Introduction to ROS Noetic

Álan e Sousa, Nadhir Messai
{alan.e-sousa, nadhir.messai}@univ-reims.fr

2023

Contents

1 TP1 – First Steps

A la fin de ce TP, vous serez capable d'installer ou de supprimer des paquets ROS et d'avoir quelques bases de programmation en Python et la bibliothèque `rospy` et de travailler dans un environnement simulé.

Au fur et à mesure, consigner vos remarques et donnez les commandes utilisées dans votre compte rendu que vous soumettrez à la fin du TP par mail à l'adresse `alan.e-sousa@univ-reims.fr`.

Tous les exercices doivent être développer en Python et avec la librairie textttrospy.
Vous trouverez une liste de quelques commandes Linux à la fin du TP.

1.1 Création et initialisation de l'environnement de travail

1) Pour initialiser l'environnement ROS, lancez la commande

```
source /opt/ros/noetic/setup.bash
```

2) Créez l'arborescence `~/tpRO32/catkin_ws/src`;
3) Lancez la commande `catkin_make` depuis le répertoire `catkin_ws`;
4) Dans le fichier `~/.bashrc` ajoutez la ligne suivante:

```
source ~/tpRO32/catkin_ws/devel/setup.bash
```

5) Compilez les paquets en utilisant la commande `catkin_make` dans le répertoire approprié et sourcez le fichier `~/.bashrc`. NB: à chaque fois qu'un nouveau paquet ou nœud est ajouté dans le workspace, Il faut répéter cette étape (1.5) avant de le tester.

1.2 Création d'un paquet ROS

1) Créez le paquet sous le répertoire `src` de `catkin_ws`, en utilisant la commande `catkin_create_pkg`.
Nom: `pulisher_subscriber`.
2) Sous le répertoire `src` de `publisher_subscriber`, créez un nouveau ROS node `publisher.py`, qui va publier en continue `Hello world %TIME%` dans un topic `hello`. Utilisez et completez le code ci-dessous:

```python
#!/usr/bin/env python3
import rospy
from std_msgs.msg import String
def talker(topicName,nodeName,message):
  pub = rospy.Publisher(topicName, String, queue_size=10)
  rospy.init_node(nodeName, anonymous=True)
  rate = rospy.Rate(10)
  while not rospy.is_shutdown():
    hello_str = message + "%s" % rospy.get_time()
    rospy.loginfo(hello_str)
    pub.publish(hello_str)
    rate.sleep()
if __name__ == '__main__':
  try:
    #code à completer
    #definir le nom de topic , le nom du Node et le messsage à envoyer
    #faire appel à la fonction de façon qu'elle publie dans topic une fois
    #toute les 2 sec
    pass
```

```
    except rospy.ROSInterruptException:
      pass
```

3) Sous le répertoire src de `publisher_subscriber`, créez un nouveau ROS node `subscriber.py`, qui va s'abonner au topic `hello` et afficher les messages dans la console. Utilisez et complétez le code ci-dessous

```
#!/usr/bin/env python3
import rospy
from std_msgs.msg import String
def callback(data):
  rospy.loginfo(rospy.get_caller_id() + "message received : %s", data.data)
def listener(nodeName,topicName):
  rospy.init_node(nodeName, anonymous=True)
  rospy.Subscriber(topicName, String, callback)
  rospy.spin()
if __name__ == '__main__':
  listener('subscriber','hello')
```

4) Créez un launch file `hello.launch` qui lance les deux ROS Nodes en même temps. La structure du launch file est la suivante:

```
<launch>
  <node name="" pkg="" type="" output=""/>
</launch>
```

1.3 Installation et configuration de turtlesim

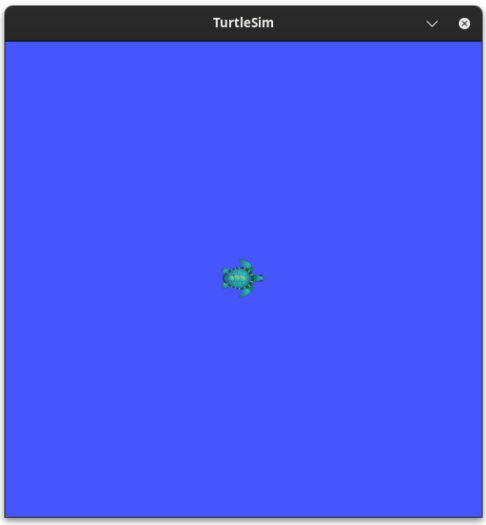

Figure 1: Fenêtre de turtlesim

1) Démarrez le ROS Master
2) Installez les paquets relatifs à turtlesim

```
sudo apt-get install ros-$(rosversion -d)-turtlesim
```

