*) – Axis along which the robot is shifted
- **shift_value** (*float* (https://docs.python.org/3/library/functions.html#float)) – In meter for X/Y/Z and radians for roll/pitch/yaw

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.jog_joints(**args*)

Jog robot joints'. Jog corresponds to a shift without motion planning. Values are expressed in radians.

Parameters:
args (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*]]*) – either 6 args (1 for each joints) or a list of 6 joints offset

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.jog_pose(**args*)

Jog robot end effector pose Jog corresponds to a shift without motion planning Arguments are [dx, dy, dz, d_roll, d_pitch, d_yaw] dx, dy & dz are expressed in meters / d_roll, d_pitch & d_yaw are expressed in radians

Parameters:
args (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float*

(https://docs.python.org/3/library/functions.html#float)*]]*) – either 6 args (1 for each coordinates) or a list of 6 offset

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.move_to_home_pose()

Move to a position where the forearm lays on shoulder

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.go_to_sleep()

Go to home pose and activate learning mode

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.forward_kinematics(**args*)

Compute forward kinematics of a given joints configuration and give the associated spatial pose

Parameters:
args (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*]]*) – either 6 args (1 for each joints) or a list of 6 joints

Return type:
PoseObject (index.html#api.objects.PoseObject)

NiryoRobot.inverse_kinematics(**args*)

Compute inverse kinematics

Parameters:
args (*Union[tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*], list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], PoseObject* (index.html#api.objects.PoseObject)*]*) – either 6 args (1 for each coordinates) or a list of 6 coordinates or a PoseObject

Returns:
List of joints value

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]

Saved Poses

NiryoRobot.get_pose_saved(*pose_name*)

Get pose saved in from Ned's memory

Parameters:

pose_name (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – Pose name in robot's memory

Returns:

Pose associated to pose_name

Return type:

PoseObject (index.html#api.objects.PoseObject)

NiryoRobot.save_pose(*pose_name*, **args*)

Save pose in robot's memory

Parameters:

args (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*], PoseObject* (index.html#api.objects.PoseObject)*]*) – either 6 args (1 for each coordinates) or a list of 6 coordinates or a PoseObject

Return type:

None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.delete_pose(*pose_name*)

Delete pose from robot's memory

Return type:

None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.get_saved_pose_list()

Get list of poses' name saved in robot memory

Return type:

list (https://docs.python.org/3/library/stdtypes.html#list)[str (https://docs.python.org/3/library/stdtypes.html#str)]

Pick & Place

NiryoRobot.pick_from_pose(**args*)

Execute a picking from a pose.

A picking is described as :

* going over the object
* going down until height = z
* grasping with tool
* going back over the object

Parameters:

args (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*], PoseObject* (index.html#api.objects.PoseObject)*]*) – either 6 args (1 for each coordinates) or a list of 6 coordinates or a PoseObject

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.place_from_pose(**args*)

Execute a placing from a position.

A placing is described as :

* going over the place
* going down until height = z
* releasing the object with tool
* going back over the place

Parameters:
args (*Union[list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)], tuple (https://docs.python.org/3/library/stdtypes.html#tuple)[float (https://docs.python.org/3/library/functions.html#float)], PoseObject (index.html#api.objects.PoseObject)]*) – either 6 args (1 for each coordinates) or a list of 6 coordinates or a PoseObject

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.pick_and_place(*pick_pose*, *place_pos*, *dist_smoothing=0.0*)

Execute a pick then a place

Parameters:
- **pick_pose** (*Union[list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)], PoseObject (index.html#api.objects.PoseObject)]*) – Pick Pose : [x, y, z, roll, pitch, yaw] or PoseObject
- **place_pos** (*Union[list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)], PoseObject (index.html#api.objects.PoseObject)]*) – Place Pose : [x, y, z, roll, pitch, yaw] or PoseObject
- **dist_smoothing** (*float* (https://docs.python.org/3/library/functions.html#float)) – Distance from waypoints before smoothing trajectory

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Trajectories

NiryoRobot.get_trajectory_saved(*trajectory_name*)

Get trajectory saved in Ned's memory

Returns:
Trajectory

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[Joints]

NiryoRobot.get_saved_trajectory_list()

Get list of trajectories' name saved in robot memory

Return type:

list (https://docs.python.org/3/library/stdtypes.html#list)[str (https://docs.python.org/3/library/stdtypes.html#str)]

NiryoRobot.execute_registered_trajectory(*trajectory_name*)

Execute trajectory from Ned's memory

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.execute_trajectory_from_poses(*list_poses*, *dist_smoothing=0.0*)

Execute trajectory from list of poses

Parameters:

- **list_poses** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]]*) – List of [x,y,z,qx,qy,qz,qw] or list of [x,y,z,roll,pitch,yaw]
- **dist_smoothing** (*float* (https://docs.python.org/3/library/functions.html#float)) – Distance from waypoints before smoothing trajectory

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.execute_trajectory_from_poses_and_joints(*list_pose_joints*, *list_type=None*, *dist_smoothing=0.0*)

Execute trajectory from list of poses and joints

Example:

```
robot.execute_trajectory_from_poses_and_joints(
   list_pose_joints = [[0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.25, 0.0, 0.0, 0.0, 0.0, 0.0]],
   list_type = ['joint', 'pose'],
   dist_smoothing = 0.01
)
```

Parameters:

- **list_pose_joints** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]]*) – List of [x,y,z,qx,qy,qz,qw] or list of [x,y,z,roll,pitch,yaw] or a list of [j1,j2,j3,j4,j5,j6]
- **list_type** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[string]*) – List of string 'pose' or 'joint', or ['pose'] (if poses only) or ['joint'] (if joints only). If None, it is assumed there are only poses in the list.
- **dist_smoothing** (*float* (https://docs.python.org/3/library/functions.html#float)) – Distance from waypoints before smoothing trajectory

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.save_trajectory(*trajectory*, *trajectory_name*, *trajectory_description*)

Save trajectory in robot memory

Parameters:

- **trajectory** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]]*) – list of Joints [j1, j2, j3, j4, j5, j6] as waypoints to create the trajectory

- **trajectory_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – Name you want to give to the trajectory
- **trajectory_description** – Description you want to give to the trajectory

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.save_last_learned_trajectory(*name*, *description*)

Save last user executed trajectory

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.delete_trajectory(*trajectory_name*)

Delete trajectory from robot's memory

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.clean_trajectory_memory()

Delete trajectory from robot's memory

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Dynamic frames

NiryoRobot.get_saved_dynamic_frame_list()

Get list of saved dynamic frames

Example:

```
list_frame, list_desc = robot.get_saved_dynamic_frame_list()
print(list_frame)
print(list_desc)
```

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

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[str (https://docs.python.org/3/library/stdtypes.html#str)], list (https://docs.python.org/3/library/stdtypes.html#list)[str (https://docs.python.org/3/library/stdtypes.html#str)]

NiryoRobot.get_saved_dynamic_frame(*frame_name*)

Get name, description and pose of a dynamic frame

Example:

```
frame = robot.get_saved_dynamic_frame("default_frame")
```

Parameters:
frame_name (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of the frame

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

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[str (https://docs.python.org/3/library/stdtypes.html#str), str (https://docs.python.org/3/library/stdtypes.html#str), list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]]

NiryoRobot.save_dynamic_frame_from_poses(*frame_name*, *description*, *pose_origin*, *pose_x*, *pose_y*, *belong_to_workspace=False*)

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

Example:

```
pose_o = [0.1, 0.1, 0.1, 0, 0, 0]
pose_x = [0.2, 0.1, 0.1, 0, 0, 0]
pose_y = [0.1, 0.2, 0.1, 0, 0, 0]

robot.save_dynamic_frame_from_poses("name", "une description test", pose_o, pose_x, pose_y)
```

Parameters:

- **frame_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of the frame
- **description** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – description of the frame
- **pose_origin** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*] [x, y, z, roll, pitch, yaw]*) – pose of the origin of the frame
- **pose_x** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*] [x, y, z, roll, pitch, yaw]*) – pose of the point x of the frame
- **pose_y** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*] [x, y, z, roll, pitch, yaw]*) – pose of the point y of 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))

NiryoRobot.save_dynamic_frame_from_points(*frame_name*, *description*, *point_origin*, *point_x*, *point_y*, *belong_to_workspace=False*)

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

Example:

```
point_o = [-0.1, -0.1, 0.1]
point_x = [-0.2, -0.1, 0.1]
point_y = [-0.1, -0.2, 0.1]

robot.save_dynamic_frame_from_points("name", "une description test", point_o, point_x, point_y)
```

Parameters:

- **frame_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of the frame
- **description** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – description of the frame
- **point_origin** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*] [x, y, z]*) – origin point of the frame
- **point_x** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*] [x, y, z]*) – point x of the frame
- **point_y** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*] [x, y, z]*) – point y of 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))

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

Modify a dynamic frame

Example:

```
robot.edit_dynamic_frame("name", "new_name", "new description")
```

Parameters:

- **frame_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of the frame
- **new_frame_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – new name of the frame
- **new_description** (*str* (https://docs.python.org/3/library/stdtypes.html#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))

NiryoRobot.delete_dynamic_frame(*frame_name*, *belong_to_workspace=False*)

Delete a dynamic frame

Example:

```
robot.delete_saved_dynamic_frame("name")
```

Parameters:

- **frame_name** (*str* (https://docs.python.org/3/library/stdtypes.html#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))

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

Move robot end of a offset in a frame

Example:

```
robot.move_relative([0.05, 0.05, 0.05, 0.3, 0, 0], frame="default_frame")
```

Parameters:

- **offset** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) – list which contains offset of x, y, z, roll, pitch, yaw
- **frame** (*str* (https://docs.python.org/3/library/stdtypes.html#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))

NiryoRobot.move_linear_relative(*offset*, *frame='world'*)

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

Example:

```
robot.move_linear_relative([0.05, 0.05, 0.05, 0.3, 0, 0], frame="default_frame")
```

Parameters:

- **offset** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) – list which contains offset of x, y, z, roll, pitch, yaw
- **frame** (*str* (https://docs.python.org/3/library/stdtypes.html#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

NiryoRobot.get_current_tool_id()

Get equipped tool Id

Return type:
ToolID

NiryoRobot.update_tool()

Update equipped tool

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.grasp_with_tool()

Grasp with tool | This action correspond to | - Close gripper for Grippers | - Pull Air for Vacuum pump | - Activate for Electromagnet

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.release_with_tool()

Release with tool | This action correspond to | - Open gripper for Grippers | - Push Air for Vacuum pump | - Deactivate for Electromagnet

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.open_gripper(*speed=500, max_torque_percentage=100, hold_torque_percentage=30*)

Open gripper

Parameters:

- **speed** (*int* (https://docs.python.org/3/library/functions.html#int)) – Between 100 & 1000 (only for Niryo One and Ned1)
- **max_torque_percentage** (*int* (https://docs.python.org/3/library/functions.html#int)) – Closing torque percentage (only for Ned2)
- **hold_torque_percentage** (*int* (https://docs.python.org/3/library/functions.html#int)) – Hold torque percentage after closing (only for Ned2)

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.close_gripper(*speed=500, max_torque_percentage=100, hold_torque_percentage=20*)

Close gripper

Parameters:

- **speed** (*int* (https://docs.python.org/3/library/functions.html#int)) – Between 100 & 1000 (only for Niryo One and Ned1)
- **max_torque_percentage** (*int* (https://docs.python.org/3/library/functions.html#int)) – Opening torque percentage (only for Ned2)
- **hold_torque_percentage** (*int* (https://docs.python.org/3/library/functions.html#int)) – Hold torque percentage after opening (only for Ned2)

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.pull_air_vacuum_pump()

Pull air of vacuum pump

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.push_air_vacuum_pump()

Push air of vacuum pump

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.setup_electromagnet(*pin_id*)

Setup electromagnet on pin

Parameters:
pin_id (*PinID or str* (https://docs.python.org/3/library/stdtypes.html#str)) –

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.activate_electromagnet(*pin_id*)

Activate electromagnet associated to electromagnet_id on pin_id

Parameters:
pin_id (*PinID or str* (https://docs.python.org/3/library/stdtypes.html#str)) –

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.deactivate_electromagnet(*pin_id*)

Deactivate electromagnet associated to electromagnet_id on pin_id

Parameters:
pin_id (*PinID or str* (https://docs.python.org/3/library/stdtypes.html#str)) –

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.enable_tcp(*enable=True*)

Enables or disables the TCP function (Tool Center Point). If activation is requested, the last recorded TCP value will be applied. The default value depends on the gripper equipped. If deactivation is requested, the TCP will be coincident with the tool_link.

