Overview of the LabVIEW-ROS Toolkit

Overview

Robot Operating System (ROS) has rapidly become one of the premiere architectures for developing new robotics technology. Its flexibility and community provides an excellent way to quickly integrate new sensors and share new code. However, it is not suitable to control real-time systems nor is it designed to interface with measurement and test systems. LabVIEW is uniquely suited to address some of these deficiencies. The Clearpath Robotics LabVIEW-ROS Toolkit allows the power of LabVIEW hardware and software to be brought to ROS systems.

System Architecture

The ROS Toolkit establishes an interface between LabVIEW and ROS, allowing communication to be established as long as a connection can be established between the computer running ROS and the computer running LabVIEW. This connection can be on the same machine via the *localhost* interface, or it can be over the internet.

Two general configurations are possible. When the robot is ROS-enabled and LabVIEW is used as a high-level control system, it allows for the easy creation of GUIs with LabVIEW VIs. When the robot is LabVIEW-enabled and ROS is used as a high-level control system, LabVIEW and NI hardware can be used for low level real-time control.

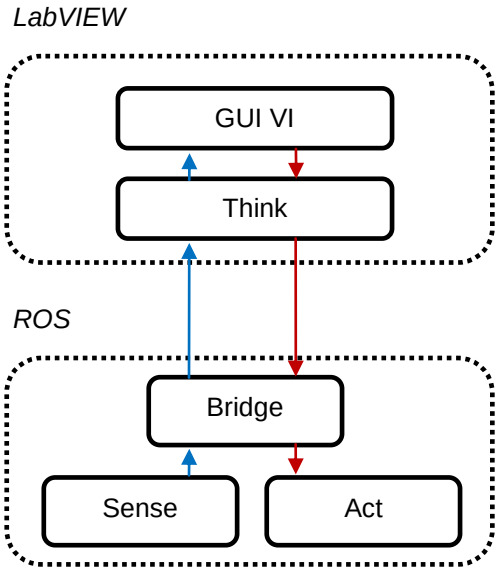

Figure 1: Controlling a ROS-enabled robot

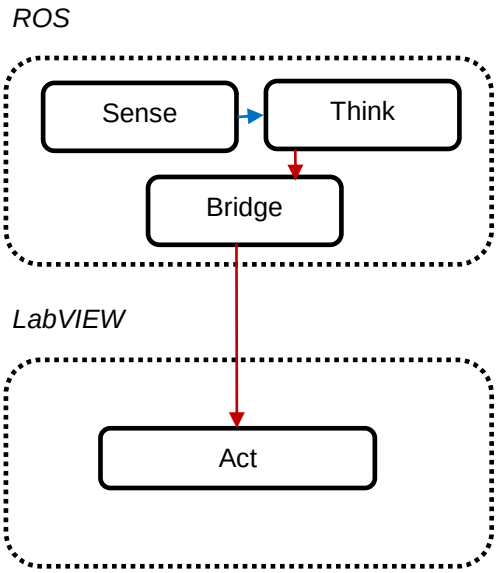

Figure 2: Controlling a LabVIEW-enabled robot

The toolkit fits cleanly into existing LabVIEW and ROS software. On the ROS side, it is only necessary to install and run the rosbridge node to provide the toolkit with access. Likewise, the various components of the toolkit can

be manipulated like any other LabVIEW sub-VI. For example, the included Joystick Demo program only requires three toolkit specific blocks; the first to open the connection (1), the second to generate control messages from standard mathematical operations (2), and the third (3) to close the connection.

