

Ned - Documentation ROS

Cette documentation contient tout ce dont vous avez besoin pour comprendre le fonctionnement de Ned et comment le contrôler via ROS.

Il s'adresse aussi bien aux utilisateurs qui utilisent le robot en « physique » qu'à ceux qui souhaitent utiliser une version simulée.



Introduction

Avant de se plonger dans la documentation ROS, vous pouvez en savoir plus sur le développement du robot dans la section [vue d'ensemble](#) ([index.html#document-source/stack/overview](#)).

Vous pouvez ensuite aller lire la section [Guide de démarrage](#) ([index.html#document-source/installation/getting_started](#)) pour mettre en place votre environnement et essayer la stack par vous-même. Si vous n'avez pas de robot réel à disposition, vous pouvez toujours le simuler via la section [Utiliser les robots Niryo via la simulation](#) ([index.html#document-source/simulation](#)).

Contrôler Ned via ROS

Ned est entièrement basé sur ROS.

Contrôle direct du Ned avec ROS

❗ Important

Pour contrôler le robot directement avec ROS, il est nécessaire d'être connecté en SSH au robot physique, ou d'utiliser la simulation.

ROS est le moyen le plus direct de contrôler le robot. Il permet :

- D'envoyer une commande via le terminal pour appeler des services, déclencher une action, ...
- D'écrire un nœud Python/C++ entier dans un script afin de réaliser un processus complet.

Se diriger vers la section [ROS](#) ([index.html#document-source/stack/high_level](#)) pour voir tous les Topics et Services disponibles.

Python ROS Wrapper

❗ Important

Pour utiliser Python ROS Wrapper, il faut être connecté en SSH au robot physique, ou alors utiliser la simulation.

Le Python ROS Wrapper est une surcouche à ROS pour permettre un développement plus rapide que ROS. Les programmes sont exécutés directement sur le robot, ce qui permet de les déclencher avec le bouton du robot une fois qu'un ordinateur n'est plus nécessaire.

Rendez vous sur la documentation [Python ROS Wrapper](#) ([index.html#document-source/ros_wrapper](#)) pour voir les fonctions accessibles et des exemples d'utilisation.

Autres alternatives

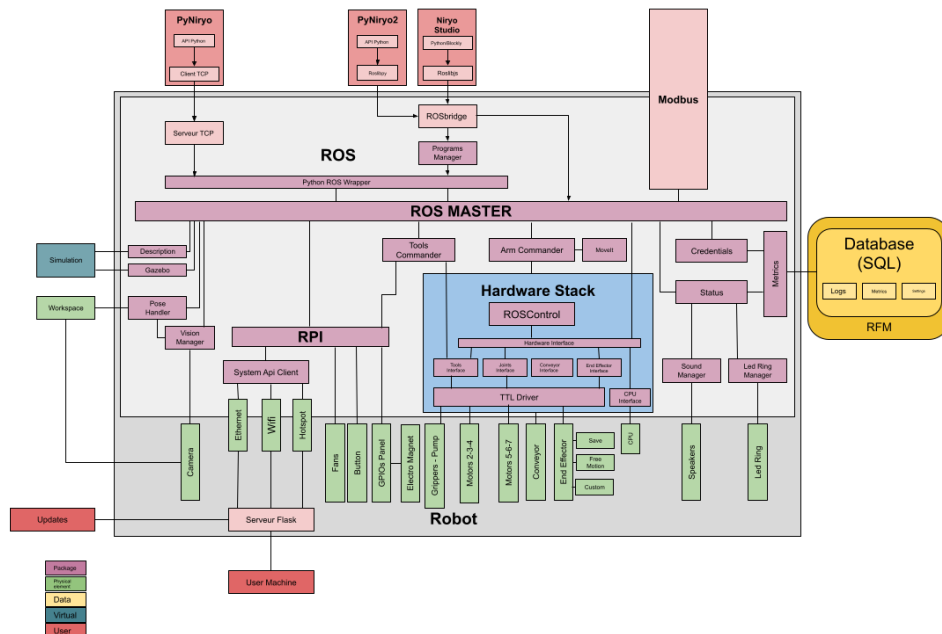
D'autres méthodes sont disponibles pour contrôler Ned afin de permettre à l'utilisateur de coder et d'exécuter des programmes en dehors de son terminal.

En savoir plus sur cette [section](#) ([index.html#document-source/more](#)).

Aperçu de la stack ROS

Ned est un robot basé sur Raspberry, Arduino & ROS. Il utilise ROS afin de créer une interface entre le matériel et les liaisons de haut niveau.

Voici sur le schéma suivant un aperçu global du logiciel de Ned afin de comprendre où sont placées chaque partie du logiciel.



Logiciel V3 du robot Niryo

Utiliser votre Robot Niryo



Chaque Robot Niryo est utilisable tel quel quand il est allumé la première fois, avec Niryo Studio par exemple. Cependant, ce Robot peut être utilisé de bien d'autres manières si vous voulez plonger plus profondément dans sa compréhension.

Dans ce tutoriel, nous allons expliquer comment le Robot est configuré et les différentes options disponibles pour le contrôler.

Se connecter au Robot

Vous pouvez vous connecter au robot de multiples manières différentes (Ethernet direct, hotspot wifi, LAN).

Vous pouvez trouver plus d'information sur comment se connecter au robot [ici](https://docs.niryo.com/product/niryo-studio/source/connection.html?lang=fr) (<https://docs.niryo.com/product/niryo-studio/source/connection.html?lang=fr>).

Une fois que le robot est accessible depuis votre ordinateur, vous pouvez y accéder de trois manières différentes :

- Via Niryo Studio

Niryo Studio vous fournit tous les outils nécessaires pour contrôler le robot. Veuillez vous référer à la [documentation Niryo Studio](https://docs.niryo.com/product/niryo-studio/v3.2.1/en/index.html) (<https://docs.niryo.com/product/niryo-studio/v3.2.1/en/index.html>) pour plus d'information.

- Via ROS Multimachine.

ROS implémente une manière de communiquer entre des nodes lancées sur différentes machines. En indiquant à votre ordinateur où se trouve le Node Master ROS, vous pouvez communiquer avec n'importe quel Node ROS comme si vous étiez directement connecté sur le robot. Voir le [tutoriel ROS wiki](https://wiki.ros.org/ROS/Tutorials/MultipleMachines) (<https://wiki.ros.org/ROS/Tutorials/MultipleMachines>) pour plus d'information.

- Via ssh (pour les utilisateurs avancés seulement).

Le port 22 est ouvert sans restriction d'accès. L'utilisateur par défaut est « niryo » et son mot de passe est « robotics ».

Organisation du robot

Pour vous aider à comprendre comment l'OS est installé sur le robot, nous vous indiquons quelques aperçus de son fonctionnement.

ⓘ Attention

Ce document n'a pas pour vocation d'expliquer comment installer complètement un robot depuis une carte SD vide. Il n'est écrit que pour vous donner des pistes sur son architecture. Quelques étapes d'installation sont référencées dans [Installer sous Ubuntu 18](#) ([index.html#ubuntu-18-installation](#)) au cas où vous voudriez réinstaller quelques uns de ses éléments (catkin_ws par exemple).

Organisation du système

Le Robot tourne sur une Ubuntu 18.04.5 pour ARM, personnalisé pour fonctionner avec une raspberry pi 4B. suivants : - ROS melodic et ses prérequis - Driver de son - Driver du Led Ring - Services du système du Robot (connectivité, base de donnée, serveur flask) - Outils de développement de base (compilation, outils d'édition)

Il comprend tous les éléments suivants:

- ROS melodic et ses prérequis
- Driver de son
- Driver du Led Ring
- Services du système du Robot (connectivité, base de donnée, serveur flask)
- Basic development tools (compilation, editing tools)

Nous avons pris soin de mettre à jour le système avant de vous l'envoyer

ⓘ Attention

Nous ne pouvons pas assurer que la stabilité du système sera conservée si vous essayez de le mettre à jour vous-même (en utilisant apt). Nous vous recommandons fortement de ne pas le faire.

Organisation du Home

Le système a été configuré avec un utilisateur par défaut « niryo ». Le coeur du programme du Robot est installé dans le dossier home de l'utilisateur niryo `/home/niryo`.

Dans ce dossier, vous avez :

- `catkin_ws` : contient le code source et les binaires compilés de la stack Niryo ROS
- `firmware_updater` : Outil de mise à jour des moteurs Steppers et de l'End Effector
- `niryo_robot_saved_files` : ensemble de fichiers sauvegardés sur le robot, utilisés par Niryo Studio
- `system_software` : fichiers de configuration pour les fonction globales du système

Services et daemons

Deux services sont utilisés sur le Robot :

- `niryo_system_software` : Lance le serveur flask pour les communications API avec le robot
- `niryo_robot_ros` ; Lance la stack avec le script `/opt/start_ros.sh` au démarrage.

Fichier `/opt/start_ros.sh` sur le robot Ned 2 :

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

Si vous souhaitez démarrer, arrêter ou désactiver l'un de ces services, veuillez vous référer à [ladocumentation dédiée](https://manpages.ubuntu.com/manpages/bionic/man8/service.8.html) (https://manpages.ubuntu.com/manpages/bionic/man8/service.8.html).

Démarrer le robot manuellement (pour les utilisateurs avancés seulement)

Soyez conscient de ce que vous faites avant de continuer.

Vous aurez besoin d'un accès ssh pour continuer.

Arrêter le service

Vous devez tout d'abord arrêter la stack Niryo ROS qui est automatiquement démarrée quand le robot s'allume. Utiliser la commande `linuxsystem` pour cela :

```
sudo service niryo_robot_ros stop
```

Démarrer le robot

Pour démarrer le robot, lancer les commandes suivantes dans un terminal ssh :

Pour le Ned

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

Pour le Ned2

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

Options de lancement du Robot

Nom	Valeur par défaut	Description
log_level	INFO	Niveau de log à afficher pour les loggers ROS
ttn_enabled	true	Active ou désactive l'utilisation du port TTL. Cette fonctionnalité est principalement utilisée pour déboguer et peut mener à une stack instable.

Nom	Valeur par défaut	Description
can_enabled	true	Active ou désactive l'utilisation du bus CAN. Cette fonctionnalité est principalement utilisée pour déboguer et peut mener à une stack instable.
debug	false	Lance en mode debug. Pour du développement et du debug seulement.
timed	true	Pour lancer le noeud en utilisant timed_roslaunch. Si activé, lancera d'abord les nodes de son et de lumière pour une meilleure expérience utilisateur. Si désactivé, le node sera directement lancé

Changer le niveau de log

Avant de lancer le robot, vous pouvez changer le fichier de configuration du Logueur ROS afin de changer les niveaux de log affichés dans le terminal. Ce fichier est localisé dans `/home/niryo/catkin_ws/src/niryo_robot_bringup/config/niryo_robot_trace.conf`.

Il définit les niveaux de log pour tous les packages cpp, basé sur la syntaxe du fichier de configuration de log4cxx. Veuillez vous référer à la documentation de [rosconsole](http://wiki.ros.org/rosconsole) (<http://wiki.ros.org/rosconsole>) ou [log4cxx](https://logging.apache.org/log4cxx/latest_stable/index.html) (https://logging.apache.org/log4cxx/latest_stable/index.html) pour plus d'information.

Par défaut, le niveau est défini à INFO, vous pouvez changer cette valeur si vous voulez plus de logs.

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

ⓘ Attention

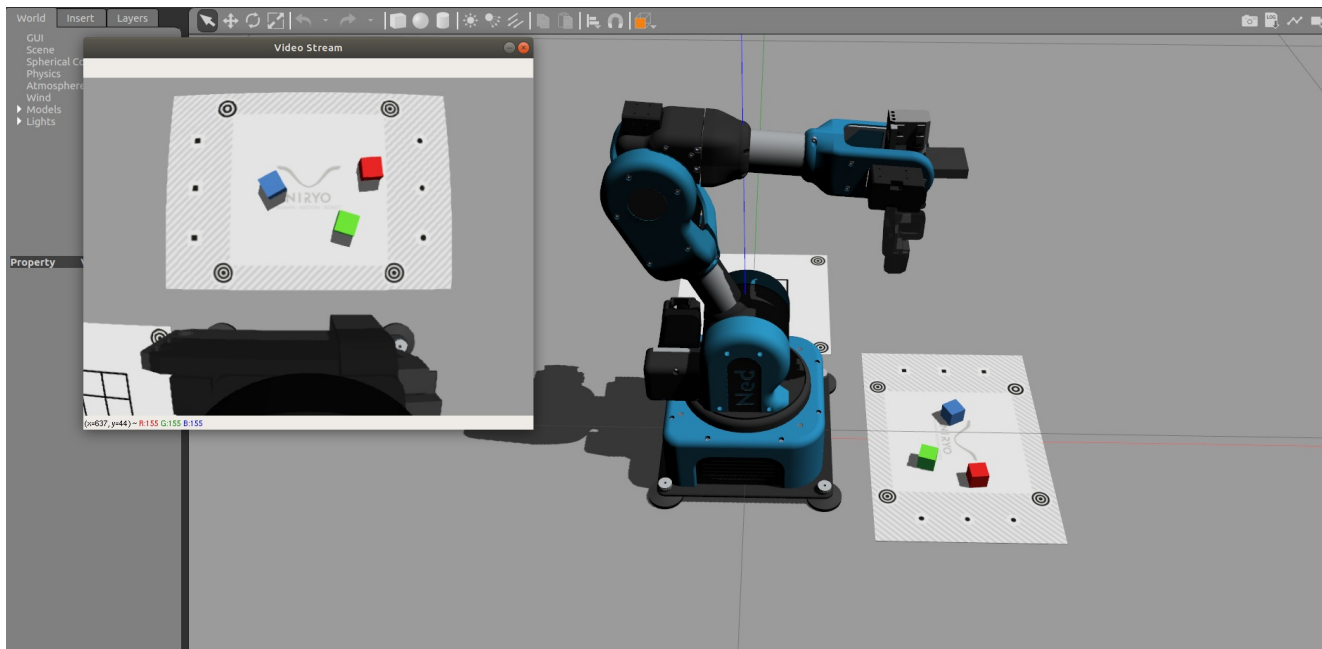
Le niveau DEBUG est très verbeux, vous pouvez détériorer les performances du robot en faisant cela.

Vous pouvez aussi choisir de changer seulement les niveaux de log d'un package cpp spécifique en décommentant l'une des lignes suivantes et éventuellement changer le niveau de log associé.

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

Utiliser les robots Niryo via la simulation

La simulation permet de contrôler virtuellement Ned depuis votre ordinateur.



Ned dans la simulation de Gazebo

Avec ce tutoriel, apprenez à configurer une simulation sur un ordinateur.

ⓘ Note

Il est possible d'utiliser [Niryo Studio avec la simulation](https://docs.niryo.com/product/niryo-studio/source/connection.html#using-ned-in-simulation-with-niryo-studio/) (<https://docs.niryo.com/product/niryo-studio/source/connection.html#using-ned-in-simulation-with-niryo-studio/>). Pour cela il vous suffit de connecter Niryo Studio en « Localhost ».

Installation de l'environnement de simulation

ⓘ Attention

L'ensemble de la stack ROS est développée et testée sur ROS **Melodic** qui nécessite **Ubuntu 18.04** pour fonctionner correctement. L'utilisation d'une autre version de ROS ou d'un autre système d'exploitation peut entraîner les dysfonctionnements de certains packages. Veuillez suivre les étapes indiquées dans [Installer sous Ubuntu 18](#) ([index.html#ubuntu-18-installation](#)) pour installer un environnement fonctionnel.

Utilisation de la simulation

❗ Important

- Les fonctionnalités matérielles sont simulées comme si vous utilisiez un robot réel.
- Les données retournées par les drivers fictifs est arbitraire et immuable. Parmi ces données, vous trouverez : voltage, température, état d'erreur (toujours 0), ping (toujours vrai), état du end effector (immuable)

La simulation est un outil puissant qui permet de tester de nouveaux programmes directement sur votre ordinateur en évitant de transférer un nouveau code sur le robot.

Cela aide également à des fins de développement → aucun besoin de transférer du code, de compiler et de redémarrer le robot, ce qui est beaucoup plus lent que de le faire sur un ordinateur de bureau.

Sans physique - Pas de Visualisation

Ce mode est principalement utilisé pour la simulation et les tests, vous rapprochant le plus possible du contrôle d'un robot réel. Il est disponible pour toutes les architectures actuellement supportées. Vous pouvez y accéder par la commande :

- Simulation du One :

```
roslaunch niryo_robot_bringup niryo_one_simulation.launch
```

- Simulation du One :

```
roslaunch niryo_robot_bringup niryo_ned_simulation.launch
```

- Simulation du Ned2 :

```
roslaunch niryo_robot_bringup niryo_ned2_simulation.launch
```

Ce mode est utile si vos capacités de calcul sont limitées ou si vous n'avez pas de serveur X disponible.

Options

Ce mode est le mode le plus flexible, car il procure toutes les options possibles pour personnaliser la simulation. Les autres modes de simulation (avec RViz et Gazebo), certains paramètres seront forcés à certaines valeurs.

Options de la simulation sans visualisation

Nom	Valeur par défaut	Description
log_level	INFO	Niveau de log à afficher pour les loggers ROS
ttn_enabled	true	Active ou désactive l'usage du bus TTL. Cette fonctionnalité est principalement utilisée pour déboguer et peut mener à un fonctionnement instable de la stack.
can_enabled	true	Active ou désactive l'usage du bus CAN. Cette fonctionnalité est principalement utilisée pour déboguer et peut mener à un fonctionnement instable de la stack.
debug	false	Lance la stack en mode debug. Pour le développement et le debugage seulement.
conf_location	version.txt	Localisation du fichier version.txt. Un chemin vers ce fichier est requis.
simu_gripper	true	Simule la présence d'un outil d'id 11 sur le bus
simu_conveyor	true	Simule la présence d'un convoyeur (CAN pour le One et le Ned, TTL pour le Ned2, d'après les fichiers de configuration) sur le bus
vision_enabled	true	Active le kit vision
gazebo	false	Active les paramètres dédiés à gazebo (Cependant cela ne lance pas gazebo, utilisez le fichier launch dédié à gazebo pour cela)

Sans physique - RViz Visualisation

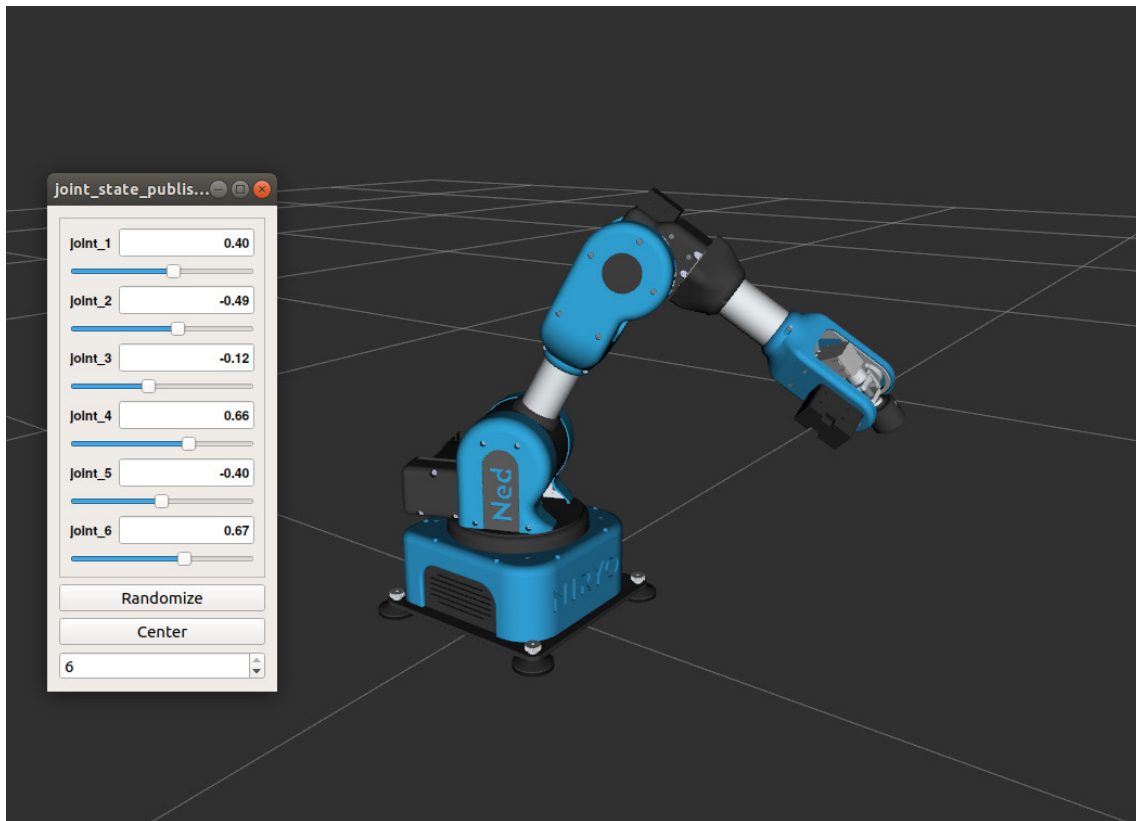
Une simple visualisation du robot est possible via un outil appelé RViz. Cette application va simuler le robot avec sa géométrie et ses positions correctes mais sans moteur physique pour éviter d'utiliser trop de processeur.

Contrôler avec Trackbar

Cette visualisation permet un premier contrôle facile du robot et aide à comprendre la position des axes. Utiliser la commande pour y accéder :

```
roslaunch niryo_robot_description display.launch
```

RViz doit s'ouvrir avec une fenêtre contenant 6 trackbars. Chacune de ces trackbars permet de contrôler l'axe correspondant.



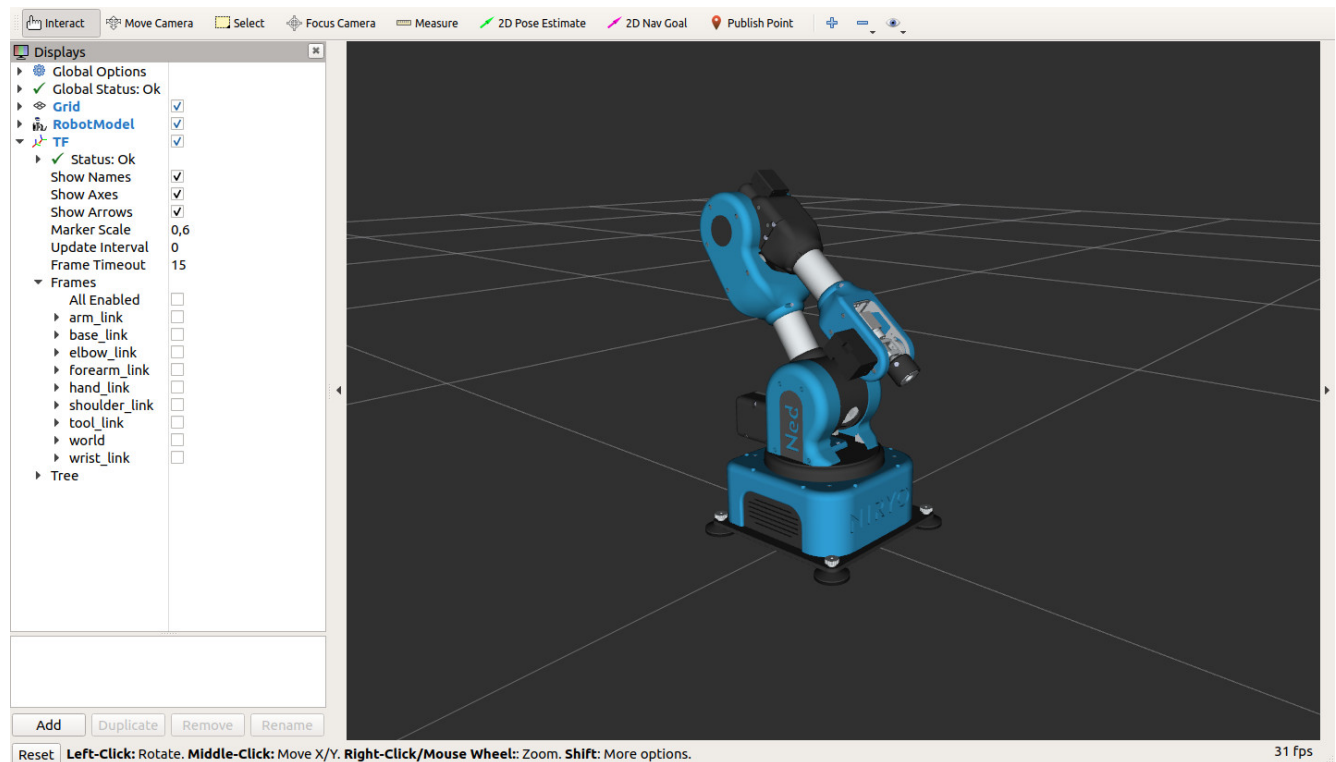
Exemple d'utilisation des trackbars.

Contrôler via ROS

Non seulement [RViz](#) peut afficher le robot, mais il peut également être lié aux contrôleurs ROS pour afficher les actions du robot à partir des commandes ROS ! Cette méthode peut aider à debugguer les topics et services ROS ainsi que les scripts des API.

Pour l'exécuter :

```
roslaunch niryo_robot_bringup desktop_rviz_simulation.launch
```



L'ouverture de Rviz, avec le robot prêt à être contrôlé avec ROS !

Option de Visualisation RViz

Table des options de lancement de RViz

Nom	Valeur par défaut	Description
log_level	INFO	Niveau de log à afficher pour les loggers ROS
hardware_version	ned	Utilise les paramètres dédiés à cette version matérielle. Les valeurs acceptées sont « one », « ned » et « ned2 »
debug	false	Lance la stack en mode debug. Pour le développement et le debuggauge seulement.
gui	true	Active la visualisation de l'interface graphique
conf_location	version.txt	Localisation du fichier version.txt. Un chemin vers ce fichier est requis.
simu_gripper	false	Simule la présence d'un outils d'id 11 sur le bus (l'outil ne sera cependant pas visible dans la visualisation, quelque soit la valeur de ce paramètre)
simu_conveyor	false	Simule la présence d'un convoyeur (le convoyeur ne sera cependant pas visible dans la simulation, quelque soit la valeur de ce paramètre)

Sans physique - Simulation Gazebo

Pour la simulation, Ned utilise Gazebo, un outil bien connu de la communauté ROS. Il permet :

- La collision.
- La création d'un monde → Un environnement virtuel dans lequel le robot peut manipuler des objets.
- L'utilisation de la caméra et des Grippers.

Le Gripper Custom a été modélisé dans Gazebo. La caméra est également implémentée.

Note

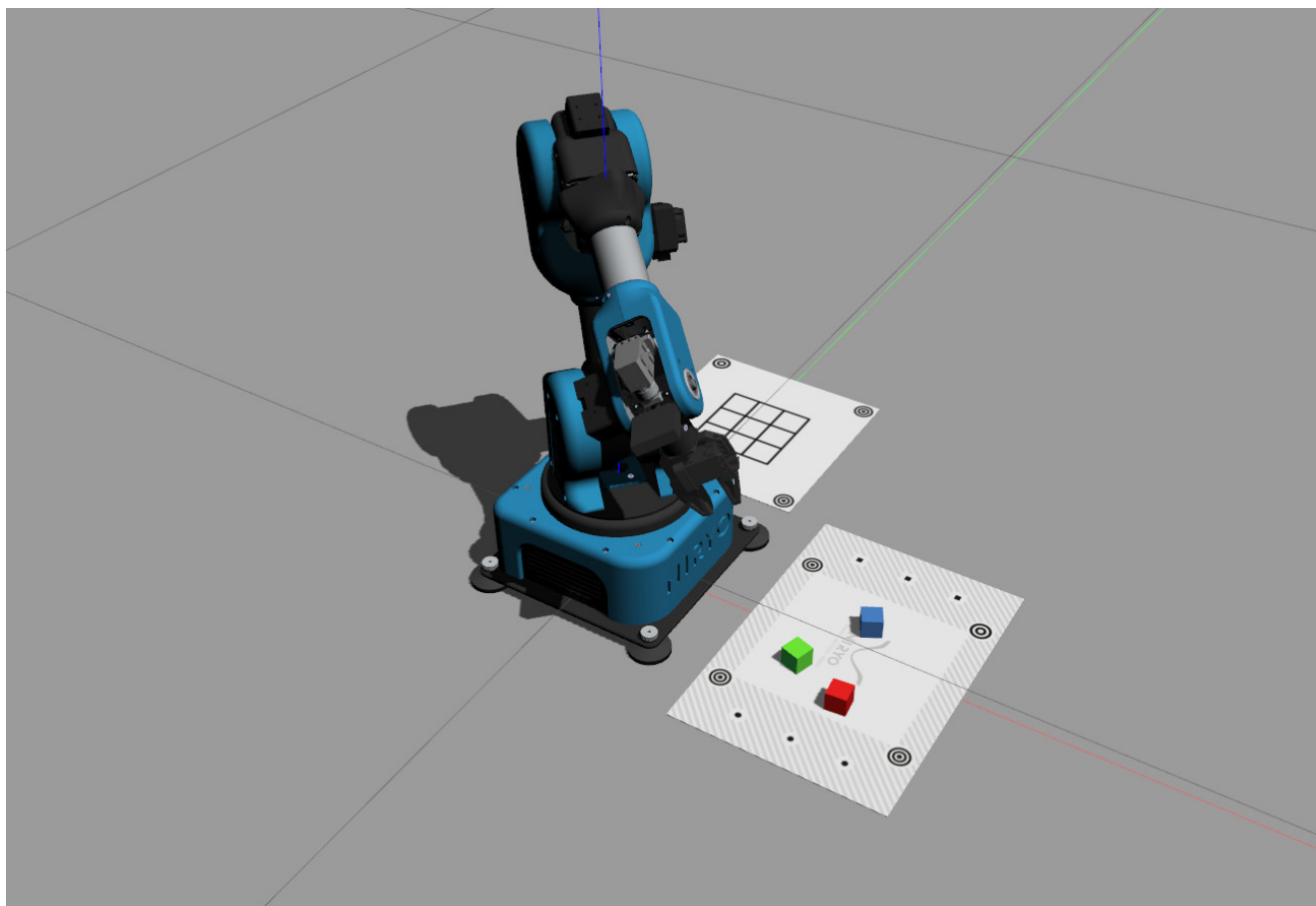
Gazebo génère également une distorsion de la caméra, ce qui rapproche encore plus la simulation de la réalité !

Lancer une simulation Gazebo

Un monde spécifique a été créé pour utiliser Ned dans Gazebo avec 2 espaces de travail.

Pour l'exécuter :

```
roslaunch niryo_robot_bringup desktop_gazebo_simulation.launch
```



Aperçu de Gazebo avec le robot prêt à être contrôlé avec ROS !

Note

Il est possible de modifier le monde Gazebo pour faire le vôtre ! Il est situé dans le dossier **mondes** du package **niryo_robot_gazebo**.

Option de simulation Gazebo

L'utilisateur peut désactiver 3 choses en ajoutant la chaîne spécifique à la ligne de commande :

- L'interface graphique Gazebo : `gui:=false`.
- La caméra et le Gripper - Les fonctions de vision et le Gripper ne seront pas utilisables : `gripper_n_camera:=false`.