3) Démarrez la simulation

```
rosrun turtlesim turtlesim_node
```

4) Listez les topics ROS et identifiez ceux qui sont relatifs à turtlesim.

5) Affichez le graphe rqt.

1.4 Manipulation de la tortue sur turtlesim

1) Créez un nouveau paquet ROS nommé `turtle_move` sous le répertoire `src` de `catkin_ws`, en utilisant la commande `catkin_create_pkg`.
2) Dans le répertoire src du paquet turtle_move, créez un nouveau script `turtle_move_cercle.py`. Ce script vous permettra de bouger la tortue sur un cercle. Ce ROS Node publiera dans le topic `cmd_vel` du turtlesim. Pour connaitre le type de message publier dans ce topic, utilisez la commande `rostopic type nomDuTopic`.
3) Dans le répertoire src du paquet turtle_move, créez un nouveau script `turtle_move_square.py`. Ce script vous permettra de bouger la tortue sur un carré.
4) Affichez le graphe rqt et comparez-le avec le premier graphe.
5) Dans le répertoire src du paquet turtle_move, créez un nouveau script `turtle_move_keyboard.py`. Ce script vous permettra de bouger la tortue en utilisant le clavier (bouger avec les touches 'z', 'q', 's', 'd' et arrêter avec 'espace').

```
#completer ce code pour manipuler le robot en utilisant le clavier
val = input("Saisissez la commande à envoyer à la tortue: ")
if val=="z":
  #code pour augmenter la vitesse linéaire
if val=="q":
  #code pour diminuer la vitesse angulaire
if val=="s":
  #code pour diminuer la vitesse linéaire
if val=="d":
  #code pour augmenter la vitesse angulaire
if val==" ":
  #code pour remttre toutes les vitesses à zero
```

6) Utilisez un Rosbag pour enregistrer le flux de donnée transmis dans le topic `cmd_vel` du turtlesim en utilisant la commande `rosbag record -O subset nomDuTopic`.
7) Rejouez les données stockées dans le Rosbag.

1.5 Liste de quelques commandes indispensables sur linux

pwd	affiche le chemin complêt de le dossier currant	`pwd`
cd	change le dossier currant	`cd /opt/ros; cd ~; cd ..`
ls	affiche le contenu de le dossier currant	`ls; ls -lh; ls -lha`
cp	copier (utilisez `-r` pour copier des dossiers)	`cp source.pdf dest.pdf; cp -r source dest`
mv	déplacer ou renommer un fichier/dossier	`mv source.pdf dest.pdf; mv file.py src/`
rm	effacer (utilisez `-r` pour effacer des dossiers)	`rm source.pdf; rm -r src`
mkdir	créer des dossier (`-p` pour crée une arborescence)	`mkdir build; mkdir -p build/src`
exit	se déconnecter de la machine	`exit`
sudo	élevation de privilèges	`sudo apt-get update`
systemctl reboot	rédemarrer le système	`systemctl reboot`
systemctl poweroff	éteindre le système	`systemctl poweroff`
sudo	élevation de privilèges	`sudo apt-get update`
./fichier	éxecuter un fichier, si executable	`./file.py`
chmod	changer les permissions d'un fichier/dossier	`chmod +x file`

Pour (beaucoup) plus d'explication et commandes, regardez https://github.com/RehanSaeed/Bash-Cheat-Sheet et https://devhints.io/bash

2 TurtleBot Simulation with Gazebo

2.1 Initializing and understanding Gazebo

Gazebo is a simulator that makes it possible to rapidly test algorithms, design robots, perform regression testing, and train AI system using realistic scenarios.

