Overview of this document

This document is intended to provide a basic introduction on how to work with ROS on a turtlebot for people with very little ROS experience. Code that runs in the ROS framework can be written in either C++ or Python, so it is important to be at least somewhat familiar with these languages. In all likelihood, code similar to what you want to run has already been written by someone and shared online, so the most important part is to learn how to make it operate as part of ROS. This document is not intended to teach you how to program in any specific language, but rather introduce you to how to make your code work with ROS.

This document assumes you have a working turtlebot and a computer with linux and a ROS distribution installed (Hydro, Indigo, Jade etc.). You will also need a WLAN connection for computer to send commands to itself for the turtlebot.

To understand exactly what ROS is, feel free to read this primer:

http://www.clearpathrobotics.com/2014/01/how-to-guide-ros-101/

Turtlebot Terminology:

Node- A collection of code that runs and performs computations. Nodes communicate with each other through topics.

Topic- a named communication channel between nodes. Nodes can publish and subscribe to topics in order to pass information between themselves

Publishing- the process of taking information from a node and passing it to the ROS framework for other nodes to use. Data types such as string or unsigned 16 bit integer must be specified.

Subscribing- the process of taking messages from a particular topic and using them as inputs for a node.

How to use Linux/Linux commands

Much of the work you do with ROS will happen inside a terminal window, which is essentially like command prompt for Linux. There are hundreds of possible commands, but the most important skill is being able to navigate directories, which can be done with a few specific commands typed into terminal. Typing

ls

into terminal will list the folder and files in your current folder

cd foldername

will open the file within your current folder with the name “foldername”

cd ..

will navigate you up a folder in your directory. These basic commands will get you where you need to go in a file system. Other commands can be found online or in the handy reference at the end of this section. Occasionally, you will need to use administrator privileges to perform an action. This can be done by typing

sudo

before the command you are trying to run. You will be prompted to enter the turtlebot password before the command will execute. Command “cheat sheets” such as the one below from the turtlebot manufacturer can be found on the internet:
http://www.clearpathrobotics.com/wp-content/uploads/2014/01/ROS-Cheat-Sheet-v1.01.pdf

Running preinstalled programs

There are many programs that are preinstalled on the turtlebot that you can run. Unfortunately, the source code for these programs is not on the turtlebot, only the compiled file data. You will have to find it elsewhere if you want to make tweaks and edits and then compile it using catkin_make as described in a later section. If you want to see the programs available on the turtlebot you can navigate to

/opt/ROS/Hydro/include

Running premade files on the turtlebot is very easy. Open a terminal window and type

roslaunch turtlebot_bringup minimal.launch

This runs the launch file for the turtlebot bringup program and starts all the specified nodes and code from the launch file. A lot of stuff happens behind the scenes, but you should hear the turtlebot beep, indicating that it has successfully connected to the computer. The program will run in the current terminal windows until you stop it by pressing ctrl+c. Open a new terminal window to launch other programs.

To launch the turtlebot follower program for example, make sure that the turtlebot is on the ground and free of any obstructions. Have someone stand in front of the kinect and then type the following command into a blank terminal window

roslaunch turtlebot_follower follower.launch

The turtlebot will begin to follow the person around. Be aware that all the program is doing is finding the centroid of the largest object in its field of view and keeping it a certain distance away. This means it can be confused if driving by large inanimate objects. When you are done with this program, type ctrl+c to end it.

Observing ROS behind the scenes

If you want to see the different nodes that are being launched, or the different topics that are being published to you can type in the following commands in terminal

rosnode list

rostopic list

When you run code inside the ROS framework, it needs to communicate with roscore, the central organizing component of ros. Programs like turtlebot bringup automatically launch roscore in order to be able to launch their nodes. If you want to launch nodes individually, however, you must launch roscore by typing

roscore

into the terminal windows. The node and topic lists mentioned above will not run if roscore is not running. To see the messages being published on a topic with the name “topicname” you can run the command

rostopic echo topicname

These tools are useful for debugging software and getting a better idea of how ROS helps nodes communicate.

