TP 1 ROS Kinetic

Mise en oeuvre de quelques commandes usuelles de ROS
Principe des NODES (subscriber et Publisher)

1. Ouvir un terminal T1:

Saisir la commande: roscore pour lancer ROS, et réveiller le Master :-)

2. Ouvrir une autre fenêtre Terminal T2 (control+shift+T)

Saisir la commande: rosrun turtlesim turtlesim_node

Rosrun: commande pour lancer un programme existant sur ROS par défaut.

On obtient une fenêtre bleu avec une tortue verte au milieu.

3. Ouvrir un 3ème terminal T3

Saisir la commande : rosrun turtlesim turtle_teleop_key

Vous pouvez contrôler la tortue avec les flèches du clavier.

4. Lancer la commande: rosrun rqt_graph rqt-graph

Nous devons voir le graphique des noeuds et des topics
interface graphique permettant d'analyser le graphe d'applications et les transferts de données via les topics.

- Lors du démarrage d'un nœud, celui-ci s'identifie auprès du *Master*. Les nœuds communiquent avec d'autres nœuds via les *topics* (communication asynchrone) ou via les *services* (communication synchrone).
- Le *Master* est donc un intermédiaire qui permet à des nœuds de se connaître et de pouvoir communiquer entre eux.
- Un *message* est une structure de données utilisée pour la communication entre des nœuds (*topic* et *service*).

ROS Services

On refait les mêmes étapes:

1. Lancer: roscore (T1)
2. rosrun turtlesim turtlesim_node (T2)
3. rosrun turtlesim turtle_teleop_key (T3)

Lancer un service: T4

1. Saisir la commande: rossrv

list mdr Package show

Pour lancer une requête (un service) ont doit avoir des nodes qui s'exécutent:

Vérifiez si des packages existent:

Saisir la commande: rossrv list

Faire appel à un service de suppression par exemple:

rosservice call /clear

Ce service va supprimer la trajectoire réalisée par la tortue.

Autre exemple à tester: saisir la commande:

Rosservice info /spawn

Node: /turtlesim
URL: rosrpc…
Type: turlesim/spawn
Args x y theta name

Saisir la commande: Rosservice info /clear
(suppression des arguments)
Node: /turtlesim
URL: rosrpc…
Type: turlesim/spawn
Args

Saisir la commande: Rosservice info /spawn

Node: /turtlesim
URL: rosrpc…
Type: turlesim/spawn
Args x y theta name

Saisir la commande: Rosservice call /spawn

x=0.0. poser 5.0
y=0.0. poser 5.0
theta= 1,57   en radian
Name : robot1
On doit obtenir le nom du nouveau robot, c’est la requête.

Files system Tools:
Rescore
rosrun
rosnode
rostopic
rosmsg
rosservice
rosparam

File system Tools:
rospack find [package_name] Find packages,
exemple: rospack find roscpp
roscd [locationname[/subdir]] Go to package location.
exemple: roscd roscpp
rosls [locationname[/subdir]] Printing the containing files,
exemple: rosls roscpp_tutorials

Lien pour d'autres commandes:

http://wiki.ros.org/ROS/CommandLineTools

Tester quelques commandes:

roses
rospack
rospack find roscpp (affiche le chemin du package)
rosls roscpp
roscd roscpp

Retour au début de l'arborescence: cd ~

Relancer roscore (T1)
ensuite rosrun turtlesim turtlesim_node (T2),

et rosrun turtlesim turtle_teleop_key (T3)
Dans T4 on commence à tester les commandes:
rosnode
Cleanup Info kill list Machine ping

rosenode list (affiche tous les nodes qui fonctionnent)
//résout
/teleop_turtle
/turtles

rosetopic list (affiche tous les topics qui fonctionnent)
//rosout
//rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose

rostopic echo /turtle1/pose

Affiche les échos,
Faire déplacer la tortue, les positions changent…

Tester la commande: rostrum rqt_graph rqt_graph
On va apercevoir le topic turtle1/pose.

Si on arrête le topic, on rafraîchie le rqt et on ne verra pas le topic turtle1/pose

Maintenant il faut publier !

