ROS User's Guide

February 27

2013

This guide is intended to aid end users when using ROS on a CoroWare robot.

The End User's Guide to ROS

ROS User's Guide

Version 7

Table of Contents

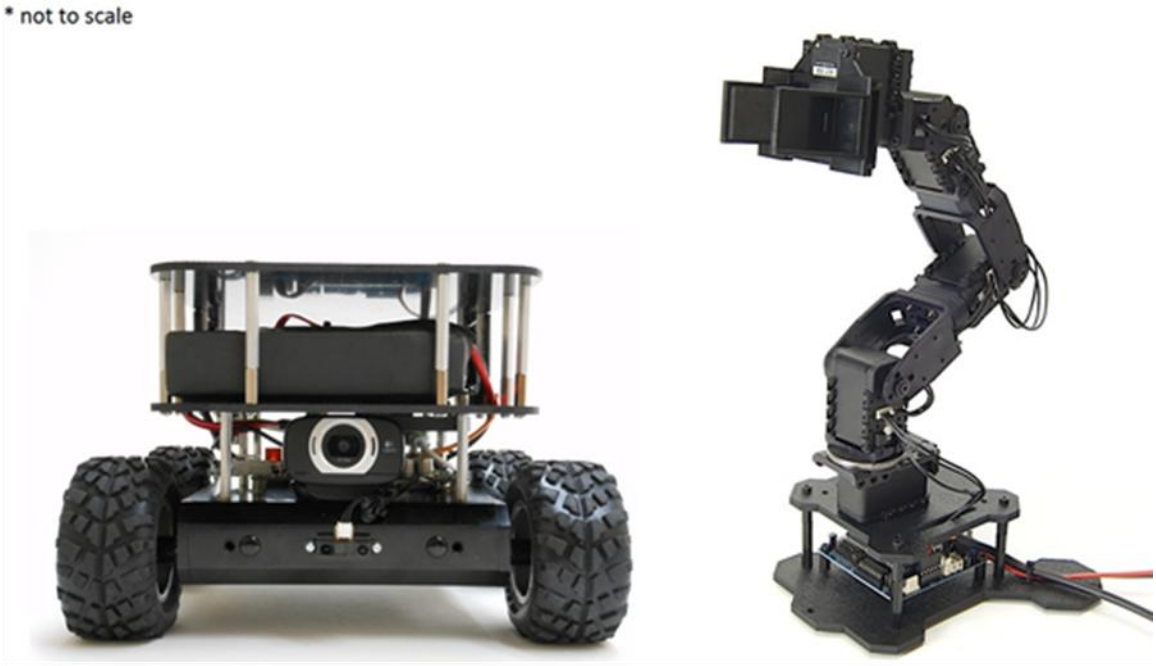

Figure 1- CoroBot and Arm

1. ROS architecture

Robot Operating System is an open-source framework for robotics development. It is available both in C++ and in Python, and is in beta version in Java. ROS is composed by a master that will coordinate one or several processes (nodes). Each node can use the ROS library to publish (or subscribe) some messages over topics, easily get the value of some parameters from the user, etc…

Nodes are grouped to form a package (but we often see one node per package) and packages are grouped to form a stack.

ROS' key points will be explained in this section.

1.1. Topics

A topic is a named bus over which nodes can exchange messages of a specific type. Messages type are defined with .msg files, several types already exist but it is possible for anyone to create new ones. Nodes can either publish to that topic or subscribe to that topic. Subscribing nodes will receive the new message in a callback function as soon as a new message has been published.

For more information, please see the topics wiki page and the tutorial:

http://www.ros.org/wiki/Topics

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

1.2. Services

Services are some function defined in a node which execution can be called by another node. Some parameters can be sent when calling the service. Services are indeed similar to topics, the difference being that at the end of the execution of the service a response will be sent to the calling node. However, being slower than topics, it is advice to use only if necessary.