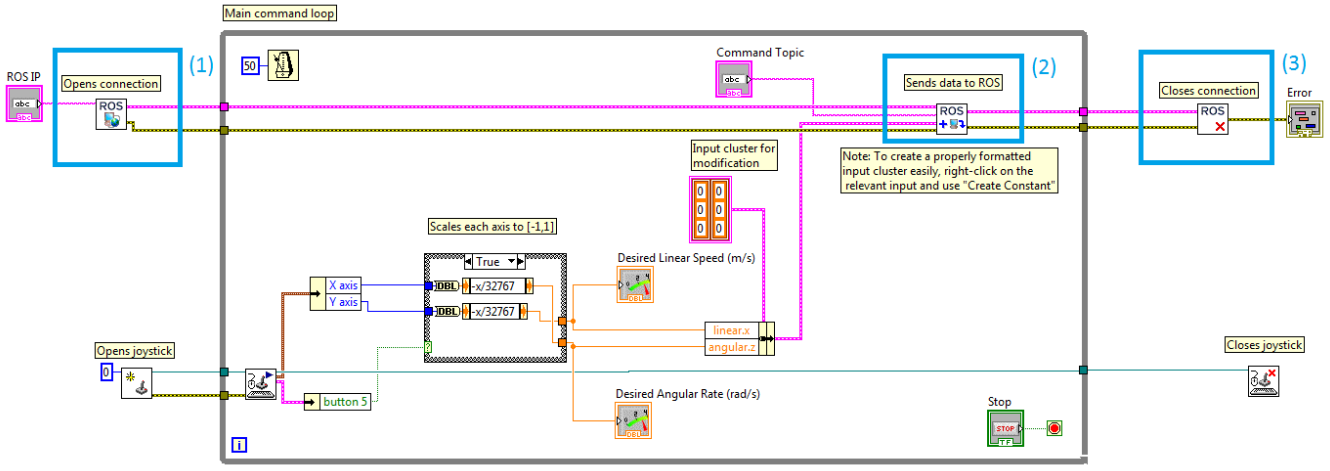

Figure 3: Using ROS Toolkit to control a mobile robot

There still remain a few limitations to the software. Namely, if the toolkit is to be deployed on targets running LabVIEW Real-Time, the blocks which allow for dynamic creation of ROS messages from LabVIEW clusters and vice-versa will not work.

Control of a ROS-based Simulation Engine

This toolkit can be explored even if access is not available to a ROS-enabled robot. The ROS ecosystem provides a number of easy-to-install simulated robots based on the Gazebo simulation environment.

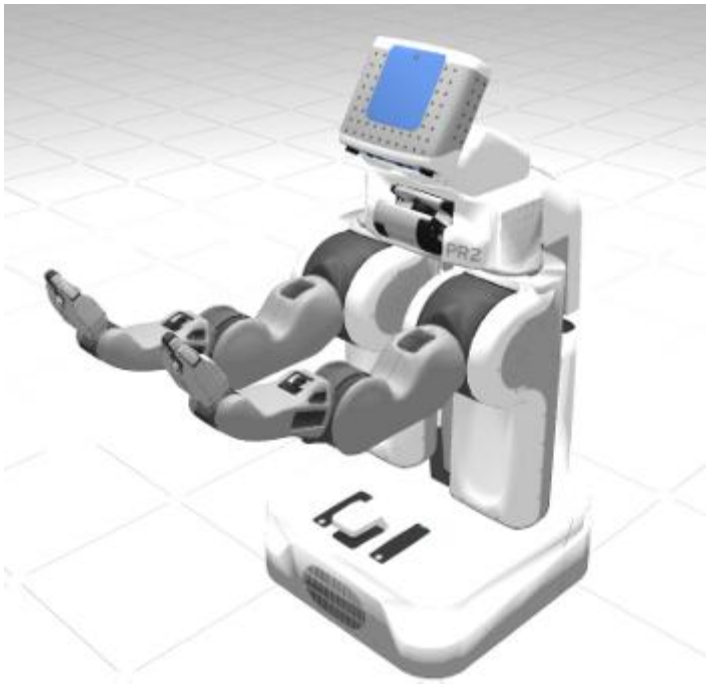

Figure 4: Simulated PR2 robot

First, the main ROS programs and some supporting packages must be installed on the computer you have selected to run ROS. Though ROS is tentatively supported on a number of platforms, using Ubuntu 10.04-11.10 is advised at the present time.

1. Follow the installation instructions for ROS Electric and your particular platform.
2. Install the "rosbridge" package via:
   sudo apt-get install ros-electric-brown-remotelab
3. Install the TurtleBot simulator with:
   sudo apt-get install ros-electric-turtlebot-simulator-desktop
4. At this point, open a terminal window and start the simulator by running:
   roslaunch turtlebot_gazebo turtlebot_empty_world.launch

