[Lab 1]

ROS/Turtlebot3 libraries setup and ROS essentials.

Resources

[1] Robotis doc (remember to select the Melodic tab)
[2] ROS Melodic
[3] GitHub of the lab: https://github.com/emarche/Mobile-robotics-4S009023-UniVR

Setup

It is recommended to install the following libraries in a Miniconda environment (remember to check your Python version to select the correct Miniconda version).

After the Miniconda installation, remember to source the configuration file, before creating (and activating) your conda environment.

Requirements

- Ubuntu 18.04
- Python 3.7 (installed directly in the conda environment)

```
conda create -n <env_name> python=3.7
```

- [Bonus] Terminator (to easily handle several command windows)

ROS Installation

Follow [2] for the ROS installation from point 1.1 to point 1.6.1 (included):

Pay attention to the following:

- At point 1.4, choose the recommended version (i.e., the full install)
- When you source your configuration file at point 1.5 you will return to the base conda environment. Remember to re-activate the previously created environment before proceeding to point 1.6 (or just add the conda environment activation in the configuration file).

Troubleshooting

1.4) sudo apt update

error > E: Il repository "http://ppa.launchpad.net/gnome-terminator/ppa/ubuntu bionic Release" non ha un file Release.

solution > Aprire "Software & Aggiornamenti -> Altro Software" e rimuovere il repository indicato

1.6) sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-wstool build-essential

error > E: Impossibile scaricare alcuni pacchetti. Potrebbe essere utile eseguire "apt-get update" o provare l'opzione "--fix-missing".

solution > sudo apt-get update

ROS workspace

We have to create a workspace for ROS packages.

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
```

Finally we have to source the setup file in the 'devel' folder (and possibly add it to the configuration file for simplicity):

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

Turtlebot3 libraries

Now back to [1] (in the melodic tab) for the Turtlebot3 libraries, following from point 1.1.3 to 1.1.5 (included):

- Install ROS dependent packages (we can get a similar error to the previous 1.6 point, fix it in the same way or with the --fix-missing option)

```
$ sudo apt-get install ros-melodic-joy ros-melodic-teleop-twist-joy \
  ros-melodic-teleop-twist-keyboard ros-melodic-laser-proc \
  ros-melodic-rgbd-launch ros-melodic-depthimage-to-laserscan \
  ros-melodic-rosserial-arduino ros-melodic-rosserial-python \
  ros-melodic-rosserial-server ros-melodic-rosserial-client \
  ros-melodic-rosserial-msgs ros-melodic-amcl ros-melodic-map-server \
  ros-melodic-move-base ros-melodic-urdf ros-melodic-xacro \
  ros-melodic-compressed-image-transport ros-melodic-rqt* \
  ros-melodic-gmapping ros-melodic-navigation ros-melodic-interactive-markers
```

Turtlebot3 libraries

- Install Turtlebot3 packages from GitHub (i.e., from sources)

```
$ sudo apt-get remove ros-melodic-dynamixel-sdk
$ sudo apt-get remove ros-melodic-turtlebot3-msgs
$ sudo apt-get remove ros-melodic-turtlebot3
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src/
$ git clone -b melodic-devel https://github.com/ROBOTIS-GIT/DynamixelSDK.git
$ git clone -b melodic-devel https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
$ git clone -b melodic-devel https://github.com/ROBOTIS-GIT/turtlebot3.git
$ cd ~/catkin_ws && catkin_make
$ echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
```

- Set the Turtlebot3 model name

```
$ echo "export TURTLEBOT3_MODEL=waffle_pi" >> ~/.bashrc
```

ROS essentials

- ROS Nodes: data/information processing software units:
  - Python scripts that rely on rospy
- ROS Topics: transport information between Nodes.

In a real application

- A lot of Nodes and Topics communicating with each other
- Which Nodes are talking to each other and how?
- Which Topics are being passed around between the Nodes?

ROS packages

Each piece of ROS software is contained in a "package", which is built using the previous 'catkin_make' command to make it available to other users.

There are several useful commands to explore the ROS packages:

```
$ rospack find [package_name]
```

```
$ roscd [locationname[/subdir]]
```

An exhaustive list of ROS bash commands is available here.

Create a ROS package

An empty package is characterized by a manifest and an input file for the build of the package. It is typically contained in the 'src' of the workspace:

```
workspace_folder/        -- WORKSPACE
  src/                   -- SOURCE SPACE
    CMakeLists.txt       -- 'Toplevel' CMake file, provided by catkin
    package_1/
      CMakeLists.txt     -- CMakeLists.txt file for package_1
      package.xml        -- Package manifest for package_1
    ...
    package_n/
      CMakeLists.txt     -- CMakeLists.txt file for package_n
      package.xml        -- Package manifest for package_n
```

We can create an empty ROS package with the built in function:

```
$ catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
```

And then build it:

```
$ cd ~/catkin_ws
$ catkin_make
```

ROS Nodes

Is an executable file within a ROS package. Nodes use a ROS client library to communicate with other Nodes, can publish or subscribe to a Topic and can also provide or use a Service.

A ROS master node is the first thing your should run using ROS:

```
$ roscore
```

Then, we can visualize the list of active nodes, or informations about nodes:

```
$ rosnode list
$ rosnode info /rosout
```

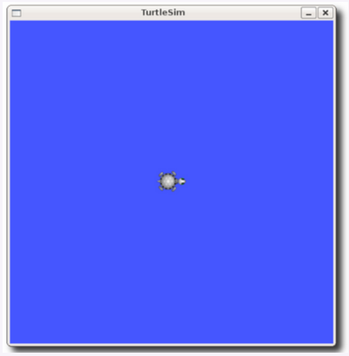
Finally, we can run a ROS Node:

```
$ rosrun [package_name] [node_name]
```

Turtlesim node

With a complete installation of ROS we can use the built in 'turtlesim' package to explore other ROS functionalities.

```
$ rosrun turtlesim turtlesim_node
```



And we can alias node's name for multiple instances:

```
$ rosrun turtlesim turtlesim_node __name:=my_turtle
```

To check if a node is running we can ping it:

```
$ rosnode ping my_turtle
```

ROS Topics

We can think of topics as blackboards, where different Nodes can write and read messages if they are Publisher or Subscriber to them.

First, we will control the previous turtlesim using our keyboard:

```
$ rosrun turtlesim turtle_teleop_key
```

Now both the simulator and the teleoperation nodes are communicating using a Topic. We can also retrieve active topics and more informations:

```
rostopic bw     display bandwidth used by topic
rostopic echo   print messages to screen
rostopic hz     display publishing rate of topic
rostopic list   print information about active topics
rostopic pub    publish data to topic
rostopic type   print topic type
```

ROS Topics

It is possible to manually send a message through the command line using:

```
pub <topic-name> <topic-type> [data...]
      Publish data to a topic.
      $ rostopic pub /topic_name std_msgs/String hello
```

There are several useful parameters, the most important ones are:

```
-r RATE
      Enable rate mode. Rate mode is the default (10hz) when using piped or file input.
-1, --once
      Enable once mode.
```

These allow to publish the message at a certain rate, or only once.

Visualize Topics

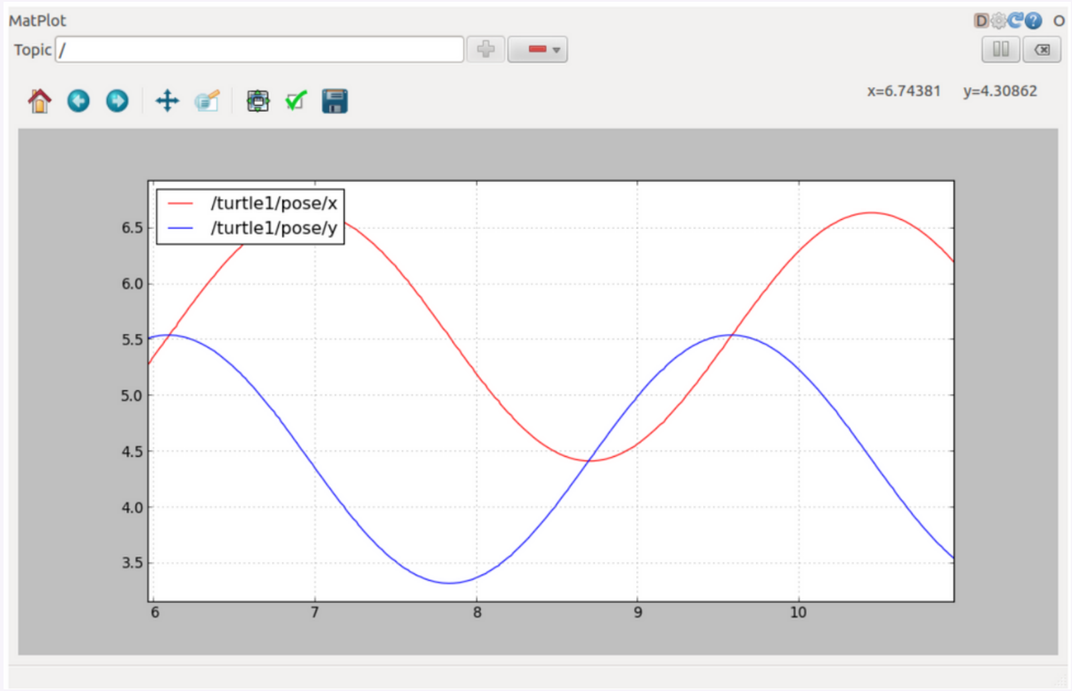
There is an additional feature that allows to visualize a dynamic graph of what is going on in the ROS system (note that rqt_graph requires additional modules, e.g., rospkg, PyQT5, Pyside2, which can be easily installed with "pip3 install <module_name>"):