For more information, Please see the wiki page and the tutorial:

http://www.ros.org/wiki/Services

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

1.3. More information

For more information on ROS architecture and goals please refer to the page:

http://www.ros.org/wiki/ROS/Introduction

Several tutorial can be found on ROS website and we advise you to take a look at them:

http://www.ros.org/wiki/ROS/Tutorials

2. Useful ROS commands

Here is a still of some important commands that can be written in a terminal. A more complete list of commands can be found on ROS website:

http://www.ros.org/wiki/Documentation?action=AttachFile&do=view&target=ROScheatsheet.pdf

2.1. Start ROS Master

This command starts ROS, it is necessary if you want to start a node:

```
roscore
```

2.2. Start a Node

Once you node the name of the package and the node, use this command to execute the node:

```
rosrun package_name node_name
```

The launch files are configuration files that permit to launch more than one node at a time, all in the same terminal. To start a launch file just do:

```
roslaunch package_name launchfile
```

In the case of CoroBot, you can start the remote control node and the all the necessary nodes by executing the command:

```
roslaunch launch_corobot_gui corobot_ros_gui.launch
```

2.3. Create a stack

To create a ROS stack just use the command:

```
roscreate-stack stack_name
```

2.4. Create a package

To create a ROS package just use the command:

```
roscreate-pkg package_name
```

2.5. Compiling a ROS package

ROS uses Cmake to compile packages, therefore you shouldn't do any Cmake command, because ROS does it for you. Once you have put a package folder in a package path folder (like ~/ros_packages), just compile it:

```
rosmake --pre-clean package_name
```

Note that the –pre-clean command is optional and is used to clean compilation files and recreate the makefile from the Cmake file.

2.6. Find a Package

This command displays all the packages and their directory path:

```
rospack list
```

This command displays the directory path of the package, if found:

```
rospack find package_name
```

2.7. Find a Running Node

This command displays all the nodes that are actually running. The command roscore has to be running on another terminal to make it work:

```
rosnode list
```

Also, rosnode can kill a running node:

```
rosnode kill node_name
```

2.8. Topics

It is possible to display the different topics that are actually advertised:

```
rostopic list
```

You can also publish any message on a topic:

```
rostopic pub -1 topic message_type -- arguments
```

The -1 is here to tell that we want one message and exit. The double-dash is here to tell that none of the arguments are an option (necessary if you have some negative values).

Finally, you can display every message being published on a topic:

```
rostopic echo topic_name
```

2.9. Services

To see the list of all the services of any package, you can type this command:

```
rossrv package package_name
```

2.10. Show the Connection of the Running Nodes

It is possible to display a graph showing how the nodes are connected to each other with the command:

```
rxgraph
```

2.11. Console

You can see the all the output of any node in one window by typing in a new terminal:

```
rxconsole
```

There are different types of output (Debug, Info, Warning, Error, Fatal). The logger level by default is Info, that means you will see all messages from Info level and higher priority. If you want to see all output possible (put the logger level to Debug) or see only important messages (setting the logger level to Error), you have to type in a new terminal:

```
rxloggerlevel
```

A new window will open where you can change the logger level of any node.

3. Development

3.1. Launch files

When you want to create an application using several nodes you will want to launch them all at once, that is the purpose of launch files. You can also give values to some parameters that will be read by nodes. For more information about launch files, please see the tutorial:

http://www.ros.org/wiki/roslaunch/Tutorials/Roslaunch%20tips%20for%20larger%20projects

In the CoroBot stack, the package corobot_teleop has a launch file. Some parameters are found on this file, such as selection of the material, port, and baud rate.

To change them, open the following file with your preferred editor:

```
corobot_teleop/launch/corobot_ros_gui.launch
```

Alternatively you can do the following:

```
roscd corobot_teleop; gedit launch/corobot_ros_gui.launch
```

In this file, you will see some lines that look like this:

```
<arg name="ssc32" value="1" />
```

The name parameters are corresponding to a material. Setting the value to 1 enables the material and setting the value to 0 disables it.

Also, there are some other parameters such as the port path of the material:

```
<arg name="camera_port" value="/dev/video0" >
```

ATTENTION:

The software may not work if values are set incorrectly.

3.2. Connect two computers with ROS

You will need two computers, for example a laptop and a Corobot. Once a network is set up between those two machines, you have to configure ROS.

One machine will be the server (the robot), the other will be the client.

On the server you have to execute in the terminal the command:

```
export ROS_IP=xxx.xxx.xxx.xxx
export ROS_MASTER_URI=http://localhost:11311
```

Where xxx.xxx.xxx.xxx is the ip address of the server.

NOTE: *You can edit the ~/.bashrc file if you don't want to execute this command at every new terminal.*

On the client, you will have to execute the command:

```
export ROS_IP=xxx.xxx.xxx.xxx
export ROS_MASTER_URI=http://xxx.xxx.xxx.xxx:11311
```

Where the first xxx.xxx.xxx.xxx is the ip address of the client machine and the second is the ip address of the server.

