Lab 1: Setup and ROS Basics

Background: ROS and Gazebo

For our labs we will use ROS which is a widely used robotic middleware in the open-source community. ROS provides an easy way to exchange information between sensors, actuators, and processing nodes using a publish-subscribe communication model. The two main concepts in ROS are nodes and topics. Nodes retrieve data from sensors, process and transform data, and write data to actuators. Topics are channels through which nodes communicate with each other. A node publishes data to a topic and other nodes that need that data subscribe to the topic. In this way, we can create a network of interconnected nodes that collectively implement a robotic task. Nodes can reside on a single processing platform (e.g. a CPU board on the robot) or can be distributed across multiple systems (e.g., a robot controlled from an off-board server over the network). Best practice in ROS is to keep the nodes small, have them do one well-defined task and interconnect the nodes to implement the control system.

There are two versions of ROS available: ROS1 and ROS2. ROS1 is a stable and mature implementation that is well documented, but its development has stopped and the only updates available are bug fixes. ROS2 is a relatively recent overhaul of ROS, which is not backward compatible with ROS1. ROS2 is recommended for new development, so we will use ROS2. In further text, when we say ROS, we mean ROS2.

The release we will use is called ROS2 Humble and it is supported on Ubuntu 22.04. If you are setting it up on your own computer, please use Ubuntu 22.04. If you use Windows, we recommend that you enable Windows Subsystem for Linux (WSL) and install Ubuntu 22.04. For Apple platforms you would have to build from the source, and you will be on your own to figure out any issues. We will provide a shell script for Ubuntu 22.04 that automates the installation of ROS and all packages you need to do the labs. For other platforms you will have to install manually.

Gazebo is a robotic simulation tool that has been designed to work with ROS, but can be also used as a standalone simulator without ROS. Gazebo consists of the physics engine and the graphical interface for visualization. The physics engine will allow us to model the robot dynamics, the environment the robot is in (world), and simulate the interaction between the robot and the environment. The ROS nodes that produce actuation signals would typically feed into Gazebo models that will translate these signals into forces and torques and use them to advance the state of the world mode. Likewise, the sensor models in Gazebo will produce data on ROS topics that nodes can use to process and construct the robot's perception of the world. The level of physical detail that the models capture is up to the model designer. The general tradeoff is that more complex and more sophisticated models provide higher fidelity, but require

more computational power to execute. The visualization graphical interface of Gazebo will allow us to see the robots moving through the world and interact with the environment.

There are two variants of Gazebo: Gazebo Classic and Gazebo Ignition (recently renamed to just Gazebo). We will use Gazebo Classic, which has reached its end-of-life, but Turtlebot 3 models are available only for this version. If you continue to work in robotics in your career and continued education, you will probably end up using Gazebo Ignition. We will do very limited development of models, so for the purpose of lab assignments, Ignition vs. Classic will not make much difference.

Lab Environment

Your first task for this lab is to create the working environment. This will be followed by trying out some basic ROS tasks to verify that everything works. In subsequent labs you will build your own models and control them using ROS. Because of the way our lab is set up, you will have to prepare your work environment as described below. Please follow the instructions below carefully. **The simulation will not work if you deviate from the instructions.**

Understanding the Restrictions

Computers in our lab are set up to accommodate the needs of other courses besides E6911. They run Red Hat Enterprise Linux 8, but to use ROS we need Ubuntu 22.04. To work around this restriction, we will use Docker containers along with the Distrobox tool set that will allow us to open a command shell in Ubuntu 22.04 and still use the desktop environment provided by the host operating system. If you are setting up ROS on your private computer and you are not running Ubuntu 22.04, you can follow the same procedure described herein. If your private computer is already running Ubuntu 22.04, you can skip the Distrobox part and install ROS on the host OS.

The second restriction is that your Columbia University network account does not have the permission to create containers, so for this course **you must use the local account**. The local account should have been issued to you by the EE department support staff if you were registered for this course after the add/drop period has passed. Local accounts are tied to the particular computer, so you will be assigned one workstation on which you will have the local account and you must use that workstation throughout the semester. After the course has completed, the local accounts will be deleted, so make sure you backup your work. We recommend that you keep your code revision controlled using Git and push the code to Github regularly.