Tweaking code: creating a workspace/downloading programs from GitHub

The easiest way to to tweak and compile code is to find it in a GitHub repository. Effective programmers know that borrowing someone else’s open source work is a much better proposition than trying to reinvent the wheel for many applications by rewriting code. GitHub is the internet’s most popular code repository, and it is easy to bring existing code onto your machine. If you wanted the source code for turtlebot follower, for example, you can navigate to the folder you want to save the program in and run

git clone https://github.com/turtlebot/turtlebot_apps/tree/hydro/turtlebot_follower.git

Which will download the turtlebot follower program and its associated data files. Alternatively, you can download and unzip the files through an internet browser. If you

open the follower.cpp file inside the src folder that was downloaded, you can see the C++ code that makes the follower program work. You can do everything from edit parameters to coding new behaviour, to publishing data from the node to other ROS nodes. Once you've made your changes, you will need to compile your code using catkin_make as described below.

Using Catkin_make to build code and manage dependencies

Programs downloaded from Github will have many parts, but two of the most important are the CMakeLists.txt and the Package.xml file. These list the dependencies on various parts of the ROS code base that are required to support the node. They will already be formatted on downloaded programs, but if you need to add or remove dependencies you can find the formatting guidelines in Clearpath's robotics cheat sheet from earlier in this document.

These dependencies are required to correctly compile to code. To compile your code, navigate to the directory above the src folder and type the following command into terminal

catkin_make

This will compile your code and let you know if there are any compilation errors.

Sourcing and running the code

If your code is not in the default directory for ROS you will need to source it with the following command

source /devel/setup.bash

You can now run your code as usual

Implementing publishing and subscribing

Implementing publishing and subscribing capabilities within programs is a fairly straightforward. Syntax varies between C++ code and python, but the idea is the same. If you want a node to publish information, you can add a few lines of code to your program to specify the name of the topic you want to publish to and what kind of message you want to send, whether it be a string or unsigned integer or other data type. Subscribing works similarly, where you add code to a node that you want to be able to receive published messages, specifying the topic it should subscribe to and the data type it should expect. For a step by step example on how to write publishing and subscribing code you can work through the following tutorial:

http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber(c%2B%2B)

Using rosserial to communicate with arduino.

If you want to integrate an Arduino with ROS, you will need to use a USB cable to establish a serial data connection with the computer. To write code for an Arduino and run it on the board, you will need the Arduino Integrated Development Environment (IDE) for Linux (64 bit) which can be downloaded from

https://www.arduino.cc/en/Main/Software

You will also need the rosserial package for your computer which can be downloaded from

https://github.com/ros-drivers/rosserial.git

Once you have the IDE and rosserial package downloaded and installed, you can write C++ code just like you would for other non-ROS based applications of the Arduino. The rosserial package allows the Arduino to communicate with the laptop over

the USB cable when you need to publish or subscribe to topics. To use standard integer variables as your data type, make sure to write

#include <stdint.h>

somewhere at the top of the Arduino code.

Plenty of examples of code using Arduino with ROS can be found at: http://wiki.ros.org/rosserial_arduino/Tutorials or by browsing the examples located within the Arduino IDE:

File>Examples>ros_lib

The final step in communicating with an Arduino is that once your are running code on the Arduino, you need to run the following command in a terminal window on your computer

rosrun rosserial_python serial_node.py /dev/ttyUSB0

Where “USB0” is replaced with the appropriate name of the COM port that the Arduino is plugged into. If you are unsure, this can usually be found in the tools menu in the IDE. This command enables the Arduino to become a part of the ROS network running on the computer and send or receive messages.

Conclusion

Hopefully this document has provided the reader with insight into some aspects of ROS and how to go about running and creating basic programs. The possibilities of ROS in robotics extend far past the scope of this document, but once you overcome the learning curve of understanding how different components of ROS interact with each other, it becomes much easier to capitalize on the ability of ROS to combine different code bases into a functioning project. Armed with this basic knowledge, it becomes easier to leverage the open source nature of the robotics software community and the availability of knowledge bases on the internet to create successful robotics projects.