1) Testing Gazebo with a blank world

```
roslaunch gazebo_ros empty_world.launch
```

2) Testing a simple cube. Models are described with XML files following the URDF (Unified Robot Description Format) specifications.
   a) clone the package `poppy_ergo_jr_gazebo` into your workspace src directory

```
git clone https://github.com/poppy-project/poppy_ergo_jr_gazebo
```

   b) In `poppy_ergo_jr_gazebo/urdf`, we can see the cube definition in the URDF file.
   c) upload the cube model to the simulation

```
rosrun gazebo_ros spawn_model -file cube.urdf -urdf -model test -x 0 -y 0 -z 1
```

   d) find the cube dimension and mass.
   e) we can check the cube position with

```
rostopic echo -n 1 /gazebo/model_states
```

   f) modify the cube position using

```
rosservice call /gazebo/set_model_state
```

   g) remove the cube model from the scene

```
rosservice call gazebo/delete_model "model_name: 'test'"
```

   h) close gazebo.

2.2 Simulating the robot

1) Install Simulation Package.
   The TurtleBot3 Simulation Package requires `turtlebot3` and `turtlebot3_msgs` packages as prerequisite. Without these prerequisite packages, the Simulation cannot be launched.

```
sudo apt-get install ros-noetic-{dynamixel-sdk,slam-gmapping,turtlebot3{,-msgs}}
sudo apt-get install ros-noetic-dwa-local-planner
cd ~/catkin_ws/src/
git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3_simulations
cd ..
catkin_make
```

2) Launch the simulation world.

```
export TURTLEBOT3_MODEL=waffle_pi
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

3) SLAM Simulation (Simultaneous Localization and Mapping)

```
roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping
```

4) Operate the turtlebot. In order to teleoperate the TurtleBot3 with the keyboard, launch the teleoperation node with the command below in a new terminal window.

```
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

5) Save Map

```
rosrun map_server map_saver -f ~/map
```

6) Run Navigation Node

```
roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=$HOME/map.yaml
```

7) Estimate Initial Pose.
Initial Pose Estimation must be performed before running the Navigation as this process initializes the AMCL parameters that are critical in Navigation. TurtleBot3 has to be correctly located on the map with the LDS sensor data that neatly overlaps the displayed map.
   a) Click the `2D Pose Estimate` button in the RViz menu.
   b) Click on the map where the actual robot is located and drag the large green arrow toward the direction where the robot is facing.
   c) Repeat step 1 and 2 until the LDS sensor data is overlayed on the saved map
   d) Launch keyboard teleoperation node to precisely locate the robot on the map
   e) Move a bit with the robot to collect the surrounding environment information
   f) Terminate the keyboard teleoperation node

8) Set Navigation Goal
   a) Click the `2D Nav Goal` button in the RViz menu
   b) Click on the map to set the destination of the robot and drag the green arrow toward the direction where the robot will be facing.

9) Creating a node to control the robot.
The `actionlib` Library provides the tools and interface to set up an Action Server to execute the requested goals sent by the Client. The `move_base` node implements a `SimpleActionServer`, an action server with a single goal policy, taking in goals of `geometry_msgs/PoseStamped` message type
Use this function to create a script that controls the robot

```
import rospy
import actionlib
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal
def movebase_client():
  client = actionlib.SimpleActionClient('move_base', MoveBaseAction)
  client.wait_for_server()
  goal = MoveBaseGoal()
  goal.target_pose.header.frame_id = "map"
  goal.target_pose.header.stamp = rospy.Time.now()
  goal.target_pose.pose.position.x = 0.5
  goal.target_pose.pose.orientation.w = 1.0
  client.send_goal(goal)
  wait = client.wait_for_result()
  if not wait:
    rospy.logerr("Action server not available!")
    rospy.signal_shutdown("Action server not available!")
  else:
    return client.get_result()
```