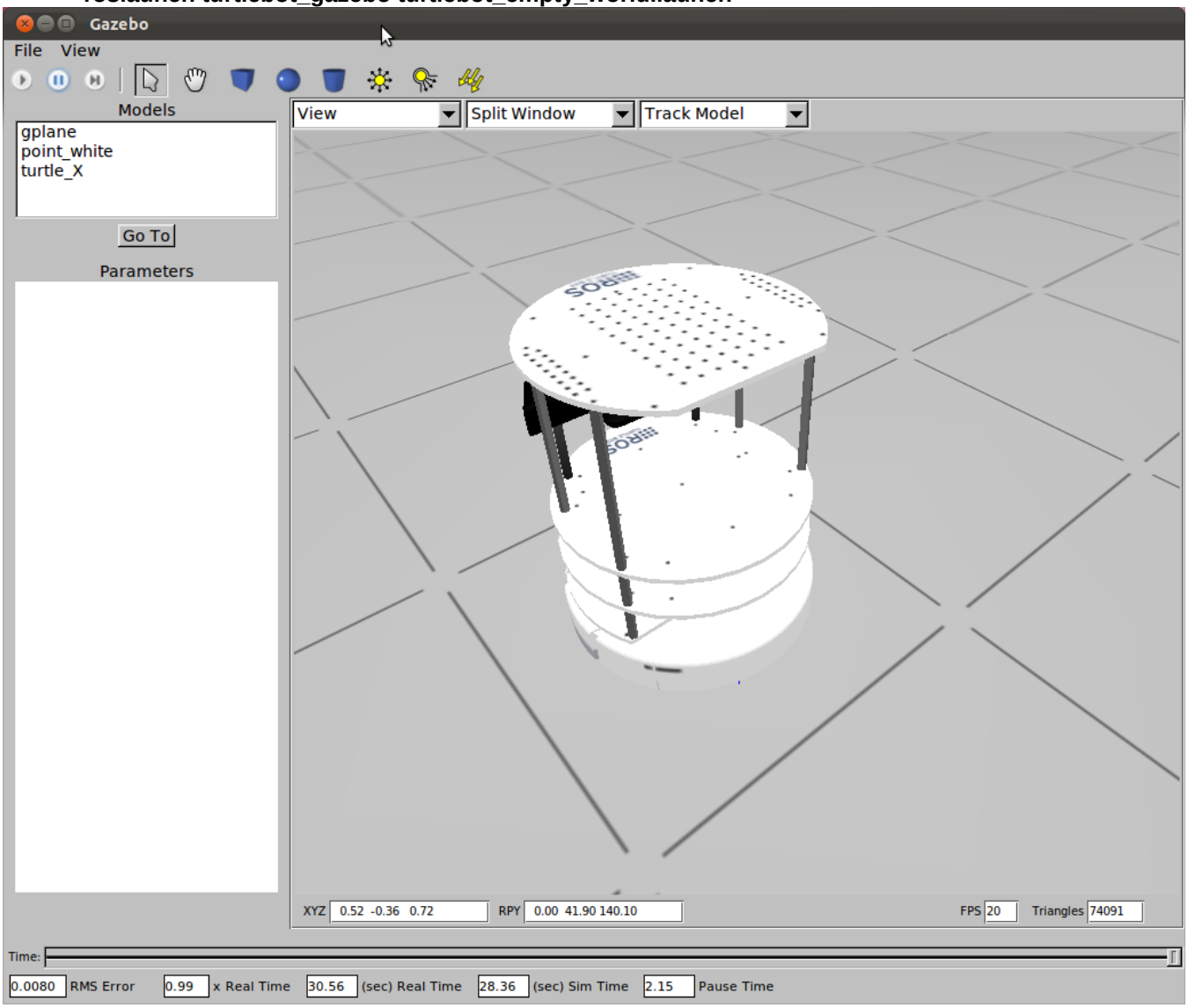

Figure 5: Gazebo simulation of a TurtleBot

5. With the TurtleBot simulation running, use **rosbridge** to allow LabVIEW to connect. Open a third terminal and run:
   rosrun rosbridge rosbridge.py
6. Finally, open a fourth terminal on the ROS computer and use **ifconfig** to determine the computer's IP.
7. On the LabVIEW computer, verify that you can access the simulation computer by using **ping**.
8. Load the Joystick Demo from the Example Finder.