❗ Indication

Gazebo peut être très lent. Si vos tests ne nécessitent pas de Gripper ou de caméra, Il faut envisager d'utiliser Rviz pour alléger votre CPU.

Table des options de lancement de Gazebo

Nom	Valeur par défaut	Description
log_level	INFO	Niveau de log à afficher pour les loggers ROS
hardware_version	ned	Utilise les paramètres dédiés à cette version matérielle. Les valeurs acceptées sont « one », « ned » et « ned2 »
debug	false	Lance la stack en mode debug. Pour le développement et le debuggauge seulement.
gui	true	Active la visualisation de l'interface graphique
conf_location	version.txt	Localisation du fichier version.txt. Un chemin vers ce fichier est requis.
gripper_n_camera	true	Simule la présence d'un outil d'id 11 sur le bus ainsi que de la caméra
simu_conveyor	true	Simule la présence d'un convoyeur (le convoyeur ne sera cependant pas visible dans la simulation, quelque soit la valeur de ce paramètre)

Démarrage rapide

Bienvenue dans le démarrage rapide du robot. Ici, vous apprendrez les fonctionnalités essentielles du robot pour vous aider à démarrer.

Connexion au robot

Il existe 4 façons de connecter votre ordinateur au robot :

Hotspot

- **Type** : Wi-Fi
- **Difficulté** : Facile
- **Description** : Le robot émet son propre réseau Wi-Fi. Dans ce mode, vous pouvez vous connecter au robot comme n'importe quel autre réseau Wi-Fi. Le nom du réseau est au format **NiryoRobot xx-xx-xx** et le mot de passe par défaut est **niryorobot**. Pour changer le nom du robot, reportez-vous à la section : [Nom du robot](#).
- **Plus d'informations** : [Paramètres Wi-Fi](#), [Utiliser le robot en Hotspot](#).
- **Avantage** : Facile, aucun câble requis.
- **Inconvénient** : Connexion Ethernet nécessaire sur l'ordinateur pour avoir accès à Internet. Le robot n'a pas accès à Internet et ne peut pas se mettre à jour.
- **Adresse IP** : 10.10.10.10

Mode connecté

- **Type** : Wi-Fi
- **Difficulté** : Moyenne
- **Description** : Le robot est connecté à un réseau Wi-Fi existant.
- **Plus d'informations** : [Paramètres Wi-Fi](#), [Utiliser Ned sur votre réseau Wi-Fi](#).
- **Avantage** : Connexion Ethernet nécessaire sur l'ordinateur pour avoir accès à Internet. Le robot n'a pas accès à Internet et ne peut pas se mettre à jour.
- **Inconvénient** : Connexion Wi-Fi stable requise.
- **Adresse IP** : En fonction de votre réseau

Ethernet direct

- **Type** : Ethernet
- **Difficulté** : Moyenne
- **Description** : Le robot est connecté directement à l'ordinateur via un câble Ethernet.
- **Plus d'informations** : [Paramètres Ethernet](#), [Utiliser Ned avec un câble Ethernet](#).
- **Avantage** : L'ordinateur peut avoir accès à Internet via le wifi. Une communication plus sûre et plus robuste avec le robot.
- **Inconvénient** : Câble requis. Le robot n'a pas accès à Internet et ne peut pas se mettre à jour.
- **Adresse IP** : 169.254.200.200

Ethernet via le réseau

- **Type** : Ethernet
- **Difficulté** : Moyenne
- **Description** : Le robot est connecté au réseau via un câble Ethernet et l'ordinateur est connecté au réseau via un câble Ethernet ou via Wi-Fi.
- **Plus d'informations** : [Paramètres Ethernet](#).
- **Avantage** : Le robot et l'ordinateur peuvent avoir accès à Internet. Meilleure communication avec le robot.
- **Inconvénient** : Câble requis.
- **Adresse IP** : En fonction de votre réseau

Programmation de robots

Il existe 6 façons de programmer les robots Niryo :

Nom	Langue	Difficulté	Documentation	Description
Niryo Studio	Blockly	Débutant	Blockly	Programmation simplifiée par blocs. Programmez vos cas d'usages le plus rapidement possible.
PyNiryo	Python	Intermédiaire	PyNiryo	Programmez votre robot à distance via une API Python 2.7 et 3.X.
Python ROS wrapper	Python	Intermédiaire	Python ROS wrapper	Programmez et exécutez votre code Python directement dans le robot. Aucun logiciel ou configuration requis, à l'exception de Niryo Studio ou d'un terminal ssh.
ROS	Python, C++	Avancé	Niryo Ros	Programmez et exécutez votre nœud ROS directement sur le robot, ou à distance via ROS Multimachine.
MODBUS	Tous	Avancé	MODBUS	Les programmes peuvent communiquer via le réseau MODBUS avec les robots dans toutes les langues disponibles.
Serveur TCP	Tous	Avancé	Serveur TCP	Les programmes peuvent communiquer via le réseau TCP avec les robots dans toutes les langues disponibles.

Astuces Niryo One et Ned

Programmez votre premier mouvement en 30 secondes

Le moyen le plus rapide de programmer le robot est via [Blockly](https://docs.niryo.com/product/niryo-studio/source/blockly_api.html) (https://docs.niryo.com/product/niryo-studio/source/blockly_api.html). Lorsque vous êtes sur la page Blockly et connecté au robot, passez en mode d'apprentissage via le bouton toggle. Vous pouvez ensuite appuyer une fois sur le bouton au-dessus de la base du robot pour faire apparaître un bloc avec la position actuelle du robot. Ainsi, déplacez votre robot à la main, appuyez sur le bouton, reliez les blocs. Félicitations, vous avez programmé un robot à la vitesse de l'éclair !

En haut à droite de la fenêtre Blockly, vous pouvez choisir d'enregistrer les positions en mode **Joints** ou **Pose**.

Articulations & Poses, quelle est la différence ?

Les **joints** sont les différentes articulations du robot. En mode **joints**, vous donnez au robot une commande sur chacun des moteurs du robot. L'unité par défaut utilisée est le radian. 6,28318530718 radian vaut 2π et correspond à 360°. Sur Niryo Studio vous pouvez passer aux degrés pour plus de simplicité.

La **Pose** correspond aux coordonnées x, y, z et à l'orientation roulis, tangage, lacet (respectivement, la rotation autour des axes x, y, z) de l'extrémité du robot. L'axe x est dirigé vers l'avant du robot et l'axe y vers la gauche du robot. Une coordonnée x positive fera avancer le robot. Une coordonnée y positive déplacera le robot vers la gauche et une coordonnée y négative déplacera le robot vers la droite.

Parfois, il peut y avoir plusieurs configurations d'axes du robot qui correspondent aux mêmes coordonnées. C'est pourquoi il est recommandé d'utiliser les commandes **Joints**. La **Pose** est cependant plus facile et plus intuitive à utiliser pour demander au robot d'aller par exemple 10cm plus haut, ou 10 vers la droite.

Utiliser un outil

Pour utiliser un outil, pensez à utiliser la fonction **scan** pour détecter l'outil connecté. Vous pouvez ensuite utiliser les grippers, la Pompe à vide ou l'Electroaimant à votre guise.

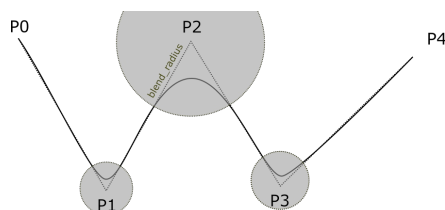
Pensez à ajouter la fonction **scan** au début de chacun de vos programmes pour éviter toute surprise.

Nos différents outils sont intelligents, ainsi le robot pourra adapter ses mouvements en fonction de l'outil sélectionné pour *unpick and place* avec vision. De plus, vous pouvez programmer vos mouvements avec **Pose**. En activant la fonction TCP ([de l'anglais Tool Center Point, Point Central de l'Outil en Français](https://docs.niryo.com/product/niryo-studio/source/settings.html#robot-tcp-tool-center-point)) (<https://docs.niryo.com/product/niryo-studio/source/settings.html#robot-tcp-tool-center-point>), le TCP du robot, et donc les mouvements, s'adapteront à l'outil équipé.

Déplacements standards, linéaires, points de passage, quelle est la différence ?

Il existe de nombreux types de mouvements différents possibles pour les bras de robot. Les 3 plus utilisés sont les suivants :

- **Mouvements standards** : Aussi appelé PTP (Point To Point). C'est le mouvement le plus simple. Dans ce type de mouvement, la durée du mouvement est minimisée, chaque articulation atteint la position finale en même temps. Le robot dessine une sorte d'arc de cercle selon les positions initiale et finale.
- **Mouvements linéaires** : Le robot trace une ligne droite entre la position de départ et la position finale. Cependant, un mouvement linéaire n'est pas toujours possible entre deux points en fonction des contraintes du robot. Veillez donc bien faire attention à ce que le mouvement soit réalisable. Dans le cas contraire le robot retournera une erreur.
- **Mouvements lissés par points de passage** : C'est ici que nous demandons au robot d'effectuer un mouvement jusqu'à un point final en passant par des points intermédiaires. Le robot dessine des trajectoires linéaires, ou PTP si le mouvement linéaire n'est pas possible, entre chaque point de passage sans s'arrêter. Il est également possible d'enregistrer le rayon de courbure pour lisser le mouvement et dessiner des courbes entre les points. Ce mouvement est idéal pour éviter les obstacles.



Trajectoire par points de passage avec rayon de fusion (https://ros-planning.github.io/moveit_tutorials/doc/pilz_industrial_motion_planner/pilz_industrial_motion_planner.html#user-interface-sequence-capability)

Démarrer, Pause, Annuler l'exécution d'un programme

Vous ne le saviez peut-être pas, mais le bouton situé en haut de la base du robot permet également de démarrer, de mettre en pause et d'arrêter un programme.

Lorsqu'un programme est en cours d'exécution :

- 1 appui met le programme en pause
- 2 appuis mettront le programme en pause et activeront le mode apprentissage

Lorsqu'un programme est en pause :

- 1 appui reprend le programme
- 2 appuis arrête le programme
- S'il n'y a pas d'intervention pendant 30 secondes, le programme s'arrête automatiquement

Lorsque le programme est en pause, la LED à l'arrière clignote en blanc.

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

Guide de démarrage

La stack ROS Niryo peut être installée sur plusieurs Système d'Exploitation cibles :

- Raspberry Pi 3 (déprécié, n'est plus supporté)
- Raspberry Pi 4
- Ordinateur de bureau

Comme la stack est basée sur ROS Melodic ou Kinetic (déprécié), vous devez être sur un système basé sur Ubuntu. Nous supportons actuellement les version suivantes :

Nous supportons actuellement les versions suivantes:

- Ubuntu 18.04 [Installer sous Ubuntu 18](#)
- Windows Subsystem pour Linux 2 (WSL 2) - Ubuntu 18.04 [Installation pour le Sous-système Windows pour Linux \(expérimental\)](#)

Installer sous Ubuntu 18

Ce guide va dérouler les étapes nécessaires pour installer la stack des robots Niryo sous Ubuntu 18. Vous pouvez appliquer ces étapes pour mettre en place un environnement de simulation sur un ordinateur de développement quelconque, ou bien l'installer sur une Raspberry Pi en vu de l'utiliser sur un robot réel.

Sommaire :

- [Prérequis](#)
- [Installer les dépendances de ROS](#)
- [Mise en place de l'environnement ROS](#)

Prérequis

La stack Niryo ROS tourne sur ROS Melodic ou Kinetic (déprécié). Cette version de ROS est fortement dépendante de la version 18.04 de Ubuntu, ainsi cet OS est aujourd'hui le seul système d'exploitation supporté.

Vérifiez que vous avez un système à jour avant de continuer

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

Paquets Ubuntu

La stack Niryo ROS nécessite les paquets suivants pour fonctionner correctement :

- build-essential
- sqlite3
- ffmpeg

Note

Ces paquets sont principalement utiles sur un robot réel, mais comme le code est identique entre la simulation et le robot physique, un paquet manquant peut amener à des instabilités de la simulation.

Environnement Python

L'environnement Python est installé avec le fichier requirements_ned2.txt

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

Note

ROS Melodic utilise python 2 en interne. Nous avons donc gardé cette version de python en interne pour être aligné avec. Vous aurez donc besoin d'utiliser pip de python 2 pour installer ces dépendances

Mise en place de ROS

Note

Toutes les commandes de terminal listées sont pour des utilisateurs Ubuntu.

Positionnez-vous dans un dossier de votre choix et créez un dossier **catkin_ws_niryo_ned** ainsi qu'un sous-dossier **src**:

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

Ensuite aller dans le dossier **catkin_ws_niryo_ned** et cloner le repository Ned dans le dossier **src**. Pour les prochaines commandes, vérifiez que vous êtes bien dans le dossier **catkin_ws_niryo_ned** :

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

Installer les dépendances de ROS

Installer ROS

Vous devez tout d'abord installer ROS Melodic. Pour cela, suivez les instructions du tutoriel officiel de ROS [ici](http://wiki.ros.org/melodic/Installation/Ubuntu) (<http://wiki.ros.org/melodic/Installation/Ubuntu>) et choisissez l'option **Desktop-Full Install**.

Installetez les paquets additionnels

Pour assurer le fonctionnement de tous les paquets du Ned, vous devez installer quelques paquets supplémentaires :

Méthode 1 : Installation rapide via ROSDep

Pour chaque package, les dépendances requises sont listées dans leur fichier *package.xml* respectifs, ce qui permet l'installation de chacun d'entre eux via la commande *roscdep* :

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

Méthode 2 : Installation Complète

Les packages ROS requis sont :

- catkin
- python-catkin-pkg
- python-pymodbus
- python-roscdistro
- python-rospkg
- python-roscdep-modules
- python-rosinstall python-rosinstall-generator
- python-wstool

Pour installer un package sur Ubuntu :

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

Les packages melodic spécifiques requis sont :

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

Pour installer un package ROS Melodic sur Ubuntu :

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

Mise en place de l'environnement ROS

Note

Vérifier que vous êtes bien positionnez dans le dossier **catkin_ws_niryo_ned**.

Lancez ensuite la compilation de la stack ROS Niryo via la commande suivante :

```
catkin_make
```

Si aucune erreur ne se produit pendant la phase de compilation, la mise en place de votre environnement est presque complète !

Il est nécessaire de sourcer le fichier de configuration pour ajouter tous les packages du Ned dans l'environnement ROS. Pour faire cela, lancer la commande :

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

Il est nécessaire de lancer cette commande à chaque fois que vous lancez un nouveau terminal. Si vous voulez l'appliquer à chaque ouverture d'un terminal, vous pouvez ajouter cette ligne à votre fichier **.bashrc** :

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

L'installation est maintenant terminée !

Installation pour le Sous-système Windows pour Linux (expérimental)

Microsoft a développé depuis 2016 une couche de compatibilité pour faire tourner les executables Linux nativement sous Windows 10. Avec la version 2 sortie en 2019, ce « coeur Linux caché » est maintenant assez mûr pour faire faire tourner des programmes aussi complexes que la stack ROS complète ^[2].

Ainsi vous pourrez lancer des simulations pour les robots Ned, le Niryo One et Ned 2 sur une machine Windows.

Note

Il vous faut à minima Windows 10 version 2004 (Build 19041) ou plus pour faire tourner WSL2.

Avertissement

L'installation sous WSL n'étant pas officiellement supportée par Niryo, ce guide n'est fourni qu'à titre d'information seulement.

Ce guide est globalement adapté de ce poste de blog écrit par Jack Kawell, soyez libres de vous y référer pour de plus amples informations^[1]

Installer WSL2 ^[3]

1. Activer le Sous-système Linux pour Windows (WSL) sur votre machine (dans un terminal powershell)

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

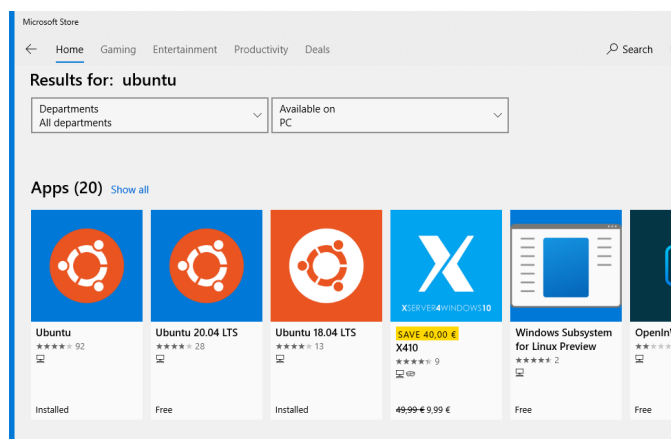
2. Mettre la version de WSL par défaut à 2 (dans un terminal powershell)

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

3. Vous devez ensuite redémarrer votre ordinateur pour terminer l'installation de WSL et la mise à jour vers WSL2.
4. Mettre la version de WSL par défaut à 2 (dans un terminal powershell)

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

5. Installer une distribution Ubuntu 18.04 via le Windows Store



Ubuntu 18.04 sur le Windows Store

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

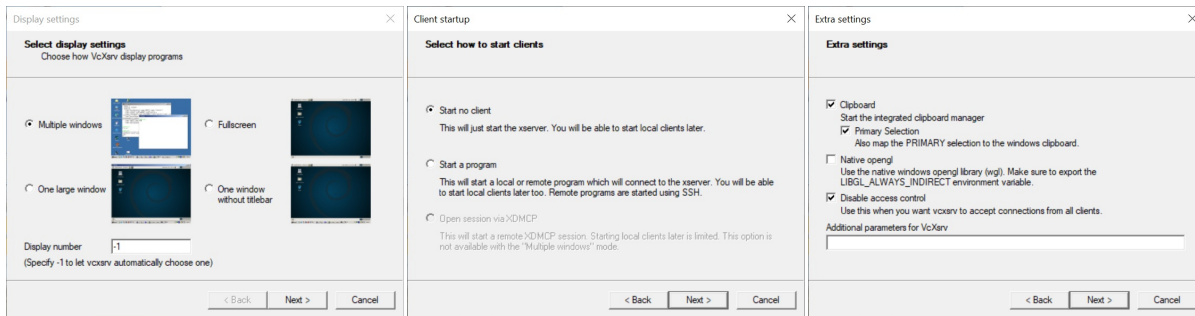
Your Ubuntu OS is now ready. You can continue the build of the stack using the tutorial.

Mise en place de la redirection d'Interface Graphique

WSL n'est pas fourni avec un serveur X. Vous ne pourrez donc pas lancer d'application graphique dans des fenêtres pour le moment. Mais il est possible d'y remédier en utilisant un serveur X dédié pour Windows et d'y rediriger les sorties graphiques via la redirection d'interface graphique (GUI forwarding).

Beaucoup de serveurs X existent pour Windows 10. Nous avons testé VcXsrv en interne, et il fait correctement le job. <https://sourceforge.net/projects/vcxsrv/> (<https://sourceforge.net/projects/vcxsrv/>)

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



Note

Vous pouvez directement utiliser cette configuration en utilisant ce [fichier](#) de configuration (`_downloads/818503a538e687731e85c64365865076/wsl_config.xlaunch`)

2. Vous avez besoin d'exporter l'adresse de votre serveur X dans Ubuntu 18 pour rediriger l'interface graphique

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

Vous pouvez ajouter cette ligne à votre fichier `bashrc`

3. Vous pouvez vérifier que la redirection fonctionne en lançant une application X11 simple par exemple :

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

4. Installer ROS melodic (voir les instructions [ici](#))

5. Essayer de lancer rviz

```
roscore & rosrn rviz rviz
```

6. Vous devriez maintenant être capable de lancer n'importe quelle simulation du One, du Ned ou du Ned2 en utilisant rviz ou gazebo

Dépannage

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

Essayer :

- Vérifiez que vous avez correctement lancé le serveur X avec les options nécessaires (Désactiver le contrôle d'accès est nécessaire)
- Vérifier que l'adresse IP que vous fournissez est correcte (vous avez besoin de l'adresse dans `/etc/resolv.conf` pour que cela fonctionne)

Problème d'OpenGL Quelques personnes ont indiqué avec eu des souci avec les applications OpenGL comme Rviz. Si c'est le cas pour vous, essayez de définir la variable d'environnement `LIBGL_ALWAYS_INDIRECT=0` dans un terminal WSL2 (vous pouvez par exemple ajouter `LIBGL_ALWAYS_INDIRECT=0` à la fin de votre fichier `.bashrc`).

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

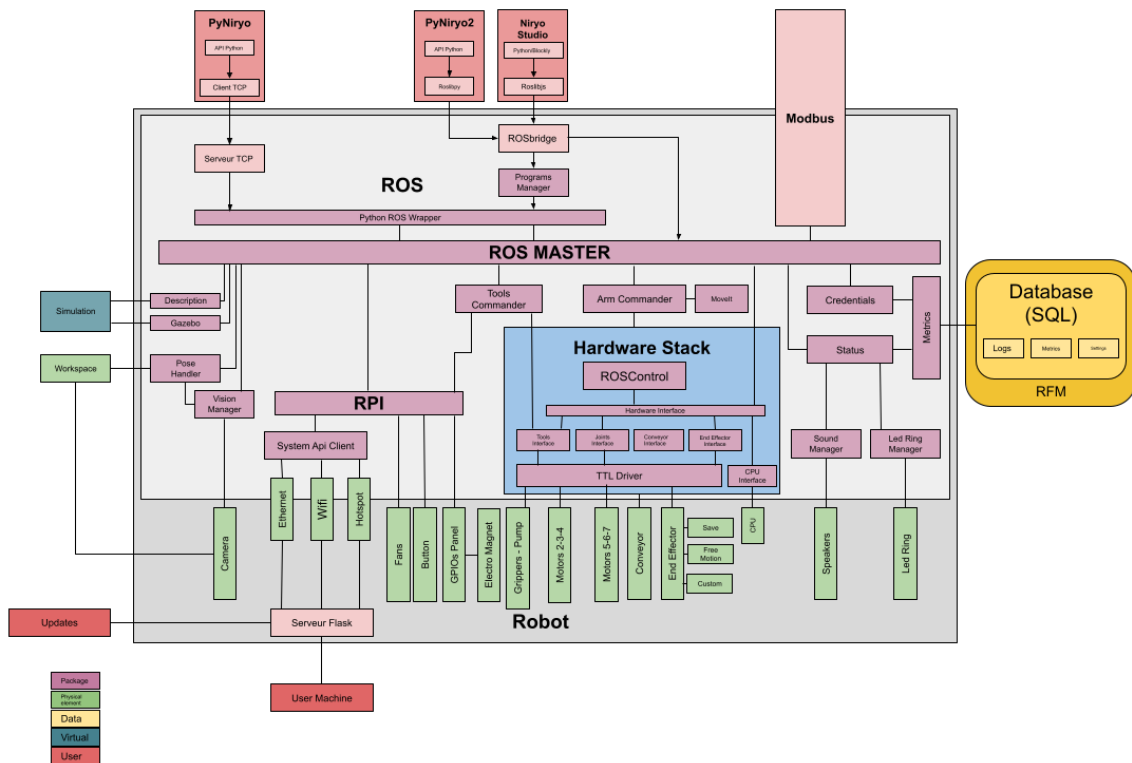
[2] <https://docs.microsoft.com/en-us/windows/wsl/compare-versions>

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

Vue d'ensemble

Le Ned est un robot basé sur Raspberry Pi, Arduino et ROS. Il utilise ROS pour faire l'interface entre le matériel et les fonctions de haut niveau.

Sur la figure suivante, vous trouverez une vue globale de l'architecture logicielle du robot de Niryo, afin de mieux comprendre où sont définies chaque composants logiciels.



Niryo robot Logiciel V3



ROS (Robot Operating System) est une infrastructure (Framework) robotique Open Source permettant de faciliter le développement logiciel des robots. Elle est utilisée dans presque tous les composants logiciels du Ned.

La stack est divisée en deux parties :

- La partie **Paquets Haut Niveau** (motion planner, vision, ...), développée en Python pour une meilleure lisibilité
- La partie **Paquets Bas Niveau** (drivers, gestion matérielle, ...), développée en C++ pour assurer des performances temps réelles.

Note

Pour en savoir plus sur ROS, vous pouvez lire le [Wiki Officiel ROS](http://wiki.ros.org/) (<http://wiki.ros.org/>).

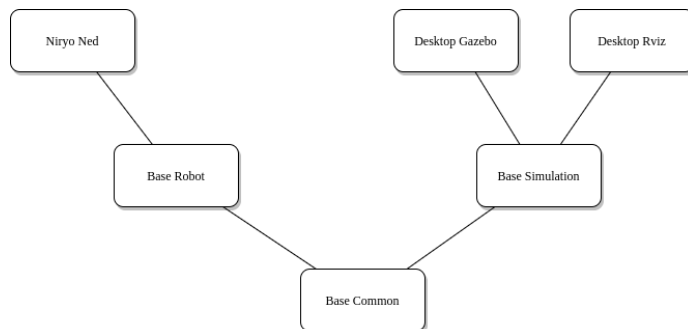
Paquets Haut Niveau

Dans cette section, vous aurez accès aux informations concernant chaque paquet ROS des robots Niryo dédiés aux interfaces Haut Niveau.

Niryo_robot_bringup

Ce package fournit des fichiers de configuration et de lancement pour démarrer les packages Ned et ROS avec divers paramètres.

Les fichiers de lancement sont placés dans le dossier de *lancement*. Seuls les fichiers avec **.lancement** peuvent être exécutés.



Organisation des fichiers de lancement Bring Up

Sur RaspberryPI

One

Le fichier **niryo_one_robot.launch** permet de lancer ROS sur une Raspberry Pi 3. Ce fichier est automatiquement lancé au démarrage du Niryo One (image Niryo One RPi3B).

Commande pour lancer la stack ROS du Niryo One:

```
roslaunch niryo_robot_bringup niryo_none_robot.launch
```

Ned

Le fichier **niryo_ned_robot.launch** permet de lancer ROS sur une Raspberry Pi 4.
Ce fichier est automatiquement lancé au démarrage de Ned (image Ned RPi4B).

Commande pour lancer la stack ROS de Ned :

```
roslaunch niryo_robot_bringup niryo_ned_robot.launch
```

Ned2

Le fichier **niryo_ned2_robot.launch** permet de lancer ROS sur une Raspberry Pi 4.
Ce fichier est automatiquement lancé au démarrage de Ned2 (image Ned2 RPi4B).

Commande pour lancer la stack ROS de Ned2 :

```
roslaunch niryo_robot_bringup niryo_ned2_robot.launch
```

Sur le bureau (Simulation)

Comme la simulation se produit sur un ordinateur, les éléments liés au matériel ne sont pas utilisés.

Pour les deux fichiers de lancement suivants, vous pouvez définir :

- *gui* à « false » pour désactiver l'interface graphique.

La simulation via Gazebo

Exécuter la simulation Gazebo. Le robot peut faire tout ce qui n'est pas lié au matériel :

- `move`, `get_pose`.
- Utiliser la caméra (pour la désactiver, régler le paramètre « caméra » sur “faux”).
- Utilisez le Gripper Custom (pour le désactiver, définissez le paramètre « `simu_gripper` » sur “faux”).
- Enregistrer/exécuter des programmes, accéder à la pose enregistrée, ...

La commande pour lancer la simulation est :

```
roslaunch niryo_robot_bringup desktop_gazebo_simulation.launch
```

La commande pour désactiver la caméra et le Gripper est :

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

La commande pour lancer la simulation est :

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

La simulation via Rviz

Exécuter la simulation Rviz. Il est possible d'accéder aux mêmes fonctionnalités que Gazebo, à l'exception de la caméra et du Gripper Custom.

La commande pour lancer la simulation est :

```
roslaunch niryo_robot_bringup desktop_rviz_simulation.launch
```

La commande pour lancer la simulation est :

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

Notes - Ned Bringup

Les fichiers *niryo_robot_base* configurent de nombreux rosparms, ces fichiers doivent être lancés avant tout autre package.

Les fichiers suivants sont utilisés pour configurés les logs des robots:

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

Niryo_robot_arm_commander

Ce package est celui qui traite tous les éléments liés aux commandes.

Il est composé d'un seul nœud, qui exécute séparément les commandes du bras et les commandes de l'outil.

Node Commander

Le nœud ROS est conçu pour interagir avec :

- Le bras via MoveIt!
- Les outils à travers le "tool_controller".

Toutes les commandes sont d'abord reçues sur le serveur actionlib qui :

- Gère les demandes simultanées.
- Vérifie si la commande ne peut pas être traitée en raison d'autres facteurs (ex: mode d'apprentissage).
- Valide les paramètres.
- Appelle les contrôleurs requis et renvoie le statut et le message appropriés.

Il appartient au namespace ROS : `/niryo_robot_arm_commander/`.

Paramètres - Commandes

Paramètres des commandes

Nom	Description
<code>reference_frame</code>	Cadre de référence utilisé par MoveIt! pour moveP. Par défaut : "monde"
<code>move_group_commander_name</code>	Nom du groupe contrôlé par MoveIt. Par défaut : « bras »
<code>jog_timer_rate_sec</code>	Taux de diffusion pour le contrôleur du Jog
<code>simu_gripper</code>	Si vous êtes en train d'utiliser le Gripper simulé et souhaitez contrôler le Gripper

Serveur d'action - Commandes

Package des commandes du serveur d'action

Nom	Type de message	Description
<code>robot_action</code>	<code>RobotMove</code>	Commandez le bras et les outils via un serveur d'action

Services - Commandes

Package des commandes des services

Nom	Type de message	Description
<code>is_active</code>	<code>GetBool</code>	Indiquer si une commande est en cours d'exécution ou non
<code>stop_command</code>	<code>Trigger</code>	Arrêter la commande en cours
<code>set_max_velocity_scaling_factor</code>	<code>SetInt</code>	Définir un pourcentage de vitesse maximale
<code>/niryo_robot/kinematics/forward</code>	<code>GetFK</code>	Calculer une cinématique avant
<code>/niryo_robot/kinematics/inverse</code>	<code>GetIK</code>	Calculer une cinématique inverse

Messages - Commandes

Package des commandes des messages

Nom	Description
<code>ArmMoveCommand</code>	Message pour commander le bras
<code>ShiftPose</code>	Message pour changer de pose
<code>PausePlanExecution</code>	Suspendre l'exécution du mouvement

Tous ces services sont disponibles dès le démarrage du nœud.