Parameters:
enable (*Bool*) – True to enable, False otherwise.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.set_tcp(**args*)

Activates the TCP function (Tool Center Point) and defines the transformation between the tool_link frame and the TCP frame.

Parameters:
args (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*], PoseObject* (index.html#api.objects.PoseObject)*]*) – either 6 args (1 for each coordinates) or a list of 6 coordinates or a PoseObject

Return type:

None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.reset_tcp()

Reset the TCP (Tool Center Point) transformation. The TCP will be reset according to the tool equipped.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.tool_reboot()

Reboot the motor of the tool equparam_list = [workspace_name]

Example:

```
for pose in (pose_origin, pose_2, pose_3, pose_4):
    pose_list = self.__args_pose_to_list(pose)
    param_list.append(pose_list)ipped. Useful when an Overload error occurs. (cf HardwareStatus)
```

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Hardware

NiryoRobot.set_pin_mode(*pin_id, pin_mode*)

Set pin number pin_id to mode pin_mode

Parameters:

- **pin_id** (*PinID or str* (https://docs.python.org/3/library/stdtypes.html#str)) –
- **pin_mode** (*PinMode*) –

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.digital_write(*pin_id, digital_state*)

Set pin_id state to digital_state

Parameters:

- **pin_id** (*PinID or str* (https://docs.python.org/3/library/stdtypes.html#str)) –
- **digital_state** (*PinState*) –

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.digital_read(*pin_id*)

Read pin number pin_id and return its state

Parameters:
pin_id (*PinID or str* (https://docs.python.org/3/library/stdtypes.html#str)) –

Return type:
PinState

NiryoRobot.get_hardware_status()

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

Returns:
Infos contains in a HardwareStatusObject

Return type:
HardwareStatusObject (index.html#api.objects.HardwareStatusObject)

NiryoRobot.get_digital_io_state()

Get Digital IO state : Names, modes, states.

Example:

```
digital_io_state = robot.digital_io_state
digital_io_state = robot.get_digital_io_state()
```

Returns:
List of DigitalPinObject instance

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[DigitalPinObject (index.html#api.objects.DigitalPinObject)]

NiryoRobot.get_analog_io_state()

Get Analog IO state : Names, modes, states

Example:

```
analog_io_state = robot.analog_io_state
analog_io_state = robot.get_analog_io_state()
```

Returns:
List of AnalogPinObject instance

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[AnalogPinObject (index.html#api.objects.AnalogPinObject)]

NiryoRobot.analog_write(*pin_id*, *value*)

Set and analog pin_id state to a value

Parameters:
- **pin_id** (*PinID or str* (https://docs.python.org/3/library/stdtypes.html#str)) –
- **value** (*float* (https://docs.python.org/3/library/functions.html#float)) – voltage between 0 and 5V

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.analog_read(*pin_id*)

Read the analog pin value

Parameters:
pin_id (*PinID or str* (https://docs.python.org/3/library/stdtypes.html#str)) –

Return type:
float (https://docs.python.org/3/library/functions.html#float)

NiryoRobot.get_custom_button_state()

Get the Ned2's custom button state

Returns:
True if pressed, False else

Return type:
bool (https://docs.python.org/3/library/functions.html#bool)

Conveyor

NiryoRobot.set_conveyor()

Activate a new conveyor and return its ID

Returns:
New conveyor ID

Return type:
ConveyorID

NiryoRobot.unset_conveyor(*conveyor_id*)

Remove specific conveyor.

Parameters:
conveyor_id (*ConveyorID*) – Basically, ConveyorID.ONE or ConveyorID.TWO

NiryoRobot.run_conveyor(*conveyor_id*, *speed=50*, *direction=<ConveyorDirection.FORWARD: 1>*)

Run conveyor at id 'conveyor_id'

Parameters:
- **conveyor_id** (*ConveyorID*) –
- **speed** (*int* (https://docs.python.org/3/library/functions.html#int)) –
- **direction** (*ConveyorDirection*) –

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.stop_conveyor(*conveyor_id*)

Stop conveyor at id 'conveyor_id'

Parameters:
conveyor_id (*ConveyorID*) –

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.control_conveyor(*conveyor_id*, *control_on*, *speed*, *direction*)

Control conveyor at id 'conveyor_id'

Parameters:
- **conveyor_id** (*ConveyorID*) –
- **control_on** (*bool* (https://docs.python.org/3/library/functions.html#bool)) –
- **speed** (*int* (https://docs.python.org/3/library/functions.html#int)) – New speed which is a percentage of maximum speed
- **direction** (*ConveyorDirection*) – Conveyor direction

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.get_connected_conveyors_id()

Returns:
List of the connected conveyors' ID

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[ConveyorID]

Vision

NiryoRobot.get_img_compressed()

Get image from video stream in a compressed format. Use uncompress_image from the vision package to uncompress it

Returns:
string containing a JPEG compressed image

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

NiryoRobot.set_brightness(*brightness_factor*)

Modify video stream brightness

Parameters:
brightness_factor (*float* (https://docs.python.org/3/library/functions.html#float)) – How much to adjust the brightness. 0.5 will give a darkened image, 1 will give the original image while 2 will enhance the brightness by a factor of 2.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.set_contrast(*contrast_factor*)

Modify video stream contrast

Parameters:
contrast_factor (*float* (https://docs.python.org/3/library/functions.html#float)) – While a factor of 1 gives original image. Making the factor towards 0 makes the image greyer, while factor>1 increases the contrast of the image.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.set_saturation(*saturation_factor*)

Modify video stream saturation

Parameters:
saturation_factor (*float* (https://docs.python.org/3/library/functions.html#float)) – How much to adjust the saturation. 0 will give a black and white image, 1 will give the original image while 2 will enhance the saturation by a factor of 2.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.get_image_parameters()

Get last stream image parameters: Brightness factor, Contrast factor, Saturation factor.

Brightness factor: How much to adjust the brightness. 0.5 will give a darkened image, 1 will give the original image while 2 will enhance the brightness by a factor of 2.

Contrast factor: A factor of 1 gives original image. Making the factor towards 0 makes the image greyer, while factor>1 increases the contrast of the image.

Saturation factor: 0 will give a black and white image, 1 will give the original image while 2 will enhance the saturation by a factor of 2.

Returns:
Brightness factor, Contrast factor, Saturation factor

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

NiryoRobot.get_target_pose_from_rel(*workspace_name*, *height_offset*, *x_rel*, *y_rel*, *yaw_rel*)

Given a pose (x_rel, y_rel, yaw_rel) relative to a workspace, this function returns the robot pose in which the current tool will be able to pick an object at this pose.

The height_offset argument (in m) defines how high the tool will hover over the workspace. If height_offset = 0, the tool will nearly touch the workspace.

Parameters:

- **workspace_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of the workspace
- **height_offset** (*float* (https://docs.python.org/3/library/functions.html#float)) – offset between the workspace and the target height
- **x_rel** (*float* (https://docs.python.org/3/library/functions.html#float)) – x relative pose (between 0 and 1)
- **y_rel** (*float* (https://docs.python.org/3/library/functions.html#float)) – y relative pose (between 0 and 1)
- **yaw_rel** (*float* (https://docs.python.org/3/library/functions.html#float)) – Angle in radians

Returns:
target_pose

Return type:
PoseObject (index.html#api.objects.PoseObject)

NiryoRobot.get_target_pose_from_cam(*workspace_name*, *height_offset=0.0*, *shape=<ObjectShape.ANY: 'ANY'>*, *color=<ObjectColor.ANY: 'ANY'>*)

First detects the specified object using the camera and then returns the robot pose in which the object can be picked with the current tool

Parameters:

- **workspace_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of the workspace
- **height_offset** (*float* (https://docs.python.org/3/library/functions.html#float)) – offset between the workspace and the target height
- **shape** (*ObjectShape*) – shape of the target
- **color** (*ObjectColor*) – color of the target

Returns:
object_found, object_pose, object_shape, object_color

Return type:
(bool (https://docs.python.org/3/library/functions.html#bool), PoseObject (index.html#api.objects.PoseObject), ObjectShape, ObjectColor)

NiryoRobot.vision_pick(*workspace_name*, *height_offset=0.0*, *shape=<ObjectShape.ANY: 'ANY'>*, *color=<ObjectColor.ANY: 'ANY'>*)

Picks the specified object from the workspace. This function has multiple phases:

1. detect object using the camera
2. prepare the current tool for picking
3. approach the object
4. move down to the correct picking pose
5. actuate the current tool
6. lift the object

Example:

```
robot = NiryoRobot(ip_address="x.x.x.x")
robot.calibrate_auto()
robot.move_pose(<observation_pose>)
obj_found, shape_ret, color_ret = robot.vision_pick(<workspace_name>,
                                                    height_offset=0.0,
                                                    shape=ObjectShape.ANY,
                                                    color=ObjectColor.ANY)
```

Parameters:

- **workspace_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of the workspace
- **height_offset** (*float* (https://docs.python.org/3/library/functions.html#float)) – offset between the workspace and the target height
- **shape** (*ObjectShape*) – shape of the target
- **color** (*ObjectColor*) – color of the target

Returns:
object_found, object_shape, object_color

Return type:
(bool (https://docs.python.org/3/library/functions.html#bool), ObjectShape, ObjectColor)

NiryoRobot.move_to_object(*workspace_name*, *height_offset*, *shape*, *color*)

Same as *get_target_pose_from_cam* but directly moves to this position

Parameters:

- **workspace_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of the workspace
- **height_offset** (*float* (https://docs.python.org/3/library/functions.html#float)) – offset between the workspace and the target height
- **shape** (*ObjectShape*) – shape of the target
- **color** (*ObjectColor*) – color of the target

Returns:
object_found, object_shape, object_color

Return type:

(bool (https://docs.python.org/3/library/functions.html#bool), ObjectShape, ObjectColor)

NiryoRobot.detect_object(*workspace_name, shape=<ObjectShape.ANY: 'ANY'>, color=<ObjectColor.ANY: 'ANY'>*)

Detect object in workspace and return its pose and characteristics

Parameters:

- **workspace_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – name of the workspace
- **shape** (*ObjectShape*) – shape of the target
- **color** (*ObjectColor*) – color of the target

Returns:

object_found, object_pose, object_shape, object_color

Return type:

(bool (https://docs.python.org/3/library/functions.html#bool), PoseObject (index.html#api.objects.PoseObject), str (https://docs.python.org/3/library/stdtypes.html#str), str (https://docs.python.org/3/library/stdtypes.html#str))

NiryoRobot.get_camera_intrinsics()

Get calibration object: camera intrinsics, distortions coefficients

Returns:

camera intrinsics, distortions coefficients

Return type:

(list (https://docs.python.org/3/library/stdtypes.html#list)[list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]], list (https://docs.python.org/3/library/stdtypes.html#list)[list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]])

NiryoRobot.save_workspace_from_robot_poses(*workspace_name, pose_origin, pose_2, pose_3, pose_4*)

Save workspace by giving the poses of the robot to point its 4 corners with the calibration Tip. Corners should be in the good order. Markers' pose will be deduced from these poses

Poses should be either a list [x, y, z, roll, pitch, yaw] or a PoseObject

Parameters:

- **workspace_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – workspace name, maximum lenght 30 char.
- **pose_origin** (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], PoseObject* (index.html#api.objects.PoseObject)*]*) –
- **pose_2** (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], PoseObject* (index.html#api.objects.PoseObject)*]*) –
- **pose_3** (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], PoseObject* (index.html#api.objects.PoseObject)*]*) –
- **pose_4** (*Union[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*], PoseObject* (index.html#api.objects.PoseObject)*]*) –

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.save_workspace_from_points(*workspace_name*, *point_origin*, *point_2*, *point_3*, *point_4*)

Save workspace by giving the points of worskpace's 4 corners. Points are written as [x, y, z] Corners should be in the good order.