(on peut arrêter le noeud publieur: rosrrun turtlesim turtle_teleop_key)

Rostopic (Affiche les topics)
Bw Echo Fin Hz Info List Pub Type

Rostopic pub /turtle1/cmd_vel geometry_msgs/twist
« linear: (ici lancer TAB deux fois)
X:0.0
Y:0.0
z:0.0
angular:
X:0.0
Y:0.0
Z:0.0

Vous avez deux vecteurs « linear » et « angular » si on pose dans angular Z=2 le robot va tourner une fois.

Si on veut que le robot tourne plusieurs fois, on rajoute -r10 après rostopic pub, dans ce cas on demande au robot turtle de tourner chaque 10 hertz. (fréquence Hz).

Rostopic pub -r10 /turtle1/cmd_vel geometry_msgs/twist
« linear: (lancer TAB deux fois)
X:0.0
Y:0.0
z:0.0
Angular:
X:0.0
Y:0.0
Z:0.0

Si on veut que le robot trace un cercle, on pose X=1 (linear) et Z=1 (Angular).

Maintenant on veut avoir des informations sur le type de fichier:

Rostopic info /turtle1/cmd_vel

Donne des informations sur le publisher/subscriber

CREATION DE WORKSPACE, on va créer notre propre NODE

1. Créer un workspace nommé: catkin_ws_3

mkdir -p -/catkin_ws_3/src. (créer le dossier)
cd -/catkin_ws_3 (accéder au dossier)
catkin_make (compiler tout le package catkin_ws_3)

source devel/setup.bash. (il faut que ROS soit au courant du nouveau package…sourcing the workspace)

cd ..

gedit .bash

gedit .bashrc (en bas du fichier on trouve: source /opt/ros/kinetic/setup.bash)

On peut le rajouter dans « source » pour que ROS n'oublie pas notre nouveau workspace. C'est pour éviter de lancer la commande source devel/setup.bash à chaque fois qu'on ouvre notre nouveau workspace… alors on lance les commandes suivantes:

cd catkin_ws_3/ pour accéder au workspace catkin_ws_3

cd devez
pwd /home/nomduPC/catkin_ws_3/devel/setup.bash

gedit .bashrc

Dans le fichier il faut rajouter (à la fin de la page)

Source /home/nomduPC/catkin_ws_3/devel/setup.bash

1. Créer un package avec des noms

cd catkin_ws_3/src
ls
catkin_create_pkg robot roscpp rospy std_msgs

(créer le package nom du package « Robot » suivi des dépendances rospy, roscpp std_msgs)

ls
cd robot/
ls

Aller voir dans le dossier « robot » le package.xml (nom du package, description, les dépendances en bas du fichier).
voir le CmakeLists.txt (il a des fonctions, find package …)

On doit tout builder !!
cd -/catkin_ws_3/
catkin_make
On doit tout compiler !!

1. Créer des NODEs

- choisir le langage C++ ou Python
- créer un dossier « /src » pour le C++ et « /script » pour le python.
- écrire votre code
- compiler votre code

Enfin passer aux commandes : cd ~ /catkin_ws_3 ensuite le catkin_make. Pour tout compiler

Ecrire vos propres codes de Publisher/subscriber en python

Créer un dossier « Script » dans le workspace « robot »
Dans « script » on créer new-document nommé
« Publisher.py » on l'ouvre:

Print("hello") on enregistre et ferme le fichier et on clique bouton droit 'open in terminal':
Lancer la commande: python publisher.py … « hello » s'affiche.