Dépendances - Commandes

- `actionlib`
- `actionlib_msgs`
- `control_msgs`
- `geometry_msgs`
- `MoveIt!`
- `moveit_msgs`
- `Niryo_robot_msgs`
- Package Niryo robot tools commander
- `python-numpy`
- `ros_controllers`
- `rosbridge_server`
- `sensor_msgs`

- [std_msgs](#)
- [tf2_web_republisher](#)
- [trajectory_msgs](#)

Fichier du Serveur d'action - Commandes

RobotMove

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

Fichiers des Services - Commandes

GetFK

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

GetIK

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

JogShift

```
int32 JOINTS_SHIFT = 1
int32 POSE_SHIFT = 2

int32 cmd

float32[] shift_values

---
int32 status
string message
```

Fichier des Messages - Commandes

ArmMoveCommand

```
int32 JOINTS = 0           # uses joints
int32 POSE = 1             # uses position and rpy
int32 POSITION = 2          # uses position
int32 RPY = 3              # uses rpy
int32 POSE_QUAT = 4        # uses position and orientation
int32 LINEAR_POSE = 5      # uses position and rpy
int32 SHIFT_POSE = 6       # uses shift
int32 SHIFT_LINEAR_POSE = 7 # uses shift
int32 EXECUTE_TRAJ = 8     # uses dist_smoothing, list_poses
int32 DRAW_SPIRAL = 9
int32 DRAW_CIRCLE = 10
int32 EXECUTE_FULL_TRAJ = 11
int32 EXECUTE_RAW_TRAJ = 12

int32 cmd_type

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

geometry_msgs/Pose[] list_poses
float32 dist_smoothing

trajectory_msgs/JointTrajectory trajectory

float64[] args
```

PausePlanExecution

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

float64 PAUSE_TIMEOUT = 30.0

int8 state
```

ShiftPose

```
int32 axis_number
float64 value
```

Niryo_robot_description

Ce package contient des fichiers URDF et des maillages (collada + stl) pour Ned.

Pour afficher Ned sur Rviz :

```
roslaunch niryo_robot_description display.launch
```

Pour afficher Ned sur Rviz :

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

Note : la visualisation 3D n'est pas disponible sur l'image Ned Raspberry Pi4. Pour l'utilisation des commandes suivantes, il est nécessaire de configurer la stack ROS pour Ned sur votre ordinateur.

Niryo_robot_gazebo



Usage

Ce package contient des modèles, des matériaux et des mondes Gazebo.

Lors du lancement de la version Gazebo de la stack ROS, le fichier **niryo_robot_gazebo_world.launch.xml** sera appelé pour générer le monde Gazebo.

Créer son propre monde

Pour créer le fichier de votre monde et il faut le placer dans le dossier **mondes**. Ensuite, il va falloir changer le paramètre **world_name** dans le fichier **niryo_robot_gazebo_world.launch.xml**.

Vous pouvez jeter un œil au monde Gazebo en le lançant sans robot en précisant le nom du monde dans l'arg *world_name* :

```
roslaunch niryo_robot_gazebo niryo_gazebo_world.launch world_name:=niryo_cube_world hardware_version:=ned # one, ned2
```

Niryo_robot_msgs

Ce package contient des messages standards qui peuvent être utilisés par tous les autres packages.

Les messages Niryo

Les messages Ned

Nom	Description
État du bus TTL	État du bus TTL
CommandStatus	Code de statut sous forme d'énumération
ObjectPose	x, y, z, roulis (roll), tangage (pitch), lacet (yaw)
RobotState	position, rpy, quaternion
RPY	roulis (roll), tangage (pitch), lacet (yaw)
HardwareStatus	Plusieurs informations sur le matériel
SoftwareVersion	Plusieurs versions de logiciel

Les services Niryo

Les services Ned

Nom	Description
GetBool	Renvoie un booléen
GetInt	Renvoie un entier
GetNameDescriptionList	Retourne un list de noms et un liste de descriptions
GetString	Renvoie une chaîne de caractères
GetStringList	Renvoie une liste de chaîne de caractères

Nom	Description
Ping	Utilisé pour ping les API
SetBool	Définir une valeur booléenne et renvoyer le statut
SetFloat	Définir une valeur décimale et renvoyer le statut
SetInt	Définir un entier et renvoyer le statut
SetString	Définir une chaîne et renvoyer le statut
Trigger	Déclencher une tâche

Dépendances des messages Niryo

- geometry_msgs

Fichiers de messages Niryo

État du bus TTL

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

CommandStatus

```
int32 val

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

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

int32 BAD_HARDWARE_VERSION = -10
int32 ROS_ERROR = -20

int32 FILE_ALREADY_EXISTS = -30
int32 FILE_NOT_FOUND = -31

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

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

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

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

int32 COLLISION = -226

int32 PAUSE_TIMEOUT= -227
int32 CANCEL_PAUSE= -228

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

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

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

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

```

int32 WORKSPACE_CREATION_FAILED = -308

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

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

# - Credentials
int32 CREDENTIALS_FILE_ERROR = -400
int32 CREDENTIALS_UNKNOWN_ERROR = -401

# - System Api Client
int32 SYSTEM_API_CLIENT_UNKNOWN_ERROR = -440
int32 SYSTEM_API_CLIENT_INVALID_ROBOT_NAME = -441
int32 SYSTEM_API_CLIENT_REQUEST_FAILED = -442
int32 SYSTEM_API_CLIENT_UNKNOWN_COMMAND = -443
int32 SYSTEM_API_CLIENT_COMMAND_FAILED = -444

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

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

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

# - I2C interface
int32 MISSING_I2C = -510

```

ObjectPose

```

float64 x
float64 y
float64 z

float64 roll
float64 pitch
float64 yaw

```

RobotState

```

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

geometry_msgs/Twist twist
float64 tcp_speed

```

RPY

```

# roll, pitch and yaw

float64 roll
float64 pitch
float64 yaw

```

HardwareStatus

```
std_msgs/Header header

# Raspberry Pi board
int32 rpi_temperature

# Ned, One, ....
string hardware_version

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

int8 hardware_state

# Motors
bool connection_up
string error_message
bool calibration_needed
bool calibration_in_progress

string[] motor_names
string[] motor_types

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

SoftwareVersion

```
string rpi_image_version
string ros_niryo_robot_version
string robot_version

string[] motor_names
string[] stepper_firmware_versions
```

Fichiers du service Niryo

GetBool

```
---
bool value
```

GetInt

```
---
int32 value
```

GetNameDescriptionList

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

GetString

```
---
string value
```

GetStringList

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

Ping

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

SetBool

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

SetFloat

```
float32 value
---
int32 status
string message
```

SetInt

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

SetString

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

Trigger

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

Niryo_robot_modbus

Niryo_robot_poses_handlers

Ce package est en charge du traitement des transformations, de l'espace de travail, des outils et des trajectoires.

Noeud Pose Handler

Description - Poses handlers

Le nœud ROS est composé de plusieurs services pour gérer les transformations, l'espace de travail, les outils et les trajectoires.

Il appartient au namespace ROS : `/niryo_robot_poses_handlers/`.

Workspaces (Espaces de travail)

Un espace de travail est défini par 4 marqueurs qui forment un rectangle. À l'aide de la pointe de calibration du robot, les positions sont enregistrées. La caméra renvoie des poses (x, y, yaw) par rapport à l'espace de travail. Nous pouvons alors déduire la pose absolue de l'objet en coordonnées du robot.

Outils

Lorsque nous connaissons la pose de l'objet en coordonnées du robot, nous ne pouvons pas envoyer directement cette pose au robot car nous spécifions la pose cible du lien `tool_link` et non du TCP réel (point central de l'outil). Ainsi nous introduisons la notion d'outil. Chaque effecteur terminal a son propre outil qui spécifie où placer le robot par rapport à l'objet. Actuellement, la notion d'outil ne fait pas partie de l'interface `python/tcp/blockly` car elle ajouterait une couche supplémentaire de complexité qui n'est pas vraiment nécessaire pour le moment. Par conséquent, nous avons un outil par défaut pour tous les outils qui est sélectionné automatiquement en fonction de l'ID de l'outil actuel. Cependant, tout est prêt si vous souhaitez définir des outils personnalisés, par exemple pour des mors personnalisés ou pour des positions d'outils personnalisés.

Actuellement, la notion de prise ne fait pas partie de l'interface `python / tcp / Blockly` car cela nécessiterait une couche de difficultés supplémentaire qui n'est, pour le moment, pas nécessaire.

Toutefois, nous proposons une prise par défaut pour tous les outils qui sont automatiquement en fonction de l'ID de l'outil. Toutefois, tout est prêt pour vous permettre de définir votre propre prise sur mesure, pour les outils personnalisés ou pour des position de saisie particulières.

La boucle de saisie par vision

1. La caméra détecte l'objet par rapport aux marqueurs et envoie x_{rel} , y_{rel} , yaw_{rel} .
2. L'objet est placé sur l'espace de travail, révélant la pose de l'objet en coordonnées du robot x, y, z, roll, pitch, yaw.
3. L'outil s'applique sur la position absolue de l'objet et donne la position vers laquelle le robot doit se déplacer.

Positions & trajectoires

Liste des positions

Paramètres - Poses handlers

Les paramètres poses handler

Nom	Description
<code>workspace_dir</code>	Chemin d'accès au dossier mère de stockage de l'espace de travail
<code>grip_dir</code>	Chemin vers le dossier mère de stockage de l'outil
<code>poses_dir</code>	Chemin vers le dossier mère de stockage des positions
<code>dynamic_frame_dir</code>	Chemin vers le dossier mère de stockage des repères dynamiques

Les services - Poses handlers

Les services poses handlers

Nom	Type du message	Description
<code>manage_workspace</code>	ManageWorkspace	Sauvegarder/Supprimer un espace de travail
<code>get_workspace_ratio</code>	GetWorkspaceRatio	Obtenir le ratio d'un espace de travail
<code>get_workspace_list</code>	GetNameDescriptionList	Obtenir le nom et la description de la liste des espaces de travail
<code>get_workspace_poses</code>	GetWorkspaceRobotPoses	Obtenir les positions de l'espace de travail du robot
<code>get_workspace_points</code>	GetWorkspacePoints	Obtenir les points de l'espace de travail du robot
<code>get_workspace_matrix_poses</code>	GetWorkspaceMatrixPoses	Obtenir les matrices de l'espace de travail du robot
<code>get_target_pose</code>	GetTargetPose	Obtenir le nom des programmes enregistrés
<code>manage_pose</code>	ManagePose	Sauvegarder/Supprimer une position
<code>get_pose</code>	GetPose	Obtenir une position
<code>get_pose_list</code>	GetNameDescriptionList	Obtenir la liste des noms et des descriptions des positions
<code>manage_dynamic_frame</code>	ManageDynamicFrame	Sauvegarde/Modifie/Supprime un repère dynamique
<code>get_dynamic_frame_list</code>	GetNameDescriptionList	Obtient la liste des repères dynamiques
<code>get_dynamic_frame</code>	GetDynamicFrame	Obtenir le repère dynamique
<code>get_transform_pose</code>	GetTransformPose	Obtient la transformation entre deux repères

Tous ces services sont disponibles dès le démarrage du nœud.

Dépendances - Poses handlers

- [geometry_msgs](#)
- [moveit_msgs](#)
- [Niryo_robot_msgs](#)
- [tf](#)

Fichiers de services et de messages - Poses handlers

GetDynamicFrame (Service)

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

GetPose (Service)

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

GetTargetPose (Service)

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

GetTransformPose (Service)

```
string source_frame
string local_frame

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

GetWorkspaceMatrixPoses (Service)

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

GetWorkspacePoints (Service)

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

GetWorkspaceRatio (Service)

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

GetWorkspaceRobotPoses (Service)

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

ManageDynamicFrame (Service)

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

int32 cmd

niryo_robot_poses_handlers/DynamicFrame dynamic_frame

---
int32 status
string message
```

ManagePose (Service)

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

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

ManageWorkspace (Service)

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

int32 cmd

niryo_robot_poses_handlers/Workspace workspace

---
int32 status
string message
```

NiryoPose (Message)

```
string name
string description

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

Workspace (Message)

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

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

Niryo_robot_programs_manager

Ce package est en charge de l'interprétation/exécution/sauvegarde des programmes. Il est utilisé par Niryo Studio.

Noeud Programs manager

Le nœud ROS est composé de plusieurs services pour gérer le stockage et l'exécution des programmes.

Les appels ne sont pas disponibles à partir du Python ROS Wrapper, car il a été conçu pour exécuter ses programmes avec le Python ROS Wrapper.

Il appartient au namespace ROS : `/niryo_robot_programs_manager/` .

Paramètres - Programs manager

Paramètres de Programs manager

Nom	Description
<code>autorun_file_name</code>	Nom du fichier contenant les informations d'exécution automatique
<code>programs_dir</code>	Chemin d'accès au dossier mère de stockage du programme

Services - Programs manager

Services de Programs manager

Nom	Type de message	Description
<code>execute_program</code>	<code>ExecuteProgram</code>	Exécuter un programme
<code>execute_program_autorun</code>	<code>Trigger</code>	Exécuter le programme d'exécution automatique
<code>get_program</code>	<code>GetProgram</code>	Récupérer le programme enregistré
<code>get_program_autorun_infos</code>	<code>GetProgramAutorunInfos</code>	Définir les paramètres d'exécution automatique
<code>get_program_list</code>	<code>GetProgramList</code>	Obtenir le nom des programmes enregistrés
<code>manage_program</code>	<code>ManageProgram</code>	Enregistrer et supprimer des programmes
<code>set_program_autorun</code>	<code>SetProgramAutorun</code>	Définir les paramètres d'exécution automatique
<code>stop_program</code>	<code>Trigger</code>	Arrêter le programme en cours d'exécution

Tous ces services sont disponibles dès le démarrage du nœud en mode autonome ou non.

Dépendances - Programs manager

- `Niryo_robot_msgs`
- `python-yaml`
- `std_msgs`

Services files - Programs manager

ExecuteProgram

```
bool execute_from_string

string name
string code_string

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

GetProgram »

```
string name

niryo_robot_programs_manager/ProgramLanguage language
---
int32 status
string message

string code
string description
```

GetProgramAutorunInfos

```
---
int32 status
string message

niryo_robot_programs_manager/ProgramLanguage language
string name

# Mode
int8 ONE_SHOT = 1
int8 LOOP = 2

int8 mode
```

GetProgramList

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

ManageProgram

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

# Program Name
string name

# - Creation
niryo_robot_programs_manager/ProgramLanguage language

string code
string description

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

SetProgramAutorun

```
# Program language
niryo_robot_programs_manager/ProgramLanguage language

# Program Name
string name

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

int8 mode

---
int16 status
string message
```

Fichiers services et messages - Programs manager

ProgramIsRunning

```
bool program_is_running

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

int8 last_execution_status
string last_execution_msg
```

ProgramLanguage

```
int8 NONE = -1

int8 ALL = 0

# Runnable
int8 PYTHON2 = 1
int8 PYTHON3 = 2

# Not Runnable
int8 BLOCKLY = 66

int8 used
```

ProgramLanguageList

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

ProgramList

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

Niryo_robot_rpi

Ce package traite des éléments liés à Raspberry Pi (Bouton, ventilateurs, E/S, LED,...).

Noeud Raspberry Pi (RPI)

Ce noeud ROS gère les composants suivants :

- Bouton supérieur physique : exécute une action quand un appui sur le bouton est réalisé.
- Panneau d'E/S numériques : reçoit les commandes et envoie l'état actuel des E/S numériques. Permet aussi de contrôler des outils tels que l'électroaimant.
- Panneau d'E/S analogiques : reçoit les commandes et envoie l'état actuel des E/S analogiques.
- Panneau d'E/S de du poignet du robot : reçoit les commandes et envoie l'état actuel des E/S numériques du Ned2. Permet aussi de contrôler des outils tels que l'électroaimant.
- Les ventilateurs du robot.

- LED : règle la couleur de la LED.
- Gestionnaire d'arrêt : arrête ou redémarre la Raspberry PI.
- ROS Log : permet de supprimer tous les états précédents au redémarrage pour éviter le manque d'espace disque à long terme (les cartes SD n'ont pas un stockage infini).

L'espace de nom utilisé est : `/niryo_robot_rpi/`

Noter que ce package ne doit pas être utilisé lorsque vous utilisez la stack ROS Ned sur votre ordinateur, en mode simulation. Il faut exécuter des actions lorsque le bouton est pressé.

Publisher - Raspberry Pi

Publishers du package RPI

Nom	Type du message	Description
<code>pause_state</code>	<code>PausePlanExecution</code>	Publier l'état d'exécution actuel lancé lorsque le bouton est enfoncé
<code>/niryo_robot/blockly/save_current_point</code>	<code>std_msgs/Int32</code>	Publier le point actuel lorsque l'utilisateur est sur Blockly pour enregistrer le bloc en appuyant sur le bouton
<code>/niryo_robot/rpi/is_button_pressed</code>	<code>std_msgs/Bool</code>	Publier l'état du bouton (vrai si appuyé)
<code>digital_io_state</code>	<code>DigitalIOState</code>	Publier l'état des E/S numériques en donnant pour chacun son pin / nom / mode / état
<code>analog_io_state</code>	<code>AnalogIOState</code>	Publier l'état des E/S analogiques en donnant pour chacun son pin / nom / mode / état
<code>/niryo_robot/rpi/led_state</code>	<code>std_msgs/Int8</code>	Publier la couleur actuelle de la LED
<code>ros_log_status</code>	<code>LogStatus</code>	Publier l'état actuel du journal (taille du journal / disque disponible / booléen si doit supprimer le journal ros au démarrage)

Services - Raspberry Pi

Services RPI

Nom	Type du message	Description
<code>shutdown_rpi</code>	<code>SetInt</code>	Arrêter le Raspberry Pi
<code>/niryo_robot/rpi/change_button_mode</code>	<code>SetInt</code>	Changer le mode du bouton supérieur (programme d'exécution automatique, blockly, rien,...)
<code>get_analog_io</code>	<code>GetAnalogIO</code>	Obtenir la liste des états d'E/S analogiques
<code>get_digital_io</code>	<code>GetDigitalIO</code>	Obtenir la liste des états d'E/S numériques
<code>set_analog_io</code>	<code>SetAnalogIO</code>	Régler une E/S analogique sur la valeur indiquée
<code>set_digital_io</code>	<code>SetDigitalIO</code>	Régler une E/S numérique sur la valeur indiquée
<code>set_digital_io_mode</code>	<code>SetDigitalIO</code>	Régler une E/S numérique sur le mode indiqué
<code>set_led_state</code>	<code>std_msgs/SetInt</code>	Définir l'état de la LED
<code>set_led_custom_blinker</code>	<code>LedBlinker</code>	Régler la LED en mode clignotant avec la couleur donnée
<code>purge_ros_logs</code>	<code>SetInt</code>	Purge des logs ROS
<code>set_purge_ros_log_on_startup</code>	<code>SetInt</code>	Modifier les paramètres permanents qui indiquent si le robot doit purger son ROS log à chaque démarrage

Dépendances - Raspberry Pi

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

Services - Raspberry Pi

ChangeMotorConfig (Service)

```
int32[] can_required_motor_id_list
int32[] dx1_required_motor_id_list
...
int32 status
string message
```

GetAnalogIO (Service)

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

float64 value
```

GetDigitalIO (Service)

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

bool value
```

LedBlinker (Service)

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

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

SetDigitalIO (Service)

```
string name
bool value

---
int32 status
string message
```

SetAnalogIO (Service)

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

SetIOMode (Service)

```
string name

int8 OUTPUT = 0
int8 INPUT = 1
int8 mode

---
int32 status
string message
```

SetPullup (Service)

```
string name
bool enable

---
int32 status
string message
```

Fichiers Messages - Raspberry Pi**AnalogIO**

```
string name
float64 value
```

AnalogIOState (Topic)

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

DigitalIO

```
string name
bool value
```

DigitalIOState (Topic)

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

LogStatus (Topic)

```
std_msgs/Header header

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

Niryo_robot_sound

Ce package contrôle du son du robot.

Nœud de son

Le noeud ROS est composé de services pour jouer, arrêter, importer et supprimer un son sur le robot. Il est également possible de modifier le volume du robot.

Il appartient à l'espace de noms ROS : `/niryo_robot_sound/`.

Paramètres - Son

Voici une liste des différents paramètres qui permettent d'ajuster les paramètres par défaut du robot et les sons du système.

Paramètres du volume sonore

Nom	Description	Valeur par défaut
<code>default_volume</code>	Volume par défaut sur le vrai robot	100
<code>default_volume_simulation</code>	Volume par défaut dans la simulation	10
<code>min_volume</code>	Volume minimum du robot	0
<code>max_volume</code>	Volume maximum du robot	200
<code>chemin_fichier_volume</code>	Fichier où est stocké le volume du robot réel défini par l'utilisateur	« ~/niryo_robot_saved_files/robot_sound_volume.txt »
<code>volume_file_path_simulation</code>	Fichier où est stocké le volume en simulation défini par l'utilisateur	« ~/.niryo/simulation/robot_sound_volume.txt »

Paramètres du son

Nom	Description	Valeur par défaut
<code>path_user_sound</code>	Volume par défaut sur le vrai robot	« ~/niryo_robot_saved_files/niryo_robot_user_sounds »
<code>path_user_sound_simulation</code>	Volume par défaut dans la simulation	« ~/.niryo/simulation/niryo_robot_user_sounds »
<code>path_robot_sound</code>	Volume minimum du robot	« niryo_robot_state_sounds »
<code>robot_sounds/error_sound</code>	Son joué lorsqu'une erreur se produit	error.2.wav
<code>robot_sounds/turn_on_sound</code>	Son joué au démarrage du robot	booting.wav
<code>robot_sounds/turn_off_sound</code>	Son joué à l'arrêt	stop.wav
<code>robot_sounds/connection_sound</code>	Son joué sur une connexion Niryo Studio	connected.wav
<code>robot_sounds/robot_ready_sound</code>	Son joué lorsque le robot est prêt	ready.wav
<code>robot_sounds/calibration_sound</code>	Son joué au début de l'étalonnage	calibration.wav

Etat du son

Etat	Description	Son
Démarrage	Son joué lorsque lors du démarrage	Your browser does not support the audio element.
Prêt	Son joué lorsque le robot est prêt après un démarrage	Your browser does not support the audio element.
Calibration	Son joué au début de l'étalonnage	Your browser does not support the audio element.
Connecté	Informe d'une connexion de Niryo Studio	Your browser does not support the audio element.
Redémarrage	Son joué au démarrage du redémarrage des robots	Your browser does not support the audio element.
Avertissement	Son joué lorsqu'un avertissement se produit	Your browser does not support the audio element.
Erreur	Son joué lorsqu'un une erreur robot/moteur/raspberry/programme/surchauffe se produit	Your browser does not support the audio element.
Arrêt	Son joué à l'arrêt	Your browser does not support the audio element.

Publisher - Son

Les Publisher du package Son

Nom	Message Type	Description
/niryo_robot_sound/sound	std_msgs/String	Publier le son en cours de lecture
/niryo_robot_sound/volume	std_msgs/UInt8	Publier le volume du robot
/niryo_robot_sound/sound_database	SoundList	Publier les sons (et leur durée) sur le robot

Services - Son

Services de Son

Nom	Message Type	Description
/niryo_robot_sound/play	PlaySound	Jouer un son de la base de données du robot
/niryo_robot_sound/stop	Trigger	Arrêter le son en cours de lecture
/niryo_robot_sound/set_volume	SetInt	Réglez le pourcentage de volume entre 0 et 200%
/niryo_robot_sound/text_to_speech	TextToSpeech	Prononcez une phrase via GTTS
/niryo_robot_sound/manage	ManageSound	Arrêter un son en cours de lecture

Abonnés - Son

Abonnés du package Son

Nom du Topic	Message type	Description
/niryo_robot_status/robot_status	RobotStatus	Récupère l'état actuel du robot et contrôle le son en conséquence Niryo_robot_status
/niryo_studio_connection	std_msgs/Empty	Attrapez la connexion Niryo Studio pour faire un son.

Dépendances - Son

- std_msgs
- niryo_robot_msgs
- niryo_robot_status

Fichiers Services & Messages - Son

SoundList (message)

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

SoundObject (Message)

```
string name
float64 duration
```

ManageSound (Service)

```
string sound_name

int8 ADD = 1
int8 DELETE = 2
int8 action

# Data to add a new sound
string sound_data

---
int32 status
string message
```

PlaySound (Service)

```
string sound_name

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

bool wait_end

---
int32 status
string message
```

TextToSpeech (Service)

```
string text # < 100 char

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

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

Fonctions API Son

Afin de contrôler le robot plus facilement que d'appeler chaque topic et service un par un, un Python ROS Wrapper a été conçu.

Par exemple, un script jouant du son via Python ROS Wrapper ressemble à :

```
from niryo_robot_led_ring.api import SoundRosWrapper

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

Cette classe permet de contrôler le son du robot via une API interne.

Liste des sous-sections de fonctions :

- [Jouer son](#)
- [Base de donnée de son](#)

Jouer son

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

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

Jouer un son du robot En cas d'échec, lève NiryoRosWrapperException

- Paramètres:**
- **sound_name** (*str*) – Nom du son à jouer
 - **start_time_sec** (*float*) – démarrer le son à partir de cette valeur en secondes
 - **end_time_sec** (*float*) – terminer le son à cette valeur en secondes
 - **wait_end** (*bool*) – attendre la fin du son avant de quitter la fonction

Renvoie: status, message

Type renvoyé: ([int](https://docs.python.org/3/library/functions.html#int) (<https://docs.python.org/3/library/functions.html#int>), [str](https://docs.python.org/3/library/stdtypes.html#str) (<https://docs.python.org/3/library/stdtypes.html#str>))

set_volume(sound_volume)

Définit le pourcentage de volume du robot. Si échec, soulève « NiryoRosWrapperException ».

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

Renvoie: status, message

Type renvoyé: ([int](#), [str](#))

stop()

Arrête un son en cours de lecture. En cas d'échec, soulève NiryoRosWrapperException »

Renvoie: status, message

Type renvoyé: ([int](#), [str](#))

say(text, language=0)

Utilisez gtts (Google Text To Speech) pour interpréter une chaîne de caractères comme un son. Les langues disponibles sont : - Anglais : 0 - Français : 1 - Espagnol : 2 - Mandarin : 3 - Portugais : 4

- Paramètres:**
- **text** (*string*) – du texte à la parole < 100 char
 - **language** (*int*) – langue du texte

Renvoie: status, message

Type renvoyé: ([int](https://docs.python.org/3/library/functions.html#int) (<https://docs.python.org/3/library/functions.html#int>), [str](https://docs.python.org/3/library/stdtypes.html#str) (<https://docs.python.org/3/library/stdtypes.html#str>))

Base de donnée de son

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

sounds

Obtenir la liste des noms de sons

Renvoie: liste des sons du robot

Type renvoyé: `list[string]`

delete_sound(sound_name)

Supprimer un son sur le RaspberryPi du robot. Si échec, soulève NiryoRosWrapperException

Paramètres: **sound_name** (*str*) – nom du son qui doit être supprimé

Renvoie: status, message

Type renvoyé: (`int`, *str*)

import_sound(sound_name, sound_data)

Supprimer un son sur le RaspberryPi du robot. Si échec, soulève NiryoRosWrapperException

Paramètres:

- sound_name** (*str*) – nom du son qui doit être supprimé
- sound_data** (*str*) – sChaine de caractère contenant la donné encodée du fichier son.

Renvoie: status, message

Type renvoyé: (`int` (<https://docs.python.org/3/library/functions.html#int>), *str* (<https://docs.python.org/3/library/stdtypes.html#str>))

get_sound_duration(sound_name)

Retourne la durée en secondes d'un son stocké dans le robot. Lance SoundRosWrapperException si le son n'existe pas.

Paramètres: **sound_name** (*str*) – Nom du son

Renvoie: durée du son en secondes

Type renvoyé: `float`

Niryo_robot_status

Nœud d'état du robot

Le nœud ROS est à l'écoute des topics du robot pour en déduire son état actuel. Il gère l'état du robot, l'état des logs et informe de la surchauffe du Raspberry Pi et du hors limite des articulations.

Il appartient à l'espace de noms ROS : `/niryo_robot_status/` .

Niryo Robot Status Table

Nom	Description	Dépannage
SHUTDOWN	Le robot est en procédure d'arrêt	
FATAL_ERROR	Crash de ROS	Veuillez redémarrer le robot
MOTOR_ERROR	Erreur de tension du moteur, surchauffe, surcharge	Vérifiez le code d'erreur sur Niryo Studio. Redémarrez le robot et vérifiez le câblage. Si le problème persiste, contactez le service client
COLLISION	Collision de bras détectée	Redémarrez votre mouvement ou passez en mode apprentissage pour supprimer cette erreur.
USER_PROGRAM_ERROR	Erreur de programme utilisateur	Lancez un mouvement ou passez en mode apprentissage pour supprimer cette erreur.
UNKNOWN	Nœud non initialisé	
BOOTING	ROS et la Raspberry sont en cours de démarrage.	Si le démarrage semble avoir raté, redémarrez le robot électriquement. Si le problème persiste, mettez à jour le robot en ssh, ou changez la carte SD, ou contactez le service client.
UPDATE	Mise à jour du robot en cours	Attendez et soyez patient :)
CALIBRATION_NEEDED	Nouvel étalonnage demandé	Exécutez un nouvel étalonnage avant d'effectuer tout mouvement.
CALIBRATION_IN_PROGRESS	Étalonnage en cours	Si l'étalonnage échoue ou prend plus de 30 secondes, le statut reviendra à CALIBRATION_NEED .
LEARNING_MODE	Mouvement libre activé, les couples sont désactivés	
STANDBY	Mouvement libre désactivé, les couples sont activés et aucun programme utilisateur n'est en cours d'exécution	
MOVING	Un mouvement unique ou un jog est en cours et aucun programme utilisateur n'est en cours d'exécution	

Nom	Description	Dépannage
<code>RUNNING_AUTONOMOUS</code>	Un programme utilisateur est en cours d'exécution et les couples sont activés	
<code>RUNNING_DEBUG</code>	Une procédure de débogage est en cours	Un appui court sur le bouton du haut l'annule.
<code>PAUSE</code>	Erreur de programme utilisateur	Un appui court sur le bouton du haut reprend le programme, un appui long (sur Ned2) ou un double appui (sur Ned et One) annule l'exécution du programme. Après 30 secondes, le programme s'arrête automatiquement.
<code>LEARNING_MODE_AUTONOMOUS</code>	Un programme utilisateur est en cours d'exécution et les couples sont désactivés	

Robot status chart

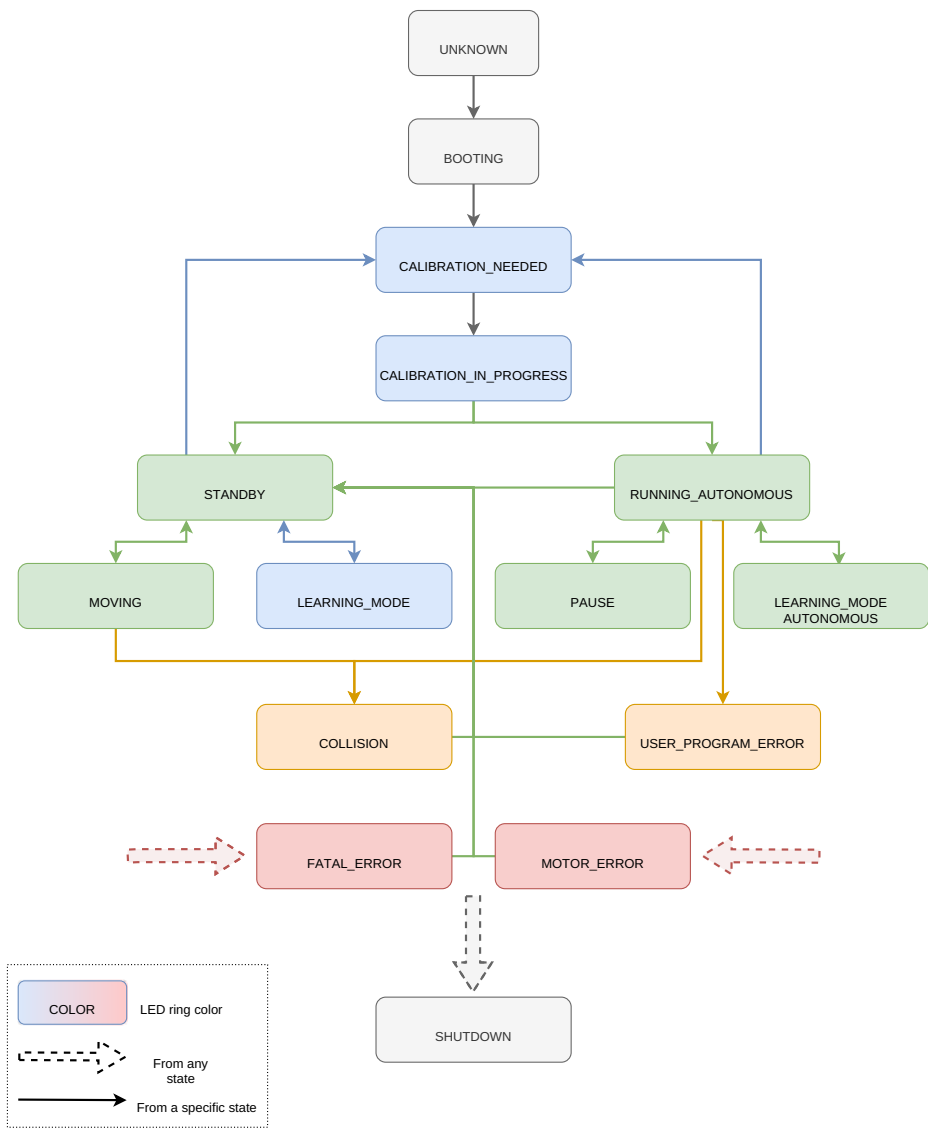


Diagramme d'état du robot Niryo

Publisher - Statut du robot

Publisher du package Robot Status

Nom	Message Type	Mode Latch	Description
<code>/niryo_robot_status/robot_status</code>	<code>RobotStatus</code>	<code>True</code>	Publier le statut du robot, des logs, de la surchauffe et du hors limites.

Services - Statut du robot

Services d'état du robot

Nom	Message Type	Description
/niryo_robot_status/advertising_shutdown	Trigger	Notifier une demande d'arrêt

Abonnés - Statut du robot

Abonnés au Robot Status Package

Nom du topic	Type de message	Description
/niryo_robot_hardware_interface/hardware_status	HardwareStatus	Détection d'un défaut moteur ou panneau effecteur, surchauffe de la raspberry
niryo_robot_rpi/pause_state	PausePlanExecution	Détection de l'état de pause
/niryo_robot_arm_commander/is_active	std_msgs/Bool	Détection d'un mouvement
/niryo_robot_arm_commander/is_debug_motor_active	std_msgs/Bool	Détection d'une procédure de débogage
/niryo_robot/jog_interface/is_enabled	std_msgs/Bool	Détection d'un mouvement de jog
/niryo_robot_programs_manager/program_is_running	ProgramIsRunning	Détection d'un programme utilisateur
/niryo_robot_user_interface/is_client_connected	std_msgs/Bool	Détection d'un utilisateur de pyniryo
/robot_niryo/learning_mode/state	std_msgs/Bool	Détection du mode mouvement libre
/niryo_robot_arm_commander/collision_detected	std_msgs/Bool	Détection d'une collision
/joint_states	sensor_msgs/JointState	Obtenir l'état du joint afin de détecter un hors limites
/ping_pyniryo	std_msgs/Bool	Détection d'un utilisateur pyniryo2

Dépendances - Statut du robot

- std_msgs
- sensor_msgs
- niryo_robot_msgs
- niryo_robot_programs_manager
- niryo_robot_arm_commander

Fichiers de message - Robot Status

RobotStatus

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

int8 robot_status
string robot_status_str
string robot_message

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

int8 logs_status
string logs_status_str
string logs_message

bool out_of_bounds
bool rpi_overheating
```