Parameters:

- **workspace_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – workspace name, maximum lenght 30 char.
- **point_origin** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) –
- **point_2** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) –
- **point_3** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) –
- **point_4** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) –

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.delete_workspace(*workspace_name*)

Delete workspace from robot's memory

Parameters:
workspace_name (*str* (https://docs.python.org/3/library/stdtypes.html#str)) –

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.get_workspace_ratio(*workspace_name*)

Get workspace ratio from robot's memory

Parameters:
workspace_name (*str* (https://docs.python.org/3/library/stdtypes.html#str)) –

Return type:
float (https://docs.python.org/3/library/functions.html#float)

NiryoRobot.get_workspace_list()

Get list of workspaces' name store in robot's memory

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[str (https://docs.python.org/3/library/stdtypes.html#str)]

Led Ring

NiryoRobot.set_led_color(*led_id*, *color*)

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

Example:

```
robot.set_led_color(5, [15, 50, 255])
```

Parameters:

- **led_id** (*int* (https://docs.python.org/3/library/functions.html#int)) – Id of the led: between 0 and 29
- **color** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) – Led color in a list of size 3[R, G, B]. 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))

NiryoRobot.led_ring_solid(*color*)

Set the whole Led Ring to a fixed color.

Example:

```
robot.led_ring_solid([15, 50, 255])
```

Parameters:
color (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) – Led color in a list of size 3[R, G, B]. 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))

NiryoRobot.led_ring_turn_off()

Turn off all LEDs

Example:

```
robot.led_ring_turn_off()
```

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))

NiryoRobot.led_ring_flashing(*color, period=0, iterations=0, wait=False*)

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

Examples:

```
robot.led_ring_flashing([15, 50, 255])
robot.led_ring_flashing([15, 50, 255], 1, 100, True)
robot.led_ring_flashing([15, 50, 255], iterations=20, wait=True)

frequency = 20  # Hz
total_duration = 10 # seconds
robot.flashing([15, 50, 255], 1./frequency, total_duration * frequency , True)
```

Parameters:

- **color** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) – Led color in a list of size 3[R, G, B]. RGB channels from 0 to 255.
- **period** (*float* (https://docs.python.org/3/library/functions.html#float)) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **iterations** (*int* (https://docs.python.org/3/library/functions.html#int)) – Number of consecutive flashes. If 0, the Led Ring flashes endlessly.
- **wait** (*bool* (https://docs.python.org/3/library/functions.html#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))

NiryoRobot.led_ring_alternate(*color_list, period=0, iterations=0, wait=False*)

Several colors are alternated one after the other.

Examples:

```
color_list = [
    [15, 50, 255],
    [255, 0, 0],
    [0, 255, 0],
]

robot.led_ring_alternate(color_list)
robot.led_ring_alternate(color_list, 1, 100, True)
robot.led_ring_alternate(color_list, iterations=20, wait=True)
```

Parameters:

- **color_list** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]]*) – Led color list of lists of size 3[R, G, B]. RGB channels from 0 to 255.
- **period** (*float* (https://docs.python.org/3/library/functions.html#float)) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **iterations** (*int* (https://docs.python.org/3/library/functions.html#int)) – Number of consecutive alternations. If 0, the Led Ring alternates endlessly.
- **wait** (*bool* (https://docs.python.org/3/library/functions.html#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))

NiryoRobot.led_ring_chase(*color, period=0, iterations=0, wait=False*)

Movie theater light style chaser animation.

Examples:

```
robot.led_ring_chase([15, 50, 255])
robot.led_ring_chase([15, 50, 255], 1, 100, True)
robot.led_ring_chase([15, 50, 255], iterations=20, wait=True)
```

Parameters:

- **color** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) – Led color in a list of size 3[R, G, B]. RGB channels from 0 to 255.
- **period** (*float* (https://docs.python.org/3/library/functions.html#float)) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **iterations** (*int* (https://docs.python.org/3/library/functions.html#int)) – Number of consecutive chase. If 0, the animation continues endlessly. One chase just lights one Led every 3 LEDs.
- **wait** (*bool* (https://docs.python.org/3/library/functions.html#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))

NiryoRobot.led_ring_wipe(*color, period=0, wait=False*)

Wipe a color across the Led Ring, light a Led at a time.

Examples:

```
robot.led_ring_wipe([15, 50, 255])
robot.led_ring_wipe([15, 50, 255], 1, True)
robot.led_ring_wipe([15, 50, 255], wait=True)
```

Parameters:

- **color** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) – Led color in a list of size 3[R, G, B]. RGB channels from 0 to 255.
- **period** (*float* (https://docs.python.org/3/library/functions.html#float)) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **wait** (*bool* (https://docs.python.org/3/library/functions.html#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))

NiryoRobot.led_ring_rainbow(*period=0, iterations=0, wait=False*)

Draw rainbow that fades across all LEDs at once.

Examples:

```
robot.led_ring_rainbow()
robot.led_ring_rainbow(5, 2, True)
robot.led_ring_rainbow(wait=True)
```

Parameters:

- **period** (*float* (https://docs.python.org/3/library/functions.html#float)) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **iterations** (*int* (https://docs.python.org/3/library/functions.html#int)) – Number of consecutive rainbows. If 0, the animation continues endlessly.
- **wait** (*bool* (https://docs.python.org/3/library/functions.html#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))

NiryoRobot.led_ring_rainbow_cycle(*period=0, iterations=0, wait=False*)

Draw rainbow that uniformly distributes itself across all LEDs.

Examples:

```
robot.led_ring_rainbow_cycle()
robot.led_ring_rainbow_cycle(5, 2, True)
robot.led_ring_rainbow_cycle(wait=True)
```

Parameters:

- **period** (*float* (https://docs.python.org/3/library/functions.html#float)) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **iterations** (*int* (https://docs.python.org/3/library/functions.html#int)) – Number of consecutive rainbow cycles. If 0, the animation continues endlessly.
- **wait** (*bool* (https://docs.python.org/3/library/functions.html#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))

NiryoRobot.led_ring_rainbow_chase(*period=0, iterations=0, wait=False*)

Rainbow chase animation, like the led_ring_chase method.

Examples:

```
robot.led_ring_rainbow_chase()
robot.led_ring_rainbow_chase(5, 2, True)
robot.led_ring_rainbow_chase(wait=True)
```

Parameters:

- **period** (*float* (https://docs.python.org/3/library/functions.html#float)) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **iterations** (*int* (https://docs.python.org/3/library/functions.html#int)) – Number of consecutive rainbow cycles. If 0, the animation continues endlessly.
- **wait** (*bool* (https://docs.python.org/3/library/functions.html#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))

NiryoRobot.led_ring_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:

```
robot.led_ring_go_up([15, 50, 255])
robot.led_ring_go_up([15, 50, 255], 1, 100, True)
robot.led_ring_go_up([15, 50, 255], iterations=20, wait=True)
```

Parameters:

- **color** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) – Led color in a list of size 3[R, G, B]. RGB channels from 0 to 255.
- **period** (*float* (https://docs.python.org/3/library/functions.html#float)) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **iterations** (*int* (https://docs.python.org/3/library/functions.html#int)) – Number of consecutive turns around the Led Ring. If 0, the animation continues endlessly.
- **wait** (*bool* (https://docs.python.org/3/library/functions.html#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))

NiryoRobot.led_ring_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:

```
robot.led_ring_go_up_down([15, 50, 255])
robot.led_ring_go_up_down([15, 50, 255], 1, 100, True)
robot.led_ring_go_up_down([15, 50, 255], iterations=20, wait=True)
```

Parameters:

- **color** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) – Led color in a list of size 3[R, G, B]. RGB channels from 0 to 255.
- **period** (*float* (https://docs.python.org/3/library/functions.html#float)) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **iterations** (*int* (https://docs.python.org/3/library/functions.html#int)) – Number of consecutive turns around the Led Ring. If 0, the animation continues endlessly.
- **wait** (*bool* (https://docs.python.org/3/library/functions.html#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))

NiryoRobot.led_ring_breath(*color*, *period=0*, *iterations=0*, *wait=False*)

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

Examples:

```
robot.led_ring_breath([15, 50, 255])
robot.led_ring_breath([15, 50, 255], 1, 100, True)
robot.led_ring_breath([15, 50, 255], iterations=20, wait=True)
```

Parameters:

- **color** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) – Led color in a list of size 3[R, G, B]. RGB channels from 0 to 255.
- **period** (*float* (https://docs.python.org/3/library/functions.html#float)) – Execution time for a pattern in seconds. If 0, the default time will be used.
- **iterations** (*int* (https://docs.python.org/3/library/functions.html#int)) – Number of consecutive turns around the Led Ring. If 0, the animation continues endlessly.
- **wait** (*bool* (https://docs.python.org/3/library/functions.html#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))

NiryoRobot.led_ring_snake(*color*, *period=0*, *iterations=0*, *wait=False*)

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

Examples:

```
robot.led_ring_snake([15, 50, 255])
robot.led_ring_snake([15, 50, 255], 1, 100, True)
```

Parameters:

- **color** (*list* (https://docs.python.org/3/library/stdtypes.html#list)[*float* (https://docs.python.org/3/library/functions.html#float)]) – Led color in a list of size 3[R, G, B]. RGB channels from 0 to 255.
- **period** (*float* (https://docs.python.org/3/library/functions.html#float)) – Execution time for a pattern in seconds. If 0, the default duration will be used.
- **iterations** (*int* (https://docs.python.org/3/library/functions.html#int)) – Number of consecutive turns around the Led Ring. If 0, the animation continues endlessly.
- **wait** (*bool* (https://docs.python.org/3/library/functions.html#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))

NiryoRobot.led_ring_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)]
robot.led_ring_custom(led_list)
```

Parameters:
led_colors (*list* (https://docs.python.org/3/library/stdtypes.html#list)[*list* (https://docs.python.org/3/library/stdtypes.html#list)[*float* (https://docs.python.org/3/library/functions.html#float)]]) – List of size 30 of led color in a list of size 3[R, G, B]. 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))

Sound

NiryoRobot.get_sounds()

Get sound name list

Returns:
list of the sounds of the robot

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[string]

NiryoRobot.play_sound(*sound_name*, *wait_end=True*, *start_time_sec=0*, *end_time_sec=0*)

Play a sound from the robot

Parameters:
- **sound_name** (*string*) – Name of the sound to play
- **wait_end** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – wait for the end of the

sound before exiting the function

- **start_time_sec** (*float* (https://docs.python.org/3/library/functions.html#float)) – start the sound from this value in seconds
- **end_time_sec** (*float* (https://docs.python.org/3/library/functions.html#float)) – end the sound at this value in seconds

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.set_volume(*sound_volume*)

Set the volume percentage of the robot.

Parameters:
sound_volume (*int* (https://docs.python.org/3/library/functions.html#int)) – volume percentage of the sound (0: no sound, 100: max sound)

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.stop_sound()

Stop a sound being played.

Return type:
None (https://docs.python.org/3/library/constants.html#None)

NiryoRobot.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 (*string*) – name of sound

Returns:
sound duration in seconds

Return type:
float (https://docs.python.org/3/library/functions.html#float)

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

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

Example