Finally, the home directories for local accounts are mounted off the root volume which is rather small. So, please **do not use the home directory as your ROS workspace.** Instead, you should use the supplementary volume mounted under `/run/host/workdir` directory. We will provide you with scripts and aliases that you should use to enter your workspace. As long as you stick to using the provided aliases, you will end up having your files at the right place. We will also periodically review how you are using the local account to make sure you remain compliant.

You may decide to use your private computer as long as it is understood that our support staff, TAs, and instructors cannot provide individualized troubleshooting support if something goes wrong with your installation and setup. We will do our best to help you, but if you are having troubles setting up or running ROS or Gazebo or the combination of the two, we will ask you to first reproduce the problem on the lab machine.

Setting up the Environment

With the above restrictions understood, now is the time to prepare your local account for using ROS. Please do not proceed until you have fully understood the above restrictions. If in doubt, please ask the instructor or the TA.

Create the Container using Distrobox

Distrobox is a set of wrapper scripts around Docker that makes it easy to create a container that once entered gives you a command shell that looks identical to your host machine. The container will give you the OS platform that matches the requirements of ROS and also inside the container you will be able to install software (a privileged operation that you are not allowed to do on the host account). For more details about Distrobox, see its Github page:

https://github.com/89luca89/distrobox

To install Distrobox, log in to your *local* account on the *host* machine and type this command (all one line):

```
curl -s https://raw.githubusercontent.com/89luca89/distrobox/main/install | sh
-s -- --prefix ~/.local
```

This will put Distrobox scripts in `.local/` directory under your home directory. No special privileges are needed to install Distrobox. They are simply a set of scripts that wrap around the underlying Docker commands. Next, update your `.bashrc` file to add Distrobox tools to your search path. Type this:

```
echo 'export PATH=$PATH:~/.local/bin:.' >> ~/.bashrc
```

To make the above change take effect, log out and log back in or simply source your `.bashrc` file like this (you should not do both):

```
. ~/.bashrc
```

You should now be able to run Distrobox tools to create the container. Type this:

```
distrobox create --name humble-e6911 --image ubuntu:22.04 --hostname humble-e6911
```

This will create the container and you should be able to see it by typing

```
distrobox list
```

The result will look like this:

```
ID           | NAME                 | STATUS              | IMAGE
2790cce0fb12 | humble-e6911         | Created             | docker.io/library/ubuntu:22.04
```

Setup ROS Inside the container

Your container is now running so it's time to enter it. Type this:

```
distrobox enter humble-e6911
```

First time you do this, it may take a few minutes to set up the container. Subsequent entries will be instantaneous. Remember, each time you open a new window and want to work on your lab, you must enter the container. ROS will not work outside the container.

Next, you must install ROS and a few other packages that lab assignments depend on. A shell script is provided to facilitate the installation:

https://github.com/ihadzic/prob_rob_labs_ros_2/blob/master/tools/setup_lab_workstation

Download the script and place it in your home directory. You will probably need to change the permissions to make the script executable:

```
chmod u+x setup_lab_workstation
```

Now run the script as root (remember you must be **inside the container!**):

```
sudo setup_lab_workstation
```

It will take a while to download and install all packages, so be patient. The script will also install three popular editors: emacs, vim, and nano. If you prefer some other editor you can install it manually using `apt` tool. You can also install other more sophisticated editors, such as Eclipse or VS Code inside the container by following the vendor's instructions.

Next, update the ROS dependencies for your account. Type this:

```
rosdep update
```

When you are done, close the terminal window and open a new one for changes to take effect. Alternatively, you can source your .bashrc file for changes to take effect. This will give you access to ROS commands. Type '`which colcon`' to verify that your setup is correct. If it displays the full path to '`colcon`' command, your environment is set up correctly.