NOTE: *You can edit the ~/.bashrc file if you don't want to execute those commands at every new terminal.*

The client and the server are now connected.

You can specify the ROS_IP and ROS_MASTER_URI in the interface of the corobot_teleop package, which will work only for the corobot_teleop node and no other.

3.3. The CoroBot stack

We will go through every package of the CoroBot stack describing briefly there goal and the list of published and subscribed topics, parameters and services. You can also find this information on these webpages:

- http://www.ros.org/wiki/Robots/Corobot
- http://www.ros.org/wiki/Corobot

3.3.1. corobot_arm

This package is used to control any of the Corobot arm, having either a ssc32, phidgets servo or arbotix controller.

Topics:

Input:

Command	Topic Type	Use
armPosition	corobot_msgs/MoveArm	Give an order of angle position for a specific servo motor.

Output:

Command	Topic Type	Use
phidgetServo_setPosition	corobot_msgs/ServoPosition	Give an order of angle position for a specific servo motor.
phidgetServo_setType	corobot_msgs/ServoType	Give the type of servo motor for a specific servo connected to the Phidget.

Parameters:

Command	Parameter	Use
dynamixels	default = None	Configuration of all the servos. Parameter given in the yalm file.
controller_type	default = None	Type of servo controller board: arbotix, ssc32 or phidget. Parameter given in yalm file.

3.3.2. corobot_camera

This package is used to acquire images from the camera.

Services:

Command	Service Type	Use
camera/set_control	corobot_srvs/setcontrol	Set camera control.

Topics:

Input:

Command	Topic Type	Use
camera/set_videomode	corobot_msgs/videomode	Set the video mode
camera/set_state	corobot_msgs/state	Set the camera state

Output:

Command	Topic Type	Use
/camera/image_raw (or value of the topic parameter)	sensor_msgs/Image or sensor_msgs/CompressedImage	Image of the camera

Parameters:

Command	Parameter	Use
device	default = "/dev/video0"	path to the camera
topic	default= "/camera/image_raw"	topic name for the image
width	default = 960	width of the image
height	default = 720	height of the image
fps	default = 30	frames per second
camera_parameter_file	default = "../camera_parameters.txt"	path to the configuration file for the camera
immediately	default = true	Start the camera automatically after the node starts

3.3.3. corobot_gps

This package is used to acquire data from a GPS, if present of the robot.

Topics:

Output:

Command	Topic Type	Use
extended_fix	corobot_msgs/GPSFix	Send the GPS data
fix	sensor_msgs/NavSatFix	Send the GPS data

Parameters:

Command	Parameter	Use
use_gps_time	default = true	use the time from the GPS or not
host	default = "localhost"	host name or ip address of the computer connected to

		the gps
port	default = 2947	port where the gps sends data to

3.3.4. corobot_hokuyo

This package is used to distances data from a laser range finder Hokuyo.

Topics:

Output:

Command	Topic Type	Use
scan	sensor_msgs/LaserScan	Scan data from the laser
diagnostics	diagnostic_msgs/DiagnosticStatus	Diagnostic status information.

Services:

Command	Service Type	Use
self test	diagnostic_msgs/SelfTest	Starts the Hokuyo self test, which runs a series of tests on the device. The laser stops publishing scans during the test, which takes about a minute. The result of the test is in the response returned by this service. At the conclusion of the test, the laser is returned to its normal operating mode.

Parameters:

Command	Parameter	Use
min_ang	default = -π/2	The angle of the first range measurement in radians (range is [-π,π], though most devices have a smaller feasible range).
max_ang	default = π/2	The angle of the last range measurement in radians (range is [-π,π], though most devices have a smaller feasible range).
intensity	default = false	Whether or not the hokuyo returns intensity values.
cluster	default = 1	The number of adjacent range measurements to cluster into a single reading;