```
robot.say("Hello", 0)
robot.say("Bonjour", 1)
robot.say("Hola", 2)
```

Parameters:

- **text** (*string*) – Text that needs to be spoken < 100 char
- **language** (*int* (https://docs.python.org/3/library/functions.html#int)) – language of the text

Return type:
None (https://docs.python.org/3/library/constants.html#None)

Enums

Enums are used to pass specific parameters to functions.

For instance, **shift_pose()** will need a parameter from **RobotAxis** enum

```
robot.shift_pose(RobotAxis.Y, 0.15)
```

List of enums:

- **CalibrateMode**
- **RobotAxis**
- **ToolID**
- **PinMode**
- **PinState**
- **PinID**
- **ConveyorID**
- **ConveyorDirection**
- **ObjectColor**
- **ObjectShape**

class **CalibrateMode**(*value*)

Enumeration of Calibration Modes

AUTO= *0*

MANUAL= *1*

class **RobotAxis**(*value*)

Enumeration of Robot Axis : it used for Shift command

X= *0*

Y= *1*

Z= *2*

ROLL= *3*

PITCH= *4*

YAW= *5*

class **ToolID**(*value*)

Enumeration of Tools IDs

NONE= *0*

GRIPPER_1= *11*

GRIPPER_2= *12*

GRIPPER_3= *13*

ELECTROMAGNET_1= *30*

VACUUM_PUMP_1= *31*

class **PinMode**(*value*)

Enumeration of Pin Modes

OUTPUT= *0*

INPUT= *1*

class **PinState**(*value*)

Pin State is either LOW or HIGH

LOW= *False*

HIGH= *True*

class **PinID**(*value*)

Enumeration of Robot Pins

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**(*value*)

Enumeration of Conveyor IDs used for Conveyor control

NONE= *0*

ID_1= *-1*

ID_2= *-2*

class **ConveyorDirection**(*value*)

Enumeration of Conveyor Directions used for Conveyor control

FORWARD= *1*

BACKWARD= *-1*

class **ObjectColor**(*value*)

Enumeration of Colors available for image processing

RED= *'RED'*

BLUE= *'BLUE'*

GREEN= *'GREEN'*

ANY= *'ANY'*

class **ObjectShape**(*value*)

Enumeration of Shapes available for image processing

SQUARE= *'SQUARE'*

CIRCLE= *'CIRCLE'*

ANY= *'ANY'*

Python object classes

Special objects

class **PoseObject**(*x*, *y*, *z*, *roll*, *pitch*, *yaw*)

Pose object which stores x, y, z, roll, pitch & yaw parameters

copy_with_offsets(*x_offset=0.0*, *y_offset=0.0*, *z_offset=0.0*, *roll_offset=0.0*, *pitch_offset=0.0*, *yaw_offset=0.0*)

Create a new pose from copying from copying actual pose with offsets

Return type:
PoseObject (index.html#api.objects.PoseObject)

to_list()

Return a list [x, y, z, roll, pitch, yaw] corresponding to the pose's parameters

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[float (https://docs.python.org/3/library/functions.html#float)]

class **HardwareStatusObject**(*rpi_temperature*, *hardware_version*, *connection_up*, *error_message*, *calibration_needed*, *calibration_in_progress*, *motor_names*, *motor_types*, *motors_temperature*, *motors_voltage*, *hardware_errors*)

Object used to store every hardware information

class **DigitalPinObject**(*pin_id*, *name*, *mode*, *state*)

Object used to store information on digital pins

class **AnalogPinObject**(*pin_id*, *name*, *mode*, *value*)

Object used to store information on digital pins

Start with Image Processing

Discover how to create your own image processing pipelines!

Overview & examples

This file illustrates few image processing pipeline using vision module from niryo_edu package. This module is based in OpenCV (https://opencv.org/) and its functions are detailed in Functions documentation (index.html#document-source/vision/image_processing_api).

The package niryo_edu comes up with the **vision** module which contains image processing functions

including thresholding, blob detection, …
To use it, add to your imports `from pyniryo.vision import *` .

> **Note**
>
> It is also possible to merge both import lines by using `from pyniryo import *` .

Play with Robot video stream

We are firstly going to take a look at robot's functions which can be find at API - Vision (index.html#vision)

Get & display image from stream

Ned can share its video stream through TCP. As sending raw images will lead to heavy packets which can saturate the network, it sends compressed images. You access it through the robot's function: **get_img_compressed()** (index.html#api.tcp_client.NiryoRobot.get_img_compressed). Once your image is received, you firstly need to uncompress via **uncompress_image()** (index.html#vision.image_functions.uncompress_image) and you can then display it with **show_img_and_wait_close()** (index.html#vision.image_functions.show_img_and_wait_close).

```
from pyniryo import *

# Connecting to robot
robot = NiryoRobot("10.10.10.10")

# Getting image
img_compressed = robot.get_img_compressed()
# Uncompressing image
img = uncompress_image(img_compressed)

# Displaying
show_img_and_wait_close("img_stream", img)
```

> **Note**
>
> **show_img_and_wait_close()** (index.html#vision.image_functions.show_img_and_wait_close) will wait for the user to press either *Q* or *Esc* key, before closing the window.

Undistort and display video stream

In this section, we are going to display the raw video stream & the undistorted video stream.

As Ned's camera is passing raw images to the robot, these images are distorted due to the camera lens. In order to undistort them, we need to use Ned's camera intrinsics.

To undistort the raw image, we use **undistort_image()** (index.html#vision.image_functions.undistort_image) which needs to be called with the parameters given by Ned through **get_camera_intrinsics()** (index.html#api.tcp_client.NiryoRobot.get_camera_intrinsics).

Once, we have both raw & undistorted images, we can concatenate them in order to display them in once with **concat_imgs()** (index.html#vision.image_functions.concat_imgs). Finally, we display the image **show_img()** (index.html#vision.image_functions.show_img).

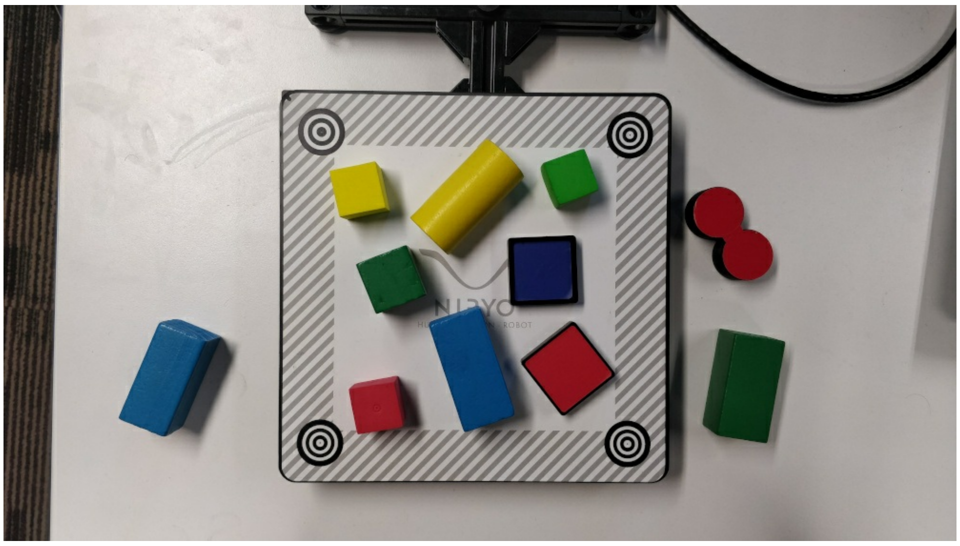
```
from pyniryo import *

observation_pose = PoseObject(
    x=0.18, y=0.0, z=0.35,
    roll=0.0, pitch=1.57, yaw=-0.2,
)

# Connecting to robot
robot = NiryoRobot("10.10.10.10")
robot.calibrate_auto()

# Getting calibration param
mtx, dist = robot.get_camera_intrinsics()
# Moving to observation pose
robot.move_pose(observation_pose)

while "User do not press Escape neither Q":
    # Getting image
    img_compressed = robot.get_img_compressed()
    # Uncompressing image
    img_raw = uncompress_image(img_compressed)
    # Undistorting
    img_undistort = undistort_image(img_raw, mtx, dist)

    # - Display
    # Concatenating raw image and undistorted image
    concat_ims = concat_imgs((img_raw, img_undistort))

    # Showing images
    key = show_img("Images raw & undistorted", concat_ims, wait_ms=30)
    if key in [27, ord("q")]:  # Will break loop if the user press Escape or Q
        break
```

Note

To see more about camera distortion/undistortion, go on OpenCV Documentation about Camera Calibration (https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html).

Pure image processing functions

In order to illustrate functions, we are going to use the following image.



Attention

In this section it is supposed that:

- You have imported pyniryo.vision

- The variable img is containing the image on which image processing is applied

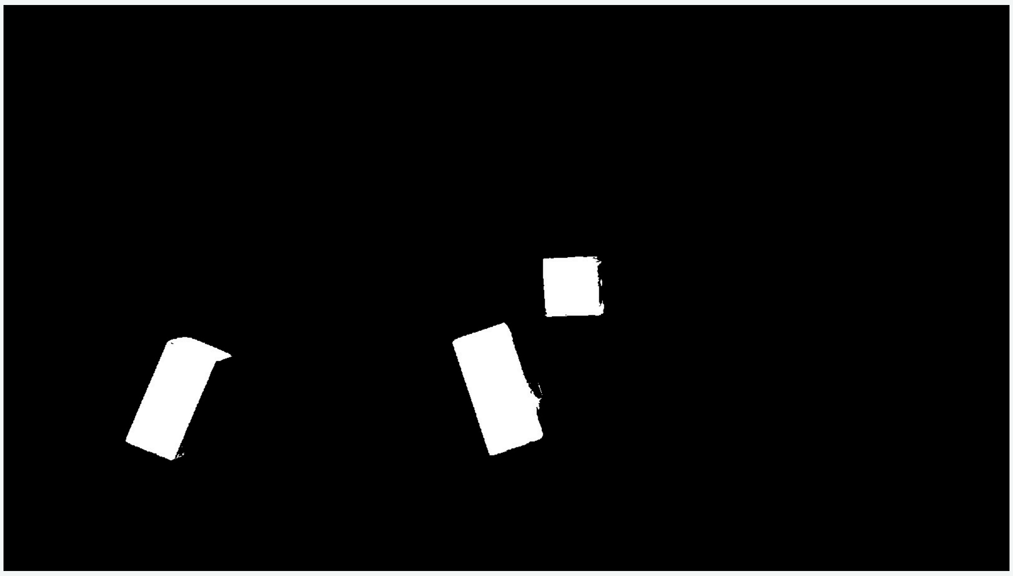
Color thresholding

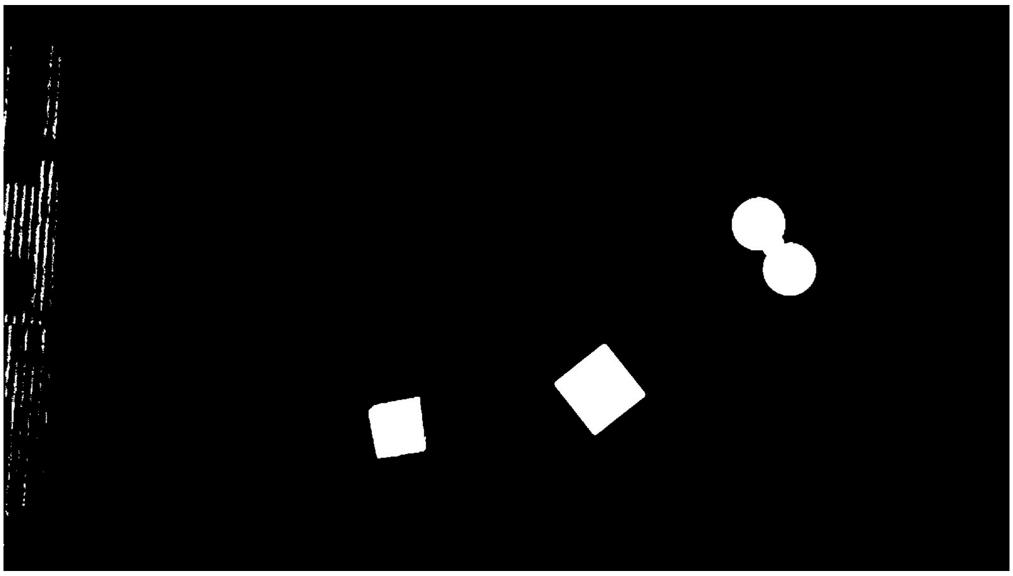
Color thresholding is very useful in order to detect object with an uniform color. The implemented function to realize this operation is **threshold_hsv()** (index.html#vision.image_functions.threshold_hsv).

The following code is using parameters from **ColorHSV** (index.html#vision.enums.ColorHSV) enum in order to threshold Red features & *hand made* parameters to extract Blue:

```
img_threshold_red = threshold_hsv(img_test, *ColorHSV.RED.value)

blue_min_hsv = [90, 85, 70]
blue_max_hsv = [125, 255, 255]

img_threshold_blue = threshold_hsv(img_test, list_min_hsv=blue_min_hsv,
                                   list_max_hsv=blue_max_hsv, reverse_hue=False)

show_img("img_threshold_red", img_threshold_red)

show_img_and_wait_close("img_threshold_blue", img_threshold_blue)
```

Images result

Thresh color	Image result
Blue	

Thresh color	Image result
Red	

Morphological transformations

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. It needs two inputs, one is our original image, second one is called structuring element or kernel which decides the nature of operation. Two basic morphological operators are Erosion (https://en.wikipedia.org/wiki/Mathematical_morphology#Erosion) and Dilation (https://en.wikipedia.org/wiki/Mathematical_morphology#Dilation).

Then its variant forms like Opening (https://en.wikipedia.org/wiki/Mathematical_morphology#Opening), Closing (https://en.wikipedia.org/wiki/Mathematical_morphology#Closing) also comes into play. Learn more on Wikipedia page (https://en.wikipedia.org/wiki/Mathematical_morphology).

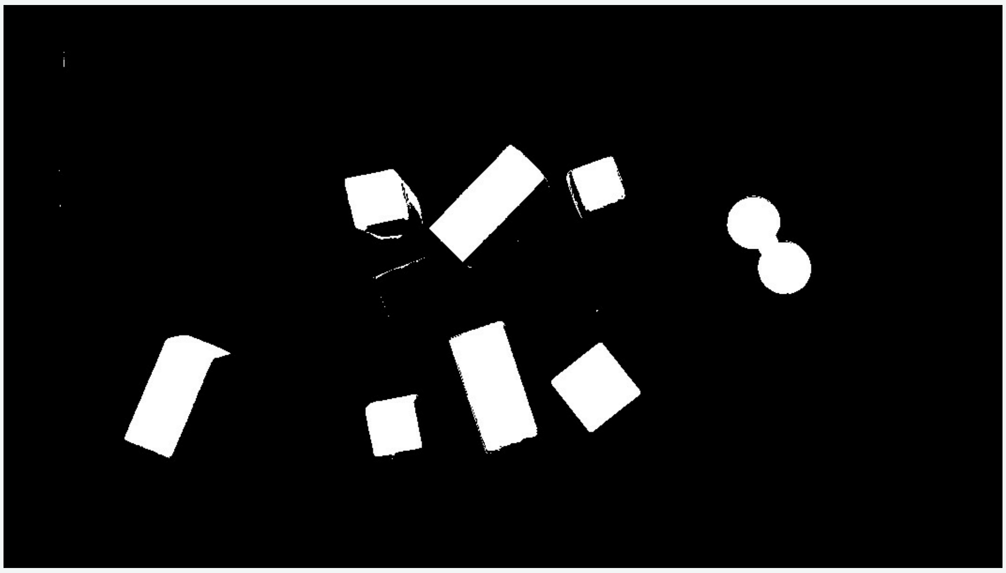
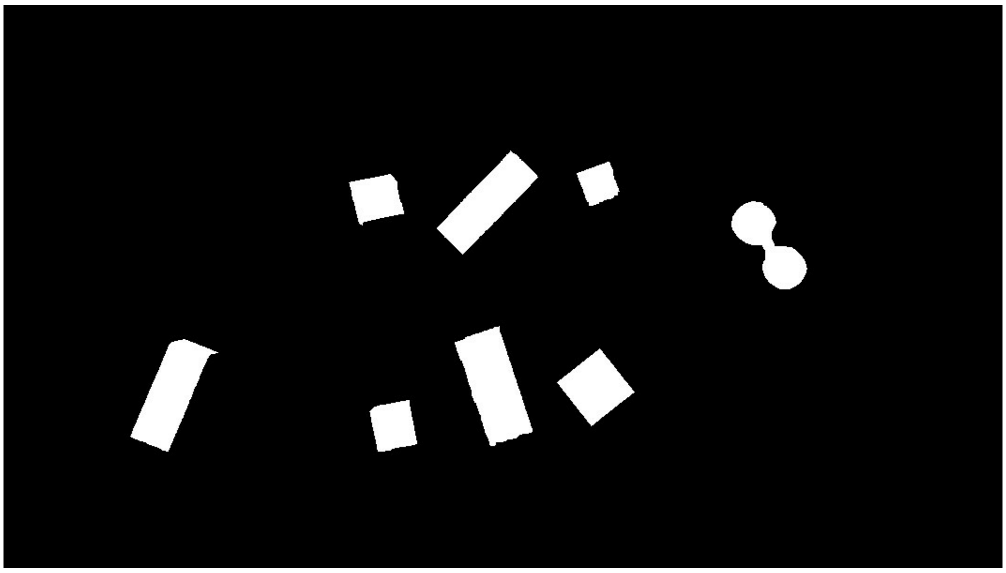
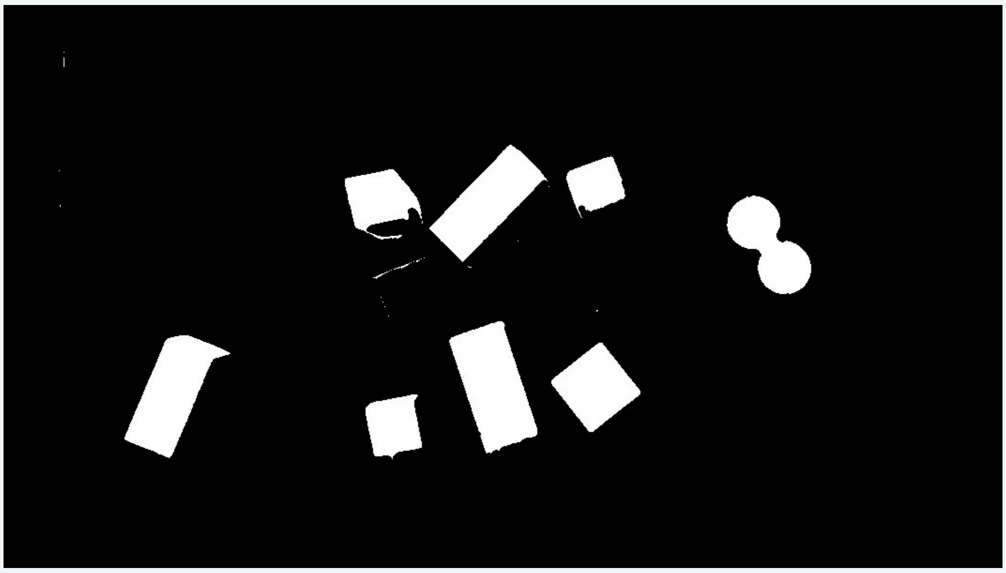
The implemented function to realize these operations is **morphological_transformations()** (index.html#vision.image_functions.morphological_transformations). It uses **MorphoType** (index.html#vision.enums.MorphoType) and **KernelType** (index.html#vision.enums.KernelType) to determine which operation should be applied on the image.

The code shows how to do a Closing & an Erosion:

```
img_threshold = threshold_hsv(img_test, *ColorHSV.ANY.value)

img_close = morphological_transformations(img_threshold, morpho_type=MorphoType.CLOSE,
                                          kernel_shape=(11, 11), kernel_type=KernelType.ELLIPSE)

img_erode = morphological_transformations(img_threshold, morpho_type=MorphoType.ERODE,
                                          kernel_shape=(9, 9), kernel_type=KernelType.RECT)