Niryo_robot_system_api_client

Ce package permet la communication avec le serveur flask pour gérer:

- Le nom du robot
- Les paramètres wifi
- Les paramètres ethernet

Publisher - System API Client

Publisher du package System API Client

Nom	Type de Message	Description
<code>/niryo_robot/wifi/status</code>	<code>WifiStatus</code>	Publier le status actuel du wifi

Services - System API Client

System API Client Services

Nom	Type de Message	Description
<code>/niryo_robot/wifi/set_robot_name</code>	<code>SetString</code>	Changer le nom du robot
<code>/niryo_robot/wifi/manage</code>	<code>ManageWifi</code>	Changer le mode du wifi
<code>/niryo_robot/ethernet/manage</code>	<code>ManageEthernet</code>	Changer les paramètres ethernet (adresse IP, masques réseaux, gateway, dhcp) basés sur l'interface nmcli.

Services files - System API Client

ManageEthernet (Service)

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

int8 profile

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

---
int32 status
string message
```

ManageWifi (Service)

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

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

Messages files - System API Client

WifiStatus (Message)

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

int8 status
```

Package Niryo robot tools commander

Ce package fournit des fonctionnalités pour contrôler les effecteurs et les accessoires pour Ned.

Ce package permet de contrôler le PCO du robot (Point Central de l'Outil, ou plus connu sous le nom de TCP pour Tool Center Point en anglais). Si cette fonctionnalité est activée, tous les mouvement (en coordonnées cartésiennes [x, y, z, roulis, tangage, lacet]) du robot seront effectuées en fonction du PCO. Un même programme peut donc fonctionner avec plusieurs outils différents en adaptant le TCP du robot à eux. Par défaut, cette fonctionnalité est désactivée, mais elle peut être activée via les services du noeud.

Package des commandes des outils du robot Niryo

Le nœud ROS est constitué de services pour équiper l'outil, d'un serveur d'action pour commander l'outil et de rubriques pour l'outil ou l'état de ce dernier.

Il appartient au namespace ROS : `/niryo_robot_tools_commander/`.

Serveur d'action - Outils

Serveur d'action du package d'outils

Nom	Type du message	Description
<code>action_server</code>	<code>ToolAction</code>	Commander l'outil via un serveur d'action

Diffuseur - outils

Diffuseurs de packages d'outils

Nom	Type du message	Description
<code>current_id</code>	<code>std_msgs/Int32</code>	Publier l'ID d'outil actuel
<code>tcp</code>	<code>TCP</code>	Publie l'état activé/désactivé du PCO (Point Central de l'Outil) ainsi que la transformation entre tool_link et le PCO

Services - outils

Services de package d'outils

Nom	Type du message	Description
<code>update_tool</code>	<code>std_srvs/Trigger</code>	Scanner pour voir si un moteur dynamixel est flashé avec un ID correspondant à un outil et l'équiper
<code>equip_electromagnet</code>	<code>SetInt</code>	Equiper l'électroaimant avec l'ID moteur donné comme paramètre
<code>enable_tcp</code>	<code>SetBool</code>	Activer ou désactiver le PCO (Point central de l'outil). Lorsqu'on l'active, la transformation appliquée au PCO sera la dernière enregistrée depuis le démarrage du robot. Celle par défaut, est celle de l'outil équipé.
<code>set_tcp</code>	<code>SetTCP</code>	Activer le PCO (Point Central de l'outil) et définir un nouveau PCO.
<code>reset_tcp</code>	<code>std_srvs/Trigger</code>	Réinitialiser la transformation du PCO avec celle de l'outil équipé.

Dépendances - outils

- [Niryo_robot_msgs](#)
- [std_msgs](#)
- [geometry_msgs](#)

Fichier du serveur d'actions - Outils

ToolAction (Action)

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

Fichiers de messages - outils

ToolCommand (Message)

```
# Gripper
int8 OPEN_GRIPPER = 1
int8 CLOSE_GRIPPER = 2

# Vacuum pump
int8 PULL_AIR_VACUUM_PUMP = 10
int8 PUSH_AIR_VACUUM_PUMP = 11

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

uint8 cmd_type

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

# if gripper Ned1/One
uint16 speed

# if gripper Ned2
uint8 max_torque_percentage
uint8 hold_torque_percentage

# if vacuum pump or electromagnet grove
bool activate

# if tool is set by digital outputs (electromagnet)
string gpio
```

TCP (Message)

```
bool enabled

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

Fichiers de services - outils

SetTCP (Service)

```

geometry_msgs/Point position

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

```

Niryo_robot_user_interface

Ce package gère les commandes d'interface utilisateur de haut niveau provenant des requêtes MATLAB ainsi que les fonctionnalités relatives au système comme les E/S, LED et ventilateurs internes.

Vous pouvez trouver leurs documentations ici :

- [TCP Server](#)

Utilisation du serveur TCP Ned

Ned exécute en permanence un serveur TCP pour acquérir des requêtes. Ce serveur est construit sur le [Ned Python ROS Wrapper](#) ([index.html#document-source/ros_wrapper](#)).

Il offre un moyen simple aux développeurs de créer des programmes pour que le robot les contrôle via une communication à distance sur un ordinateur, sur un mobile ou sur tout autre appareil doté d'une installation réseau.

Les programmes peuvent communiquer via le réseau TCP avec les robots dans n'importe quelle langue disponible.

Connexion

Pour accéder au serveur, il est nécessaire d'utiliser l'adresse IP du robot et communiquer via le **port 40001**.

Communication

Un seul client peut communiquer avec le serveur (reconnexion effective mais pas de multi clients).

Le serveur ne répond qu'une fois la commande terminée, il ne peut donc pas traiter plusieurs commandes en même temps.

Convention de paquet

Format général

Pour une utilisation et un débogage simplifiés, la communication est basée sur le format JSON.

Chaque package a la forme suivante : `<json_packet_size>{<json_content>}<payload>` .

La taille du paquet JSON est un code court non signé sur 2 octets.

Le JSON contient le nom et les paramètres de la commande.

La charge utile contient des données lourdes telles qu'une image.

Requête

Format - Requête

Comme aucune fonction ne demande une charge utile en entrée (input), les demandes ont les caractéristiques suivantes.

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

Requête - Exemples

Calibration automatique : `{'param_list': ['AUTO'], 'command': 'CALIBRATE'}`

Déplacer axes : `{'param_list': [0.0, 0.0, 0.0, 0.0, 0.0, 0.0], 'command': 'MOVE_JOINTS'}`

Réponses

Format - Réponse

Premièrement, les réponses indiquent à l'utilisateur si sa commande a été correctement exécutée. Ceci est indiqué dans le JSON par le paramètre « status ».

Une réponse réussie aura le format :

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

Une réponse infructueuse aura le format : `{'status': 'KO', 'message': <message_str>}`

Exemples - Réponse

Calibration automatique : `{'status': 'OK', 'list_ret_param': [], 'payload_size': 0, 'command': 'CALIBRATE'}`

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

Commandes enum pour le serveur TCP

class CommandEnum

Enumération de toutes les commandes utilisées

CALIBRATE= 0

SET_LEARNING_MODE= 1

GET_LEARNING_MODE= 2

SET_ARM_MAX_VELOCITY= 3

SET_JOG_CONTROL= 4

GET_JOINTS= 10

GET_POSE= 11

GET_POSE_QUAT= 12

MOVE_JOINTS= 20

MOVE_POSE= 21

SHIFT_POSE= 22

MOVE_LINEAR_POSE= 23

SHIFT_LINEAR_POSE= 24

JOG_JOINTS= 25

JOG_POSE= 26

FORWARD_KINEMATICS= 27

INVERSE_KINEMATICS= 28

GET_POSE_SAVED= 50

SAVE_POSE= 51

DELETE_POSE= 52

GET_SAVED_POSE_LIST= 53

PICK_FROM_POSE= 60

PLACE_FROM_POSE= 61

PICK_AND_PLACE= 62

GET_TRAJECTORY_SAVED= 80

GET_SAVED_TRAJECTORY_LIST= 81

EXECUTE_REGISTERED_TRAJECTORY= 82

EXECUTE_TRAJECTORY_FROM_POSES= 83

EXECUTE_TRAJECTORY_FROM_POSES_AND_JOINTS= 84

SAVE_TRAJECTORY= 85

SAVE_LAST_LEARNED_TRAJECTORY= 86

UPDATE_TRAJECTORY_INFOS= 87

DELETE_TRAJECTORY= 88

CLEAN_TRAJECTORY_MEMORY= 89

GET_SAVED_DYNAMIC_FRAME_LIST= 95

GET_SAVED_DYNAMIC_FRAME= 96

SAVE_DYNAMIC_FRAME_FROM_POSES= 97

SAVE_DYNAMIC_FRAME_FROM_POINTS= 98

EDIT_DYNAMIC_FRAME= 99

DELETE_DYNAMIC_FRAME= 100

MOVE_RELATIVE= 101

MOVE_LINEAR_RELATIVE= 102

UPDATE_TOOL= 120

OPEN_GRIPPER= 121

CLOSE_GRIPPER= 122

PULL_AIR_VACUUM_PUMP= 123

PUSH_AIR_VACUUM_PUMP= 124

SETUP_ELECTROMAGNET= 125

ACTIVATE_ELECTROMAGNET= 126

DEACTIVATE_ELECTROMAGNET= 127

GET_CURRENT_TOOL_ID= 128

GRASP_WITH_TOOL= 129

RELEASE_WITH_TOOL= 130

ENABLE_TCP= 140

SET_TCP= 141

RESET_TCP= 142

TOOL_REBOOT= 145

SET_PIN_MODE= 150

DIGITAL_WRITE= 151

DIGITAL_READ= 152

GET_DIGITAL_IO_STATE= 153

GET_HARDWARE_STATUS= 154

ANALOG_WRITE= 155

ANALOG_READ= 156

GET_ANALOG_IO_STATE= 157

CUSTOM_BUTTON_STATE= 158

SET_CONVEYOR= 180

UNSET_CONVEYOR= 181

CONTROL_CONVEYOR= 182

GET_CONNECTED_CONVEYORS_ID= 183

GET_IMAGE_COMPRESSED= 200

GET_TARGET_POSE_FROM_REL= 201

GET_TARGET_POSE_FROM_CAM= 202

VISION_PICK= 203

MOVE_TO_OBJECT= 205

DETECT_OBJECT= 204

GET_CAMERA_INTRINSICS= 210

SAVE_WORKSPACE_FROM_POSES= 220

SAVE_WORKSPACE_FROM_POINTS= 221

DELETE_WORKSPACE= 222

GET_WORKSPACE_RATIO= 223

GET_WORKSPACE_LIST= 224

SET_IMAGE_BRIGHTNESS= 230

SET_IMAGE_CONTRAST= 231

SET_IMAGE_SATURATION= 232

GET_IMAGE_PARAMETERS= 235

PLAY_SOUND= 240

SET_VOLUME= 241

STOP_SOUND= 242

DELETE_SOUND= 243

IMPORT_SOUND= 244

GET_SOUNDS= 245

GET_SOUND_DURATION= 246

SAY= 247

LED_RING_SOLID= 250

LED_RING_TURN_OFF= 251

LED_RING_FLASH= 252

LED_RING_ALTERNATE= 253

LED_RING_CHASE= 254

LED_RING_WIPE= 255

LED_RING_RAINBOW= 256

LED_RING_RAINBOW_CYCLE= 257

LED_RING_RAINBOW_CHASE= 258

LED_RING_GO_UP= 259

LED_RING_GO_UP_DOWN= 260

LED_RING_BREATH= 261

LED_RING_SNAKE= 262

LED_RING_CUSTOM= 263

LED_RING_SET_LED= 264

Niryo_robot_vision

Ce package est celui qui traite de tous les éléments liés à la vision.

Noeud Vision

Le nœud ROS est composé de plusieurs services pour gérer le streaming vidéo, la détection d'objets... Le nœud fonctionne exactement de la même manière si vous avez choisi de l'utiliser sur la simulation ou la réalité.

Ce nœud peut être lancé localement en mode autonome via la commande :

```
roslaunch niryo_robot_vision vision_node_local.launch
```

La configuration (image par seconde, port de caméra, résolution vidéo) peut être modifiée dans le fichier de configuration :

- Pour un nœud « standard » : `niryo_robot_vision/config/video_server_setup.yaml`
- Pour un nœud local : `niryo_robot_vision/config/video_server_setup_local.yaml`

Il appartient au namespace ROS : `/niryo_robot_vision/`.

Paramètres - Vision

Paramètres du Package de Vision

Nom	Description
<code>frame_rate</code>	Fréquence du flux d'images
<code>simulation_mode</code>	Définir sur vrai si vous utilisez la simulation Gazebo. Il adaptera la façon dont le nœud obtient son flux de vidéo
<code>debug_compression_quality</code>	Déboguer la qualité de la compression du flux
<code>stream_compression_quality</code>	Qualité de compression de flux
<code>subsampling</code>	Facteur de sous-échantillonnage de flux

Diffuseur - Vision

Diffuseurs du package Vision

Nom	Type du message	Description
<code>compressed_video_stream</code>	<code>sensor_msgs/CompressedImage</code>	Publier la dernière image lue sous forme d'image compressée
<code>video_stream_parameters</code>	<code>ImageParameters</code>	Publie les paramètres de luminosité, contraste et saturation du stream vidéo

Services - Vision

Gestionnaire de programmes services

Nom	Type du message	Description
<code>debug_colors</code>	<code>DebugColorDetection</code>	Renvoie une image annotée pour souligner ce qu'il s'est passé avec la détection des couleurs
<code>debug_markers</code>	<code>DebugMarkers</code>	Renvoie une image annotée pour souligner ce qu'il s'est passé avec la détection des marqueurs
<code>obj_detection_rel</code>	<code>ObjDetection</code>	Service de détection d'objets

Nom	Type du message	Description
<code>start_stop_video_streaming</code>	SetBool	Démarrer ou arrêter la diffusion vidéo
<code>take_picture</code>	TakePicture	Enregistrer une image dans le dossier spécifié
<code>set_brightness</code>	SetImageParameter	Définir la luminosité du stream vidéo
<code>set_contrast</code>	SetImageParameter	Définir le contraste du stream vidéo
<code>set_saturation</code>	SetImageParameter	Définir la saturation du stream vidéo
<code>visualization</code>	Visualization	Ajoute des marqueurs sur rviz des objets détectés par le kit vision

Tous ces services sont disponibles dès le démarrage du nœud.

Dépendances - Vision

- [Niryo_robot_msgs](#)
- [sensor_msgs](#)

Fichiers de services - Vision

ImageParameters (Topic)

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

Fichiers de services - Vision

DebugColorDetection (Service)

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

DebugMarkers (Service)

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

ObjDetection (Service)

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

niryo_robot_msgs/ObjectPose obj_pose

string obj_type
string obj_color

sensor_msgs/CompressedImage img
```

TakePicture (Service)

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

SetImageParameter (Service)

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

Visualization (Service)

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

Niryo_robot_led_ring

Ce package permet de contrôler l'anneau LED du Ned2.

Il est composé d'un nœud, recevant les commandes et l'état actuel du robot, et publiant les états de l'anneau LED.

Le LED Ring est composé de 30 LEDs RGB WS2811, contrôlées par le package grâce la bibliothèque [rpi_ws281x](#).

Nœud de l'anneau LED

Le nœud ROS est fait pour gérer l'état du LED Ring, et publie son état actuel sur un topic ROS. Il utilise une classe implémentant plusieurs animations, permettant de contrôler le LED Ring ou d'afficher l'état actuel du robot. L'anneau LED est également implémenté dans [Rviz](#).

L'anneau LED peut être :

- en mode **ROBOT STATUS** : la LED affiche l'état du robot.
- En mode **USER** : l'utilisateur peut contrôler le LED Ring avec les différentes méthodes mises en œuvre, via :
[Blockly](#), [Pyniryo](#) or [Python ROS Wrapper](#).

Mode d'état du robot

Lors de l'affichage du **robot status**, l'anneau LED a plusieurs états qui représentent différents modes et états d'erreur. Reportez-vous au tableau suivant. Le nœud s'abonne au topic ROS `/niryo_robot_status/robot_status`, publié par le package [RobotStatus](#) ([index.html#robotstatus](#)).



Animation et couleur	Description	Dépannage
Breath blanc	Le robot démarre	N/A
Chase bleu	Un étalonnage est nécessaire	Appuyez sur le bouton <i>Custom</i> , ou lancez un étalonnage
Snake bleu	Étalonnage en cours	N/A
Breath bleu	Mouvement libre activé	N/A
3 clignotements jaunes	Début de l'étalonnage	N/A
Breath vert	Free Motion désactivé, couple activé	N/A
Vert fixe	Programme en cours	N/A
Chase vert	Programme suspendu	Appui long sur le bouton sur haut du robot pour annuler le programme, appui court pour reprendre
Breath orange	Erreur d'exécution du programme	Lancer une nouvelle action pour effacer cet état
Flashing Orange	Collision	Lancer une nouvelle action pour effacer cet état
Orange fixe	Joint hors limites	Passez en mode mouvement libre pour ramener les articulations dans les limites.
1 clignotement violet	Nouveau formulaire de connexion Niryo Studio	N/A
2 clignotements violets	Enregistrer les positions d'un robot à partir du bouton "Enregistrer"	N/A
Flashing rouge	Erreur moteur / Surchauffe Raspberry	Veuillez vérifier l'erreur sur Niryo Studio.
Rouge Fixe	Crash de ROS	Veuillez redémarrer le robot.







Mode utilisateur






Plusieurs animations sont implémentées pour permettre à l'utilisateur différentes manières de contrôler le LED Ring. Reportez-vous au tableau suivant. Le nœud reçoit des commandes via le service `/niryo_robot_led_ring/set_user_animation` (voir [la section services](#))

ⓘ Important

Ned doit être en mode autonome afin de permettre à l'utilisateur de contrôler l'anneau LED.

Animation	Apparence	Gif
None	Les LEDs sont éteintes	
Solid	Réglez l'ensemble de l'anneau LED sur la même couleur à la fois	

Animation	Apparence	Gif
Flashing	Clignote une couleur selon une fréquence.	
Alternate	Les différentes couleurs s'alternent les unes après les autres.	
Chase	Une LED sur trois est allumée. Toutes se décalent une LED après l'autre.	
Color Wipe	Allume les LEDs une à une jusqu'à allumer complètement l'anneau. Similaire à <i>Go Up</i> , mais les LEDs ne sont pas éteintes à la fin.	
Rainbow	Dessine un arc-en-ciel qui s'estompe sur toutes les LED à la fois.	
Rainbow cycle	Dessine un arc-en-ciel qui se répartit sur toutes les LEDs.	

Animation	Apparence	Gif
Rainbow chase	Animation de <i>chase</i> arc-en-ciel.	
Go up	Les LEDs s'allument comme un cercle de chargement jusqu'à allumer tout l'anneau LED. et sont ensuite tous éteints en même temps.	
Go up and down	Comme <i>Go UP</i> , mais les LEDs sont éteintes de la même manière qu'elles sont allumées.	
Breath	Variation de l'intensité lumineuse pour imiter la respiration.	
Snake	Serpent lumineux qui tourne autour de l'anneau LED.	

Note

Lors de l'affichage de l'état du robot, le LED Ring commander utilise ces méthodes, avec les paramètres par défaut définis ci-dessous.

Il appartient à l'espace de noms ROS : `/niryo_robot_led_ring/`.

Paramètres - Anneau LED

Tout d'abord, le composant LED Ring, contrôlé avec la librairie `rpi_ws281x` (<https://github.com/rpi-ws281x/rpi-ws281x-python>), via la classe Python `PixelStrip`. Les paramètres par défaut sont définis dans le fichier `led_strim_params.yaml` du dossier `/config` du package.

Paramètres du composant LED Ring

Nom	Description	Valeur par défaut
<code>led_count</code>	Nombre de pixels LED dans l'anneau LED	30
<code>led_pin</code>	Pin de la Raspberry Pi connecté aux pixels Il doit prendre en charge PWM.	13
<code>led_freq_hs</code>	Fréquence du signal LED en Hertz	800 kHz
<code>led_dma</code>	Canal DMA à utiliser pour générer un signal	10
<code>led_luminosité</code>	Luminosité des LEDs. Régulé sur 0 pour le plus sombre et 255 pour le plus clair	255
<code>led_invert</code>	True pour inverser le signal (lors de l'utilisation du décalage de niveau du transistor NPN)	True
<code>led_channel</code>	le canal PWM à utiliser	0

Un autre fichier de configuration, le `led_ring_params.yaml`, définit les paramètres par défaut des animations de l'anneau LED.

Paramètres des animations de l'anneau LED

Nom	Description	Valeur par défaut
<code>default_flashing_period</code>	Période par défaut en secondes de l'animation Flashing	0.25
<code>default_alternate_period</code>	Période par défaut en secondes de l'animation Alternate .	1
<code>default_chase_period</code>	Période par défaut en secondes de l'animation Chase .	4
<code>default_colorwipe_period</code>	Période par défaut en secondes de l'animation Wipe .	5
<code>default_rainbow_period</code>	Période par défaut en secondes de l'animation Rainbow .	5
<code>default_rainbowcycle_period</code>	Période par défaut en secondes de l'animation Rainbow cycle	5
<code>default_rainbowchase_period</code>	Période par défaut en secondes de l'animation Rainbow chase	5
<code>default_goup_period</code>	Période par défaut en secondes de l'animation Go up .	5
<code>default_goupanddown_period</code>	Période par défaut en secondes de l'animation Go up and down .	5
<code>default_breath_period</code>	Période par défaut en secondes de l'animation Breath .	4
<code>default_snake_period</code>	Période par défaut en secondes de l'animation Snake .	1.5
<code>led_offset</code>	ID de décalage entre la LED avec l'ID 0 et l'ID de la led à l'arrière du robot.	8
<code>simulation_led_ring_markers_publish_rate</code>	Taux de publication des marqueurs annulaires Rviz en mode simulation	20
<code>led_ring_markers_publish_rate</code>	Les marqueurs d'anneau à led Rviz publient le taux sur un robot real	5

Services - Anneau LED

Le nœud ROS implémente un service, conçu pour que l'utilisateur contrôle l'anneau LED.

Services du package d'anneau de LED

Nom	Message type	Description
<code>set_user_animation</code>	LedUser	Permet à l'utilisateur de contrôler l'anneau LED, avec des animations implémentées. Une nouvelle demande va interrompre la précédente, si elle est toujours en cours de lecture. En fonction du champ booléen <code>wait</code> et le champ <code>itérations</code> de la requête, le service sera soit répondre immédiatement après avoir lancé l'animation, ou attendre que l'animation se termine pour répondre.
<code>set_led_color</code>	SetLedColor	Allume une LED identifiée par un identifiant

Publishers - Anneau LED

Publishers du package d'anneau LED

Nom	Message type	Description
<code>led_ring_status</code>	LedRingStatus	Publie le statut du LED Ring fournissant des informations sur le mode actuel (affichage de l'état du robot ou contrôlé par l'utilisateur si le robot fonctionne en mode AUTONOME) l'animation en cours jouée et la couleur de l'animation (sauf pour les méthodes <i>Rainbow</i> , où la couleur de l'animation n'est pas définie). Publie à chaque fois au moins un champ modifié .
<code>visualization_marker_array</code>	visualization_msgs/MarkerArray	Publie des formes représentant des LEDs lorsque Ned est utilisé en simulation avec Rviz , sous forme de liste de 30 visualization_msgs/Marker de taille 30.

Subscribers - Anneau LED

Subscribers du package anneau LED

Nom du topic	Message type	Description
--------------	--------------	-------------

Nom du topic	Message type	Description
<code>/niryo_robot_status/robot_status</code>	RobotStatus	Récupère l'état actuel du robot et contrôle la LED en conséquence (voir la section Niryo_robot_status)
<code>/niryo_robot/blockly/save_current_point</code>	std_msgs/Int32	Capturez l'action <i>Enregistrer le point</i> pour faire clignoter l'anneau LED.
<code>/niryo_studio_connection</code>	std_msgs/Vide	Récupère la connexion Niryo Studio pour faire clignoter l'anneau LED.

Dépendances - Anneau LED

- [niryo_robot_msgs](#)
- [std_msgs](#)
- [visualization_msgs](#)
- [rpi_ws281x==4.3.0](#)

Fichiers de service - Anneau LED

LedUser (Service)

```
niryo_robot_led_ring/LedRingAnimation animation_mode

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

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

---
int32 status
string message
```

SetLedColor (Service)

```
int8 led_id
std_msgs/ColorRGBA color

---
int32 status
string message
```

Messages files - LED Ring

LedRingAnimation (Message)

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

int32 animation
```

LedRingCurrentState (Message)

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

LedRingStatus (Message)

```
int32 ROBOT_STATUS = 1
int32 USER = 2

int32 led_mode

niryo_robot_led_ring/LedRingAnimation animation_mode

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

Fonctions de l'API de l'anneau LED

Afin de contrôler le robot plus facilement que d'appeler chaque Topic et Service un par un, un Python ROS Wrapper de ROS a été développé.

Par exemple, un script activant le LED Ring via Python ROS Wrapper ressemblera à :

```
from niryo_robot_led_ring.api import LedRingRosWrapper

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

Cette classe vous permet de contrôler le robot via une API interne.

Liste des sous-sections de fonctions :

- [Fonctions d'animations personnalisées](#)
- [Fonctions d'animations prédéfinies](#)

Fonctions d'animations personnalisées

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

`set_led_color(led_id, color)`

Allume une LED d'une seule couleur. Couleur RVB entre 0 et 255.

Exemple:

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

Paramètres:

- **led_id** (*int*) – Id de la led : entre 0 et 29
- **color** (*list[float]* or *ColorRGBA*) – Couleur de la LED dans une liste de taille 3 [R, V, B] ou dansun objet ColorRGBA. Les canaux RVB ont une valeur comprise entre 0 et 255.

Renvoie: status, message

Type renvoyé: ([int](https://docs.python.org/3/library/functions.html#int) (<https://docs.python.org/3/library/functions.html#int>), [str](https://docs.python.org/3/library/stdtypes.html#str) (<https://docs.python.org/3/library/stdtypes.html#str>))

`custom(led_colors)`

Envoie une commande de couleur à toutes les LED de l'anneau LED.La fonction attend une liste de couleurs pour les 30 LED durobot.

Exemple:

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

Paramètres: **led_colors** (*list[list[float]* or *ColorRGBA*) – Liste de taille 30 de couleur de led dans une liste de taille3[R, V, B] ou dans un objet ColorRGBA. Les canaux RVB ont une valeur comprise entre 0 et 255. »

Renvoie: status, message

Type renvoyé: ([int](#), [str](#))

Fonctions d'animations prédéfinies

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

`solid(color, wait=False)`

Réglez l'ensemble de l'anneau LED sur une couleur fixe.

Exemple:

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

Paramètres:

- **color** (*list[float]* or *ColorRGBA*) – Couleur de la LED dans une liste de taille 3 [R, V, B] ou dansun objet ColorRGBA. Les canaux RVB ont une valeur comprise entre 0 et 255.
- **wait** (*bool*) – Le service attend que l'animation se termine ou ne réponde pas.Pour cette méthode, l'action se fait rapidement, donc l'attente ne prend pas beaucoup de temps.

Renvoie: status, message

Type renvoyé: ([int](https://docs.python.org/3/library/functions.html#int) (<https://docs.python.org/3/library/functions.html#int>), [str](https://docs.python.org/3/library/stdtypes.html#str) (<https://docs.python.org/3/library/stdtypes.html#str>))

`turn_off(wait=False)`

Éteignez toutes les LEDs

Exemple:

```
led_ring.turn_off()
```

Paramètres: **wait** (*bool*) – Le service attend que l'animation se termine ou ne réponde pas.Pour cette méthode, l'action se fait rapidement, donc l'attente ne prend pas beaucoup de temps.

Renvoie: status, message

Type renvoyé: ([int](#), [str](#))

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

Clignote une couleur selon une fréquence. La fréquence est égale à 1/période.

Exemples:

```
from std_msgs.msg import ColorRGBA

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

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

- Paramètres:**
- **color** (*list*[float](#) or *ColorRGBA*) – Couleur de la LED dans une liste de taille 3 [R, V, B] ou dans un objet ColorRGBA. Les canaux RVB ont une valeur comprise entre 0 et 255.
 - **period** (*float*) – Temps d'exécution d'un motif en secondes. Si 0, la période par défaut sera utilisée.
 - **iterations** (*int*) – Nombre de clignotements consécutifs. Si 0, la LED Ring clignote sans indéfiniment.
 - **wait** (*bool*) – Le service attend que l'animation termine toutes les itérations ou non pour répondre. Si le nombre d'itérations est à 0, le service répond immédiatement.