Retour dans le fichier « Publisher.py » et modifier le contenu:
Aller dans ros.wiki dans tutoriels writting a simple publisher and subscriber (python) on copie le code: http://wiki.ros.org/ROS/Tutorials

Copier le code de la fonction « talker »

```
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String   # Dans ROS il faut importer le string de la dépendance std_msgs

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
# variable « pub » avec « chatter » nom du topic, type de la variable:string, queue_size:  buffer de 10octets)

rospy.init_node('talker', anonymous=True)

#(initialisation du Node, nom du Node 'talker',, anonymous=True, si y'avait un autre Node nommé 'talker', on l'ecrase et on le remplace par le nouveau.

    rate = rospy.Rate(10) # 10hz
#Fréquence de publication 10Hz


    while not rospy.is_shutdown():   # tant qu'on n'a pas arrêté le processus (ctl C) :
        hello_str = "hello world %s" % rospy.get_time()
# On affiche la variable string hello_str « hello world » et le temps via la fonction rospy.get_time() .
```

```
        rospy.loginfo(hello_str).  # pour
print sur le terminal..pas encore publié sur
le topic
        pub.publish(hello_str) # publier la
variable hello_str
        rate.sleep(). # Delay jusqu'au
prochain cycle/période avec f=10hz

if __name__ == '__main__':  # par défaut
Programme principal
    try:
        talker()        # appel de la
fonction « talker »
    except rospy.ROSInterruptException: #Si
interrupttion, le Node est en attente..
        pass
```

////Nous avons terminé le publisher////////////////////////////////

Maintenant: click droit, open in terminal le fichier « publisher.py »

Et lancer les commandes suivantes:

```
ls.   #On obtient le publisher.py.
Chmod +x publisher.py   # le fichier publisher.py est devenu
un exécutable.
ls.   # publisher.py est coloré en vert, c'est un exécutable.
```

test:

Cd ..
Cd catkin_ws_3/
Catkin_make
Rosrun robot publisher.py (n'oubliez pas de lancer roscore dans un autre terminal)

On doit voir la publication de « hello word »

Dans un autre terminal on lance la commande:
rostrun rqt_graph rqt_graph

Pour visualiser l'affichage, on lance la commande rostopic echo/
Ensuite choisir le fichier /chatter/rostopic écho /chatter

On retrouve les data dans l'écho,
On fait ctrl C dans le terminal du publisher, on va s'apercevoir que la data ne s'affiche plus dans l'écho.

Créer un script subscriber.py:

Dans « script » on créer new-document nommé « subscriber.py » on l'ouvre:

Aller dans ros.wiki dans tutoriels writting a simple publiasher and subscriber (python) on copie le code: http://wiki.ros.org/ROS/Tutorials

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def callback(receive):  #receive c'est la
donnée
    rospy.loginfo( "I heard %s",
receive.data)

def listener():

    # In ROS, nodes are uniquely named. If
two nodes with the same
    # name are launched, the previous one is
kicked off. The
    # anonymous=True flag means that rospy
will choose a unique
    # name for our 'listener' node so that
multiple listeners can
    # run simultaneously.
    rospy.init_node('listener',
anonymous=True)
# listener: nom du node
    rospy.Subscriber("chatter", String,
callback)
# Chatter le topic , string= type de
message, callback = fonction
    # spin() simply keeps python from
exiting until this node is stopped
    rospy.spin()
Print("done")
if __name__ == '__main__':
    listener()
```

Commande permettant de connaitre le type des données:

```
Rosmsg show std_msgs/string
String data
```

Ensuite:
On doit se positionner dans Script:

```
ls
Publisher.py. subscriber.py
Chmod +x subscriber.py
ls
Publisher.py subscriber.py #nous avons un nouveau mode subscriber.py
```

Dans un autre terminal:

```
Roscore (lancer roscore dans un autre terminal)

rosrun robot
publisher.py subscriber.py
rosrun robot subscribe.py
Y'a rien !
Lancer dans un autre terminal:
rosrun robot publisher.py
Dans le terminal précédent, les données s'affichent.
rosrun rqt_graph rqt_graph
```

Prochaine étape:

TP2 Gazebo (version 9.0) et simulation d'un robot dans son environnement !

TEST pour valider la version ROS Noetic

Après avoir installé ROS, on doit vérifier qu'il est bien installé:

Ouvrez un terminal: et lancez la commande **roscore**

Ouvrez deux autres terminaux:

- Dans le premier rentrez la commande :

```
rostopic echo /test
```

- Dans le deuxième rentrez la commande :

```
rostopic pub /test std_msgs/Int32 1
```