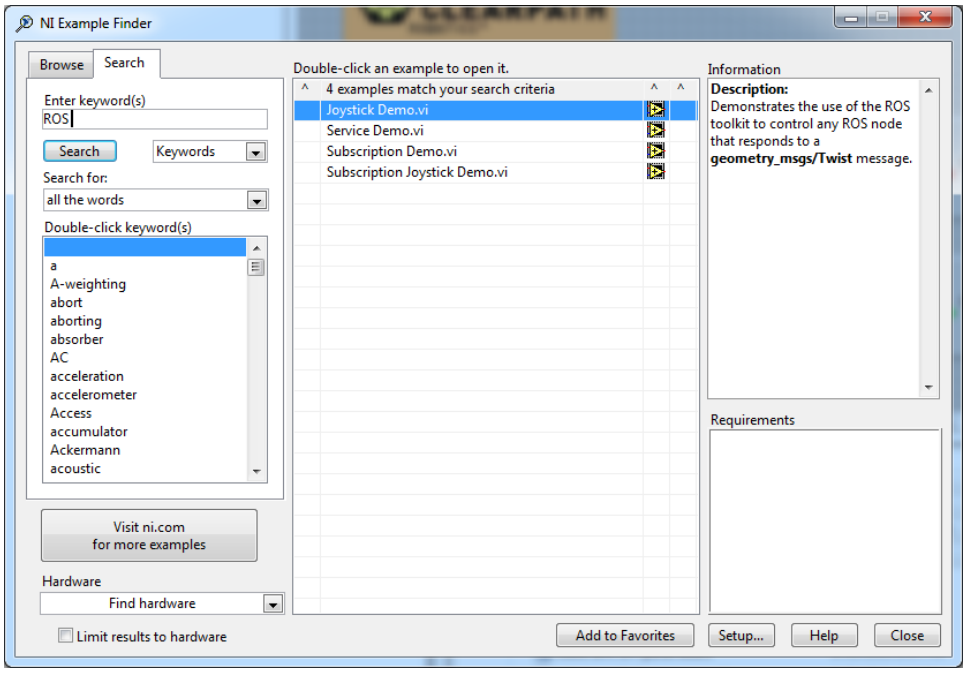

Figure 6: ROS Toolkit Examples

9. Plug a joystick into the LabVIEW computer.
10. Enter in the IP of the ROS computer that was found in Step #6, and run the demo.

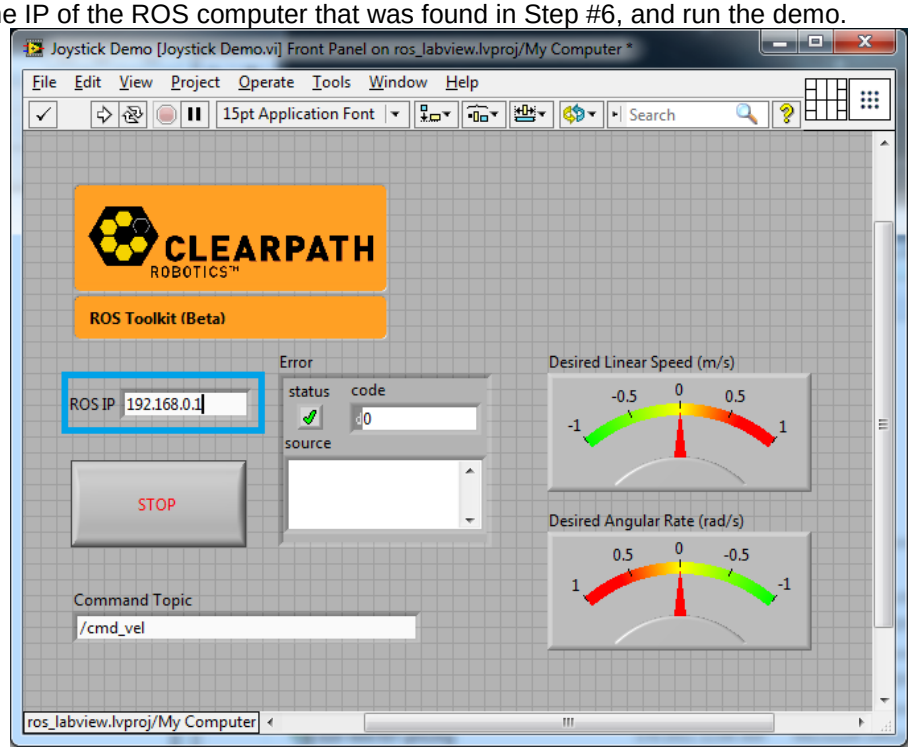

Figure 7: Joystick Control Demo

11. Depress Button #1 on the joystick and begin driving your simulated robot!

If you would like to examine the ROS commands that the toolkit is generating, open a terminal on the ROS computer and type:

rostopic echo /cmd_vel

Control of a TurtleBot

The easiest ROS-enabled robot to work with is the TurtleBot educational platform. The computer comes with ROS installed and tested, so no additional modifications are necessary. To drive this robot from LabVIEW, only one extra software package is required.

Figure 8: TurtleBot

1. Install the "rosbridge" package on the TurtleBot netbook via:
   sudo apt-get install ros-electric-brown-remotelab
2. Since ROS starts up automatically on the TurtleBot, all that is necessary to allow LabVIEW to connect is to use **rosbridge**. Open a terminal on the TurtleBot netbook and run:
   rosrun rosbridge rosbridge.py
3. Open a second terminal on the TurtleBot netbook and use **ifconfig** to determine the netbook's IP.

The remaining steps are identical to those used to control a simulated robot. Since both the simulated TurtleBot and the real TurtleBot subscribe to ROS Twist messages, the only thing that needs to be changed in the Joystick Demo VI is the IP which the LabVIEW computer is targeting.