```
$ sudo apt-get install ros-<distro>-rqt
$ sudo apt-get install ros-<distro>-rqt-common-plugins
$ rosrun rqt_graph rqt_graph
```

Plot published data

Finally, we can plot in a 2d graph the published data over time:

```
$ rosrun rqt_plot rqt_plot
```



ROS Services

Services represent another communication mechanism between Nodes. In detail, they allow to send (data) requests and receive answers.

We will not use Services during the course but for a quick overview we can threat them as Topics, using similar ROS bash commands:

```
rosservice list         print information about active services
rosservice call         call the service with the provided args
rosservice type         print service type
rosservice find         find services by service type
rosservice uri          print service ROSRPC uri
```

Launch files

We can design '.launch' files to run multiple nodes with a single command

```
$ roslaunch [package] [filename.launch]
```

Let's create a launch file for the previoulsy created package:

```
$ cd ~/catkin_ws
$ source devel/setup.bash
$ roscd beginner_tutorials
```

```
$ mkdir launch
$ cd launch
```

(It is not mandatory to create a launch directory, however it is considered good practice)

Launch files

Create the '.launch' file and paste the following commands:

```
<launch>

  <group ns="turtlesim1">
    <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
  </group>

  <group ns="turtlesim2">
    <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
  </group>

  <node pkg="turtlesim" name="mimic" type="mimic">
    <remap from="input" to="turtlesim1/turtle1"/>
    <remap from="output" to="turtlesim2/turtle1"/>
  </node>

</launch>
```

Launch files

Finally, we can launch the specified nodes using a single instruction:

```
$ roslaunch beginner_tutorials turtlemimic.launch
```

Exercise

- Try to manually publish a message to the velocity Topic of the above node
- Visualize the Topics structure using 'rqt_graph'
- Visualize the published commands with 'rqt_plot'

Publisher and Subscriber in Python

These Python scripts will be contained in a 'scripts' folder in our package.

- We will use the Python Publisher example available here.
- We will use the Python Subscriber example available here.

ROS bag

It is possible to save the data passed through the Topics in a '.bag' file, which can also be retrieved to reproduce the same behaviors.

- Launch the 'turtlesim' simulator and the teleoperation node

We will save the keyboard commands in a rosbag:

```
mkdir ~/bagfiles
cd ~/bagfiles
rosbag record -a
```

ROS bag

We can retrieve the informations from a rosbag file:

```
rosbag info <your bagfile>
```

```
path:        2014-12-10-20-08-34.bag
version:     2.0
duration:    1:38s (98s)
start:       Dec 10 2014 20:08:35.83 (1418270915.83)
end:         Dec 10 2014 20:10:14.38 (1418271014.38)
size:        865.0 KB
messages:    12471
compression: none [1/1 chunks]
types:       geometry_msgs/Twist [9f195f881246fdfa2798d1d3eebca84a]
             rosgraph_msgs/Log   [acffd30cd6b6de30f120938c17c593fb]
             turtlesim/Color     [353891e354491c51aabe32df673fb446]
             turtlesim/Pose      [863b248d5016ca62ea2e895ae5265cf9]
topics:      /rosout                4 msgs    : rosgraph_msgs/Log   (2 connections)
             /turtle1/cmd_vel     169 msgs    : geometry_msgs/Twist
             /turtle1/color_sensor 6149 msgs  : turtlesim/Color
             /turtle1/pose         6149 msgs  : turtlesim/Pose
```

ROS bag

And replay the same data (remember to quit the teleoperation node):

```
rosbag play <your bagfile>
```

There are also several options to store the data of only one Topic, or replay the data with a specific frequency:

```
rosbag record -O subset /turtle1/cmd_vel /turtle1/pose
rosbag play -r 2 <your bagfile>
```

Exercise

- Create a Publisher (Python) script to send velocity commands to a turtlesim simulator.
- Visualize the data flow using rqt_graph and rqt_plot.
- Store the published data in a bag and replay them.