**Prise en main de ROS**

|  |
| --- |
| **Note 1:** Si vous avez installé ROS depuis un gestionnaire de paquets comme apt, alors ces paquets ne vous seront pas accessibles en écriture et ne doivent pas être modifiés par vous l'utilisateur. Lorsque vous travaillez avec des packages source ROS ou lors de la création d'un nouveau package ROS, vous devez toujours travailler dans un répertoire auquel vous avez accès, comme votre dossier de départ. |

|  |
| --- |
| **Note 2:**   * *Package* : Le package est l'unité de base du code ROS.Chaque package peut contenir des bibliothèques, des exécutables, des scripts ou d'autres objets. * *Manifest* (package.xml): Un manifeste est une description d'un package. Ca sert à définir les dépendances entre paquets et de capturer des méta-informations sur le package comme la version, le gestionnaire, licence, etc .. * *Node:* C’est rien de plus qu'un exécutable dans un paquet ROS. Les nodes ROS utilisent une bibliothèque 'client' ROS afin de communiquer avec les autres nodes. Un node peut publier ou souscrire à un Topic. Un node peut également fournir ou utiliser un Service. * *Bibliothèque:* rospy = bibliothèque client python/roscpp = bibliothèque client c++ |

***GÉRER SON ENVIRONNEMENT***

* Il faut s'assurer que les [environment variables](http://wiki.ros.org/ROS/EnvironmentVariables) comme [ROS\_ROOT](http://wiki.ros.org/ROS/EnvironmentVariables#ROS_ROOT) et [ROS\_PACKAGE\_PATH](http://wiki.ros.org/ROS/EnvironmentVariables#ROS_PACKAGE_PATH) sont configurées :

|  |
| --- |
| $ export | grep ROS |

* Vous devrez lancer cette commande dans chaque nouveau shell que vous ouvrirez pour avoir accès aux commandes ROS, à moins que vous ajoutiez cette ligne à votre fichier .bashrc

|  |
| --- |
| $ source /opt/ros/indigo/setup.bash |

***CRÉER UN ROS WORKSPACE***

1. Créons un [catkin workspace](http://wiki.ros.org/catkin/workspaces):

|  |
| --- |
| $ mkdir -p ~/catkin\_ws/src $ cd ~/catkin\_ws/src $ catkin\_init\_workspace |

1. "Construire" l'espace de travail

|  |
| --- |
| $ cd ~/catkin\_ws/ $ catkin\_make |

Même si l'espace de travail est vide (il n'y a pas de paquets dans le dossier 'src', juste un lien unique CMakeLists.txt), vous pouvez quand même "construire" l'espace de travail.

1. "Sourcez" votre nouveau fichier setup.\*sh

La commande [catkin\_make](http://wiki.ros.org/catkin/commands/catkin_make) est un outil qui facilite le travail avec [catkin workspaces](http://wiki.ros.org/catkin/workspaces). Si vous regardez dans le dossier courrant vous verrez les dossiers 'build' et 'devel'. A l'interieur du dossier 'devel' vous pouvez voir maintenant qu'il y a plusieurs fichiers setup.\*sh. "Sourcer" un de ces fichiers va positionner ce workspace en tête de votre environnement.

|  |
| --- |
| $ source devel/setup.bash |

***CREATION PACKAGE CATKIN***

1. D'abord il faut se rendre dans le répertoire des fichiers sources de l'espace de travail catkin (catkin workspace) que l'on a créé précédemment

|  |
| --- |
| $ cd ~/catkin\_ws/src |

1. Créer un nouveau package appelé "new\_package" qui dépend de std\_msgs, roscpp et Rospy

|  |
| --- |
| $ catkin\_create\_pkg new\_package std\_msgs rospy roscpp |

Cela va créer un dossier “new\_package” qui contient un package.xml et un CMakeLists.txt

***CREATION PACKAGE ROS***

1. On a déjà un espace de travail catkin ([catkin workspace](http://wiki.ros.org/fr/catkin/Tutorials/create_a_workspace)) et un nouveau package catkin appelé new\_package. Pour c[réer un Package](http://wiki.ros.org/fr/ROS/Tutorials/CreatingPackage), allez dans l'espace de travail catkin (catkin\_ws) et construire ce package en utilisant [catkin\_make](http://wiki.ros.org/catkin/commands/catkin_make) :

|  |
| --- |
| $ cd ~/catkin\_ws/src  $ catkin\_make |

Plusieurs dossiers ont été créés dans votre espace de travail catkin: build, devel, src

***COMMANDES UTILES***

1.    Pour trouver le chemin d’un package:

|  |
| --- |
| $ rospack find [package\_name] |

2.    Pour changer de répertoire directement à un package ou à une stack

|  |
| --- |
| $ roscd [package\_name] |

3.    Réécrire un nom de package long plus rapidement: Utilisation de la touche TAB

|  |
| --- |
| $ cd Working<<< maintenant appuyez sur la touche TAB >>> |

Après avoir appuyé sur la touche TAB, la ligne de commande doit se remplir autant que possible:

|  |
| --- |
| Working\_Folder\_Montrac/ |

Toutefois, dans ce cas, il y a plusieurs packages qui commencent par Working. Essayez de taper TAB une autre fois. Ceci devrait afficher tous les paquets ROS qui commencent par Working :

|  |
| --- |
| Working\_Folder\_Baxter/  Working\_Folder\_Montrac/ |

***ROS MESSAGES***

1.    Communication on topics happens by sending ROS **messages** between nodes. For the publisher (turtle\_teleop\_key) and subscriber (turtlesim\_node) to communicate, the publisher and subscriber must send and receive the same **type** of message. This means that a topic **type** is defined by the message **type** published on it. The **type** of the message sent on a topic can be determined using rostopic type.

2.    *rostopic type* returns the message type of any topic being published.

|  |
| --- |
| $ rostopic type [topic] |

3.     La fonction rostopic pub [topic] [msg\_type] [args] permet d’envoyer un seul message avec en argument la commande à effectuer

|  |
| --- |
| $ rostopic pub -1 /turtle1/cmd\_vel geometry\_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]' |

Commande pour répéter le message par fréquence: (Ici repeat tous les 1 Hz)

|  |
| --- |
| $ rostopic pub /turtle1/cmd\_vel geometry\_msgs/Twist -r 1 -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, -1.8]' |

4.    Pour afficher les courbes des coordonnées de la tortue:

|  |
| --- |
| $ rosrun rqt\_plot rqt\_plot |

Et rentrer les commandes dans la fenêtre qui s’est ouverte:

/turtle1/pose/x et /turtle1/pose/y