2.3 Broadcaster

1) Dans cette partie nous allons créer un `TF broadcaster`. Dans ROS, l'outil tf nous permet de simplifier les changements de repère.
   a) Créez un nouveau nœud `tf_broadcaster.py`

```
import roslib
import rospy
```

```
import tf
import turtlesim.msg
def handle_turtle_pose(msg, turtlename):
  br = tf.TransformBroadcaster()
  br.sendTransform((msg.x, msg.y, 0),
    tf.transformations.quaternion_from_euler(0, 0,msg.theta),
    rospy.Time.now(),
    turtlename,
    "world")
if __name__ == '__main__':
  rospy.init_node('turtle_tf_broadcaster')
  turtlename = rospy.get_param('~turtle')
  rospy.Subscriber('/%s/pose' % turtlename,
    turtlesim.msg.Pose,
    handle_turtle_pose,
    turtlename)
  rospy.spin()
```

b) Créez maintenant le launch file `start_demo.launch` permettant de lancez la simulation turtlesim et le tf broadcaster en même tamps

```
<launch>
  <!-- Turtlesim Node-->
  <node pkg="turtlesim" type="turtlesim_node" name="sim"/>
  <node pkg="turtlesim" type="turtle_teleop_key" name="teleop" output="screen"/>
  <node name="turtle1_tf_broadcaster" pkg="learning_tf" type="tf_broadcaster.py" respawn="false" out
    <param name="turtle" type="string" value="turtle1" />
  </node>
</launch>
```

2) Executez le launch file pour tester la demo
3) Vérifiez les résultats avec la commande

```
rosrun tf tf_echo /world /turtle1
```

3 TurtleBot

To use the turtlebot remotely, some configuration steps are necessary.

1) Connect to the robot using SSH:

```
ssh pi@IP_ADDRESS_OF_RASPBERRY_PI
```

2) Since ROS1 has terrible security requirements, we need to disable the firewall:

```
sudo ufw disable
```

3) We need the hostnames of both the PC and the robot, so run this command on both and take note of the returned values:

```
hostname
```

4) Set the environment variables for the PI:

```
{
  echo "export TURTLEBOT3_MODEL=waffle_pi"
  echo "export ROS_MASTER_URI='http://IP_ADDRESS_OF_RASPBERRY_PI_3:11311'"
  echo "export ROS_HOSTNAME=$(hostname)"
} >> ~/.bashrc
source ~/.bashrc
```

5) In the PC, run:

```
{
  echo "export TURTLEBOT3_MODEL=waffle_pi"
  echo "export ROS_MASTER_URI='http://IP_ADDRESS_OF_RASPBERRY_PI_3:11311'"
  echo "export ROS_HOSTNAME=$(hostname)"
} >> ~/.bashrc
source ~/.bashrc
```

6) Now, edit the file `/etc/hosts` (use `sudo -e /etc/hosts`) and make sure that all hostnames are listed with their respective IPs (to find the IP, run `ip a`).
7) Bringup TurtleBot3 Sur le turtlebot (la connexion SSH), executez

```
roscore
roslaunch turtlebot3_bringup turtlebot3_robot.launch
```

8) Basic Operation: Teleoperation depuis le PC

```
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

9) Visualisez les topics lancés par le turtlebot et affichez le graphe RQT
10) Identifiez les topics qui permettent de bouger le robots, de savoir sa position actuelle et de lui envoyer des goals.
11) SLAM: Lancez le nœud de slam depuis votre PC, et créez une carte
12) Affin d'effectuer le SLAM, le Turtlebot utilise un algorithme nomé GMAPPING. Tous les paramètres de cet algorithme peuvent être changé en accédant au ficher `turtlebot3_slam/config/gmapping_params.yaml`.

    - maxUrange: permet de chager la distance du lidar.
    - map_update_interval: Plus la valeur est petite, plus la carte est mise à jour fréquemment.
    - minimumScore: Ce paramètre définit la valeur de score minimum qui détermine le succès ou l'échec du test de correspondance des données de scan du capteur. Cela peut réduire les erreurs dans la position attendue du robot dans une grande zone. Si le paramètre est correctement défini, vous verrez des informations similaires à celles illustrées ci-dessous.
    - linearUpdate: Lorsque le robot traduit une distance plus longue que cette valeur, il exécute le processus de scan.
    - angularUpdate: Lorsque le robot tourne plus que cette valeur, il exécute le processus de scan. Il est recommandé de définir cette valeur sur une valeur inférieure à linearUpdate.

Modifier ces valeurs pour avoir une carte la plus précise possible.

13) Utilisez las carte pour controller le robot.
14) Créez un nouveau nœud permettant de manipuler le robot (envoyer un goal).
15) Créez un nouveau nœud dans le quelle vous définissez deux points. Le robot doit naviguer en boucle entre ces deux points.