Renvoie: status, message

Type renvoyé: ([int](https://docs.python.org/3/library/functions.html#int) (<https://docs.python.org/3/library/functions.html#int>), [str](https://docs.python.org/3/library/stdtypes.html#str) (<https://docs.python.org/3/library/stdtypes.html#str>))

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

Plusieurs couleurs sont alternées les unes après les autres.

Exemples:

```
from std_msgs.msg import ColorRGBA

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

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

- Paramètres:**
- **color_list** (*list*[list](#)[float](#) or *ColorRGBA*) – Liste des couleurs des leds des listes d'objets de taille 3 [R, G, B] ou ColorRGBA. Les canaux RVB ont une valeur comprise entre 0 et 255.
 - **period** (*float*) – Temps d'exécution d'un motif en secondes. Si 0, la période par défaut sera utilisée.
 - **iterations** (*int*) – Nombre d'alternances consécutives. Si 0, le LED Ring alterne indéfiniment.
 - **wait** (*bool*) – Le service attend que l'animation termine toutes les itérations ou non pour répondre. Si le nombre d'itérations est à 0, le service répond immédiatement.

Renvoie: status, message

Type renvoyé: ([int](https://docs.python.org/3/library/functions.html#int) (<https://docs.python.org/3/library/functions.html#int>), [str](https://docs.python.org/3/library/stdtypes.html#str) (<https://docs.python.org/3/library/stdtypes.html#str>))

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

Animation de Chase de style lumière de cinéma.

Exemples:

```
from std_msgs.msg import ColorRGBA

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

- Paramètres:**
- **color** (*list* or *ColorRGBA*) – Couleur de la LED dans une liste de taille 3 [R, V, B] ou dans un objet ColorRGBA. Les canaux RVB ont une valeur comprise entre 0 et 255.
 - **period** (*float*) – Temps d'exécution d'un motif en secondes. Si 0, la période par défaut sera utilisée.
 - **iterations** (*int*) – Nombre de Chase consécutives. Si 0, l'animation continue indéfiniment. Un Chase allume juste une LED sur 3. »
 - **wait** (*bool*) – Le service attend que l'animation termine toutes les itérations ou non pour répondre. Si le nombre d'itérations est à 0, le service répond immédiatement.

Renvoie: status, message

Type renvoyé: ([int](https://docs.python.org/3/library/functions.html#int) (<https://docs.python.org/3/library/functions.html#int>), [str](https://docs.python.org/3/library/stdtypes.html#str) (<https://docs.python.org/3/library/stdtypes.html#str>))

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

Essayez une couleur sur l'anneau LED, allumez une LED à la fois.

Exemples:

```
from std_msgs.msg import ColorRGBA

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

- Paramètres:**
- **color** (*list*/*float*) or *ColorRGBA* – Couleur de la LED dans une liste de taille 3 [R, V, B] ou dansun objet ColorRGBA. Les canaux RVB ont une valeur comprise entre 0 et 255.
 - **period** (*float*) – Temps d'exécution d'un motif en secondes. Si 0, la période par défautsera utilisée.
 - **wait** (*bool*) – Le service attend que l'animation se termine ou ne réponde pas.

Renvoie: status, message

Type renvoyé: ([int](https://docs.python.org/3/library/functions.html#int) (<https://docs.python.org/3/library/functions.html#int>), [str](https://docs.python.org/3/library/stdtypes.html#str) (<https://docs.python.org/3/library/stdtypes.html#str>))

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

Dessine un arc-en-ciel qui s'estompe sur toutes les LED à la fois.

Exemples:

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

- Paramètres:**
- **period** (*float*) – Temps d'exécution d'un motif en secondes. Si 0, la période par défautsera utilisée.
 - **iterations** (*int*) – Nombre d'arcs-en-ciel consécutifs. Si 0, l'animation continueindéfiniment.
 - **wait** (*bool*) – Le service attend que l'animation se termine ou ne réponde pas.Si le nombre d'itérations vaut 0, le service est non bloquant.

Renvoie: status, message

Type renvoyé: ([int](https://docs.python.org/3/library/functions.html#int) (<https://docs.python.org/3/library/functions.html#int>), [str](https://docs.python.org/3/library/stdtypes.html#str) (<https://docs.python.org/3/library/stdtypes.html#str>))

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

Dessine un arc-en-ciel qui se répartit sur toutes les LEDs.

Exemples:

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

- Paramètres:**
- **period** (*float*) – Temps d'exécution d'un motif en secondes. Si 0, la période par défautsera utilisée.
 - **iterations** (*int*) – Nombre de cycles arc-en-ciel consécutifs. Si 0, l'animation continueindéfiniment.
 - **wait** (*bool*) – Le service attend que l'animation se termine ou ne réponde pas.Si le nombre d'itérations vaut 0, le service est non bloquant.

Renvoie: status, message

Type renvoyé: ([int](https://docs.python.org/3/library/functions.html#int) (<https://docs.python.org/3/library/functions.html#int>), [str](https://docs.python.org/3/library/stdtypes.html#str) (<https://docs.python.org/3/library/stdtypes.html#str>))

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

Animation de Chase arc-en-ciel, comme la méthode led_ring_chase.

Exemples:

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

- Paramètres:**
- **period** (*float*) – Temps d'exécution d'un motif en secondes. Si 0, la période par défautsera utilisée.
 - **iterations** (*int*) – Nombre de cycles arc-en-ciel consécutifs. Si 0, l'animation continueindéfiniment.
 - **wait** (*bool*) – Le service attend que l'animation se termine ou ne réponde pas.Si le nombre d'itérations vaut 0, le service est non bloquant.

Renvoie: status, message

Type renvoyé: ([int](https://docs.python.org/3/library/functions.html#int) (<https://docs.python.org/3/library/functions.html#int>), [str](https://docs.python.org/3/library/stdtypes.html#str) (<https://docs.python.org/3/library/stdtypes.html#str>))

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

Les LED s'allument comme un cercle de chargement, puis s'éteignenttoutes en même temps.

Exemples:

```
from std_msgs.msg import ColorRGBA

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

- Paramètres:**
- **color** (*listffloat*] or *ColorRGBA*) – Couleur de la LED dans une liste de taille 3 [R, V, B] ou dansun objet ColorRGBA. Les canaux RVB ont une valeur comprise entre 0 et 255.
 - **period** (*float*) – Temps d'exécution d'un motif en secondes. Si 0, la période par défautsera utilisée.
 - **iterations** (*int*) – Nombre de tours consécutifs autour du LED Ring. Si 0, l'animationcontinue indéfiniment.
 - **wait** (*bool*) – Le service attend que l'animation se termine ou ne réponde pas.Si le nombre d'itérations vaut 0, le service est non bloquant.

Renvoie: status, message

Type renvoyé: ([int](https://docs.python.org/3/library/functions.html#int) (<https://docs.python.org/3/library/functions.html#int>), [str](https://docs.python.org/3/library/stdtypes.html#str) (<https://docs.python.org/3/library/stdtypes.html#str>))

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

Les LED s'allument comme un cercle de chargement et s'éteignentde la même manière.

Exemples:

```
from std_msgs.msg import ColorRGBA

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

- Paramètres:**
- **color** (*listffloat*] or *ColorRGBA*) – Couleur de la LED dans une liste de taille 3 [R, V, B] ou dansun objet ColorRGBA. Les canaux RVB ont une valeur comprise entre 0 et 255.
 - **period** (*float*) – Temps d'exécution d'un motif en secondes. Si 0, la période par défautsera utilisée.
 - **iterations** (*int*) – Nombre de tours consécutifs autour du LED Ring. Si 0, l'animationcontinue indéfiniment.
 - **wait** (*bool*) – Le service attend que l'animation se termine ou ne réponde pas.Si le nombre d'itérations vaut 0, le service est non bloquant.

Renvoie: status, message

Type renvoyé: ([int](https://docs.python.org/3/library/functions.html#int) (<https://docs.python.org/3/library/functions.html#int>), [str](https://docs.python.org/3/library/stdtypes.html#str) (<https://docs.python.org/3/library/stdtypes.html#str>))

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

Variation de l'intensité lumineuse de l'anneau LED, similaireà la respiration humaine.

Exemples:

```
from std_msgs.msg import ColorRGBA

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

- Paramètres:**
- **color** (*listffloat*] or *ColorRGBA*) – Couleur de la LED dans une liste de taille 3 [R, V, B] ou dansun objet ColorRGBA. Les canaux RVB ont une valeur comprise entre 0 et 255.
 - **period** (*float*) – Temps d'exécution d'un motif en secondes. Si 0, la période par défautsera utilisée.
 - **iterations** (*int*) – Nombre de tours consécutifs autour du LED Ring. Si 0, l'animationcontinue indéfiniment.
 - **wait** (*bool*) – Le service attend que l'animation se termine ou ne réponde pas.Si le nombre d'itérations vaut 0, le service est non bloquant.

Renvoie: status, message

Type renvoyé: ([int](https://docs.python.org/3/library/functions.html#int) (<https://docs.python.org/3/library/functions.html#int>), [str](https://docs.python.org/3/library/stdtypes.html#str) (<https://docs.python.org/3/library/stdtypes.html#str>))

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

Un petit serpent coloré (certainement un python :D) se déplace autourde l'anneau LED.

Exemples:

```
from std_msgs.msg import ColorRGBA

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

- Paramètres:**
- **color** (*listffloat*] or *ColorRGBA*) – Couleur de la LED dans une liste de taille 3 [R, V, B] ou dansun objet ColorRGBA. Les canaux RVB ont une valeur comprise entre 0 et 255.
 - **period** (*float*) – Temps d'exécution d'un motif en secondes. Si 0, la durée pardéfaut sera utilisée.
 - **iterations** (*int*) – Nombre de tours consécutifs autour du LED Ring. Si 0, l'animationcontinue indéfiniment.
 - **wait** (*bool*) – Le service attend que l'animation se termine ou ne réponde pas.Si le nombre d'itérations vaut 0, le service est non bloquant.

Renvoie: status, message

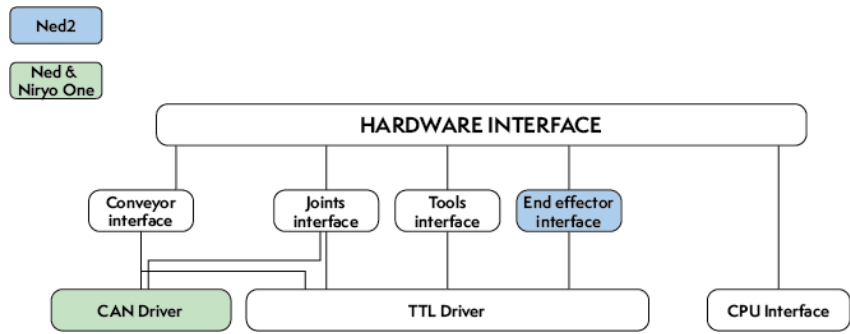
Type renvoyé: ([int](https://docs.python.org/3/library/functions.html#int) (<https://docs.python.org/3/library/functions.html#int>), [str](https://docs.python.org/3/library/stdtypes.html#str) (<https://docs.python.org/3/library/stdtypes.html#str>))

Paquets Bas Niveaux

Dans cette section, vous aurez accès ç toutes les informations concernant les paquets de la stack hardware ROS des robots de Niryo, dédiées aux interfaces bas niveau

Niryo Robot Hardware Interface

Ce package gère les packages en lien avec les composants matériels du robot.
Il lance les nodes des interfaces matérielles, des communications moteurs et des drivers.



Aperçu global de l'organisation des packages de la stack hardware.

Hardware interface Node

Ce noeud a été conçu pour instancier toutes les interfaces nécessaires pour avoir un robot complètement fonctionnel.

Parmi ces interfaces, nous avons :

- Conveyor Interface
- Joints Interface
- Tools Interface
- Cpu Interface
- End Effector Interface (Ned2 seulement)
- Can Driver (Ned et One seulement)
- Ttl Driver

Il appartient au namespace ROS : `/niryo_robot_hardware_interface/`.

Paramètres

Paramètres de l'Interface Hardware Interface

Nom	Description
<code>publish_hw_status_frequency</code>	Fréquence de publication des statuts matériels. Défaut : "2.0"
<code>publish_software_version_frequency</code>	Fréquence de publication des statuts logiciels. Défaut : "2.0"

Dépendances

- [Tools Interface](#)
- [Joints Interface](#)
- [Interface Convoyeur](#)
- [CPU Interface](#)
- [Niryo_robot_msgs](#)

Services, Topics et Messages

Topics publiés

Topics publiés de l'interface Hardware Interface

Nom	Type de message	Description
<code>hardware_status</code>	<code>niryo_robot_msgs/HardwareStatus</code>	Moteurs, bus, joints et statuts des processeurs
<code>software_version</code>	<code>niryo_robot_msgs/SoftwareVersion</code>	Version logicielle de la Raspberry Pi et de chaque composant matériel (moteurs, end effector, convoyeurs et outils)

Services

Services du package Hardware Interface

Nom	Type de message	Description
<code>launch_motors_report</code>	Trigger	Lance l'outil de diagnostic des moteurs
<code>reboot_motors</code>	Trigger	Redémarre les moteurs
<code>stop_motors_report</code>	Trigger	Arrête l'outil de diagnostic des moteurs

Joints Interface

Ce package gère les packages en lien avec le contrôleur des articulations (joints) du robot.

Il expose une interface à [ros_control](#).

Joints interface node

Il est instancié dans le package [Niryo Robot Hardware Interface](#) ([index.html#document-source/stack/low_level/niryo_robot_hardware_interface](#)).

Il a été conçu pour :

- Interfacer les moteurs du robot au contrôleur de trajectoire des articulations, depuis le package [ros_control](#).
- Créer un gestionnaire de contrôleurs, depuis le package [controller_manager](#). Il définit l'infrastructure pour charger, décharger, démarrer et arrêter les contrôleurs.
- S'interfacer avec la calibration des moteurs.
- Initialiser les paramètres des moteurs.

Il appartient au namespace ROS : `/joints_interface/`.

Paramètres

Paramètres par défaut de l'interface

Fichier default.yaml

Nom	Description	Valeur par défaut	Unité
<code>ros_control_loop_frequency</code>	Contrôle la fréquence de la boucle de contrôle.	100	Hz

Paramètres spécifiques à l'interface matérielle de Joints Interface

Ces paramètres sont spécifiques à la version matérielle (Ned, Niryo One ou Ned2). Ces fichiers sont présents en versions différentes pour chaque version matérielle. Ils sont situés dans un dossier du nom de la version matérielle.

Fichier joints_params.yaml

Nom	Description	Versions matérielles supportées
<code>joint_N/id</code>	Id de l'articulation N (1, 2, 3, 4, 5 ou 6) Défaut: -1 (id invalide)	Toutes les versions
<code>joint_N/type</code>	Type de moteur pour l'articulation N (1, 2, 3, 4, 5 ou 6) parmi : « stepper », « xl320 », « xl430 », « fakeStepper » ou « fakeDxl » Défaut: « »	Toutes les versions
<code>joint_N/bus</code>	Bus de communication pour l'articulation N (1, 2, 3, 4, 5 ou 6) parmi : « ttl » ou « can » Défaut: « »	Toutes les versions

Fichier calibration_params.yaml

Nom	Description	Valeur par défaut	Unité	Versions matérielles supportées
<code>calibration_timeout</code>	Temps d'attente entre 2 commandes pendant le processus de calibration.	30	secondes	Toutes les versions
<code>calibration_file</code>	Chemin du fichier où sont sauvegardées les valeurs de calibration des moteurs.	<code>/home/niryo/niryo_robot_saved_files/stepper_motor_calibration_offsets.txt</code>	N.A.	Toutes les versions
<code>stepper_N/id</code>	Id du moteur Stepper N (1, 2 ou 3)	-1 (id invalide)	N.A.	Toutes les versions
<code>stepper_N/v_start</code>	Vitesse de départ pour le profil d'accélération du moteur Stepper N (1, 2 ou 3)	1	0.01 RPM	Ned 2 seulement
<code>stepper_N/a_1</code>	Première accélération du profil d'accélération du moteur Stepper N (1, 2 ou 3)	0	RPM²	Ned 2 seulement
<code>stepper_N/v_1</code>	Première vitesse pour le profil d'accélération du moteur Stepper N (1, 2 ou 3)	0	0.01 RPM	Ned 2 seulement
<code>stepper_N/a_max</code>	Accélération maximale du profil d'accélération du moteur Stepper N (1, 2 ou 3)	6000	RPM²	Ned 2 seulement
<code>stepper_N/v_max</code>	Vitesse maximale du profil d'accélération du moteur Stepper N (1, 2 ou 3)	6	0.01 RPM	Ned 2 seulement
<code>stepper_N/d_max</code>	Décélération maximale du profil d'accélération du moteur Stepper N (1, 2 ou 3)	6000	RPM²	Ned 2 seulement

Nom	Description	Valeur par défaut	Unité	Versions matérielles supportées
<code>stepper_N/d_1</code>	Décélération finale pour le profil d'accélération du moteur Stepper N (1, 2 ou 3)	0	RPM²	Ned 2 seulement
<code>stepper_N/v_stop</code>	Vitesse d'arrêt pour le profile d'accélération du moteur Stepper N (1, 2 ou 3)	2	0.01 RPM	Ned 2 seulement
<code>stepper_N/stall_threshold</code>	Seuil de torque du Stepper N (1, 2 ou 3) pour la détection de la fin de course pour le moteur lors du processus de calibration	0	N.A.	Ned 2 seulement
<code>stepper_N/direction</code>	Direction de calibration pour le moteur Stepper N (1, 2 ou 3) (1 = même direction que la direction du moteur, -1 = direction inverse de la direction du moteur)	1	N.A.	Toutes les versions
<code>stepper_N/delay</code>	Délai à attendre pour la calibration	0	millisecondes	Toutes les versions

Fichier dynamixel_params.yaml

Nom	Description	Unité	Versions matérielles supportées
<code>dxl_N/offset_position</code>	Décalage de position du moteur Dynamixel N (1, 2 ou 3) à la position limite minimale Défaut: "0.0"	Rad	Toutes les versions
<code>dxl_N/home_position</code>	Position Origine du moteur Dynamixel N (1, 2 ou 3) Défaut: "0.0"	Rad	Toutes les versions
<code>dxl_N/direction</code>	Direction du moteur Dynamixel N (1, 2 ou 3) (1 = sens horaire, -1 = sens anti-horaire) Défaut: 1	N.A.	Toutes les versions
<code>dxl_N/limit_position_max</code>	Position maximale autorisée pour le moteur Dynamixel N (1, 2 ou 3) Défaut: "0.0"	Rad	Toutes les versions
<code>dxl_N/limit_position_min</code>	Position minimale autorisée du moteur Dynamixel N (1, 2 ou 3) Défaut: "0.0"	Rad	Toutes les versions
<code>dxl_N/position_P_gain</code>	Gain Proportionnel du correcteur PID en position pour le moteur Dynamixel N (1, 2 ou 3) Défaut: "0.0"	N.A.	Toutes les versions
<code>dxl_N/position_I_gain</code>	Gain Intégral du correcteur PID en position pour le moteur Dynamixel N (1, 2 ou 3) Défaut: "0.0"	N.A.	Toutes les versions
<code>dxl_N/position_D_gain</code>	Gain Dérivé du correcteur PID en position pour le moteur Dynamixel N (1, 2 ou 3) Défaut: "0.0"	N.A.	Toutes les versions
<code>dxl_N/velocity_P_gain</code>	Gain Proportionnel du correcteur PI en vitesse pour le moteur Dynamixel N (1, 2 ou 3) Défaut: "0.0"	N.A.	Toutes les versions
<code>dxl_N/velocity_I_gain</code>	Gain Intégral du correcteur PI en vitesse pour le moteur Dynamixel N (1, 2 ou 3) Défaut: "0.0"	N.A.	Toutes les versions
<code>dxl_N/FF1_gain</code>	Gain par anticipation sur la vitesse pour le moteur Dynamixel N (1, 2 ou 3) Défaut: "0.0"	N.A.	Toutes les versions
<code>dxl_N/FF2_gain</code>	Gain par anticipation sur l'accélération pour le moteur Dynamixel N (1, 2 ou 3) Défaut: "0.0"	N.A.	Toutes les versions
<code>dxl_N/acceleration_profile</code>	Paramétrage du profil d'accélération du moteur Dynamixel N (1, 2 ou 3) ^[*] Défaut: "0.0"	RPM²	Toutes les versions
<code>dxl_N/velocity_profile</code>	Paramétrage du profil de vitesse pour le moteur Dynamixel N (1, 2 ou 3) Défaut: "0.0"	RPM	Toutes les versions

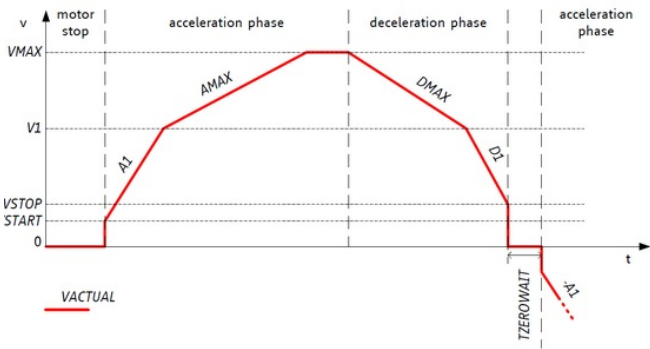
[*] Référez vous à la [documentation de référence](#) du moteur concerné pour plus d'information.

Fichier steppers_params.yaml

Nom	Description	Unité	Versions matérielles supportées
<code>stepper_N/id</code>	Id du moteur Stepper N (1, 2 ou 3) Défaut: -1 (id invalide)	N.A.	Toutes les versions
<code>stepper_N/gear_ratio</code>	Rapport de réduction du moteur Stepper N (1, 2 ou 3) Défaut: 1	N.A.	Ned et One seulement

Nom	Description	Unité	Versions matérielles supportées
stepper_N/max_effort	Effort maximal du moteur Stepper N (1, 2 ou 3) Défaut: 0	N.A.	Ned et One seulement
stepper_N/motor_ratio	Rapport de conversion du moteur Stepper N (1, 2 ou 3) pour convertir en radian Défaut: 1	N.A.	Ned 2 seulement
stepper_N/offset_position	Décalage de position du moteur Stepper N (1, 2 ou 3) à la position limite minimale Défaut: 0	Rad	Toutes les versions
stepper_N/home_position	Position Orgine du moteur Stepper N (1, 2 ou 3) Défaut: 0	Rad	Toutes les versions
stepper_N/limit_position_min	Position Limite minimale pour le moteur Stepper N (1, 2 ou 3) Défaut: 0	Rad	Toutes les versions
stepper_N/limit_position_max	Position Limite maximale pour le moteur Stepper N (1, 2 ou 3) Défaut: 0	Rad	Toutes les versions
stepper_N/direction	Direction d'assemblage (1 = Sens horaire, -1 = sens anti-horaire) du moteur Stepper N (1, 2 ou 3) Défaut: 1	N.A.	Toutes les versions
stepper_N/v_start	Vitesse de départ pour le profil d'accélération du moteur Stepper N (1, 2 ou 3) Défaut: 1	RPM	Ned 2 seulement
stepper_N/a_1	Première accélération du profil d'accélération du moteur Stepper N (1, 2 ou 3) Défaut: 0	RPM²	Ned 2 seulement
stepper_N/v_1	Première vitesse pour le profil d'accélération du moteur Stepper N (1, 2 ou 3) Défaut: 0	RPM	Ned 2 seulement
stepper_N/a_max	Accélération maximale du profil d'accélération du moteur Stepper N (1, 2 ou 3) Défaut: 6000	RPM²	Ned 2 seulement
stepper_N/v_max	Vitesse maximale du profil d'accélération du moteur Stepper N (1, 2 ou 3) Défaut: 6	RPM	Ned 2 seulement
stepper_N/d_max	Décélération maximale du profil d'accélération du moteur Stepper N (1, 2 ou 3) Défaut: 6000	RPM²	Ned 2 seulement
stepper_N/d_1	Décélération finale pour le profil d'accélération du moteur Stepper N (1, 2 ou 3) Défaut: 0	RPM²	Ned 2 seulement
stepper_N/v_stop	Vitesse d'arrêt pour le profile d'accélération du moteur Stepper N (1, 2 ou 3) Défaut: 2	RPM	Ned 2 seulement
stepper_N/stall_threshold	Seuil de torque pour la détection de la fin de course pour le moteur Stepper N (1, 2 ou 3) lors du processus de calibration Défaut :	N.A.	Ned 2 seulement

Les profils de vitesse des moteurs stepper (dans les fichiers *calibration_params.yaml* et *steppers_params.yaml*) peuvent être définis pour les moteurs stepper TTL seulement (donc pour le Ned2 seulement). Ils sont définis selon le graphique suivant :



Dépendances

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

Services, Topics et Messages

Abonnement aux Topics

Topics publiés pour Joints Interface

Nom	Type de Message	Description
<code>niryo_robot_follow_joint_trajectory_controller/follow_joint_trajectory/result</code>	<code>:control_actions:`control_msgs/FollowJointTrajectory Action<FollowJointTrajectory>`</code>	Résultats de trajectoire du contrôleur

Topics Publiés*Topics publiés pour Joints Interface*

Nom	Type de Message	Description
<code>/niryo_robot/learning_mode/state</code>	<code>std_msgs/Bool</code>	Etat du Learning Mode (FreeMotion pour le Ned2)

Services*Services du package joints interface*

Nom	Type de Message	Description
<code>/niryo_robot/joints_interface/calibrate_motors</code>	<code>SetInt</code>	Démarre la calibration des moteurs - Les valeurs possibles sont 1 pour une calibration automatique, 2 pour une calibration manuelle
<code>/niryo_robot/joints_interface/request_new_calibration</code>	<code>Trigger</code>	Réinitialise l'état de la calibration à « non calibré ». Ceci permet à l'utilisateur de demander une nouvelle calibration.
<code>niryo_robot/learning_mode/activate</code>	<code>Trigger</code>	Change l'état du mode apprentissage (Learning Mode ou Free Motion en Anglais). Quand le mode apprentissage est activé, les couples sont désactivés et les articulations peuvent bouger librement.
<code>niryo_robot/joints_interface/steppers_reset_controller</code>	<code>Trigger</code>	Réinitialise le contrôleur

Erreurs et messages d'alerte*Liste des erreurs et des messages d'alerte*

Type	Message	Description
Erreur	<code>JointHardwareInterface::init - Fail to add joint, return :</code>	L'articulation n'est pas correctement initialisée
Erreur	<code>JointHardwareInterface::init - stepper state init failed</code>	Les paramètres d'état du stepper ne sont pas correctement récupérés
Erreur	<code>JointHardwareInterface::init - dxl state init failed</code>	Les paramètres d'état du dynamixel ne sont pas correctement récupérés
Erreur	<code>JointHardwareInterface::init - Dynamixel motors are not available on CAN Bus</code>	Le robot essaye à tort d'initialiser un moteur dynamixel sur le bus CAN (ne fonctionnent que sur le TTL)
Erreur	<code>JointHardwareInterface::init - Fail to reboot motor id</code>	Le moteur échoue à redémarrer. Essayer de le redémarrer
ALERTE	<code>JointHardwareInterface::init - initialize stepper joint failure, return %d. Retrying</code>	L'initialisation du stepper a échoué. L'opération sera retentée 3 fois
ALERTE	<code>JointHardwareInterface::init - add stepper joint failure, return %d. Retrying</code>	L'ajout d'une articulation stepper a échoué. L'opération sera retentée 3 fois
ALERTE	<code>JointHardwareInterface::init - init dxl joint failure, return : %d. Retrying</code>	L'initialisation du dynamixel a échoué. L'opération sera retentée 3 fois
ALERTE	<code>JointHardwareInterface::init - add dxl joint failure, return : %d. Retrying</code>	L'ajout d'une articulation dynamixel a échoué. L'opération sera retentée 3 fois

Interface Convoyeur

Ce package gère les convoyeurs Niryo.

Il vous permet de contrôler jusqu'à deux Convoyeurs en même temps.

Deux versions du convoyeur existes: Le convoyeur v1, qui communique à travers un bus CAN et le convoyeur v2, qui communique à travers un bus TTL. Les deux sont entièrement compatible avec le Ned et le NiryoOne. Pour le Ned2, vous aurez besoin de changer la carte stepper du convoyeur belt v1 pour qu'il soit compatible; le Ned2 n'ayant pas de port CAN disponible de base.

Conveyor Interface Node (pour le développement et le debuggauge seulement)**The Node ROS a été conçu pour :**

- Utiliser le bon driver bas niveau, selon la version matérielle du robot.
- Initialiser l'interface du Convoyeur.

Convoyeur Interface Core

Il est instancié dans le package [Niryo Robot Hardware Interface](#) (index.html#document-source/stack/low_level/niryo_robot_hardware_interface).

The Node ROS a été conçu pour :

- S'interfacer avec les drivers bas niveau (CAN ou TTL pour Ned et le Niryo One, TTL uniquement pour le Ned2)
- Initialiser l'interface du Convoyeur.
- Gérer les requêtes depuis les services pour installer, contrôler ou retirer les convoyeurs.

- Fréquence de publication pour les états des convoyeurs.

Il appartient au namespace ROS : `/niryo_robot/conveyor/` .

Paramètres

Paramètres de l'interface Convoyeur

Nom	Description
<code>publish_frequency</code>	Fréquence de publication pour les états des convoyeurs. Défaut : "2.0"
<code>type</code>	Type de moteur utilisé. Défaut: "Stepper"
<code>protocol</code>	Protocole de communication. Peut être "CAN" (pour Ned ou le One) ou "TTL" (pour Ned ou le One ou le Ned2)
<code>default_id</code>	Identifiant (id) par défaut pour le convoyeur avant la connexion.
<code>Pool_id_list</code>	Identifiant du convoyeur après la connexion.
<code>Direction</code>	Direction du convoyeur.
<code>max_effort</code> (CAN seulement)	Effort maximal des moteurs stepper Défaut: "90"
<code>micro_steps</code> (CAN seulement)	Micro steps utilisés par les moteurs steppers Défaut : "8"

Topics publiés - Conveyor Interface

Liste des topics publiés par le Conveyor Interface

Nom	Type de Message	Description
<code>feedback</code>	<code>ConveyorFeedbackArray</code>	Etats des convoyeurs

Services

Services du package Interface Convoyeur

Nom	Type de Message	Description
<code>control_conveyor</code>	<code>ControlConveyor</code>	Envoie une commande au convoyeur choisi
<code>ping_and_set_conveyor</code>	<code>SetConveyor</code>	Scan et connecte un nouveau Convoyeur ou supprime un Convoyeur connecté

Dépendances - Convoyeur Interface

- [std_msgs](#)
- [Driver CAN](#)
- [TTL Driver](#)

Files de Services et messages - Conveyor interface

: `/niryo_robot/conveyor/`

```
uint8 id

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

: `/niryo_robot/conveyor/`

```
uint8 cmd
uint8 id

uint8 ADD = 1
uint8 REMOVE = 2

int16 id
int16 status
string message
```

: `/niryo_robot/conveyor/`

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

: `/niryo_robot/conveyor/`

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

Tools Interface

Ce package est dédié à la gestion des outils Niryo.