Setting up the Workspace

Workspace is the area where your code will reside, build, and execute from. The code you will use as the baseline for your labs is available on your instructor's Github. To get started you must clone the code and build it. Make sure your environment has been set up as instructed in the previous section and that the changes have taken effect (i.e. you either opened a new shell after editing the `.bashrc` file or you have re-sourced the file).

```
mkdir -p $MY_WORKSPACE/src
go_colcon
cd src
git clone --recursive https://github.com/ihadzic/prob_rob_labs_ros_2.git
cc_prebuild
cc_build
```

This will create a directory where you will keep your source code and compiled applications and put the code you will use to do the labs. The last two commands will compile the code. First, `cc_prebuild` will compile and install everything except the Python code that you will use for the lab. Next, `cc_build` will do the so-called symlink-installation of the Python code that you will use. This type of installation allows you to modify the existing Python modules without reinstalling (they run from your source directory). That means that when you change the code in the existing module, you can just run it. However, if you add new Python modules, you will have to rerun the `cc_build` step. The pre-build step is part of the one-time setup and you will not need to repeat unless we push an update to modules other than those that you will use for the labs. If that becomes necessary, you will be instructed to re-run the pre-build step. After the build has finished, exit the container and re-enter it and your overlay should be active. Test it like this and make sure it shows the valid path:

```
ros2 pkg prefix prob_rob_labs
```

The build process will generate three new directories in the workspace. This is your overlay. The way overlays work is that if there is a ROS package installed under system root path and the same ROS package installed inside the overlay, the one inside the overlay will take precedence. In this way, you can fork-off and modify available packages and use them together with other packages available under ROS. The call to `setup.bash` script above that is under the `if` statement checks if your overlay exists and activates it. Note that before you started the build the overlay did not exist, so it will not be active until you log out and log back in the container, so you can do so now. Also, each time you add (and build) a new package to your overlay you will have to re-activate your overlay by logging out and back in. Once the package in the overlay is known to the system, you can continue to modify it and rebuild and changes will be visible to you.

Using ROS

Your container is now ready to run your ROS applications. Let us first verify some basic functionality. We will use ROS shell commands to create a topic, publish a string to the topic, and subscribe to it to receive data. Open two shell windows on your host machine and enter the container by typing '`distrobox enter humble-e6911`' in each. Once each shell is inside the container type this in the first one to start a publisher:

```
ros2 topic pub -r 10 /hello std_msgs/msg/String "{data: 'Hello World'}"
```

This will create a temporary node (with ephemeral name) and a topic called /hello of type String to which the node will publish the "Hello World!" string 10 times per second. You can see the topic by typing this in the third window:

```
ros2 topic list
```

You can also create another temporary node that will subscribe to the topic and dump the received data on the screen:

```
ros2 topic echo /hello
```

Run the Simulation

Let us now run the full simulation of a ground robot. We will use the model of Turtlebot 3 robot, which is a small-size low-cost robot widely used in robotics research. You can read more about this robot here:

https://www.turtlebot.com/turtlebot3/

Enter the container and type this to start the simulation:

```
ros2 launch turtlebot3_gazebo turtlebot3_house.launch.py
```

Gazebo will open and you will see your robot inside a small race track. You can use the mouse to pan rotate and zoom in the view. Try holding each of the mouse buttons (one at the time) without selecting any objects and pan the mouse to see what happens. Set the view that is convenient for you.

Let us now move the robot. Type this (in another window inside the container):

```
ros2 run rqt_robot_steering rqt_robot_steering
```

A new graphical interface with two control sliders will open. By moving the sliders, you can set linear and angular velocity commanded to the robot. You can also drive the robot with a joystick. To do so, connect the joystick that will be provided to you to your workstation and type this:

```
ros2 launch prob_rob_labs joystick_launch.py
```

The button that is under your right thumb as you hold the joystick is a dead-man button that you must keep pressed. Press it and move the joystick. Forward-backward motion generates the linear velocity and left-right motion generates the angular velocity. Drive it around and get the feeling for steering the robot.

Let us now inspect the commands that are being sent to the robot. Open another window and use the '`ros2 topic echo`' command to see the /cmd_vel topic. You can also plot the values by starting the signal plotting tool:

```
ros2 run rqt_plot rqt_plot
```

When the tool opens, add two signals to it: `/cmd_vel/angular/z` and `/cmd_vel/linear/x`. Use the steering tool to send commands to the robot and watch the plot. Click on the "pan" button and drag the mouse while holding the right mouse button above the plot to scale the time and amplitude.