4. On the LabVIEW computer, verify that you can access the simulation computer by using **ping**.
5. Load the Joystick Demo from the Example Finder.
6. Plug a joystick into the LabVIEW computer.
7. Enter in the IP of the ROS computer that was found in Step #6, and run the demo.
8. Depress Button #1 on the joystick and begin driving your TurtleBot!

Control of a Mobile Manipulator

Using ROS to control LabVIEW-powered systems is also done in a similar way. A Husky A200 UGV was outfitted with a custom-built robotic manipulator. Due to the real-time control requirements of this manipulator and multitude of I/O, it was built around a sbRIO-9606.

Figure 9: Husky A200 equipped with custom manipulator

The ROS Toolkit was deployed on the sbRIO-9606 to allow controls development to be conducted in LabVIEW and the overall planning software to be built with ROS. This is an excellent demonstration of the complimentary power of the hardware; high-bandwidth control can be done by the sbRIO while ROS provides a multitude of drivers and supporting software specific for autonomous systems.

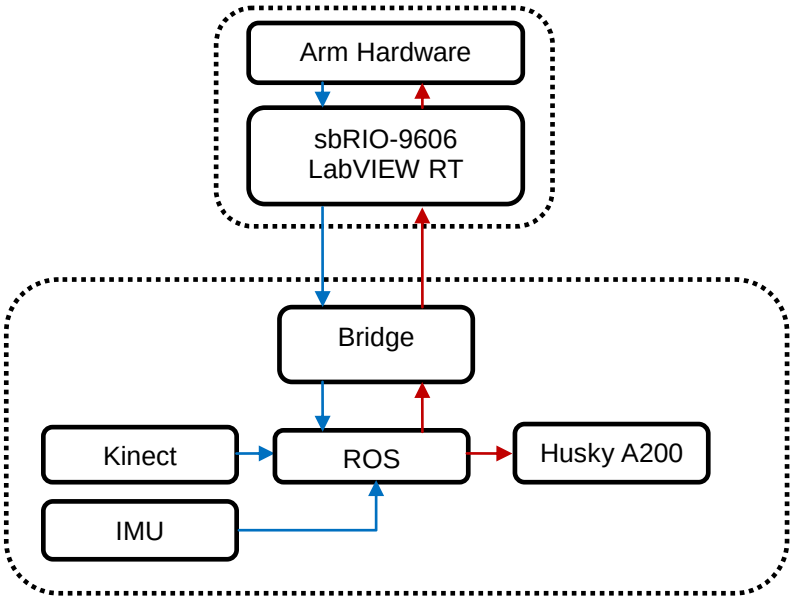

Figure 10: Architecture of a Mobile Manipulator

Dynamic Message Generation and Parsing

The toolkit supports both generation and parsing of arbitrary ROS messages. For example, a cluster can be created with the same structure as a given ROS message. When a reference to this cluster along with input data from the "ROS Read" VI is passed into the "JSON to Cluster Recurse" VI, the cluster will be populated with data from the incoming ROS message.

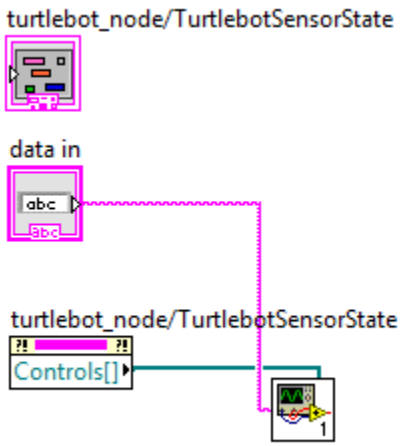

Figure 11: Parsing a user-created TurtlebotSensorState message

Likewise, passing a cluster reference, topic name, and topic type to the "ROS Send Generic" VI will dynamically create and transmit a matching ROS message on the specified topic. Due to restrictions in LabVIEW RT, neither the dynamic parsing nor the dynamic generation features will work when deployed to real-time targets.

Basic ROS Commands

There are many tutorials available for ROS, as well as a set provided specifically for the TurtleBot. As well, several basic debugging tools are made available at the command line. They are all prefixed with **ros** and most have descriptive names.

For example, **rosnode** provides information about currently running nodes. **rosnode list** will list them, and **rosnode info <NODE>** will provide information about the specified node. Likewise, **rostopic** exposes details about current topics. **rostopic list** will list the current topics being published, and **rostopic echo <TOPIC>** will display the latest data being sent on the named topic.

Reference

Please contact support@clearpathrobotics.com with questions, suggestions, or if specific message types are required.