		the shortest reading from the cluster is reported.
skip	default = 0	The number of scans to skip between each measured scan. This controls the update rate. For a UTM-30LX, the hokuyo will scan at 40Hz, so setting "skip" to 1 makes it publish at 20Hz.
port	default = /dev/ttyACM0	The port where the hokuyo device can be found.
calibrate_time	default = true	Whether the node should calibrate the hokuyo's time offset on startup. If true, the node will exchange of series of messages with the device in order to determine the time delay in the USB connection. This calibration step is necessary to produce accurate time stamps on scans.
frame_id	default = laser	The frame where laser scans will be returned. This frame should be at the optical center of the laser, with the x-axis along the zero degree ray, and the y-axis along the 90 degree ray.
time_offset	default = 0.0	An offet to add to the timestamp before publication of a scan Range: -0.25 to 0.25 (New in release 1.0.1)
allow_unsafe_settings	default = False	Turn this on if you wish to use the UTM-30LX with an unsafe angular range. Turning this option on may cause occasional crashes or bad data. This option is a tempory workaround that will hopefully be removed in an upcoming driver version. (New in release 1.0.3)

3.3.5. corobot_joystick

This package is used to control CoroBot with a gamepad.

3.3.6. corobot_map_to_jpeg

This package contains two nodes, one used to convert a map into an image and the other to convert a jpeg into a map.

map_to_image_node:

Topics:

Input:

Command	Topic Type	Use
pose	geometry_msgs/PoseStamped	position of the robot
map	nav_msgs/OccupancyGrid	map generated by a SLAM algorithm

Output:

Command	Topic Type	Use
map_image/full_with_position	sensor_msgs/Image or sensor_msgs/CompressedImage	Full map showing the position of the robot
map_image/full	sensor_msgs/Image or sensor_msgs/CompressedImage	Full map without the position of the robot
map_image/tile	sensor_msgs/Image or sensor_msgs/CompressedImage	Map of the environment surrounding the robot

image_to_map_node

Topics:

Input:

Command	Topic Type	Use
map_image_raw	sensor_msgs/Image	Map of the environment

Output:

Command	Topic Type	Use
/map_from_jpeg	nav_msgs/OccupancyGrid	Map of the environment

Parameters:

Command	Parameter	Use
resolution	default = 0.05	The map resolution in meters. This is the length of

		a grid cell edge.

3.3.7. corobot_msgs

No node is present in this package where only messages are created. Please see the msg folder for more information.

3.3.8. corobot_pantilt

This package is used to move the pan tilt camera, if one is present.

Parameters:

Command	Parameter	Use
device	default = /dev/video0	path to the camera
script_path	default = init_camera.sh	path to an initialization script necessary for the pantilt to work

Topics:

Input:

Command	Topic Type	Use
/joy	sensor_msgs/Joy	Command data from the joystick or gamepad
/pantilt	corobot_msgs/PanTilt	Pan and Tilt command

3.3.9. corobot_phidgetIK

This package interfaces the Phidget Interface Kit, which is used to get some sensor data. Also this package interfaces the Phidget Encoders board to get encoders data.

Services:

Command	Service Type	Use
phidgetnode_set_odom	corobot_srvs/SetOdom	This service is used to set the odometer.

Topics:

Output:

Command	Topic Type	Use
position_data	corobot_msgs/PosMsg	This topic sends every 50ms the position of the robot.
infrared_data	corobot_msgs/IrMsg	This topic sends every 50ms the voltage value of both

		infrared sensors. It also send the conversion in meters.
power_data	corobot_msgs/PowerMsg	This topic sends every 50ms the power of battery left(in Volt)
bumper_data	corobot_msgs/BumperMsg	This topic sends every 50ms information coming from the Bumper to tell if an obstacle has been hit or not.
gripper_data	corobot_msgs/GrippergMsg	This topic sends every 50ms information coming from the gripper.
spatial_data	corobot_msgs/spatial	This topic sends every 50ms information coming from the imu.
imu_data	sensor_msgs/Imu	This topic sends every 50ms information coming from the imu.
phidget_info	corobot_msgs/phidget_info	This topic sends once at the creation of the node information related the initialization of the phidget device.
sonar_data	corobot_msgs/RangeSensor	This topic sends every 50ms the range in meters of every sonar sensors connected, if any.

Parameters:

Command	Parameter	Use
rearBumper	default = false	Defines if some bumpers are present at the rear of the robot.
bwOutput	default = -1	pin number where the bw Output sonar connector is connected to the phidget Interface Kit, if any sonar is present.
strobeOutput	default = -1	pin number where the strobe Output sonar connector is connected to the phidget Interface Kit, if any sonar is present.

firstSonarInput	default = -1	pin number where the first sonar is connected to
lastSonarInput	default = -1	pin number where the last sonar is connected to
battery	default = 0	pin number where the battery sensor is connected
irFront	default = 1	pin number where the front infrared sensor is connected
irBack	default = 2	pin number where the back infrared sensor is connected
gripper	default = 3	pin number where gripper is connected

3.3.10. corobot_srvs

No node is present in this package where only messages are created. Please see the msg folder for more information.