Node Tools Interface (Pour le développement et le debugage)

Ce noeud ROS est conçu pour :

- Initialiser l'interface Tool Interface avec les paramètres de configuration.
- Démarrer les services et topics ROS.

Tools Interface Core

Il est instancié dans le package [Niryo Robot Hardware Interface](#) (index.html#document-source/stack/low_level/niryo_robot_hardware_interface).

Il a été conçu pour :

- Initialiser le Tool Interface.
- Procurer les services pour configurer et contrôler les outils.
- Publier l'état de connexion des outils.

Il appartient au namespace ROS : `/tools_interface/` .

Paramètres par défaut de Tool Interface

default.yaml

Nom	Description
<code>check_tool_connection_frequency</code>	Fréquence de vérification et de publication de l'état de l'outil connecté, ou enlève l'outil si il est déconnecté. Défaut: "2.0"

Paramètres spécifiques à la version matérielle du robot

Ces paramètres sont spécifiques à la version matérielle (Ned, Niryo One ou Ned2). Ces fichiers sont présents en versions différentes pour chaque version matérielle. Ils sont situés dans un dossier du nom de la version matérielle.

tools_params.yaml

Nom	Description	Versions Matérielles supportées
<code>id_list</code>	Liste des Ids par défaut de chaque outil supporté par Niryo Défaut: "[11,12,13,30,31]"	Toutes les versions
<code>type_list</code>	Liste des types de moteurs pour les outils Défaut: "xl320" pour le Ned et le One Défaut: "xl330" pour le Ned2 Défaut: "fakeDxl" pour la simulation	Toutes les versions
<code>name_list</code>	Liste des noms d'outils correspondant à la liste des ids et des types ci-dessus Défaut: "[« Standard Gripper », « Large Gripper », « Adaptive Gripper », « Vacuum Pump », « Electromagnet »]"	Toutes les versions

Dépendances

- [std_msgs](#)
- [std_srvs](#)
- [TTL Driver](#)
- [Common](#)

Services, Topics et Messages

Topics publiés

Topics publiés de Tool Interface

Nom	Type de message	Description
<code>/niryo_robot_hardware/tools/current_id</code>	std_msgs/Int32	Identifiant de l'outil actuel

Services

Services du package Tool Interface

Nom	Type de message	Description
<code>niryo_robot/tools/ping_and_set_dx1_tool</code>	<code>tools_interface/PingDxlTool</code>	Scanne et configure un outil connecté
<code>niryo_robot/tools/open_gripper</code>	<code>tools_interface/OpenGripper</code>	Ouvre la pince
<code>niryo_robot/tools/close_gripper</code>	<code>tools_interface/OpenGripper</code>	Ferme la pince
<code>niryo_robot/tools/pull_air_vacuum_pump</code>	<code>tools_interface/OpenGripper</code>	Aspire l'air avec la pompe à vide
<code>niryo_robot/tools/push_air_vacuum_pump</code>	<code>tools_interface/OpenGripper</code>	Souffle de l'air avec la pompe à vide
<code>niryo_robot/tools/reboot</code>	<code>std_srvs/Trigger</code>	Redémarre le moteur de l'outil équipé

PingDxlTool (Service)

```

---
int8 state
tools_interface/Tool tool

```

ToolCommand (Service)

```

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

```

End Effector Interface

Ce package prend en charge le panel End Effector d'un robot, il est supporté à partir du Ned2.
Il expose des services et topics spécifiques au panel End Effector pour être utilisé par un utilisateur final.

Cependant, il ne gère pas la partie bas niveau de communication sur le bus avec les composants : c'est fait dans le package [TTL Driver](#).

Node End Effector Interface (pour le développement et le debugguage)

Le Node ROS dans le package End Effector Interface est utilisé pour :

- Instancier un manager [TTL Driver](#) pour communiquer avec le matériel.
- Initialiser l'interface End Effector.

End Effector Interface Core

Il est instancié dans le package [Niryo Robot Hardware Interface](#) ([index.html#document-source/stack/low_level/niryo_robot_hardware_interface](#)).

Il a été conçu pour :

- S'interfacer avec le Driver TTL.
- Initialiser les paramètres du End Effector.
- Récupérer les données du End Effector depuis le driver TTL.
- Publier le statu des boutons.
- Publier le statut de la détection de collision.
- Démarrer le service d'état des IO.

Il appartient au namespace ROS : `/end_effector_interface/`.

Paramètres - End Effector Interface

Paramètres de end_effector_interface

Nom	Description
<code>end_effector_id</code>	Id de l'End Effector sur le bus TTL Défaut: 0
<code>check_end_effector_status_frequency</code>	Fréquence de reception des données du End Effector depuis le driver Défaut: 40.0
<code>button_2_type</code>	Bouton utilisé pour activer le mode FreeMotion Défaut: free_drive
<code>button_1_type</code>	Bouton utilisé pour sauver la position courante du robot Défaut: save_position
<code>button_0_type</code>	Bouton personnalisé utilisé par les utilisateurs pour faire l'action de leur choix Défaut: custom
<code>hardware_type</code>	Type de l'End Effector. Les valeurs possibles sont : end_effector ou fake_end_effector Défaut: end_effector

Topics publiés - End Effector Interface

Topics publiés du package end_effector_interface

Nom	Type de message	Description
/niryo_robot_hardware_interface/end_effector_interface/_free_drive_button_state_publisher	EEButtonStatus	Publie l'état du bouton FreeMotion
/niryo_robot_hardware_interface/end_effector_interface/_save_button_state_publisher	EEButtonStatus	Publie l'état du bouton Save Position
/niryo_robot_hardware_interface/end_effector_interface/_custom_button_state_publisher	EEButtonStatus	Publie l'état du bouton Custom
/niryo_robot_hardware_interface/end_effector_interface/_digital_out_publisher	EEIOState	Publie l'état du Digital IO

Services - End Effector Interface

Services du package end_effector_interface

Nom	Type de service	Description
set_ee_io_state	SetEEDigitalOut	Initialise la sortie digitale du End Effector

Dépendances - End Effector Interface

- [std_msgs](#)
- [TTL Driver](#)
- [Common](#)

Fichiers de Services et de Messages - End Effector Interface

SetEEDigitalOut (Service)

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

EEButtonStatus (Message)

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

EEIOState (Message)

```
bool digital_input
bool digital_output
```

CPU Interface

Ce package procure une interface pour la surveillance de la température des processeurs du système.

CPU Interface Node (pour le développement et le debuggage seulement)

Ce noeud ROS a été conçu pour lancer l'interface CPU de manière isolée.

CPU Interface Core

Il est instancié dans le package [Niryo Robot Hardware Interface](#) ([index.html#document-source/stack/low_level/niryo_robot_hardware_interface](#)).

Il a été conçu pour surveiller la température des processeurs et arrêter automatiquement la Raspberry Pi si celle-ci atteint un seuil critique. Deux seuils peuvent être définis via des paramètres : un seuil d'alerte et un seuil d'arrêt.

La température des processeurs est lue depuis le fichier système d'Ubuntu `/sys/class/thermal/thermal_zone0/temp`.

En simulation, la température des processeurs de l'ordinateur faisant tourner la simulation est utilisées, mais les seuils sont désactivés (pas d'arrêt en cas de température trop élevée).

Il appartient au namespace ROS : `/cpu_interface/`.

Paramètres

Paramètres de l'interface

Nom	Description
<code>read_rpi_diagnostics_frequency</code>	Fréquence de publication de la température des processeurs Défaut : "0.25"

Nom	Description
<code>temperature_warn_threshold</code>	Seuil de température des processeurs [celsius] avant d'envoyer un message d'alerte Défaut : "75"
<code>temperature_shutdown_threshold</code>	Seuil de température des processeurs [celsius] avant d'éteindre le robot Défaut : "85"

Dépendances

- [Common](#)

Services, Topics et Messages

Aucun

Common

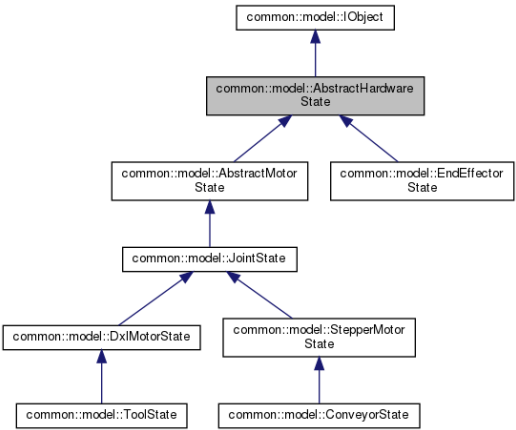
Le package Common définit les composants logiciels communs de la stack bas niveau. Il est séparé en une partie modèle et une partie utilitaire :
- The "model" subpackage defines the model tree needed to keep a virtual state of the robot up to date at any time.
- The "util" subpackage defines cpp interfaces and useful functions

Model

Le sous-package model est composé de :

States

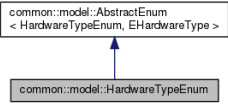
Classes représentant l'état virtuel de chaque composant matériel à tout moment. La hiérarchie permet de mettre en place un polymorphisme efficace afin de pouvoir interpréter chaque composant différemment en fonction des informations que l'on souhaite obtenir.



Graphe d'héritage de Abstract Hardware State

Enums

Enums améliorés, pour garder une trace des multiples énumérations et les doter de méthodes utilitaires pratiques (comme la conversion en chaîne de caractère par exemple).

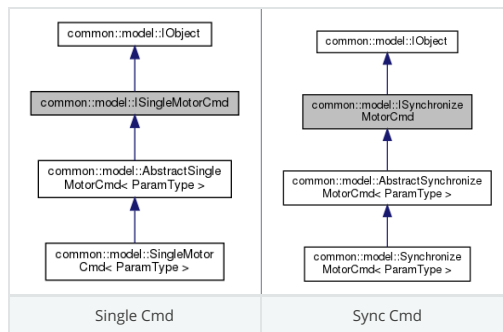


Graphe d'héritage de Hardware Type Enum

Commandes

Classes représentant les commandes simple et synchrones, pour les steppers et les dynamixels. Elles sont nécessaires pour leur utilisation au sein des queues de commandes des packages ttl_driver et can_driver.

Graph des Commandes



Chaque type de commande est un alias de versions spécialisées des deux classes template de base : `AbstractSynchronizeMotorCmd` et `AbstractSingleMotorCmd`

Util

Le sous-package util est compris de :

- Interfaces Cpp, utilisées globalement dans la stack pour du polymorphisme par exemple
- Des fonctions utilitaires, utilisées globalement dans la stack

Dépendances

Ce package ne dépend d'aucun autre package. Ce package est une dépendance des packages suivants :

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

TTL Driver

Ce package gère la communication des composants matériels sur le bus TTL.

Ce package est basé sur le SDK de DXL. Il expose une interface à [dynamixel_sdk](#).

TTL Driver Node (seulement pour le développement et le débogage)

Le noeud ROS est conçu pour :

- Initialiser l'interface TTL.
- Récupérer la configuration des composants matériels et les ajouter à l'interface TTL.

TTL Interface Core

Il est instancié dans le package [Niryo Robot Hardware Interface](#) ([index.html#document-source/stack/low_level/niryo_robot_hardware_interface](#)).

Il a été conçu pour :

- Initialiser l'interface TTL (interface utilisée par d'autres packages) et le bus physique avec les configurations.
- Ajouter, enlever et surveiller les périphériques.
- Démarrer l'acquisition et l'envoi de données sur le bus physique.
- Démarrer les Services et Topics ROS.

Il appartient au namespace ROS : `/niryo_robot/ttl_driver/`.

Paramètres - TTL Driver

Note

Ces paramètres de configuration sont choisis et testés de nombreuses fois pour fonctionner correctement. Assurez-vous de bien comprendre ce que vous faites avant de changer ces fichiers.

Paramètres du TTL driver

Nom	Description
<code>ttl_hardware_control_loop_frequency</code>	Fréquence de la boucle de contrôle du bus. Défaut: "240.0"
<code>ttl_hardware_write_frequency</code>	Fréquence d'écriture sur le bus. Défaut: "120.0"

Nom	Description
<code>t1l_hardware_read_data_frequency</code>	Fréquence de lecture sur le bus. Défaut: "120.0"
<code>t1l_hardware_read_status_frequency</code>	Fréquence de lecture pour les status des périphériques TTL. Défaut: "0.7"
<code>t1l_hardware_read_end_effector_frequency</code>	Fréquence de lecture des status du End Effector. Défaut: "13.0"
<code>bus_params/Baudrate</code>	Débit de Baude du bus TTL Défaut: "1000000"
<code>bus_params/uart_device_name</code>	Nom du port UART utilisé Défaut: "/dev/ttyAMA0"

Dépendances - TTL Driver

- [dynamixel_sdk](#)
- [Niryo_robot_msgs](#)
- [Common](#)
- [std_msgs](#)

Services - TTL Driver

Services du package TTL Driver

Nom	Type de message	Description
<code>niryo_robot/ttl_driver/set_dxl_leds</code>	SetInt	Contrôle les LED des dynamixels
<code>niryo_robot/ttl_driver/send_custom_value</code>	SendCustomValue	Ecrit des données sur une adresse registre choisi d'un périphérique TTL donné
<code>niryo_robot/ttl_driver/read_custom_value</code>	ReadCustomValue	Lit des données sur une adresse registre choisi d'un périphérique TTL donné
<code>niryo_robot/ttl_driver/read_pid_value</code>	ReadPIDValue	Lit le PID des moteurs dxl
<code>niryo_robot/ttl_driver/write_pid_value</code>	WritePIDValue	Ecrit le PID des moteurs dxl
<code>niryo_robot/ttl_driver/read_velocity_profile</code>	ReadVelocityProfile	Lit les profils de vitesse des steppers
<code>niryo_robot/ttl_driver/write_velocity_profile</code>	WriteVelocityProfile	Ecrit les profils de vitesse des steppers

Fichiers de Services et Messages - TTL Driver

SendCustomValue (Service)

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

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

ReadCustomValue (Service)

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

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

ReadPIDValue (Service)

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

uint8 id
...
uint16 pos_p_gain
uint16 pos_i_gain
uint16 pos_d_gain

uint16 vel_p_gain
uint16 vel_i_gain

uint16 ff1_gain
uint16 ff2_gain

int32 status
string message
```

WritePIDValue (Service)

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

uint8 id
---
uint16 pos_p_gain
uint16 pos_i_gain
uint16 pos_d_gain

uint16 vel_p_gain
uint16 vel_i_gain

uint16 ff1_gain
uint16 ff2_gain

int32 status
string message
```

ReadVelocityProfile (Service)

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

uint8 id
---
float64 v_start

float64 a_1
float64 v_1

float64 a_max
float64 v_max
float64 d_max

float64 d_1

float64 v_stop

int32 status
string message
```

WriteVelocityProfile (Service)

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

uint8 id

float64 v_start

float64 a_1
float64 v_1

float64 a_max
float64 v_max
float64 d_max

float64 d_1

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

MotorHardwareStatus (Message)

```
niryo_robot_msgs/MotorHeader motor_identity

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

MotorCommand (Message)

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

uint8[] motors_id
uint32[] params
```

ArrayMotorHardwareStatus (Message)

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

Driver CAN

Ce package procure une interface entre les packages ROS haut niveau et le gestionnaire de bus CAN »

Utilisé uniquement par le Ned et le One.

CAN Driver Node (Uniquement pour du développement ou du debuguage)

Le Node ROS est conçu pour :

- Initialiser l'interface CAN.

CAN Interface Core

Il est instancié dans le package [Niryo Robot Hardware Interface](#) ([index.html#document-source/stack/low_level/niryo_robot_hardware_interface](#))

Il a été conçu pour :

- Initialiser l'interface CAN et le bus physique avec les configurations.
- Ajouter, enlever et surveiller les périphériques sur le bus.
- Démarrer la boucle de contrôle pour récupérer et envoyer des données depuis/vers les moteurs.
- Démarrer les services et topics ROS s'ils existent.

Il appartient au namespace ROS `/can_driver/`.

Paramètres**Note**

Ces paramètres de configuration ont été étudiés pour faire fonctionner les robots de Niryo de manière optimale. Il est fortement déconseillé de les éditer.

Paramètres du driver CAN

Nom	Description
<code>can_hardware_control_loop_frequency</code>	Fréquence de la boucle de contrôle. Défaut: "1500.0"
<code>can_hw_write_frequency</code>	Fréquence d'écriture. Défaut: "200.0"
<code>can_hw_read_frequency</code>	Fréquence de lecture. Défaut: "50.0"
<code>bus_params/spi_channel</code>	Canal spi (Serial Peripheral Interface) utilisé pour le bus CAN. Défaut: "0"
<code>bus_params/spi_baudrate</code>	Débit en baud. Défaut: "1000000"
<code>bus_params/gpio_can_interrupt</code>	Pin d'interruption GPIO. Défaut: "25"

Dépendances

- [MCP CAN rpi](#)
- [Niryo_robot_msgs](#)
- [Common](#)
- [std_msgs](#)

Services, Topics et Messages**StepperCmd (Service)**

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

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

StepperMotorHardwareStatus (Message)

```
niryo_robot_msgs/MotorHeader motor_identity

string firmware_version
int32 temperature
int32 voltage
int32 error
```

StepperMotorCommand (Message)

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

uint8[] motors_id
int32[] params
```

StepperArrayMotorHardwareStatus (Message)

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

TTL Debug Tools

Ce package est un package de débogage pour configurer et accéder directement aux composants matériels présents sur le bus TTL. Il expose des fonctions principales comme le ping, le scan de périphérique et les opérations de lecture/écriture/lecture synchrone/écriture synchrone sur les périphériques.

Il y a deux manières d'utiliser ce package : directement avec le binaire compilé, ou via [TTL Driver](#) (index.html#document-source/stack/low_level/ttl_driver) en utilisant les scripts dédiés.

Binaire Ttl debug tool

Le binaire compilé (situé dans `install/lib/ttl_debug_tools/ttl_debug_tools`) accède directement au bus TTL en utilisant la librairie tierce [SDK Dynamixel](#) (index.html#document-source/stack/third_parties/dynamixel_sdk). Il ne peut donc pas être utilisé si la stack ROS Niryo est déjà en train de tourner et vous devez d'abord arrêter la stack du robot (`sudo service niryo_robot_ros stop`)

Cet outils peut être lancé via :

```
roslaunch ttl_debug_tools ttl_debug_tools
```

ou

```
roslaunch ttl_debug_tools ttl_debug_tools
```

Paramètres - Ttl debug tools

- **-help / -h:** Affiche l'aide
- **-baudrate / -b [Baudrate]:** Débit de Baud (1000000 par défaut)
- **-port / -p [Port]:** Indique le port à utiliser
- **-id / -i [ID]:** ID du périphérique (-1 par défaut)
- **-ids [IDs]:** Liste des ids des périphériques
- **-scan:** Scanne tous les périphériques sur le bus
- **-ping:** Ping un id spécifique
- **-get-register [Addr]:** Lire une valeur depuis un registre; le paramètre est : adresse du registre
- **-get-registers [Addr]:** Récupérer la liste des valeurs sur plusieurs périphériques à une adresse de registre, le paramètre est : adresse du registre
- **-get-size [Size]:** Taille des données à lire avec get-register ou get-registers, le paramètre est : taille des données en octets
- **-set-register [Addr] [Value] [Size]:** Ecrire une valeur en registre, les paramètres sont dans l'ordre : adresse du registre / valeur / taille (en octets) des données
- **-set-registers [Addr] [Values] [Size]:** Ecrire des valeurs en registre sur plusieurs périphériques, les paramètres sont dans l'ordre : adresse du registre / liste des valeurs / taille (en octets) des données
- **-calibrate:** Calibre tous les steppers sur le bus. Utilisable sur le NED2 seulement

Scripts

Afin de pouvoir utiliser Ttl debug tools pour déboguer une stack ROS en train de tourner, il était nécessaire de développer un autre outil. Pour cela, deux scripts python ont été développés. Ils permettent d'accéder aux données du bus TTL via deux services implémentés dans le package [TTL Driver](#) (index.html#document-source/stack/low_level/ttl_driver) :

- `read_custom_dx1_value.py` : utilise le service [ReadCustomValue](#) pour lire les valeurs depuis le bus TTL
- `send_custom_dx1_value` : utilise le service [SendCustomValue](#) pour écrire les valeurs sur le bus TTL

Niryo robot - Envoie une valeur personnalisée sur le Dynamixel

Il utilise un service de `ttl_driver` pour envoyer des données sur le registre des périphériques TTL quand la stack ROS tourne. Ce script peut être lancé via :

```
roslaunch ttl_debug_tools send_custom_dx1_value.py
```

paramètres - Send custom value

- **-id [ID]:** ID du périphérique
- **-address [Addr]:** Adresses registres à modifier
- **-value [Value]:** Valeur à écrire sur l'adresse registre donnée
- **-size [Size]:** Taille en octets pour la donnée à écrire

Niryo robot - Read DXL custom value

Il utilise un service pour lire des données sur le registre des périphériques TTL quand la stack ROS tourne. Ce script peut être lancé via :

```
roslaunch ttl_debug_tools read_custom_dx1_value.py
```

paramètres - Read custom value

- **-id [ID]:** ID du périphérique
- **-address [Addr]:** Adresses registres à modifier
- **-size [Size]:** taille en octet des données à lire

CAN Debug Tools

Ce package contient des scripts utilisables pour debugguer et configurer des périphériques CAN. Il expose plusieurs fonctions principales comme l'initialisation du bus CAN et le sniffing de données sur le bus.

Niryo robot - Outils de debuguage

Il fournit des services pour afficher les données circulant sur le bus CAN en temps réel. Ce script peut être lancé via :

```
roslaunch can_debug_tools can_debug_tools
```

Paramètres - Outil de debuguage CAN

- **-help / -h**: Affiche l'aide
- **-baudrate / -b [Baudrate]**: Débit Baud (1000000 par défaut)
- **-channel / -c [Channel]**: Canal SPI à utiliser (0 par défaut)
- **-gpio / -g**: Pin d'interruption GPIO pour le CAN (25 par défaut)
- **-freq / -f**: Fréquence de la boucle de control pour vérifier les données (100Hz par défaut)
- **-dump**: Lance le service de « dump » pour afficher en temps réel toutes les données circulant sur le bus

Quand vous affichez les données circulant sur le bus, le résultat du tableau inclut :

- Nombre de données du package
- Status du package
- Octet de contrôle
- Données sur 8 octets

Paquets tierces

Dans cette section, vous avez accès aux informations sur les libraries tierces de chaque robot Niryo

SDK Dynamixel

Ce paquet a été forké depuis le [package \[dynamixel_sdk\] officiel](https://github.com/ROBOTIS-GIT/DynamixelSDK/) (https://github.com/ROBOTIS-GIT/DynamixelSDK/).

Il a été adapté pour fonctionner avec la carte électronique maison (shield) pour Raspberry Pi 4B, en utilisant la [bibliothèque \[wiringPi\]](http://wiringpi.com/) (http://wiringpi.com/).

MCP CAN rpi

Bibliothèque pour le module MCP2515 pour Raspberry Pi (Interface Bus CAN) via les GPIOs SPI

Forked depuis [la bibliothèque \[MCP_CAN\]](https://github.com/coryjfowler/MCP_CAN_lib) (https://github.com/coryjfowler/MCP_CAN_lib).

Le module MCP2515 est une interface SPI CAN. La bibliothèque MCP_CAN utilise le protocole SPI sur Arduino pour programmer et utiliser ce module. Il a été adapté ici pour fonctionner avec les GPIOs de la Raspberry Pi 4, en utilisant les fonction SPI de la [bibliothèque \[wiringPi\]](http://wiringpi.com/) (http://wiringpi.com/).

—

Une des principales différences est que le pin SPI Chip Select PIN n'est pas gérée. Ceci est déjà mis en oeuvre via la bibliothèque wiringPi, et tous les PINs pour le SPI sont déjà prédéfinis (canal SPI 0 ou 1).

Pour sonder le module MCP2515 (pour voir si il y a des données à lire), la fonction `_digitalRead_` de wiringPi est utilisée.

Paquet ROS Tierces

- `ros_core`
- `moveit`
- `ros_control`
- `roscpp`
- `rostdoc_lite`
- `roslint`
- `rostopic`

Contrôle avec Python ROS Wrapper



Logo Python

Pour contrôler le Ned plus simplement qu'en appelant chaque topic et service l'un après l'autre, une API python d'encapsulation de ROS, appelée dans la suite Python ROS Wrapper, a été construite au dessus de ROS.

Par exemple, un script réalisant un moveJ via Python ROS Wrapper ressemblera à :

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

Ce que fait ce code en arrière plan :

- Il génère un [RobotMove Action Goal](#) et le définit comme une command de joint avec les valeurs de joints correspondantes.
- Envoie un but au Commander Action Server.
- Attend que le Commander Action Server définisse l'Action comme « finished ».
- Vérifie si l'action s'est terminée avec succès.

Dans cette section, vous trouverez quelques exemples d'utilisation de Python ROS Wrapper pour contrôler le Ned, ainsi qu'une documentation complète des fonctions présentes dans Ned Python ROS Wrapper.

🚫 Indication

Le Python ROS Wrapper force l'utilisateur à écrire son code directement dans le robot, ou à minima, copier le code sur le robot via une commande en terminal. Si vous ne souhaitez pas faire cela et lancer directement le code depuis votre ordinateur, vous pouvez utiliser le package Python [PyNiryo](#) ([index.html#pyniryo](#)).

Avant d'exécuter vos programmes

La variable PYTHONPATH

L'interpréteur Python doit avoir tous les packages utilisés dans la variable d'environnement **PYTHONPATH**, pour ce faire, vous devez avoir sourcé votre environnement ROS :

- Si vous codez directement sur votre robot, cela se fait directement dans chaque terminal.
- Si vous utilisez la simulation, assurez-vous d'avoir suivi la configuration dans la rubrique [Installer sous Ubuntu 18](#).

Code requis

Pour fonctionner, votre programme aura besoin d'importations et d'initialisations. Nous vous donnons ci-dessous le code qu'il faut utiliser pour faire fonctionner Python ROS Wrapper :

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

# Imports
from niryo_robot_python_ros_wrapper import *
import rospy

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

niryo_robot = NiryoRosWrapper()

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

Vous avez maintenant tout ce dont vous avez besoin pour contrôler le robot via son Python ROS Wrapper. Pour exécuter un script, utilisez simplement la commande `python my_script.py`.

Exemples : Les principes de base

Dans ce fichier, deux programmes courts sont implémentés et commentés afin de vous aider à comprendre la philosophie derrière Python ROS Wrapper.

🚫 Danger

Si vous utilisez le vrai robot, assurez-vous que l'environnement autour est dégagé.

Votre premier mouvement d'axes

L'exemple suivant montre un premier cas d'utilisation. C'est un simple MoveJ.

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

# Imports
from niryo_robot_python_ros_wrapper import *
import rospy

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

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

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

Détails du code - Premier MoveJ

Tout d'abord il faut indiquer au système que nous exécutons un script Python :

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

Ensuite, nous importons le package API pour pouvoir accéder aux fonctions :

```
from niryo_robot_python_ros_wrapper import *
```

Ensuite, il est nécessaire d'installer un nœud ROS afin de communiquer avec le ROS master :

```
import rospy

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

Nous lançons une instance **NiryoRosWrapper** ([index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper](#)) :

```
niryo_robot = NiryoRosWrapper()
```

Une fois la connexion établie, nous calibrons le robot à l'aide de sa fonction **calibrate_auto()** ([index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper.calibrate_auto](#)) :

```
niryo_robot.calibrate_auto()
```

Maintenant que le robot est calibré, nous pouvons faire un déplacement d'axes en donnant les 6 positions des axes en radians. Pour ce faire, nous utilisons **move_joints()** ([index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper.move_joints](#)):

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

Votre premier pick and place

Pour notre deuxième exemple, nous allons développer un algorithme de pick and place :

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

# Imports
from niryo_robot_python_ros_wrapper import *
import rospy

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

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

# Updating tool
niryo_robot.update_tool()

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

Détails du code - Premier pick and place

Tout d'abord, il faut effectuer les importations et démarrer un nœud ROS :

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

from niryo_robot_python_ros_wrapper import *
import rospy

rospy.init_node('niryo_robot_example_python_ros_wrapper')
```

Ensuite, il est nécessaire de créer une instance **NiryoRosWrapper** ([index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper](#)) et de calibrer le robot :

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

Ensuite, il vous faut équiper l'outil avec **update_tool()** ([index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper.update_tool](#)) :

```
niryo_robot.update_tool()
```

Maintenant que notre initialisation est terminée, nous pouvons ouvrir le Gripper (ou activer la Pompe à vide) avec **release_with_tool()** ([index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper.release_with_tool](#)), aller à la position de picking via **move_pose()** ([index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper.move_pose](#)) et ensuite attraper l'objet avec **grasp_with_tool()** ([index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper.grasp_with_tool](#)) !

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

Nous voilà maintenant en position finale (place position) et le robot prêt à placer l'objet.

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

Notes - Exemples basiques

Si vous n'avez pas bien compris comment déplacer le robot et utiliser les outils de Ned, c'est tout à fait normal. Vous trouverez plus de détails sur les autres pages d'exemples !

La chose importante à retenir de cette page est de savoir comment importer la bibliothèque et de se connecter au robot.

Exemples : Les mouvements

Ce document montre comment contrôler Ned afin de déplacer les axes et de changer la position de Ned.

