#1 Lecture: ROS Introduction

ROS Overview

Please read the ROS Overview from `1_Course_ROS.pdf` slides.

Ubuntu and basic terminal commands

We will be using `Ubuntu 22.04 (Jammy Jellyfish)`. From 20 up to 40 GB of free space are required. Also, 8 GB of RAM are recommended.

The `Terminal` is essential for developing ROS applications: look for the basic commands on the slides. You can find a introductory course here.

Editing your code is important. I highly recommend using `Visual Studio Code`. Find the basic installation instructions here: https://code.visualstudio.com/docs/setup/linux.

We will be using C++ for coding. You can find a basic course here.

More useful links are compiled here:

- ROS Best Practices: http://wiki.ros.org/BestPractices
- ROS C++ Coding Style: http://wiki.ros.org/CppStyleGuide
- ROS Cheat Sheet: https://clearpathrobotics.com/ros-robot-operating-system-cheat-sheet/
- ROS Map: http://metrorobots.com/rosmap.html

ROS2 Installation and Configuration

Installation

We are going to install ROS2 Humble, please open this link: https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html. And follow all the installation instructions.

> You can install the `Desktop Install` because it comes with ROS, RViz, demos and tutorials.

Configuration

Open your `.bashrc` file:

```
gedit ~/.bashrc
```

then copy the next three lines at the end of the document:

```
source /opt/ros/humble/setup.bash
export ROS_DOMAIN_ID=1 #choose a number of your choice, between 1 and 101
# export ROS_LOCALHOST_ONLY = 1
```

basically adding this lines in the bash file we get those lines to execute in every terminal that we open. The first line makes avaible for the console the ros2 command line tools. The second configures the Domain ID wich makes the computers running ROS 2 on the same network to communicate. By defaul the ID is `0`, so in order to avoid interference between different groups of computers (on the same network), a different domain ID should be set for each group. Finally if you are testing or the network is public with many restrictions, you can choose to hide (not make visible) your topics, services and actions. This will be helpful in classrooms.

Creating a ROS workspace

A ROS workspace is a directory with a particular structure. Commonly there is a `src` subdirectory. Inside that subdirectory is where the source code of ROS packages will be located. Typically the directory starts otherwise empty. Create the next folders structure:

```
mkdir -p ~/ros2_ws/src
cd ~/ros2_ws
```

Then, could be necessary to install `colcon` which is ROS build tools evolved from the original `catkin_make`:

```
sudo apt install python3-colcon-common-extensions
```

Now build the workspace:

```
colcon build --symlink-install
```

After the build is finished, we should see the `build`, `install`, and `log` directories. Add the next line to the end of your `.bashrc` file:

```
source ~/ros2_ws/install/local_setup.bash
```

Now we are ready to start building our first `package`. For example a file system will looks like this:

```
workspace_folder/
    src/
        cpp_package_1/
            CMakeList.txt
            include/cpp_package_1/
            package.xml
            src/
```

```
        py_package_1/
            package.xml
            resource/py_package_1
            setup.cfg
            setup.py
            py_package_1/
    ...
```

Basic definitions

The **ROS2 graph** is a network of ROS 2 elements processing data together at the same time. It encompasses all executables and the connections between them.

- **Nodes**: a node is a participant in the ROS 2 graph, which uses a client library to communicate with other nodes. Nodes will only establish connections with other nodes if they have compatible Quality of Service settings.
- **Messages**: Messages are a way for a ROS 2 node to send data on the network to other ROS nodes, with no response expected.
- **Topics**: Nodes can publish messages to a topic as well as subscribe to a topic
- **Services**: Services are a request/response communication, where the client (requester) is waiting for the server (responder) to make a short computation and return a result.
- **Actions**: Actions are used for long-running goals that can be preempted.

Most useful command line tools

Run a package specific executable:

```
ros2 run
```

Options to get information about the nodes that are running:

```
ros2 node -h
```

Shows a list of active nodes:

```
ros2 node list
```

Options to get information about the topics that are running:

```
ros2 topic -h
```

Show a list of active topics:

```
ros2 topic list
```

Show the messages published to a topic:

```
ros2 topic echo
```

Publish a message to a topic:

```
ros2 topic pub
```

Commands for ROS's client/service framework:

```
ros2 service -h
```

Start nodes defined in a launch file:

```
ros2 launch
```

Topics

Nodes can publish or subscribe to a topic to send or receive messages

- Topics are conceived for unidirectional, streaming communication
- Topics must have a name and associated type of message
- Messages are described and defined in .msg files composed of two parts: fields and constants.
- Find existing interfaces at: https://github.com/ros2/common_interfaces or run:

```
ros2 interface list -m
```

Examples

Run the following:

```
ros2 run examples_rclcpp_minimal_publisher publisher_member_function
```

in other terminal run:

```
ros2 node info /minimal_publisher
```

Now open another terminal an run:

```
ros2 run examples_rclcpp_minimal_subscriber subscriber_member_function
```

you can see the info of this subscriber with:

```
node info /minimal_subscriber
```

Practice with the following exercises:

1. Check which nodes are running and available topics

```
ros2 node list
ros2 topic list
```

2. Kill the subscriber node. Open a new terminal and type:

```
ros2 topic echo /topic
```

3. Run the subscriber again. Kill the publisher. Do not close the echo terminal. Publish a text message of your own:

```
ros2 topic pub -1 /topic std_msgs/msg/String "{data: 'Hi, it is me'}"
```

ROS Service

- The publish/subscribe model is not appropriate for RPC request/reply interactions.
- Request/reply is done via services, which are defined by a pair of messages: one for the request and one for the reply
- Services are defined using `.srv` files, which are compiled into source code by a ROS client library.

ROS Actions

- Actions are intended for long running tasks
- They consist of three parts: a goal, feedback, and a result
- Actions use a client-server model
- Actions are defined using `.action` files, which are compiled into source code by a ROS client library