3.3.11. corobot_ssc32

This package is used to control the movement of both the robot and the arm in the case the robot has a ssc32 device instead of Phidgets board (so only for older models).

Services:

Command	Service Type	Use
ssc32_set_speed	corobot_srvs/SetSpeed	Set movement speed to the robot
move_arm	corobot_srvs/MoveArm	Move the robotic arm
move_wrist	corobot_srvs/MoveArm	Move the wrist of the arm
move_gripper	corobot_srvs/MoveArm	Move the gripper of the arm
reset_arm	corobot_srvs/MoveArm	Reset the position of the arm

Topics:

Input:

Command	Topic Type	Use
setPositionServo	corobot_msgs/ServoPosition	Set the angle of one servo
/ssc32_velocity	corobot_msgs/MotorCommand	Set the speed of the robot

Output:

Command	Topic Type	Use

ssc32_info	corobot_msgs/ssc32_info	Give information on the connection of the ssc32 board

Parameters:

Command	Parameters	Use
ssc32_port	default = "/dev/ttyS1"	port to the ssc32 device

3.3.12. corobot_state_tf

This package is used to calculate the odometry using the encoders only.

Topics:

Input:

Command	Topic Type	Use
position_data	corobot_msgs/PosMsg	Encoders data

Output:

Command	Topic Type	Use
odometry	nav_msgs/Odometry	Odometry Information

Parameters:

Command	Parameter	Use
Explorer	default = false	Defines if the robot is a CoroBot Explorer or a CoroBot classic
publish_odom_tf	default = true	Choose to publish or not the tf transform message

3.3.13. corobot_teleop

Remote control software to control a CoroBot.

3.3.14. PhidgetMotor

This package is used to control the movement of the robot in case the robot has a Phidget Motor controller (newest models)

Topics:

Input:

Command	Topic Type	Use
PhidgetMotor	corobot_msgs/MotorCommand	Command the movement of the robot

Services:

Command	Service Type	Use
PhidgetMotor	corobot_srvs/Move	Command the movement of the robot

3.3.15. PhidgetServo

This package is an interface for the Phidget Servo controller to move the arm, if any.

Services:

Command	Service Type	Use
phidgetServo_getEngaged	corobot_srvs/GetEngaged	Get the status of a connected servo motor, to know if it is engaged or not
phidgetServo_setEngaged	corobot_srvs/SetEngaged	Set the status of a servo motor, engaged or not
phidgetServo_getPosition	corobot_srvs/GetPosition	Get the position of a servo motor.
phidgetServo_setPosition	corobot_srvs/SetPosition	Set the position of a servo motor.
phidgetServo_getPositionMax	corobot_srvs/GetPosition	Get the maximum position possible of a servo motor.
phidgetServo_getPositionMin	corobot_srvs/GetPosition	Get the minimum position possible of a servo motor
phidgetServo_getMotorCount	corobot_srvs/GetMotorCount	Get the maximum number of servo motors that can be connected to this Phidget Servo controller.
phidgetServo_setServoParameters	corobot_srvs/SetParam	Get the parameters of a servo motor.
phidgetServo_getSerialNumber	corobot_srvs/GetSerialNumber	Get the serial number of the current phidget servo controller. Useful if there are two phidget servo controllers connected.

phidgetServo_getServoType	corobot_srvs/GetType	Get the type of servo motor pluged in
phidgetServo_setServoType	corobot_srvs/SetType	Set the servo motor to one of the defined types

Topics:

Input:

Command	Topic Type	Use
phidgetServo_setPosition	corobot_msgs/ServoPosition	Set the position possible of a servo motor
phidgetServo_setType	corobot_msgs/ServoType	Set type of the concerned servo

Output:

Command	Topic Type	Use
phidgetServo_getPosition	corobot_msgs/ServoPosition	Get the position possible of a servo motor

Parameters:

Command	Parameters	Use
serialNumber	default = -1	If more than one Phidget Servo controller are present you can set the serial number of the one you want to use.