Si vous désirez en voir davantage, vous pouvez regarder la rubrique [API - Joints & Pose](#) (index.html#joints-pose).

⚠ Danger

Si vous utilisez le vrai robot, assurez-vous que l'environnement autour est dégagé.

Axes

Pour effectuer un moveJ (déplacement d'axes), il est nécessaire de passer 6 floats : (j1, j2, j3, j4, j5, j6) vers la méthode `move_joints()` (index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper.move_joints) :

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

# Imports
from niryo_robot_python_ros_wrapper import *
import rospy

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

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

Pour obtenir les axes, il faut utiliser `get_joints()` (index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper.get_joints) :

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

Position

Pour effectuer un moveP, Il est nécessaire de passer 6 floats : (x, y, z, roll, pitch, yaw) à la méthode `move_pose()` (index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper.move_pose).

Voir l'exemple ci-dessous :

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

# Imports
from niryo_robot_python_ros_wrapper import *
import rospy

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

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

Pour obtenir les positions, il faut utiliser `get_pose()` (index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper.get_pose) :

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

Exemples : Les actions avec les outils

Cette page montre comment contrôler les outils de Ned via le Python ROS Wrapper.

Si vous voulez en savoir plus, vous pouvez consulter l'[API - Tools](#) (index.html#tools).

⚠ Danger

Si vous utilisez le vrai robot, assurez-vous que l'environnement autour est dégagé.

Contrôler un outil

Équiper un outil

Afin d'utiliser un outil, celui-ci doit être branché mécaniquement sur le robot mais aussi connecté au niveau logiciel.

Pour effectuer cela, utiliser la fonction **update_tool()** ([index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper.update_tool](#)) qui ne prend aucun argument. Il analysera les connexions du moteur et définira le nouvel outil !

La ligne pour équiper un nouvel outil est :

```
niryo_robot.update_tool()
```

Saisir un objet

Pour saisir ou attraper un objet avec un outil, utiliser la fonction : **grasp_with_tool()** ([index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper.grasp_with_tool](#)). Cette action correspond à :

- Fermer le Gripper
- Activer la pompe à vide
- Activer l'Électroaimant

Le code pour saisir est :

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

# Imports
from niryo_robot_python_ros_wrapper import *
import rospy

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

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

# Updating tool
niryo_robot.update_tool()

# Grasping
niryo_robot.grasp_with_tool()
```

Saisir en spécifiant des paramètres :

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

# Imports
from niryo_robot_python_ros_wrapper import *
import rospy

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

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

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

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

Relâcher

Pour relâcher un objet avec un outil, utiliser la fonction : **release_with_tool()** ([index.html#niryo_robot_python_ros_wrapper.ros_wrapper.NiryoRosWrapper.release_with_tool](#)). Cette action correspond à :

- Ouvrir le Gripper
- Désactiver la Pompe à vide
- Désactiver l'Électroaimant

La ligne de code pour relâcher est :

```
niryo_robot.release_with_tool()
```

Relâcher en spécifiant des paramètres :

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

# Imports
from niryo_robot_python_ros_wrapper import *
import rospy

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

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

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

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

Pick and place avec un outil

Il existe de nombreuses façons de réaliser un pick and place avec le ROS Wrapper. Les méthodes seront présentées du plus bas au plus haut niveau.

Le code utilisé sera :

```
# Imports
from niryo_robot_python_ros_wrapper import *

gripper_used = ToolID.XXX # Tool used for picking

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

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

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

Première solution : la plus lourde

Tout est fait à la main :

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

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

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

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

    # Going Over Place pose
    niryo_ned.move_pose(place_pose[0], place_pose[1], place_pose[2] + height_offset,
                        place_pose[3], place_pose[4], place_pose[5])
    # Going to Place pose
    niryo_ned.move_pose(place_pose[0], place_pose[1], place_pose[2],
                        place_pose[3], place_pose[4], place_pose[5])

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

Deuxième solution : pick from pose & place from pose functions

Il est nécessaire d'utiliser des fonctions prédéfinies :

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

Troisième solution : Le tout en un

Nous utilisons LA fonction prédéfinie :

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


Exemples : Le Convoyeur

Ce document montre comment utiliser le Convoyeur Ned.

Si vous voulez en savoir plus sur les fonctions du Convoyeur Ned, vous pouvez consulter l'[API - Convoyeur](#).

Note

Les importations et l'initialisation ne sont pas mentionnées, mais il ne faut pas les oublier !

Contrôle simple du Convoyeur

Ce court exemple montre comment connecter un Convoyeur, activer la connexion et lancer son moteur :

```
niryo_robot = NiryoRosWrapper()

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

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

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

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

Contrôle avancé du Convoyeur

Cet exemple montre comment effectuer une certaine quantité de pick & place en utilisant le Convoyeur avec le capteur infrarouge :

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

# -- Setting variables
sensor_pin_id = PinID.GPIO1A

catch_nb = 5

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

# -- MAIN PROGRAM

niryo_robot = NiryoRosWrapper()

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

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

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

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

Exemples : La vision

Ce document montre comment utiliser le Set Vision de Ned.

Si vous voulez en savoir plus sur les fonctions vision de Ned, vous pouvez consulter l'[API - Vision \(index.html#vision\)](#).

Au préalable

Pour réaliser les exemples suivants, vous devez avoir créé un espace de travail.

Comme les exemples commencent toujours de la même manière, il est nécessaire d'ajouter le code au début de chacun d'entre eux :

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

# Imports
from niryo_robot_python_ros_wrapper import *
import rospy

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

niryo_robot = NiryoRosWrapper()

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

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

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



Simple Vision pick

Ce court exemple montre comment faire votre premier vision pick :

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

niryo_robot.set_learning_mode(True)
```

Exemples: Repères dynamiques

Ce document montre comment utiliser les repères dynamiques.

Si vous voulez en savoir plus sur les fonctions des repères dynamiques, vous pouvez regarder [API - Dynamic frames](#) (index.html#dynamic-frames)

⚠ Danger

Si vous utilisez un robot réel, assurez vous que son environnement est sans obstacles

Contrôle simple des repères dynamiques

Cet exemple montre comment créer un repère et effectuer un petit pick & place dans ce repère:

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

# Imports
from niryo_robot_python_ros_wrapper import *
import rospy

gripper_speed = 400

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

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

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

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

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

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

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

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

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

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

Documentation Python ROS Wrapper

Ce fichier présente les différentes fonctions, classes et énumérations disponibles avec l'API.

- [Les fonctions de l'API](#)
- [Énumération](#)

Les fonctions de l'API

Cette classe vous permet de contrôler le robot via l'API interne. Par le terme contrôler, nous signifions :

- Changer la position du robot.
- Utiliser la Vision.
- Contrôler le Convoyeur.
- Jouer avec du matériel.

Liste des sous-sections des fonctions :

- [Fonctions majeures](#)
- [Axe et Pose](#)
- [Positions sauvegardées](#)
- [Pick & place](#)
- [Trajectoires](#)
- [Repères dynamiques](#)
- [Outils](#)
- [Matériel](#)
- [Convoyeur](#)
- [Vision](#)
- [Son](#)
- [Anneau LED](#)
- [Bouton "Custom"](#)

Fonctions majeures

class NiryoRosWrapper

calibrate_auto()

Appelle le service pour calibrer les moteurs, puis attend que la calibration soit terminée. Si ce n'est pas réussi, soulève une exception NiryoRosWrapperException

Renvoie:

Statut, message

Type renvoyé:

(int, str)

calibrate_manual()

Appelle le service pour calibrer les moteurs, puis attend que la calibration soit terminée. Si ce n'est pas réussi, soulève une exception `NiryoRosWrapperException`

Renvoie: Statut, message

Type renvoyé: `(int, str)`

`get_learning_mode()`

Utilise `/niryo_robot/learning_mode/state` topic subscriber pour obtenir le statut du mode apprentissage

Renvoie: `True` si activé, sinon `False`

Type renvoyé: `bool`

`set_learning_mode(set_bool)`

Appelle le service `set_learning_mode` pour activer ou désactiver le mode apprentissage en fonction de `set_bool`. Si échec, soulève une exception `NiryoRosWrapperException`

Paramètres: `set_bool (bool)` – `True` pour activer, `False` pour désactiver

Renvoie: Statut, message

Type renvoyé: `(int, str)`

`set_arm_max_velocity(percentage)`

Définir la vitesse relative maximum (en %)

Paramètres: `percentage (int)` – Pourcentage de la vitesse maximum

Renvoie: Statut, message

Type renvoyé: `(int, str)`

Axe et Pose

`class NiryoRosWrapper`

`get_joints()`

Récupération de la valeurs des axes via le topic `/joint_states`

Renvoie: Liste des valeurs des axes

Type renvoyé: `list[float]`

`get_pose()`

Récupération de la position de l'effecteur via le topic `/niryo_robot/robot_state`

Renvoie: Objet `RobotState` [position.x/y/z ; rpy.roll/pitch/yaw ; orientation.x/y/z]

Type renvoyé: `RobotState`

`get_pose_as_list()`

Récupération de la position de l'effecteur via le topic `/niryo_robot/robot_state`

Renvoie: Une liste correspondant à [x, y, z, roll, pitch, yaw]

Type renvoyé: `list[float]`

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

Réalise une action de mouvement d'axes

Paramètres:

- `j1 (float)` –
- `j2 (float)` –
- `j3 (float)` –
- `j4 (float)` –
- `j5 (float)` –
- `j6 (float)` –

Renvoie: Statut, message

Type renvoyé: `(int` (<https://docs.python.org/3/library/functions.html#int>), `str` (<https://docs.python.org/3/library/stdtypes.html#str>))

`move_to_sleep_pose()`

Déplace le robot à la position de repos ce qui permet d'activer le mode apprentissage sans risque que le robot touche quelque chose

Renvoie: Statut, message

Type renvoyé: (int, str)

move_pose(x, y, z, roll, pitch, yaw, frame="")

Déplace l'effecteur à la position [x, y, z, roll, pitch, yaw], dans un repère particulier si précisé

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

Renvoie: Statut, message

Type renvoyé: (int (<https://docs.python.org/3/library/functions.html#int>), str (<https://docs.python.org/3/library/stdtypes.html#str>))

shift_pose(axis, value)

Réalise une action de déplacement de l'effecteur par glissement

- Paramètres:**
- **axis** (*ShiftPose*) – Valeur de l'énumérations correspondant à l'axe de glissement
 - **value** (*float*) – Valeur du glissement

Renvoie: Statut, message

Type renvoyé: (int (<https://docs.python.org/3/library/functions.html#int>), str (<https://docs.python.org/3/library/stdtypes.html#str>))

shift_linear_pose(axis, value)

Réalise un glissement de façon linéaire

- Paramètres:**
- **axis** (*ShiftPose*) – Valeur de l'énumérations correspondant à l'axe de glissement
 - **value** (*float*) – Valeur du glissement

Renvoie: Statut, message

Type renvoyé: (int (<https://docs.python.org/3/library/functions.html#int>), str (<https://docs.python.org/3/library/stdtypes.html#str>))

move_linear_pose(x, y, z, roll, pitch, yaw, frame="")

Déplacement de l'effecteur du robot à la position [x, y, z, roll, pitch, yaw] suivant une trajectoire linéaire, dans un repère particulier si précisé

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

Renvoie: Statut, message

Type renvoyé: (int (<https://docs.python.org/3/library/functions.html#int>), str (<https://docs.python.org/3/library/stdtypes.html#str>))

set_jog_use_state(state)

Active ou désactive le controle direct

Paramètres: **state** (*bool*) – **True** quand activé, **False** sinon

Renvoie: Statut, message

Type renvoyé: (int, str)

jog_joints_shift(shift_values)

Réalise un control directe sur la position des axes

Paramètres: **shift_values** (*list(float)*) – La liste correspondant au déplacement à appliquer à chaque axe

Renvoie: Statut, message

Type renvoyé: (int, str)

jog_pose_shift(shift_values)

Réalise un control direct sur la position de l'effecteur

Paramètres: **shift_values** (*list(float)*) – La liste correspondant au déplacement à appliquer à la position

Renvoie: Statut, message

Type renvoyé: (int, str)

forward_kinematics(j1, j2, j3, j4, j5, j6)

Calcule la cinématique directe

Paramètres:

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

Renvoie: Une liste correspondant à [x, y, z, roll, pitch, yaw]

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

inverse_kinematics(x, y, z, roll, pitch, yaw)

Calcule la cinématique inverse

Paramètres:

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

Renvoie: Liste des valeurs des axes

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

Positions sauvegardées

class NiryoRosWrapper

move_pose_saved(pose_name)

Déplace l'effecteur du robot à une position sauvegardées

Paramètres: **pose_name** (str) –

Renvoie: Statut, message

Type renvoyé: (int, str)

get_pose_saved(pose_name)

Récupération des position sauvegardées dans le robot. Renvoie une erreur si la position n'existe pas

Paramètres: **pose_name** (str) – Nom de la position

Renvoie: x, y, z, roulis, tangage, lacet

Type renvoyé: [tuple](#)[float]

save_pose(name, x, y, z, roll, pitch, yaw)

Sauvegarde de la position dans le robot

Paramètres:

- **name** (str) –
- **x** (float) –
- **y** (float) –
- **z** (float) –
- **roll** (float) –
- **pitch** (float) –
- **yaw** (float) –

Renvoie: Statut, message

Type renvoyé: ([int](https://docs.python.org/3/library/functions.html#int)), ([str](https://docs.python.org/3/library/stdtypes.html#str))

delete_pose(name)

Supprime la position

Paramètres: **name** (str) –

Renvoie: Statut, message

Type renvoyé: (int, str)

get_saved_pose_list(with_desc=False)

Renvoie la liste des positions sauvegardées et leur description

Paramètres: **with_desc** (*bool*) – Si Vrai renvoie les descriptions des positions

Renvoie: La liste des noms des positions

Type renvoyé: list[str]

Pick & place

class NiryoRosWrapper

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

Réalise un prélèvement depuis une position. Un prélèvement est décrit comme : - aller au dessus de l'objet - descendre jusqu'à l'hauteur z - attraper avec l'outil - retourner au dessus de l'objet

Paramètres:

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

Renvoie: Statut, message

Type renvoyé: (int (<https://docs.python.org/3/library/functions.html#int>), str (<https://docs.python.org/3/library/stdtypes.html#str>))

place_from_pose(x, y, z, roll, pitch, yaw)

Réalise un placement à partir d'une position. Un placement est décrit comme : - aller au dessus du placement - descendre jusqu'à la hauteur z - relâcher l'objet avec l'outil - retourner au dessus du placement

Paramètres:

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

Renvoie: Statut, message

Type renvoyé: (int (<https://docs.python.org/3/library/functions.html#int>), str (<https://docs.python.org/3/library/stdtypes.html#str>))

pick_and_place(pick_pose, place_pose, dist_smoothing=0.0)

Réalise un prélèvement puis un placement. Si une erreur se produit, une exception est levée

Paramètres:

- **pick_pose** (*list[float]*) –
- **place_pose** (*list[float]*) –
- **dist_smoothing** (*float*) – Distance entre les points de passage avant de lisser la trajectoire

Renvoie: Statut, message

Type renvoyé: (int (<https://docs.python.org/3/library/functions.html#int>), str (<https://docs.python.org/3/library/stdtypes.html#str>))

Trajectoires

class NiryoRosWrapper

get_trajectory_saved(trajectory_name)

Récupération des trajectoires sauvegardées dans le robot. Lève une exception si la position n'existe pas

Paramètres: **trajectory_name** (*str*) –

Lève: **NiryoRosWrapperException** – si la trajectoire n'existe pas

Renvoie: Liste de [x, y, z, qx, qy, qz, qw]

Type renvoyé: list[list[float]]

get_saved_trajectory_list()

Renvoie la liste des trajectoires sauvegardées

Renvoie: Liste des noms des trajectoires

Type renvoyé: `list[str]`

`execute_trajectory_from_poses(list_poses_raw, dist_smoothing=0.0)`

Réalise un trajectoire à partir d'une liste de position

- Paramètres:**
- **list_poses_raw** (`list[list[float]]`) – Liste de [x, y, z, qx, qy, qz, qw] ou liste de [x, y, z, roll, pitch, yaw]
 - **dist_smoothing** (`float`) – Distance entre les points de passage avant de lisser la trajectoire

Renvoie: Statut, message

Type renvoyé: (`int` (<https://docs.python.org/3/library/functions.html#int>), `str` (<https://docs.python.org/3/library/stdtypes.html#str>))

`execute_trajectory_from_poses_and_joints(list_pose_joints, list_type=None, dist_smoothing=0.0)`

Réalise une trajectoire à partir d'une liste de positions ou d'axes

- Paramètres:**
- **list_pose_joints** (`list[list[float]]`) – Liste de [x,y,z,qx,qy,qz,qw] ou liste de [x,y,z,roll,pitch,yaw] ou liste de [j1,j2,j3,j4,j5,j6]
 - **list_type** (`list[string]`) – Liste de string de valeur "pose" ou "joint", ou ["pose"] (si ce n'est que des positions) ou ["joint"] (si que des axes). Sans valeur, la fonction concidère qu'il n'y a que des positions dans la liste.
 - **dist_smoothing** (`float`) – Distance entre les points de passage avant de lisser la trajectoire

Renvoie: Statut, message

Type renvoyé: (`int` (<https://docs.python.org/3/library/functions.html#int>), `str` (<https://docs.python.org/3/library/stdtypes.html#str>))

`save_trajectory(trajectory_points, trajectory_name, trajectory_description)`

Sauvegarde de la trajectoire

- Paramètres:**
- **trajectory_name** (`str`) – Nom de la trajectoire
 - **trajectory_points** (`list[trajectory_msgs/trajectory_point]`) – Liste des trajectory_msgs/JointTrajectoryPoint

Renvoie: Statut, message

Type renvoyé: (`int` (<https://docs.python.org/3/library/functions.html#int>), `str` (<https://docs.python.org/3/library/stdtypes.html#str>))

`delete_trajectory(trajectory_name)`

Supprime une trajectoire

Paramètres: **trajectory_name** (`str`) – nom

Renvoie: Statut, message

Type renvoyé: (`int`, `str`)

Repères dynamiques

`class NiryoRosWrapper`

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

Crée un repère dynamique par 3 positions (origine, x, y)

- Paramètres:**
- **frame_name** (`str`) – nom du repère
 - **description** (`str`) – description du repère
 - **list_robot_poses** (`list[list[float]]`) – 3 positions nécessaires pour créer le repère
 - **belong_to_workspace** (`boolean`) – indique si le repère appartient à un espace de travail

Renvoie: Statut, message

Type renvoyé: (`int` (<https://docs.python.org/3/library/functions.html#int>), `str` (<https://docs.python.org/3/library/stdtypes.html#str>))

`save_dynamic_frame_from_points(frame_name, description, list_points, belong_to_workspace=False)`

Crée un repère dynamique par 3 positions (origine, x, y)

- Paramètres:**
- **frame_name** (`str`) – nom du repère
 - **description** (`str`) – description du repère
 - **list_points** (`list[list[float]]`) – 3 positions nécessaires pour créer le repère
 - **belong_to_workspace** (`boolean`) – indique si le repère appartient à un espace de travail

Renvoie: Statut, message

Type renvoyé: (`int` (<https://docs.python.org/3/library/functions.html#int>), `str` (<https://docs.python.org/3/library/stdtypes.html#str>))

`edit_dynamic_frame(frame_name, new_frame_name, new_description)`

Modifie un repère

Paramètres:

- **frame_name** (*str*) – nom du repère
- **new_frame_name** (*str*) – nouveau nom du repère
- **new_description** (*str*) – nouvelle description du repère

Renvoie: Statut, message

Type renvoyé: ([int](https://docs.python.org/3/library/functions.html#int) (<https://docs.python.org/3/library/functions.html#int>), [str](https://docs.python.org/3/library/stdtypes.html#str) (<https://docs.python.org/3/library/stdtypes.html#str>))

`delete_dynamic_frame(frame_name, belong_to_workspace=False)`

Supprime un repère

Paramètres:

- **frame_name** (*str*) – nom du repère dynamique à supprimer
- **belong_to_workspace** (*boolean*) – indique si le repère appartient à un espace de travail

Renvoie: Statut, message

Type renvoyé: ([int](https://docs.python.org/3/library/functions.html#int) (<https://docs.python.org/3/library/functions.html#int>), [str](https://docs.python.org/3/library/stdtypes.html#str) (<https://docs.python.org/3/library/stdtypes.html#str>))

`get_saved_dynamic_frame(frame_name)`

Obtient le nom, la description et la position d'un repère dynamique

Paramètres: **frame_name** (*str*) – nom du repère

Renvoie: Obtient le nom, la description et la position d'un repère dynamique

Type renvoyé: [list](#)[[str](#), [str](#), [list](#)[[float](#)]]

`get_saved_dynamic_frame_list()`

Obtient la liste des repères dynamiques sauvegardés

Renvoie: liste des noms des repères dynamiques, liste des descriptions des repères dynamiques

Type renvoyé: [list](#)[[str](#)], [list](#)[[str](#)]

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

Déplace l'effecteur d'un offset dans un repère

Paramètres:

- **offset** ([list](#)[[float](#)]) – Une liste correspondant à [x, y, z, roll, pitch, yaw]
- **frame** (*str*) – nom du repère

Renvoie: Statut, message

Type renvoyé: ([int](https://docs.python.org/3/library/functions.html#int) (<https://docs.python.org/3/library/functions.html#int>), [str](https://docs.python.org/3/library/stdtypes.html#str) (<https://docs.python.org/3/library/stdtypes.html#str>))

`move_linear_relative(offset, frame='world')`

Déplace l'effecteur avec un mouvement linéaire d'un offset dans un repère

Paramètres:

- **offset** ([list](#)[[float](#)]) – Une liste correspondant à [x, y, z, roll, pitch, yaw]
- **frame** (*str*) – nom du repère

Renvoie: Statut, message

Type renvoyé: ([int](https://docs.python.org/3/library/functions.html#int) (<https://docs.python.org/3/library/functions.html#int>), [str](https://docs.python.org/3/library/stdtypes.html#str) (<https://docs.python.org/3/library/stdtypes.html#str>))

Outils

`class NiryoRosWrapper`

`get_current_tool_id()`

Récupération de l'id de l'outil courant via le topic /niryo_robot_tools_commander/current_id

Renvoie: Id de l'outil

Type renvoyé: [ToolID](#)

`update_tool()`

Mise à jour de l'outil en appelant le service niryo_robot_tools_commander/update_tool

Renvoie: Statut, message

Type renvoyé: ([int](#), [str](#))

`grasp_with_tool(pin_id="")`

Attrape avec l'outil lié à l'id de l'outil. L'action correspond à - Fermer la pince pour une pince - Tirer l'air pour la pompe à vide - Activer l'électroaimant

Paramètres: **pin_id** (*PinID*) – [Seulement pour l'électroaimant] L'ID de la broche de l'électroaimant

Renvoie: Statut, message

Type renvoyé: (*int*, *str*)

release_with_tool(pin_id="")

Relâcher avec l'outil lié à l'id de l'outil. L'action correspond à - Ouvri la pince pour une pince - Expulser l'air pour la pompe à vide - Désactiver l'électroaimant

Paramètres: **pin_id** (*PinID*) – [Seulement pour l'électroaimant] L'ID de la broche de l'électroaimant

Renvoie: Statut, message

Type renvoyé: (*int*, *str*)

open_gripper(speed=500, max_torque_percentage=100, hold_torque_percentage=20)

Ouvrir la pince avec une vitesse "speed"

Paramètres:

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

Renvoie: Statut, message

Type renvoyé: (*int* (<https://docs.python.org/3/library/functions.html#int>), *str* (<https://docs.python.org/3/library/stdtypes.html#str>))

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

Fermer pince avec une vitesse "speed"

Paramètres:

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

Renvoie: Statut, message

Type renvoyé: (*int* (<https://docs.python.org/3/library/functions.html#int>), *str* (<https://docs.python.org/3/library/stdtypes.html#str>))

pull_air_vacuum_pump()

Tirer l'air

Renvoie: Statut, message

Type renvoyé: (*int*, *str*)

push_air_vacuum_pump()

Tirer l'air

Renvoie: Statut, message

Type renvoyé: (*int*, *str*)

setup_electromagnet(pin_id)

Installation de l'électroaimant sur la pin

Paramètres: **pin_id** (*PinID*) – Id de la broche

Renvoie: Statut, message

Type renvoyé: (*int*, *str*)

activate_electromagnet(pin_id)

Désactiver l'électroaimant associé à electromagnet_id sur pin_id

Paramètres: **pin_id** (*PinID*) – Id de la broche

Renvoie: Statut, message

Type renvoyé: (*int*, *str*)

deactivate_electromagnet(pin_id)

Désactiver l'électroaimant associé à electromagnet_id sur pin_id

Paramètres: **pin_id** (*PinID*) – Id de la broche

Renvoie: Statut, message

Type renvoyé: (int, str)

enable_tcp(enable=True)

Active ou désactive la fonctions PCO (Point Central Objet). Si activé, la dernière valeur du PCO sera appliquée. La valeur par défaut dépend de l'outil équipé. Si désactivé, le PCO coïncidera avec le tool_link (point d'attache de l'outil sur le bras)

Paramètres: enable (Bool) – True pour activer, sinon false

Renvoie: Statut, message

Type renvoyé: (int, str)

set_tcp(x, y, z, roll, pitch, yaw)

Active la fonction PCO (Point Central Objet) et définit la transformation entre le repère tool_link et le repère du PCO

Paramètres:

- x (float) –
- y (float) –
- z (float) –
- roll (float) –
- pitch (float) –
- yaw (float) –

Renvoie: Statut, message

Type renvoyé: (int (<https://docs.python.org/3/library/functions.html#int>), str (<https://docs.python.org/3/library/stdtypes.html#str>))

reset_tcp()

Réinitialise la transformation du PCO (Point Central Objet). Le PCO est réinitialisé selon l'outil équipé

Renvoie: Statut, message

Type renvoyé: (int, str)

Matériel

class NiryoRosWrapper

set_pin_mode(pin_id, pin_mode)

Initialise le numéro de broche pin_id dans le mode pin_mode

Paramètres:

- pin_id (PinID) –
- pin_mode (PinMode) –

Renvoie: Statut, message

Type renvoyé: (int (<https://docs.python.org/3/library/functions.html#int>), str (<https://docs.python.org/3/library/stdtypes.html#str>))

digital_write(pin_id, digital_state)

Initialise l'état de la broche pin_id à pin_state

Paramètres:

- pin_id (Union[PinID, str]) – Nom de la broche
- digital_state (Union[PinState, bool]) –

Renvoie: Statut, message

Type renvoyé: (int (<https://docs.python.org/3/library/functions.html#int>), str (<https://docs.python.org/3/library/stdtypes.html#str>))

digital_read(pin_id)

Lecture de l'état pin_state de la broche pin_id

Paramètres: pin_id (Union[PinID, str]) – Nom de la broche

Renvoie: état

Type renvoyé: PinState

get_digital_io_state()

Récupération de l'état des Entres/Sorties digitales: nom, mode, état

Renvoie: Informations contenues dans l'objet IOState (voir niryo_robot_msgs)

Type renvoyé: IOState

get_hardware_status()

Récupération de l'état du matériel : Température, version matériel, nom des moteurs et types

Renvoie: Informations contenues dans l'object `HardwareStatus` (voir `niryo_robot_msgs`)

Type renvoyé: `HardwareStatus`

Convoyeur

class `NiryoRosWrapper`

`set_conveyor()`

Analyse la présence d'un convoyeur sur le bus can. Si un convoyeur est détecté, renvoie l'id du convoyeur

Lève: `NiryoRosWrapperException` –

Renvoie: ID

Type renvoyé: `ConveyorID`

`unset_conveyor(conveyor_id)`

Retire un convoyeur spécifique

Paramètres: `conveyor_id` (`ConveyorID`) – Essentiellement, `ConveyorID.ONE` ou `ConveyorID.TWO`

Lève: `NiryoRosWrapperException` –

Renvoie: Statut, message

Type renvoyé: (`int`, `str`)

`control_conveyor(conveyor_id, bool_control_on, speed, direction)`

Controle du convoyeur associé à `conveyor_id`. Stop si `bool_control_on` est False, sinon rafraichit la vitesse et la direction

- Paramètres:**
- `conveyor_id` (`ConveyorID`) – `ConveyorID.ID_1` ou `ConveyorID.ID_2`
 - `bool_control_on` (`bool`) – True pour activer, False pour désactiver
 - `speed` (`int`) – vitesse ciblée
 - `direction` (`ConveyorDirection`) – Direction ciblées

Renvoie: Statut, message

Type renvoyé: (`int` (<https://docs.python.org/3/library/functions.html#int>), `str` (<https://docs.python.org/3/library/stdtypes.html#str>))

Vision

class `NiryoRosWrapper`

`get_compressed_image(with_seq=False)`

Récupération de la dernière image au format compressé

Renvoie: chaine de caractère contenant l'image compressé en JPEG

Type renvoyé: `str`

`set_brightness(brightness_factor)`

Modifie la luminosité de l'image

Paramètres: `brightness_factor` (`float`) – Comment ajuster la luminosité. 0.5 donnera une image assombrie, 1 l'image d'origine et 2 augmentera la luminosité par un facteur 2

Renvoie: Statut, message

Type renvoyé: (`int`, `str`)

`set_contrast(contrast_factor)`

Modifie le contraste de l'image

Paramètres: `contrast_factor` (`float`) – Plus le facteur est proche de 0, plus l'image deviendra grise

Renvoie: Statut, message

Type renvoyé: (`int`, `str`)

`set_saturation(saturation_factor)`

Modifie la saturation de l'image

Paramètres: `saturation_factor` (`float`) – Le facteur permet de modifier la valeur de saturation appliquée à l'image.

Renvoie: Statut, message

Type renvoyé: (int, str)

get_target_pose_from_rel(workspace_name, height_offset, x_rel, y_rel, yaw_rel)

Pour une pose (x_rel, y_rel, yaw_rel) relative à l'espace de travail, retourne la position du robot dans laquelle l'outil courant sera capable de prendre un objet. L'argument height_offset (en mètre) définit à quelle hauteur l'outil sera placé au dessus de l'espace de travail. Si height_offset=0, l'outil touchera presque l'espace de travail.