show_img("img_threshold", img_threshold)
show_img("img_erode", img_erode)
show_img_and_wait_close("img_close", img_close)
```

Images result

Morpho type	Image result

Morpho type	Image result
None	
Erode	
Close	

Contours finder

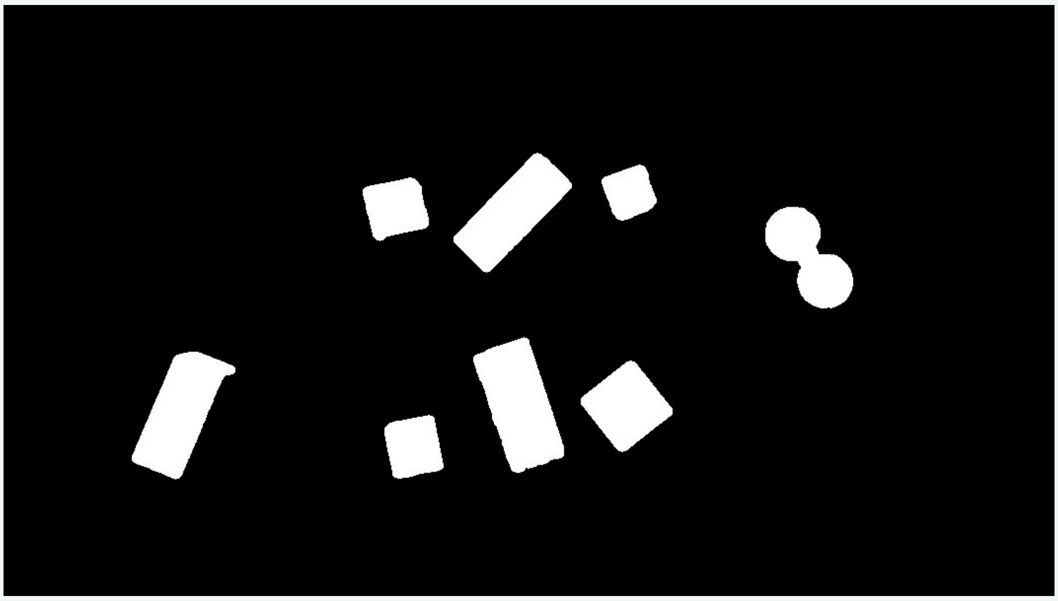
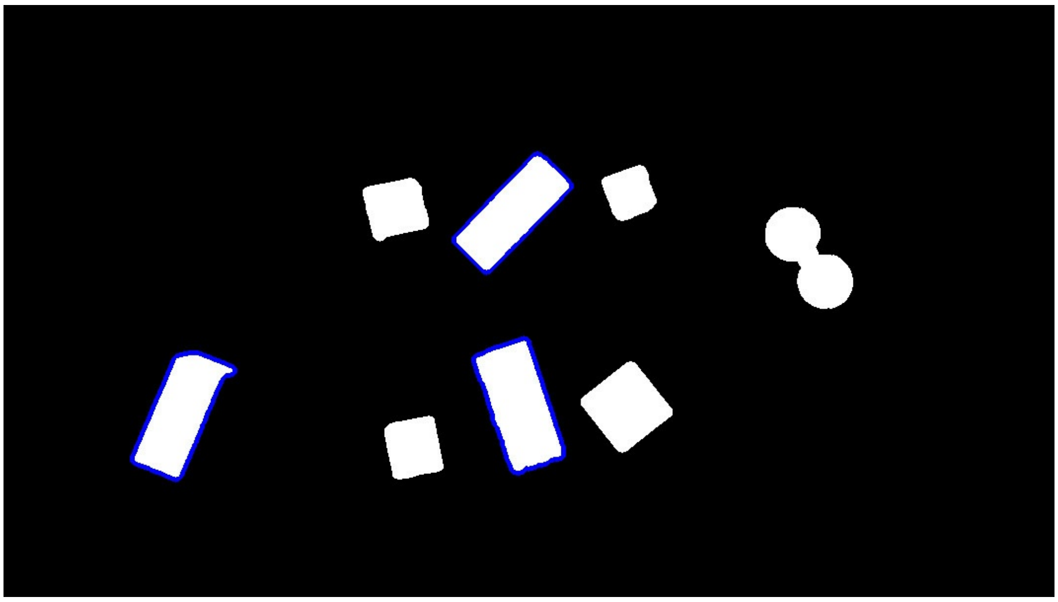
Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition. See more on OpenCV Documentation (https://docs.opencv.org/3.4/d3/d05/tutorial_py_table_of_contents_contours.html).

The implemented function to realize these operations is **biggest_contours_finder()** (index.html#vision.image_functions.biggest_contours_finder) which takes a Black & White image, and extracts the biggest (in term of area) contours from it.

The code to extract and draw the 3 biggest contours from an image is the following:

```
img_threshold = threshold_hsv(img_test, *ColorHSV.ANY.value)
img_threshold = morphological_transformations(img_threshold, morpho_type=MorphoType.OPEN,
                                              kernel_shape=(11, 11), kernel_type=KernelType.ELLIPSE)

cnts = biggest_contours_finder(img_threshold, 3)

img_contours = draw_contours(img_threshold, cnts)

show_img("init", img_threshold)
show_img_and_wait_close("img with contours", img_contours)
```

Images result

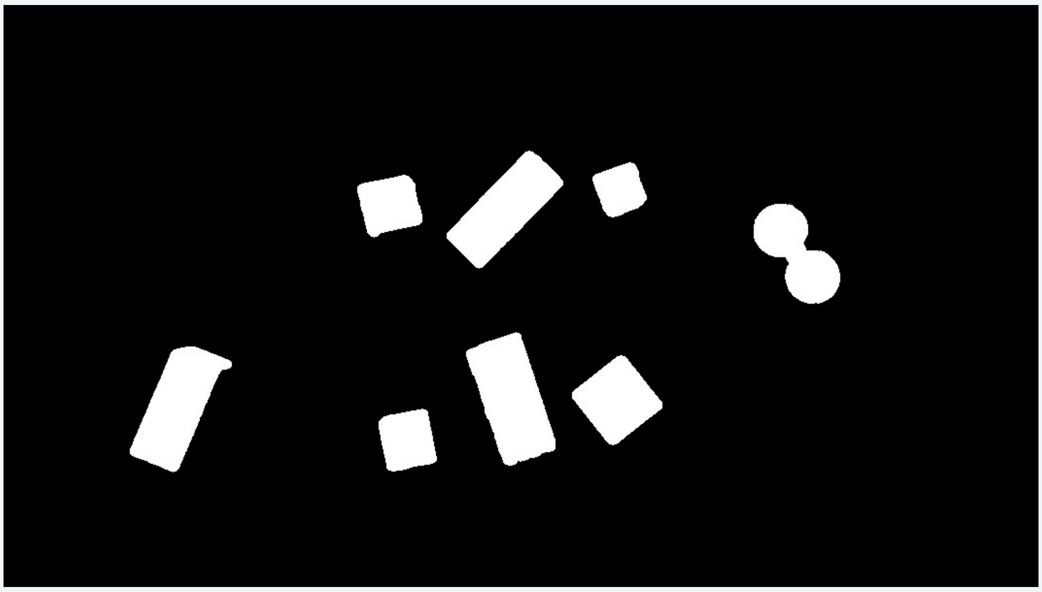
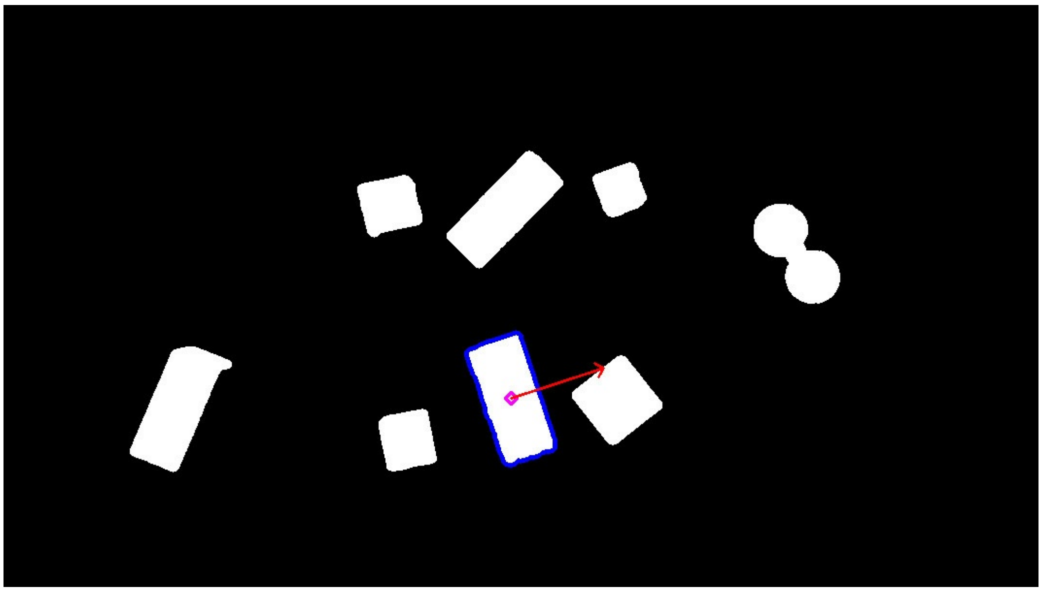
Thresh + Opening	
3 contours	

Find object center position

In order to catch an object, we need to find a pose from where the end effector can grasp the object. The following method uses contours which have been found in the previous section and finds their barycenter and orientation via the functions **get_contour_barycenter()** (index.html#vision.image_functions.get_contour_barycenter) & **get_contour_angle()** (index.html#vision.image_functions.get_contour_angle).

```
img_threshold = threshold_hsv(img_test, *ColorHSV.ANY.value)
img_threshold = morphological_transformations(img_threshold, morpho_type=MorphoType.OPEN,
                                              kernel_shape=(11, 11), kernel_type=KernelType.ELLIPSE)

cnt = biggest_contour_finder(img_threshold)

cnt_barycenter = get_contour_barycenter(cnt)
cx, cy = cnt_barycenter
cnt_angle = get_contour_angle(cnt)

img_debug = draw_contours(img_threshold, [cnt])
img_debug = draw_barycenter(img_debug, cx, cy)
img_debug = draw_angle(img_debug, cx, cy, cnt_angle)
show_img("Image with contours, barycenter and angle", img_debug, wait_ms=30)
```

Images result

Thresh + Opening	
Barycenter + Angle	

Note

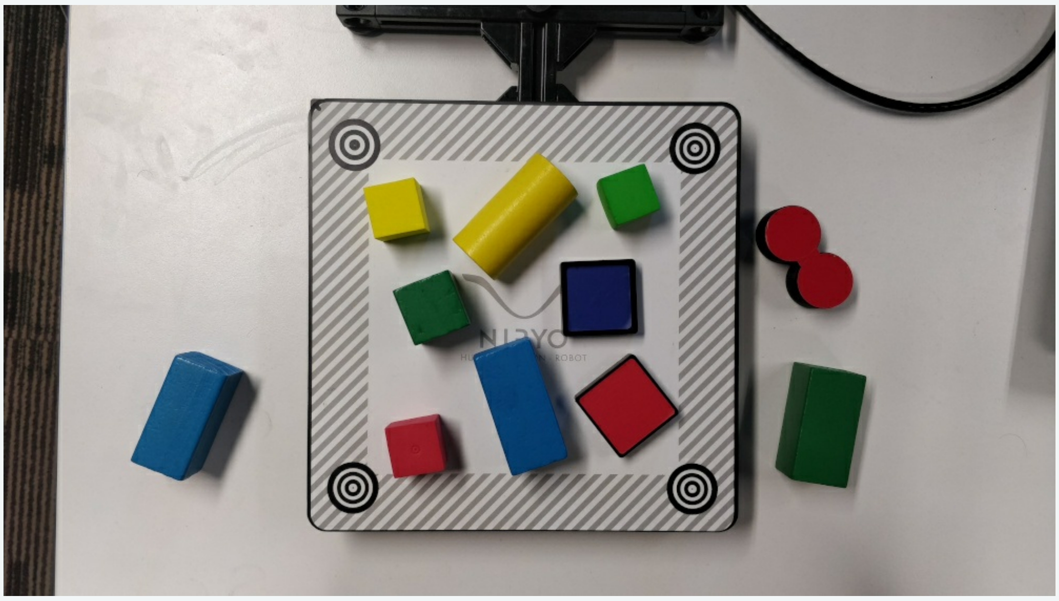
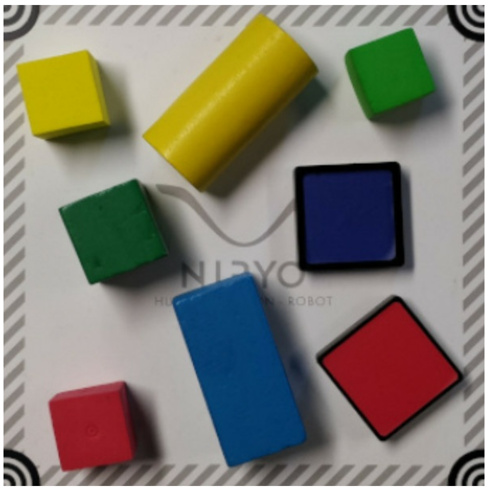
The drawn vector is normal to the contour's length because we want Ned to catch the object by the width rather than the length. Indeed, it leads to least cases where the gripper cannot open enough.

Markers extraction

As image processing happens in a workspace, it is important to extract the workspace beforehand! To do so, you can use the function **extract_img_workspace()** (index.html#vision.image_functions.extract_img_workspace).