Signals you are plotting are commands that the steering tool is sending to the robot. The robot has an on-board motion controller (whose hardware and software are captured by the robot model) which does the best effort to deliver the requested velocity. To see what the robot is actually delivering (or to be more precise what the robot **believes** it is delivering), add odometry signals to the plot:

```
/odom/angular/z
/odom/linear/x
```

Now stop the simulation, steering tool, and the plotting tool. You will next make the robot move from your program.

Your First ROS Program

After a tour of ROS, we finally arrive at your first assignment. Start Jackal simulation in an empty world:

```
ros2 launch turtlebot3_gazebo empty_world.launch.py
```

Assignment 1:

Instead of controlling the robot using the steering tool, write a Python program that moves the robot in a circle with radius of two meters and tangential velocity of 1 m/s. To do that, you will have to write a publisher that will send the command to `/cmd_vel` topic and the topic type will be `geometry_msgs/msg/Twist`. Submit the code with the assignment.

For a simple publisher example, see this tutorial:

https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Py-Publisher-And-Subscriber.html

Don't worry about creating a ROS package and the launch file (we will do that in the next lab). Just write the Python code that will create a node and publish the command, and execute it as a stand-alone program. The simple publisher shown in Section 2 of the above tutorial is a good starting point that you need to adapt to publish the robot command instead of a "hello world" string. Twist message is defined in geometry_msgs package and its definition is here:

https://docs.ros2.org/foxy/api/geometry_msgs/msg/Twist.html

You can make a new directory called `misc/` under `src/prob_rob_labs/` and put your Python file there. In the next lab, we will learn about the recommended directory hierarchy in a ROS package.

When you stop your program, the robot will continue to move. This is actually a safety flaw of Turtlebot 3. A safe robot should implement a timer and stop if no motion commands are coming, but Turtlebot requires explicit stop command. To stop the robot you can just publish zero-velocity from command line:

```
ros2 topic pub /cmd_vel geometry_msgs/msg/Twist \
    '{"angular": {"z": 0}, "linear": {"x": 0}}'
```

Appendix: Personal Device Setup

You are welcome to use your own device, laptop or a desktop PC, to do your labs. Understandably, it will be difficult for your instructor and the TA to provide personalized support for each student, but we will try to provide reasonable assistance to get you going. This appendix provides some guidelines for commonly used platforms. Note that these procedures have received much less testing than the one that use the lab workstation, so if you find problems and fixed them, please report so that we can apply the fix for other users.

Windows

ROS is designed for Ubuntu Linux, but if you have a Windows machine, you can use Windows Subsystem for Linux (WSL). The rough steps are the following:

1. Enable WSL in settings (you can find exact instructions on the Internet easily).
2. Go to Microsoft Store and find Ubuntu Linux 22.04. Install it.
3. Instead of running `setup_lab_workstation` script, run `setup_home_workstation` located. Remember that you must use `sudo` to run it. Do not become `root` first and then run the script because it will incorrectly deduce for which user it needs to set the environment.

After the setup has finished, close the terminal window (shell) and open a new one to make it re-read the environment. First test that you can publish and subscribe to topics using `ros2 topic` tools. Next test if graphical applications work (try starting Gazebo or RVIZ). If the window fails to open and your PC has NVidia GPU, try adding the following two lines to your `.bashrc` file:

```
export MESA_D3D12_DEFAULT_ADAPTER_NAME=nvidia
alias gl_software='export LIBGL_ALWAYS_SOFTWARE=1'
```

Keep in mind that this will fall back to rendering graphics in software, so things may be slow, but still usable. For Intel GPU, things should work out of the box.

Linux

If you have Ubuntu Linux 22.04 on your computer, you can just use `setup_home_workstation` without any additional steps. If you are running some other distribution, use Distrobox and follow the instructions for lab workstation set up, but after the script completes, edit your .bashrc file and change this line:

```
export MY_WORKSPACE=/run/host/workdir/$(whoami)/ros2_ws
```

to this:

```
export MY_WORKSPACE=/home/$(whoami)/ros2_ws
```

Apple

ROS support for Apple platforms is limited, so we recommend that you set up a Linux Virtual Machine and then follow the instructions for Windows setup.