Paramètres:

- **workspace_name** (*str*) – nom de l'espace de travail
- **height_offset** (*float*) – Décalage entre l'espace de travail et l'hauteur ciblée
- **x_rel** (*float*) –
- **y_rel** (*float*) –
- **yaw_rel** (*float*) –

Renvoie: target_pose

Type renvoyé: RobotState

get_target_pose_from_cam(workspace_name, height_offset, shape, color)

Commence par détecter l'objet spécifié en utilisant la caméra puis retourne la position du robot dans laquelle l'objet peut être pris avec l'outil courant

Paramètres:

- **workspace_name** (*str*) – nom de l'espace de travail
- **height_offset** (*float*) – Décalage entre l'espace de travail et l'hauteur ciblée
- **shape** (*ObjectShape*) – Forme de la cible
- **color** (*ObjectColor*) – Couleur de la cible

Renvoie: object_found, object_pose, object_shape, object_color

Type renvoyé: ([bool](https://docs.python.org/3/library/functions.html#bool) (<https://docs.python.org/3/library/functions.html#bool>)), [RobotState](https://docs.python.org/3/library/stdtypes.html#str), [str](https://docs.python.org/3/library/stdtypes.html#str) (<https://docs.python.org/3/library/stdtypes.html#str>), [str](https://docs.python.org/3/library/stdtypes.html#str) (<https://docs.python.org/3/library/stdtypes.html#str>)

vision_pick_w_obs_joints(workspace_name, height_offset, shape, color, observation_joints)

Déplace les axes à observation_joints, puis réalise une prise par vision

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

Déplace la position à observation_pose, puis réalise une prise par vision

vision_pick(workspace_name, height_offset, shape, color)

Prise d'un objet spécifique dans un espace de travail. Les phases 1. détection de l'objet via la caméra 2. préparation de l'outil courant pour la prise 3. rapprochement vers l'objet 4. descente jusqu'à la position de prise 5. Actionne l'outil courant 6. soulève l'outil

Paramètres:

- **workspace_name** (*str*) – nom de l'espace de travail
- **height_offset** (*float*) – Décalage entre l'espace de travail et l'hauteur ciblée
- **shape** (*ObjectShape*) – Forme de la cible
- **color** (*ObjectColor*) – Couleur de la cible

Renvoie: object_found, object_shape, object_color

Type renvoyé: ([bool](https://docs.python.org/3/library/functions.html#bool) (<https://docs.python.org/3/library/functions.html#bool>)), [ObjectShape](#) (index.html#niryo_robot_python_ros_wrapper.ros_wrapper_enums.ObjectShape), [ObjectColor](#) (index.html#niryo_robot_python_ros_wrapper.ros_wrapper_enums.ObjectColor)

move_to_object(workspace, height_offset, shape, color)

idem que *get_target_pose_from_cam* mais déplacement direct vers la position

Paramètres:

- **workspace** (*str*) – nom de l'espace de travail
- **height_offset** (*float*) – Décalage entre l'espace de travail et l'hauteur ciblée
- **shape** (*ObjectShape*) – Forme de la cible
- **color** (*ObjectColor*) – Couleur de la cible

Renvoie: object_found, object_shape, object_color

Type renvoyé: ([bool](https://docs.python.org/3/library/functions.html#bool) (<https://docs.python.org/3/library/functions.html#bool>)), [ObjectShape](#) (index.html#niryo_robot_python_ros_wrapper.ros_wrapper_enums.ObjectShape), [ObjectColor](#) (index.html#niryo_robot_python_ros_wrapper.ros_wrapper_enums.ObjectColor)

detect_object(workspace_name, shape, color)

Paramètres:

- **workspace_name** (*str*) – nom de l'espace de travail
- **shape** (*ObjectShape*) – Forme de la cible
- **color** (*ObjectColor*) – Couleur de la cible

Renvoie: object_found, object_pose, object_shape, object_color

Type renvoyé: ([bool](https://docs.python.org/3/library/functions.html#bool) (<https://docs.python.org/3/library/functions.html#bool>), [RobotState](#), [str](https://docs.python.org/3/library/stdtypes.html#str) (<https://docs.python.org/3/library/stdtypes.html#str>), [str](https://docs.python.org/3/library/stdtypes.html#str) (<https://docs.python.org/3/library/stdtypes.html#str>))

`get_camera_intrinsics()`

Récupération de l'objet calibration : paramètres intrinsèques, coefficients de distortions

Renvoie: paramètres intrinsèques, coefficients de distortions

Type renvoyé: ([list](#), [list](#))

`save_workspace_from_poses(name, list_poses_raw)`

Sauvegarde de l'espace de travail en donnant les positions du robot permettant de cibler les 4 coins avec l'outil de calibration. Les coins doivent être donnés dans le bon ordre

Paramètres:

- **name** ([str](#)) – nom de l'espace de travail. Max 30 caractères
- **list_poses_raw** ([list\(list\)](#)) – Liste des 4 positions des coins

Renvoie: Statut, message

Type renvoyé: ([int](https://docs.python.org/3/library/functions.html#int) (<https://docs.python.org/3/library/functions.html#int>), [str](https://docs.python.org/3/library/stdtypes.html#str) (<https://docs.python.org/3/library/stdtypes.html#str>))

`save_workspace_from_points(name, list_points_raw)`

Sauvegarde de l'espace de travail en donnant la listes des 4 positions des coins dans le bon ordre

Paramètres:

- **name** ([str](#)) – nom de l'espace de travail. Max 30 caractères
- **list_points_raw** ([list\(list\(float\)\)](#)) – Liste des 4 coins [x, y, z]

Renvoie: Statut, message

Type renvoyé: ([int](https://docs.python.org/3/library/functions.html#int) (<https://docs.python.org/3/library/functions.html#int>), [str](https://docs.python.org/3/library/stdtypes.html#str) (<https://docs.python.org/3/library/stdtypes.html#str>))

`delete_workspace(name)`

Supprime un espace de travail

Paramètres: **name** ([str](#)) – Nom de l'espace de travail

Renvoie: Statut, message

Type renvoyé: ([int](#), [str](#))

`get_workspace_poses(name)`

Récupération des 4 positions des coins de l'espace de travail donné par "name"

Paramètres: **name** ([str](#)) – Nom de l'espace de travail

Renvoie: Liste des 4 positions des coins de l'espace de travail

Type renvoyé: [list\(list\)](#)

`get_workspace_ratio(name)`

Récupération du ratio longueur sur largeur de l'espace de travail

Paramètres: **name** ([str](#)) – Nom de l'espace de travail

Renvoie: ratio

Type renvoyé: [float](#)

`get_workspace_list(with_desc=False)`

Récupération de la liste des espaces de travail

Renvoie: Liste des noms des espaces de travail

Type renvoyé: [list\(str\)](#)

Son

Pour plus d'informations, se référer à: [Son API](#) ([index.html#sound-api-functions](#))

`class NiryoRosWrapper`

`sound`

Gérer le son

Exemple:

```
from niryo_robot_python_ros_wrapper.ros_wrapper import *

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

Renvoie: SoundRosWrapper API instance

Type renvoyé: SoundRosWrapper

Anneau LED

Pour plus d'informations, se référer à: [Led Ring API](#) (index.html#led-ring-api-functions)

class NiryoRosWrapper

led_ring

Contrôler l'anneau LED

Exemple:

```
from niryo_robot_python_ros_wrapper.ros_wrapper import *

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

Renvoie: LedRingRosWrapper API instance

Type renvoyé: LedRingRosWrapper

Bouton "Custom"

class NiryoRosWrapper

custom_button

Contrôler le bouton *custom*

Exemple:

```
from niryo_robot_python_ros_wrapper.ros_wrapper import *

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

Renvoie: CustomButtonRosWrapper API instance

Type renvoyé: CustomButtonRosWrapper

class CustomButtonRosWrapper(hardware_version='ned2')

state

Obtenir l'état du bouton à partir de la classe ButtonAction

Renvoie: valeur entière provenant de la classe ButtonAction

Type renvoyé: int

is_pressed()

Etat d'appuie du bouton

Type renvoyé: bool

wait_for_action(action, timeout=0)

Attendre jusqu'à ce que l'action attendue intervienne et retourne True. Retourne false si le délais d'attente est atteint.

Paramètres: action (int) – valeur entière provenant de la classe ButtonAction

Renvoie: True si une action est intervenue, false sinon

Type renvoyé: bool

wait_for_any_action(timeout=0)

Retourne l'action détectée. Retourne ButtonAction.NO_ACTION si le délais d'attente est atteint sans action.

Renvoie: Renvoie l'action détectée, ou ButtonAction.NO_ACTION si le délai est atteint sans aucune action.

Type renvoyé: int

get_and_wait_press_duration(timeout=0)

Attend que le bouton soit pressé et renvoie le temps d'appuie. Renvoie 0 si aucune pression n'est détectée après la durée du délai d'attente.

Type renvoyé: float

Enumération

class ShiftPose

AXIS_X= 0

AXIS_Y= 1

AXIS_Z= 2

ROT_ROLL= 3

ROT_PITCH= 4

ROT_YAW= 5

class ToolID

Id de l'outil (doit correspondre à l'id de l'outil du package niryo_robot_tools_commander)

NONE= 0

GRIPPER_1= 11

GRIPPER_2= 12

GRIPPER_3= 13

GRIPPER_4= 14

ELECTROMAGNET_1= 30

VACUUM_PUMP_1= 31

class PinMode

Le Mode d'une broche peut être SORTIE ou ENTREE

OUTPUT= 0

INPUT= 1

class PinState

L'état d'une broche peut être BAS ou HAUT

LOW= False

HIGH= True

class PinID

ID de la broche

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'

D02= 'D02'

D03= 'D03'

D04= 'D04'

DI1= 'DI1'

DI2= 'DI2'

DI3= 'DI3'

DI4= 'DI4'

DI5= 'DI5'

AI1= 'AI1'

AI2= 'AI2'

A01= 'A01'

A02= 'A02'

class ConveyorID

ConveyorID pour contrôler des convoyeurs CAN (id 12 and 13) and TTL (id 9 and 10)

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

NONE= 0

ID_1= -1

ID_2= -2

class ConveyorCan

ConveyorID afin de contrôler les convoyeurs avec une interface CAN

NONE= 0

ID_1= 12

ID_2= 13

class ConveyorTTL

ConveyorID afin de contrôler les convoyeurs avec une interface CAN

NONE= 0

ID_1= 9

ID_2= 10

class ConveyorDirection

FORWARD= 1

BACKWARD= -1

class ObjectColor

RED= 'RED'

GREEN= 'GREEN'

BLUE= 'BLUE'

ANY= 'ANY'

class ObjectShape

CIRCLE= 'CIRCLE'

SQUARE= 'SQUARE'

ANY= 'ANY'

class ProgramLanguage

NONE= -1

ALL= 0

PYTHON2= 1

PYTHON3= 2

BLOCKLY= 66

class AutorunMode

DISABLE= 0

ONE_SHOT= 1

LOOP= 2

class ButtonAction

HANDLE_HELD_ACTION= 0

LONG_PUSH_ACTION= 1

SINGLE_PUSH_ACTION= 2

DOUBLE_PUSH_ACTION= 3

NO_ACTION= 100

Modbus



Dans ce document, nous nous focaliserons sur le serveur Modbus/TCP.

Ned exécute en permanence un serveur Modbus TCP qui permet à Ned de communiquer avec une API ou un autre ordinateur du même réseau.

Le serveur Modbus/TCP tourne par défaut sur le **port 5020**. Il a été développé en utilisant la librairie [pymodbus](https://pymodbus.readthedocs.io/en/latest/index.html) (<https://pymodbus.readthedocs.io/en/latest/index.html>). Le serveur vous permet de faire communiquer Ned avec un CLP ou avec un autre ordinateur sur le même réseau.

Installation de la librairie Python Modbus

Votre espace de travail doit disposer d'un interpréteur Python avec **Python 3** (3.6 ou une version supérieure) ou **Python 2** (2.7 ou une version supérieure).

Note

Téléchargez Python sur son [site officiel](https://www.python.org/) (<https://www.python.org/>) et retrouvez plus d'informations sur son installation sur [ce site](https://realpython.com/installing-python/) (<https://realpython.com/installing-python/>).

Cette installation nécessitera l'utilisation de l'utilitaire [pip](https://pypi.org/project/pip/) (<https://pypi.org/project/pip/>), fourni avec Python.

Vous aurez tout d'abord besoin d'installer numpy :

```
pip install numpy
```

Pour utiliser l'**API Modbus Python**, vous aurez besoin d'installer la librairie Modbus pour python [pymodbus](https://pymodbus.readthedocs.io/en/latest/index.html) (<https://pymodbus.readthedocs.io/en/latest/index.html>) :

```
pip install pymodbus
```

⚠ Attention

- Pip peut nécessiter une autorisation de l'administrateur afin d'installer ces packages. Dans ce cas, ajoutez

```
sudo
```

avant vos lignes de commande sur Linux.

- Si pip n'est pas automatiquement installée avec Python, rendez-vous sur le site de [Pypi](https://pypi.org/project/pip/) (<https://pypi.org/project/pip/>).

Utilisation du serveur Modbus

Dans ce document, nous nous focaliserons sur le serveur Modbus/TCP.

Ned exécute en permanence un serveur Modbus TCP qui permet à Ned de communiquer avec une API ou un autre ordinateur du même réseau.

Le serveur Modbus/TCP tourne par défaut sur le **port 5020**. Il a été développé en utilisant la librairie [pymodbus](https://pymodbus.readthedocs.io/en/latest/index.html) (<https://pymodbus.readthedocs.io/en/latest/index.html>). Le serveur vous permet de faire communiquer Ned avec un CLP ou avec un autre ordinateur sur le même réseau.

Introduction

Les 4 banques de registres Modbus sont implémentées : les **coils**, les **entrées discrètes**, les **registres de maintien**, ainsi que les **registres d'entrée**. Chaque banque de registres a un ensemble de fonctionnalités.

Les entrées discrètes et les registres d'entrée sont en **lecture seule**. Ils sont utilisés pour garder les états du robot.

Les coils et les registres de maintien sont en **lecture/écriture**. Ils sont utilisés pour donner à l'utilisateur l'accès aux commandes du robot. Ces deux banques de registres ne contiennent pas les états du robot, mais les dernières commandes.

Les tables commencent par l'adresse 0.

Coils

Chaque adresse contient une valeur de 1bit.

Lecture/écriture (les valeurs stockées correspondent à la dernière commande, pas à l'état courant du robot).

Les fonctions Modbus acceptées :

- 0x01 : READ_COILS
- 0x05 : WRITE_SINGLE_COIL

Cette banque de données peut être utilisée pour définir le mode et l'état des entrées/sorties numériques. Les numéros des entrées/sorties numériques utilisées par Modbus sont :

Table des décalages d'adresse des E/S digitales

Offset adresse	Niryo One / Ned E/S digitales	Ned2 E/S digitale
0	1A	DI1
1	1B	DI2
2	1C	DI3
3	2A	DI4
4	2B	DI5
5	2C	D01
6	SW1	D02
7	SW2	d03
8		D04

Adresse	Description
0-8	Mode des entrées/sorties numériques (Entrée = 1, Sortie = 0)
100-108	État des E/S numériques (Haut = 1, Bas = 0)
200-299	Peut être utilisé pour stocker vos propres variables

Entrées discrètes

Chaque adresse contient une valeur de 1bit.

LECTURE-SEULEMENT

Les fonctions Modbus acceptées :

- 0x02 : READ_DISCRETE_INPUTS

Cette banque de données peut être utilisée pour lire le mode et l'état des entrées/sorties numériques. Se référer à la [Coils](#), ci-dessus, pour la cartographie des numéros des entrées/sorties numériques.

Adresse	Description
0-8	Mode des entrées/sorties numériques (Entrée = 1, Sortie = 0)
100-108	État des E/S numériques (Haut = 1, Bas = 0)

Registres de maintien

Chaque adresse contient une valeur de 16bit.

Lecture/écriture (les valeurs stockées correspondent à la dernière commande, pas à l'état courant du robot).

Les fonctions Modbus acceptées :

- 0x03 : READ_HOLDING_REGISTERS
- 0x06 : WRITE_SINGLE_REGISTER

Adresse	Description
0-5	Axes (mrad)
10-12	Position x, y, z (mm)
13-15	Orientation roulis, tangage, lacet (mrad)
100	Envoie une commande de mouvement d'axe avec les valeurs stockées dans Axes
101	Envoie une commande de mouvement de position avec les valeurs stockées dans Position et Orientation
102	Envoie une commande de mouvement de position linéaire avec les valeurs stockées dans Position et Orientation
110	Arrête la commande en cours d'exécution
150	Drapeau de commande en cours d'exécution
151	Résultat de la dernière commande*
152	Grandeur résultant de la dernière commande (sauf commandes de vision)
153 - 158	Vision - Position de la cible
159	Vision - Forme de l'objet trouvé (-1 : TOUS, 1 : CERCLE, 2 : CARRE, 3 : TRIANGLE)
160	Vision - Couleur de l'objet trouvé (-1 : TOUS, 1 : BLEU, 2 : ROUGE, 3 : VERT)
200-299	Peut être utilisé pour stocker vos propres variables
300	Mode apprentissage (Activé = 1, Désactivé = 0)
301	Joystick activé (Activé = 1, Désactivé = 0)
310	Demande d'une nouvelle calibration
311	Demarrage d'une calibration automatique
312	Demarrage d'une calibration manuelle
401	Vitesse d'ouverture de la pince (100-1000)
402	Vitesse de fermeture de la pince (100-1000)
500	Mettre à jour l'identifiant de l'accessoire branché (gripper 1 : 11, gripper 2 : 12, gripper 3 : 13, pompe à vide : 31)
501	Stocker l'identifiant de l'accessoire branché
510	Ouvrir la pince précédemment mise à jour
511	Fermer la pince précédemment mise à jour
512	Aspirer l'air de la pompe à vide avec l'identifiant 31
513	Expirer l'air de la pompe à vide avec l'identifiant 31
520	Mettre à jour l'identifiant du convoyeur et l'activer
521	Détacher ou désactiver le convoyeur précédemment activé et mis à jour
522	Démarrer le convoyeur précédemment activé et mis à jour
523	Configurer la direction du Convoyeur (arrière = number_to_raw_data(-1), avant = 1)
524	Configurer la vitesse du Convoyeur (0-100)(%)
525	Stocker l'identifiant du Convoyeur
526	Arrêter le convoyeur précédemment activé et mis à jour

Adresse	Description
600	TCP - Active ou désactive la fonction PCO (Point Central Objet)
601	Active la fonction PCO et définit la transformation entre le repère de la jonction de l'outil et celui du PCO.
610	Vision - Fonction "Get target pose from relative pose", à partir des 3 positions relatives et du paramètre "height offset" stockés
611	Vision - Fonction "Get target pose from camera", à partir du nom du workspace, de "height offset", de la forme et la couleur stockés
612	Vision - Fonction "Vision pick", à partir du nom du workspace, de "height offset", de la forme et la couleur stockés
613	Vision - Fonction "Move to object", à partir du nom du workspace, de "height offset", de la forme et la couleur stockés
614	Vision - Fonction "Detect object", à partir du nom du workspace, de la forme et la couleur stockés
620	Vision - Stocke le paramètre "height offset" du workspace
621	Vision - Stocke la position relative x_rel
622	Vision - Stocke la position relative y_rel
623	Vision - Stocke la position relative yaw_rel
624	Vision - Stocke la forme demandée (-1 : TOUS, 1 : CERCLE, 2 : CARRE, 3 : TRIANGLE)
625	Vision - Stocke la couleur demandée (-1 : TOUS, 1 : BLEU, 2 : ROUGE, 3 : VERT)
626 - max 641	Vision - Stocke le nom du workspace, sous la forme d'une chaîne de caractères codée en hexadécimal 16 bits (voir code d'exemple).
650	Appliquer l'E/S analogique - Arg : [Analog IO number , Tension 0V- 5000mV]

"*" Le « résultat de la dernière commande » vous donne plus d'informations sur la dernière commande exécutée

- 0 : pas encore de résultat
- 1 : succès
- 2 : la commande a été rejetée (paramètres invalides, ...)
- 3 : la commande a été avortée
- 4 : la commande a été annulée
- 5 : la commande a eu une erreur inattendue
- 6 : délais d'attente de la commande
- 7 : erreur interne

Registres d'entrée

Chaque adresse contient une valeur de 16bit.

Lecture seule.

Les fonctions Modbus acceptées :

- 0x04 : READ_INPUT_REGISTERS

Adresse	Description
0-5	Axes (mrad)
10-12	Position x, y, z (mm)
13-15	Orientation roulis, tangage, lacet (mrad)
200	Sélection de l'ID de l'outil (0 pour aucun outil)
300	Mode apprentissage activé
400	Etat de la connexion avec les moteurs (OK = 1; pas OK = 0)
401	Drapeau de calibration nécessaire
402	Drapeau de calibration en cours
403	Température de la Raspberry Pi
404	Espace libre sur le disque de la Raspberry Pi
405	Taille des fichiers de log ROS sur la Raspberry Pi
406	Version n.1 de l'image de la Raspberry Pi
407	Version n.2 de l'image de la Raspberry Pi
408	Version n.3 de l'image de la Raspberry Pi
409	Version du matériel (1 ou 2)
530	Etat de la connexion du Convoyeur 1 (Connecté = 1, Non connecté = 0)
531	Etat de contrôle du Convoyeur 1 (Activé = 1, Désactivé = 0)
532	Vitesse du Convoyeur 1 (0-100)[%]
533	Direction du Convoyeur 1 (arrière = -1, avant = 1)
540	Etat de la connexion du Convoyeur 2 (Connecté = 1, Non connecté = 0)

Adresse	Description
541	Etat de controle du Convoyeur 2 (Activé = 1, Désactivé = 0)
542	Vitesse du Convoyeur 2 (0-100)[%]
543	Direction du Convoyeur 2 (arrière = -1, avant = 1)
600 - 604	Mode E/S analogique
610 - 614	Valeur de l'E/S analogique en mV

Analog IO addresses offset table

Offset adresse	Niryo One / Ned E/S analogique	Ned2 E/S analogiques
0	/	AI1
1	/	AI2
2	/	AO1
3	/	AO2

Dépendances - Serveur Modbus TCP

- [Librairie pymodbus](#)
- [Niryo_robot_msgs](#)
- [std_msgs](#)

Exemples Modbus

Des exemples de la librairie python Modbus peuvent être trouvés ici [Python](#) [Modbus](#) [exemples](#) (https://github.com/NiryoRobotics/ned_ros/tree/master/niryo_robot_modbus/examples/). Dans le dossier exemples, vous pouvez trouver plusieurs exemples scripts qui contrôlent Ned. Ces scripts sont commentés pour vous aider à comprendre chaque étape

Client Modbus Test

Appelle plusieurs fonctions utilisant les ports IO de Ned.

Client Move Command

Ce script montre les mouvements et la calibration de Ned.

Client Modbus Conveyor Example

Ce script montre comment activer le Convoyeur grâce à l'API Python Modbus, définir une direction, une vitesse ou démarrer et arrêter le convoyeur.

Client Modbus Vision Example

Ce script montre comment utiliser la méthode de pick par vision à partir d'un client Modbus à travers l'API Python Modbus. Ned attrape un objet rouge vu dans son workspace et le relâche sur sa droite. On utilise la fonction **string_to_register** pour convertir une chaîne de caractères en un objet stockable dans les registres.

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

from pymodbus.client.sync import ModbusTcpClient
from pymodbus.payload import BinaryPayloadBuilder, BinaryPayloadDecoder
import time
from enum import Enum, unique

# Enums for shape and color. Those enums are the one used by the modbus server to receive requests
@unique
class ColorEnum(Enum):
    ANY = -1
    BLUE = 1
    RED = 2
    GREEN = 3
    NONE = 0

@unique
class ShapeEnum(Enum):
    ANY = -1
    CIRCLE = 1
    SQUARE = 2
    TRIANGLE = 3
    NONE = 0

# Functions to convert variables for/from registers

# Positive number : 0 - 32767
# Negative number : 32768 - 65535
def number_to_raw_data(val):
    if val < 0:
        val = (1 << 15) - val
    return val

def raw_data_to_number(val):
    if (val >> 15) == 1:
        val = - (val & 0x7FFF)
    return val

def string_to_register(string):
    # code a string to 16 bits hex value to store in register
    builder = BinaryPayloadBuilder()
    builder.add_string(string)
    payload = builder.to_registers()
    return payload
```

```

# ----- Modbus server related function

def back_to_observation():
    # To change
    # joint_real = [0.057, 0.604, -0.576, -0.078, -1.384, 0.253]
    joint_simu = [0, -0.092, 0, 0, -1.744, 0]

    joint_to_send = list(map(lambda j: int(number_to_raw_data(j * 1000)), joint_simu))
    client.write_registers(0, joint_to_send)
    client.write_register(100, 1)

    while client.read_holding_registers(150, count=1).registers[0] == 1:
        time.sleep(0.01)

def register_workspace_name(ws_name):
    workspace_request_register = string_to_register(ws_name)
    client.write_registers(626, workspace_request_register)

def register_height_offset(height_offset):
    client.write_registers(620, int(number_to_raw_data(height_offset * 1000)))

def auto_calibration():
    print "Calibrate Robot if needed ..."
    client.write_register(311, 1)
    # Wait for end of calibration
    while client.read_input_registers(402, 1).registers[0] == 1:
        time.sleep(0.05)

def get_current_tool_id():
    return client.read_input_registers(200, count=1).registers[0]

def open_tool():
    tool_id = get_current_tool_id()
    if tool_id == 31:
        client.write_register(513, 1)
    else:
        client.write_register(510, 1)
    while client.read_holding_registers(150, count=1).registers[0] == 1:
        time.sleep(0.05)

# Function to call Modbus Server vision pick function
def vision_pick(workspace_str, height_offset, shape_int, color_int):
    register_workspace_name(workspace_str)
    register_height_offset(height_offset)

    client.write_registers(624, number_to_raw_data(shape_int))
    client.write_registers(625, number_to_raw_data(color_int))

    # launch vision pick function
    client.write_registers(612, 1)

    # Wait for end of function
    while client.read_holding_registers(150, count=1).registers[0] == 1:
        time.sleep(0.01)

    # - Check result : SHAPE AND COLOR
    result_shape_int = raw_data_to_number(client.read_holding_registers(159).registers[0])
    result_color_int = raw_data_to_number(client.read_holding_registers(160).registers[0])

    return result_shape_int, result_color_int

# ----- Main programm

if __name__ == '__main__':
    print "--- START"

    client = ModbusTcpClient('localhost', port=5020)

    # ----- Variable definition
    # To change
    workspace_name = 'gazebo_1'
    height_offset = 0.0

    # connect to modbus server
    client.connect()
    print "Connected to modbus server"

    # launch auto calibration then go to obs. pose
    auto_calibration()
    back_to_observation()

    # update tool
    client.write_registers(500, 1)

    print 'VISION PICK - pick a red pawn, lift it and release it'
    shape = ShapeEnum.ANY.value
    color = ColorEnum.RED.value
    shape_picked, color_picked = vision_pick(workspace_name, height_offset, shape, color)

    # --- Go to release pose
    joints = [0.866, -0.24, -0.511, 0.249, -0.568, -0.016]
    joints_to_send = list(map(lambda j: int(number_to_raw_data(j * 1000)), joints))

    client.write_registers(0, joints_to_send)
    client.write_register(100, 1)

    # Wait for end of Move command
    while client.read_holding_registers(150, count=1).registers[0] == 1:
        time.sleep(0.01)

    open_tool()

    back_to_observation()

    # Activate learning mode and close connexion
    client.write_register(300, 1)
    client.close()
    print "Close connection to modbus server"
    print "--- END"

```

Plus de façons de contrôler Ned

Il existe encore plus de moyens de contrôler Ned.

Si vous êtes un débutant, consultez la section [Blockly](#) pour comprendre les principes de base de la programmation.

Si vous voulez aller plus loin, peut-être expérimenter votre propre traitement d'image, multi_robot, IA (Intelligence artificielle)... Vous pouvez regarder [PyNiryo](#) pour avoir plus d'informations.

Blockly

Découvrez Niryo Studio.

PyNiryo

Comme expliqué dans la page [Utilisation du serveur TCP Ned](#) ([index.html#use-ned-s-tcp-server](#)), un serveur TCP s'exécute sur Ned, ce qui lui permet de recevoir des commandes de n'importe quel périphérique externe.

PyNiryo est un package Python disponible sur Pip qui permet de commander les robots Niryo avec une liaison Python facile.

Release de janvier 2022 - Compatibilité Niryo One et NED - Refonte Stack Hardware

Prérequis

Paquets Ubuntu

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

Librairies Python

Voir le fichier `src/requirements_ned2.txt`

Packages

Nouveaux packages

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

Packages renommés

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

Packages supprimés

- niryo_robot_serial_number
- niryo_robot_unit_tests
- niryo_robot_hardware_stack/fake_interface

Nettoyage et refonte

- Conformité avec roslint
- Conformité avec catkin lint pour une majorité des points
- Ajout d'une validation xsd pour les fichiers launch et les fichiers packages.xml
- Mise à jour des formats des packages à la version 3
- Mise à jour de C++ en version C++14

- Amélioration de la conformité avec Clang et Clazy
- Mise en place de rosdock_lite dans tous les packages
- Conformité avec catkin_tools
- Fonctionnalité « install » fonctionnelle
- Restructuration de sphinx_doc
- Ajout de la différenciation entre les versions matérielles Ned, One et Ned2
- Ajout des fichiers de configuration Ned2 dans tous les packages
- Refonte de niryo_robot_arm_commander
- Refonte de niryo_robot_python_ros_wrapper

Fonctionnalités (Pour Ned et One seulement)

- Ajout du fichier VERSION à la racine
- Ajout du fichier CHANGELOG.rst dans chaque package (avec l'outil catkin_generate_changelog)
- Mise à jour des valeurs de PID pour les Dynamixels
- Remplacement de fake_interface par des faux drivers (mock driver) pour les steppers et les dynamixels
- Ajout de la compatibilité avec les tapis convoyeurs TTL (à venir)
- Ajout des fonctionnalités Ned2 (à venir)
- Refonte de niryo_robot_bringup
- Amélioration des boucles de contrôle pour ttl_driver et joints_interface

Problèmes connus (pour le Ned et le One seulement)

Impossibilité de détecter 2 convoyeurs branchés en même temps. Merci de scanner les convoyeurs un par un.

Limitations

- Calibration désactivée sur le One et le Ned simulés
- Non supporté officiellement sur la version matérielle Ned2
- Le mode Hotspot est toujours activé par défaut au redémarrage du Niryo One

Niryo Studio

Nouvelles Fonctionnalités

- Configurations réseau (DHCP / Static IP)
- Détection de la version matérielle One / Ned / Ned2
- Affichage de la vitesse TCP
- Blockly - Blocs dynamiques (position sauvee, workspace)

Bugs fixés

- Blockly - Conversion RAD / DEG dans les blocs

Version septembre - Nouvelles fonctionnalités

Fonctionnalités

Package Tool Commander

- Service de configuration du PCO
TCP.msg
SetTCP.srv

package Arm Commander

- Nouveaux mouvements disponibles dans ArmMoveCommand.msg
linear pose
shift linear pose
trajectoire

Package Python ROS Wrapper

- Nouvelles fonctions de mouvement
move linear pose
linear pose
jog pose shift
jog joints shift
shift linear pose
exécuter une trajectoire depuis une position
- Nouvelles fonctions PCO

set_tcp

enable_tcp

reset_tcp

- Nouvelles fonctions de configuration de la caméra

set_brightness

set_contrast

set_saturation

Améliorations

- Refactorisation des packages Tool Commander et Robot Commander.

Suppression du package Robot Commander

Redistribution du package Robot Commander entre les packages Tool Commander et Arm Commander.

- Détection d'auto-collision

Ajout de la fonction de détection d'auto-collision via MoveIt.

- Détection de collision

Amélioration de la détection de collision du robot sur tous ses axes.

Le robot se met en Mode Apprentissage en cas de collision.

[Suggérer une modification](#)

[Télécharger le PDF](#)