```
status, im_work = extract_img_workspace(img, workspace_ratio=1.0)
show_img("init", img_test)
show_img_and_wait_close("img_workspace", img_workspace)
```

Images result

Original	
Extracted	

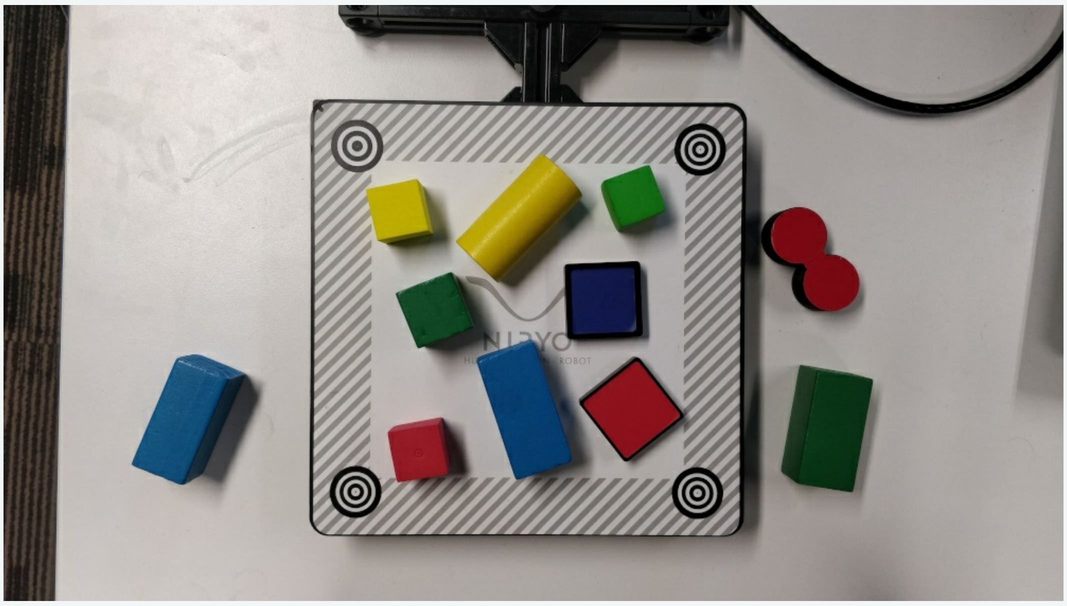
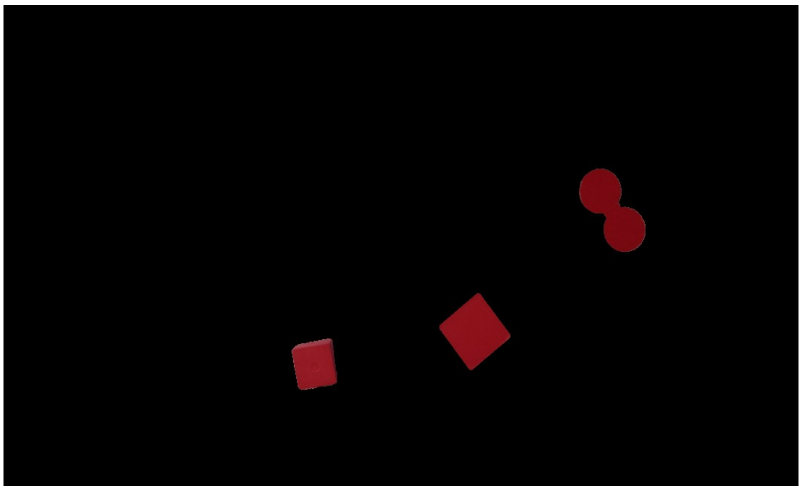
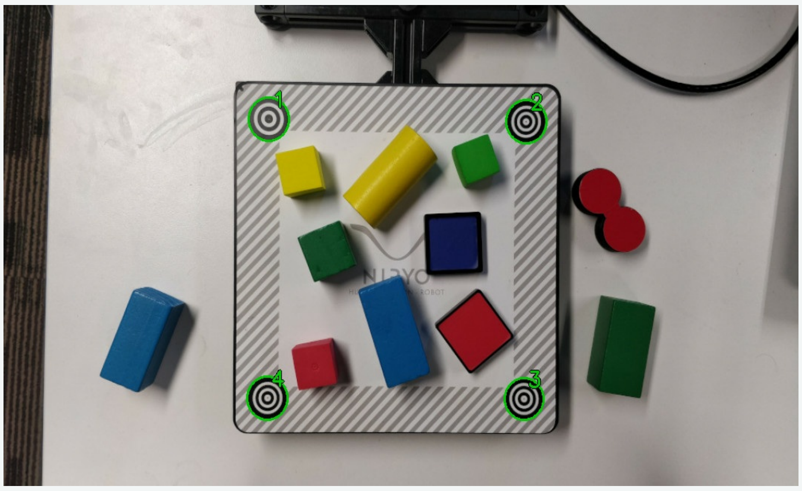
Debug mode

If Ned's functions are failing, you can use Debug functions which are **debug_threshold_color()** (index.html#vision.image_functions.debug_threshold_color) & **debug_markers()** (index.html#vision.image_functions.debug_markers) in order to display what the robot sees.

You can use the functions as follow:

```
debug_color = debug_threshold_color(img_test, ColorHSV.RED)
_status, debug_markers_im = debug_markers(img_test, workspace_ratio=1.0)

show_img("init", img_test)
show_img("debug_color", debug_color)
show_img_and_wait_close("debug_markers", debug_markers_im)
```

Images result

Original	
Debug red	
Debug Markers	

Do your own image processing!

Now that you are a master in image processing, let’s look at full examples.

Display video stream with extracted workspace

In the current state, the following code will display the video stream and the extracted workspace image. You can add your own image processing functions maybe to apply additional process.

```
from pyniryo import *

# Connecting to robot
robot = NiryoRobot("10.10.10.10")
robot.calibrate_auto()

# Getting calibration param
mtx, dist = robot.get_camera_intrinsics()
# Moving to observation pose
robot.move_pose(*observation_pose.to_list())

while "User do not press Escape neither Q":
    # Getting image
    img_compressed = robot.get_img_compressed()
    # Uncompressing image
    img_raw = uncompress_image(img_compressed)
    # Undistorting
    img_undistort = undistort_image(img_raw, mtx, dist)
    # Trying to find markers
    workspace_found, res_img_markers = debug_markers(img_undistort)
    # Trying to extract workspace if possible
    if workspace_found:
        img_workspace = extract_img_workspace(img_undistort, workspace_ratio=1.0)
    else:
        img_workspace = None

    # --- --------- --- #
    # --- YOUR CODE --- #
    # --- --------- --- #

    # - Display
    # Concatenating raw image and undistorted image
    concat_ims = concat_imgs((img_raw, img_undistort))
    # Concatenating extracted workspace with markers annotation
    if img_workspace is not None:
        resized_img_workspace = resize_img(img_workspace, height=res_img_markers.shape[0])
        res_img_markers = concat_imgs((res_img_markers, resized_img_workspace))
    # Showing images
    show_img("Images raw & undistorted", concat_ims)
    key = show_img("Markers", res_img_markers, wait_ms=30)
    if key in [27, ord("q")]:  # Will break loop if the user press Escape or Q
        break
```

Vision pick via your image processing pipeline

You may want to send coordinate to Ned in order to pick the object of your choice! To do that, use the function **get_target_pose_from_rel()** (index.html#api.tcp_client.NiryoRobot.get_target_pose_from_rel) which converts a relative pose in the workspace into a pose in the robot’s world!

```
from pyniryo import *

# -- MUST Change these variables
simulation_mode = True
if simulation_mode:
    robot_ip_address, workspace_name = "127.0.0.1", "gazebo_1"
else:
    robot_ip_address, workspace_name = "10.10.10.10", "workspace_1"

# -- Can Change these variables
grid_dimension = (3, 3)  # conditioning grid dimension
vision_process_on_robot = False  # boolean to indicate if the image processing append on the Robot
display_stream = True  # Only used if vision on computer

# -- Should Change these variables
# The pose from where the image processing happens
observation_pose = PoseObject(
    x=0.17, y=0., z=0.35,
    roll=0.0, pitch=1.57, yaw=0.0,
)

# Center of the conditioning area
center_conditioning_pose = PoseObject(
    x=0.0, y=-0.25, z=0.12,
    roll=-0., pitch=1.57, yaw=-1.57
)


# -- MAIN PROGRAM

def process(niryo_robot):
    """

    :type niryo_robot: NiryoRobot
    :rtype: None
    """
    # Initializing variables
    obj_pose = None
    try_without_success = 0
    count = 0
    if not vision_process_on_robot:
        mtx, dist = niryo_robot.get_camera_intrinsics()
    else:
        mtx = dist = None
    # Loop
    while try_without_success < 5:
        # Moving to observation pose
        niryo_robot.move_pose(observation_pose)

        if vision_process_on_robot:
            ret = niryo_robot.get_target_pose_from_cam(workspace_name,
                                                     height_offset=0.0,
                                                     shape=ObjectShape.ANY,
                                                     color=ObjectColor.ANY)
            obj_found, obj_pose, shape, color = ret

        else:  # Home made image processing
            img_compressed = niryo_robot.get_img_compressed()
            img = uncompress_image(img_compressed)
            img = undistort_image(img, mtx, dist)
            # extracting working area
            im_work = extract_img_workspace(img, workspace_ratio=1.0)
            if im_work is None:
                print("Unable to find markers")
                try_without_success += 1
                if display_stream:
                    cv2.imshow("Last image saw", img)
                    cv2.waitKey(25)
                continue

            # Applying Threshold on ObjectColor
            color_hsv_setting = ColorHSV.ANY.value
            img_thresh = threshold_hsv(im_work, *color_hsv_setting)

            if display_stream:
                show_img("Last image saw", img, wait_ms=100)
                show_img("Image thresh", img_thresh, wait_ms=100)
            # Getting biggest contour/blob from threshold image
            contour = biggest_contour_finder(img_thresh)
            if contour is None or len(contour) == 0:
                print("No blob found")
                obj_found = False
            else:
                img_thresh_rgb_w_contour = draw_contours(img_thresh, [contour])

                # Getting contour/blob center and angle
                cx, cy = get_contour_barycenter(contour)
                img_thresh_rgb_w_contour = draw_barycenter(img_thresh_rgb_w_contour, cx, cy)
                cx_rel, cy_rel = relative_pos_from_pixels(im_work, cx, cy)

                angle = get_contour_angle(contour)
                img_thresh_rgb_w_contour = draw_angle(img_thresh_rgb_w_contour, cx, cy, angle)

                show_img("Image thresh", img_thresh_rgb_w_contour, wait_ms=30)

                # Getting object world pose from relative pose
                obj_pose = niryo_robot.get_target_pose_from_rel(workspace_name,
                                                                height_offset=0.0,
                                                                x_rel=cx_rel, y_rel=cy_rel,
                                                                yaw_rel=angle)
                obj_found = True
        if not obj_found:
            try_without_success += 1
            continue
        # Everything is good, so we going to object
        niryo_robot.pick_from_pose(obj_pose)

        # Computing new place pose
        offset_x = count % grid_dimension[0] - grid_dimension[0] // 2
        offset_y = (count // grid_dimension[1]) % 3 - grid_dimension[1] // 2
        offset_z = count // (grid_dimension[0] * grid_dimension[1])
        place_pose = center_conditioning_pose.copy_with_offsets(0.05 * offset_x, 0.05 * offset_y, 0.025 * offset_z)

        # Placing
        niryo_robot.place_from_pose(place_pose)

        try_without_success = 0
        count += 1


if __name__ == '__main__':
    # Connect to robot
    robot = NiryoRobot(robot_ip_address)
    # Changing tool
    robot.update_tool()
    # Calibrate robot if robot needs calibration
    robot.calibrate_auto()
    # Launching main process
    process(robot)
    # Ending
    robot.go_to_sleep()
    # Releasing connection
    robot.close_connection()
```

Functions documentation

This file presents the different functions and Enums Image Processing available for image processing.

These functions are divided in subsections:

- Pure image processing are used to deal with the thresholding, contours detection, ..
- Workspaces wise section contains functions to extract workspace and deal with the relative position in the workspace
- The section Show allows to display images
- Image Editing contains lot of function which can compress images, add text to image, …

Pure image processing

image_functions.threshold_hsv(*list_min_hsv*, *list_max_hsv*, *reverse_hue=False*, *use_s_prime=False*)

Take BGR image (OpenCV imread result) and return thresholded image according to values on HSV (Hue, Saturation, Value) Pixel will worth 1 if a pixel has a value between min_v and max_v for all channels

Parameters:

- **img** (*numpy.array*) – image BGR if rgb_space = False
- **list_min_hsv** (*list* (https://docs.python.org/3/library/stdtypes.html#list)[*int* (https://docs.python.org/3/library/functions.html#int)]) – list corresponding to [min_value_H,min_value_S,min_value_V]
- **list_max_hsv** (*list* (https://docs.python.org/3/library/stdtypes.html#list)[*int* (https://docs.python.org/3/library/functions.html#int)]) – list corresponding to [max_value_H,max_value_S,max_value_V]

- **use_s_prime** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – True if you want to use S channel as S' = S x V else classic
- **reverse_hue** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – Useful for Red color cause it is at both extremum

Returns:
threshold image

Return type:
numpy.array

image_functions.debug_threshold_color(*color_hsv*)

Return masked image to see the effect of color threshold

Parameters:

- **img** (*numpy.array*) – OpenCV image
- **color_hsv** (*ColorHSV* (index.html#vision.enums.ColorHSV)) – Color used for debug

Returns:
Masked image

Return type:
numpy.array

image_functions.morphological_transformations(*morpho_type=<MorphoType.CLOSE: 3>*, *kernel_shape=(5, 5)*, *kernel_type=<KernelType.ELLIPSE: 2>*)

Take black & white image and apply morphological transformation

Parameters:

- **im_thresh** (*numpy.array*) – Black & White Image
- **morpho_type** (*MorphoType* (index.html#vision.enums.MorphoType)) – CLOSE/OPEN/ERODE/DILATE => See on OpenCV/Google if you do not know these words
- **kernel_shape** (*tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[float* (https://docs.python.org/3/library/functions.html#float)*]*) – tuple corresponding to the size of the kernel
- **kernel_type** (*KernelType* (index.html#vision.enums.KernelType)) – RECT/ELLIPSE/CROSS => see on OpenCV

Returns:
image after processing

Return type:
numpy.array

image_functions.get_contour_barycenter()

Return barycenter of an OpenCV Contour

Parameters:
contour (*OpenCV Contour*) –

Returns:
Barycenter X, Barycenter Y

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

image_functions.get_contour_angle()

Return orientation of a contour according to the smallest side in order to be well oriented for gripper

Parameters:
contour (*OpenCV Contour*) – contour

Returns:
Angle in radians

Return type:
float (https://docs.python.org/3/library/functions.html#float)

image_functions.biggest_contour_finder()

image_functions.biggest_contours_finder(*nb_contours_max=3*)

Function to find the biggest contour in an binary image

Parameters:
- **img** (*numpy.array*) – Binary Image
- **nb_contours_max** (*int* (https://docs.python.org/3/library/functions.html#int)) – maximal number of contours which will be returned

Returns:
biggest contours found

Return type:
list (https://docs.python.org/3/library/stdtypes.html#list)[OpenCV Contour]

image_functions.draw_contours(*contours*)

Draw a list of contour on an image and return the drawing image

Parameters:
- **img** (*numpy.array*) – Image
- **contours** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[OpenCV Contour]*) – contours list

Returns:
Image with drawing

Return type:
numpy.array

image_functions.draw_barycenter(*cx, cy, color=(255, 0, 255), marker_size=10, thickness=2*)

Draw a barycenter marker on an image and return the drawing image

Parameters:
- **img** (*numpy.array*) – Image
- **cx** (*int* (https://docs.python.org/3/library/functions.html#int)) – Barycenter X
- **cy** (*int* (https://docs.python.org/3/library/functions.html#int)) – Barycenter Y
- **color** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[uint8, utin8, uint8]*) – (R, G, B) colors of the marker
- **marker_size** (*int* (https://docs.python.org/3/library/functions.html#int)) – size of the marker
- **thickness** (*int* (https://docs.python.org/3/library/functions.html#int)) – thickness of the marker

Returns:

Image with drawing

Return type:
numpy.array

image_functions.draw_angle(*cx, cy, angle, color=(0, 0, 255), arrow_length=30, thickness=2*)

Draw an arrow that represents the orientation of an object on an image and return the drawing image

Parameters:

- **img** (*numpy.array*) – Image
- **cx** (*int* (https://docs.python.org/3/library/functions.html#int)) – Barycenter X
- **cy** (*int* (https://docs.python.org/3/library/functions.html#int)) – Barycenter Y
- **angle** – angle to display
- **color** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[uint8, utin8, uint8]*) – (R, G, B) colors of the marker
- **arrow_length** (*int* (https://docs.python.org/3/library/functions.html#int)) – length of the arrow marker
- **thickness** (*int* (https://docs.python.org/3/library/functions.html#int)) – thickness of the arrow marker

Returns:
Image with drawing

Return type:
numpy.array

Workspaces wise

image_functions.extract_img_workspace(*workspace_ratio=1.0*)

Extract working area from an image thanks to 4 Niryo's markers

Parameters:

- **img** (*numpy.array*) – OpenCV image which contain 4 Niryo's markers
- **workspace_ratio** (*float* (https://docs.python.org/3/library/functions.html#float)) – Ratio between the width and the height of the area represented by the markers

Returns:
extracted and warped working area image

Return type:
numpy.array

image_functions.debug_markers(*workspace_ratio=1.0*)

Display detected markers on an image

Parameters:

- **img** (*numpy.array*) – OpenCV image which contain Niryo's markers
- **workspace_ratio** (*float* (https://docs.python.org/3/library/functions.html#float)) – Ratio between the width and the height of the area represented by the markers

Returns:
(status, annotated image)

Return type:
numpy.array

image_functions.relative_pos_from_pixels(*x_pixels, y_pixels*)

Transform a pixels position to a relative position

Parameters:

- **img** (*numpy.array*) – Image where the object is detected
- **x_pixels** (*int* (https://docs.python.org/3/library/functions.html#int)) – coordinate X
- **y_pixels** (*int* (https://docs.python.org/3/library/functions.html#int)) – coordinate Y

Returns:

X relative, Y relative

Return type:

float (https://docs.python.org/3/library/functions.html#float), float (https://docs.python.org/3/library/functions.html#float)

Show

image_functions.show_img_and_check_close(*img*)

Display an image and check whether the user want to close

Parameters:

- **window_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – window's name
- **img** (*numpy.array*) – Image

Returns:

boolean indicating if the user wanted to leave

Return type:

bool (https://docs.python.org/3/library/functions.html#bool)

image_functions.show_img(*img, wait_ms=1*)

Display an image during a certain time

Parameters:

- **window_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – window's name
- **img** (*numpy.array*) – Image
- **wait_ms** (*int* (https://docs.python.org/3/library/functions.html#int)) – Wait time in milliseconds

Returns:

value of the key pressed during the display

Return type:

int (https://docs.python.org/3/library/functions.html#int)

image_functions.show_img_and_wait_close(*img*)

Display an image and wait that the user close it

Parameters:

- **window_name** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – window's name
- **img** (*numpy.array*) – Image

Returns:

None

Return type:

None (https://docs.python.org/3/library/constants.html#None)

Image Editing

image_functions.compress_image(*quality=90*)

Compress OpenCV image

Parameters:

- **img** (*numpy.array*) – OpenCV Image
- **quality** (*int* (https://docs.python.org/3/library/functions.html#int)) – integer between 1 - 100. The higher it is, the less information will be lost, but the heavier the compressed image will be

Returns:

status & string representing compressed image

Return type:

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

image_functions.uncompress_image()

Take a compressed img and return an OpenCV image

Parameters:

compressed_image (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – compressed image

Returns:

OpenCV image

Return type:

numpy.array

image_functions.add_annotation_to_image(*text*, *write_on_top=True*)

Add Annotation to an image

Parameters:

- **img** (*numpy.array*) – Image
- **text** (*str* (https://docs.python.org/3/library/stdtypes.html#str)) – text string
- **write_on_top** (*bool* (https://docs.python.org/3/library/functions.html#bool)) – if you write the text on top

Returns:

img with text written on it

Return type:

numpy.array

image_functions.undistort_image(*mtx*, *dist*)

Use camera intrinsics to undistort raw image

Parameters:

- **img** (*numpy.array*) – Raw Image
- **mtx** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]]*) – Camera Intrinsics matrix

- **dist** (*list* (https://docs.python.org/3/library/stdtypes.html#list)*[list* (https://docs.python.org/3/library/stdtypes.html#list)*[float* (https://docs.python.org/3/library/functions.html#float)*]]*) – Distortion Coefficient

Returns:
Undistorted image

Return type:
numpy.array

image_functions.resize_img(*width=None*, *height=None*, *inter=3*)

Resize an image. The user should precise only width or height if he wants to keep image's ratio

Parameters:

- **img** (*numpy.array*) – OpenCV Image
- **width** (*int* (https://docs.python.org/3/library/functions.html#int)) – Target Width
- **height** (*int* (https://docs.python.org/3/library/functions.html#int)) – Target Height
- **inter** (*long*) – OpenCV interpolation flag

Returns:
resized image

Return type:
numpy.array

image_functions.concat_imgs(*axis=1*)

Concat multiple images along 1 axis

Parameters:

- **tuple_imgs** (*tuple* (https://docs.python.org/3/library/stdtypes.html#tuple)*[numpy.array]*) – tuple of images
- **axis** (*int* (https://docs.python.org/3/library/functions.html#int)) – 0 means vertically and 1 means horizontally

Returns:
Concat image

Return type:
numpy.array

Enums Image Processing

Enums are used to pass specific parameters to functions.

List of enums:

- **ColorHSV**
- **ColorHSVPrime**
- **ObjectType**
- **MorphoType**
- **KernelType**

class **ColorHSV**(*value*)

MIN HSV, MAX HSV, Invert Hue (bool)

BLUE= *([90, 50, 85], [125, 255, 255], False)*

RED= *([15, 80, 75], [170, 255, 255], True)*

GREEN= *([40, 60, 75], [85, 255, 255], False)*

ANY= *([0, 50, 100], [179, 255, 255], False)*

class **ColorHSVPrime**(*value*)

MIN HSV, MAX HSV, Invert Hue (bool)

BLUE= *([90, 70, 100], [115, 255, 255], False)*

RED= *([15, 70, 100], [170, 255, 255], True)*

GREEN= *([40, 70, 100], [85, 255, 255], False)*

ANY= *([0, 70, 140], [179, 255, 255], False)*

class **ObjectType**(*value*)

An enumeration.

SQUARE= *4*

TRIANGLE= *3*

CIRCLE= *-1*

ANY= *0*

class **MorphoType**(*value*)

An enumeration.

ERODE= *0*

DILATE= *1*

OPEN= *2*

CLOSE= *3*

class **KernelType**(*value*)

An enumeration.

RECT= *0*

ELLIPSE= *2*

CROSS= *1*

Indices and tables

- Index (genindex.html)
- Module Index (py-modindex.html)
- Search Page (search.html)

Suggest a modification Download as PDF