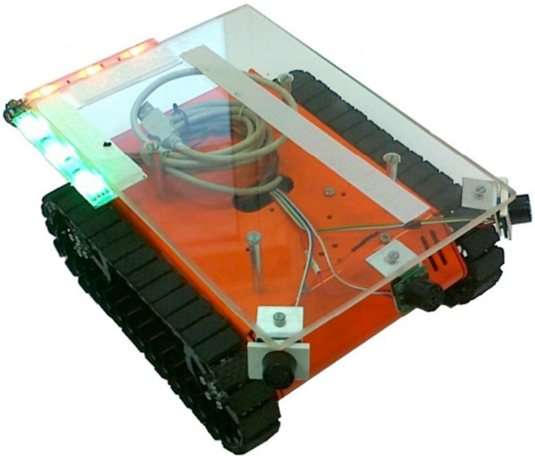


Institute of

Systems and

Robotics

TraxBot

ROS Driver Guide

André Araújo

David Portugal

Micael Couceiro

Rui P. Rocha

Document	ROS Driver Guide
Project	TraxBot Platform
Version	V1.1
Date	July 13, 2012
State	pre-release
Distribution	Public

Mobile Robotic Laboratory (MRL)

Institute of Systems and Robotics - University of Coimbra (ISR-UC)
Portugal
Contact person: Rui P. Rocha
E-mail address: rprocha@isr.uc.pt
Webpage: http://paloma.isr.uc.pt/mrl/

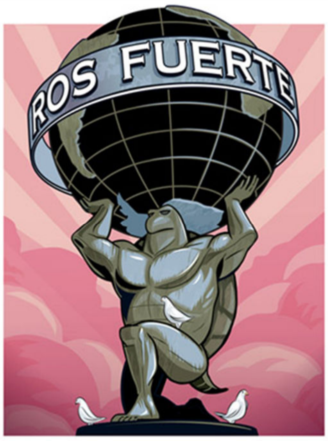
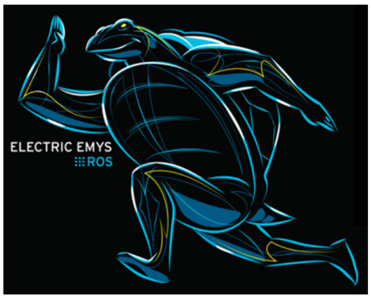
Authors contact: André Araújo, David Portugal, Micael Couceiro
E-mail address: aaraujo@isr.uc.pt , davidbsp@isr.uc.pt , micaelcouceiro@isr.uc.pt

Contents

Chapter 1 – Installation Guide

1.1. Requirements

- Install ROS according to the Ubuntu version running on your machine and your preferences and needs.

Ubuntu Compatible Distribution	ROS version	Installation Instructions
Ubuntu: - **10.04** (Lucid Lynx) - **11.10** (Oneiric Ocelot) - **12.04** (Precise Pangolin)	 Released on April 23, 2012	http://ros.org/wiki/fuerte/Installation
Ubuntu: - **10.04** (Lucid Lynx) - **10.10** (Maverick Meerkat) - **11.04** (Natty Narwhal) - **11.10** (Oneiric Ocelot)	 Released August 30, 2011	http://ros.org/wiki/electric/Installation

- Install subversion on Ubuntu, open an Ubuntu system console, then:

```
$ sudo apt-get install subversion
```

1.2. ROS files organization

For better management of the stacks, it is possible to create a folder to use all the stacks which are necessary without having to use in the root folder of stacks of ROS, it is necessary to create a folder called stacks in the home folder, and add this destination path for ROS be aware of this new location:

- Create a folder in your home directory to manage all your custom ROS packages and stacks:

```
$ cd ~
$ mkdir stacks
$ cd stacks
```

- Add the new path to yout bash initilzation file (.bashrc) in your home directory (replace USER with your username):

```
$ echo “export ROS_PACKAGE_PATH=/home/USER/stacks:$ROS_PACKAGE_PATH” >> ~/.baschrc
```

1.3. Download and Install the TraxBot driver

- Open your stack folder, created in section 1.2:

```
$ cd ~/stacks
```

- Install the serial communication stack stack:

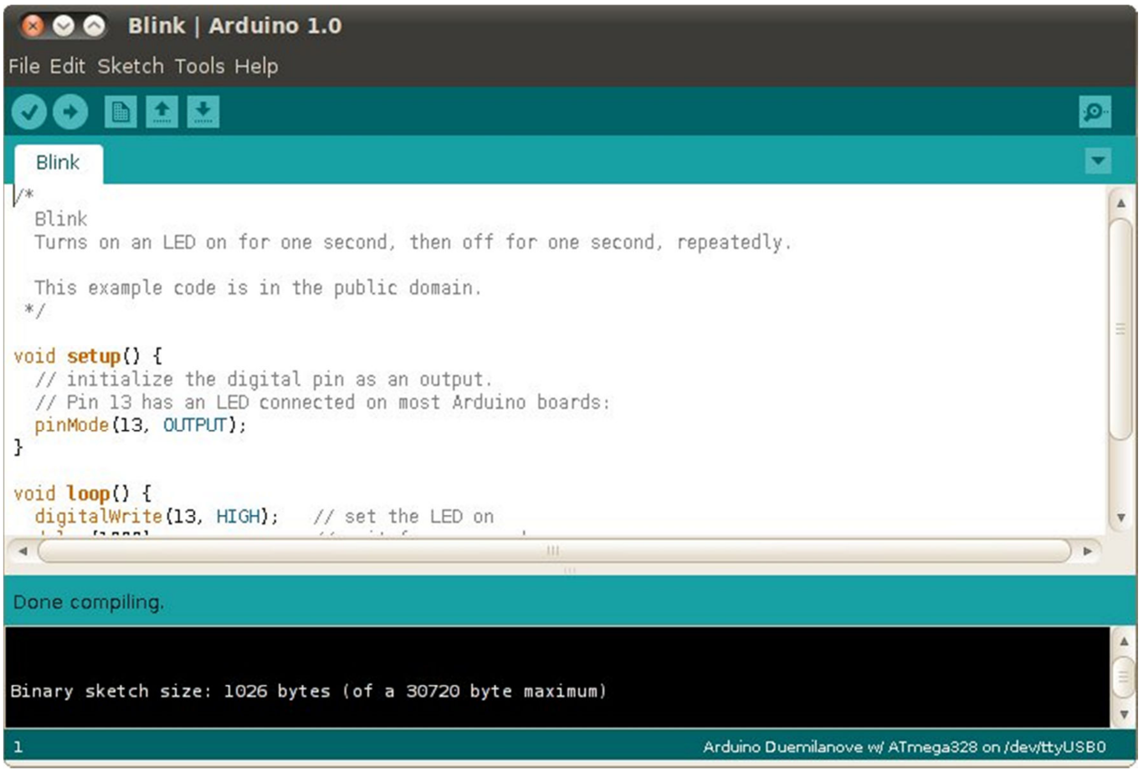
```
$ svn co http://isr-uc-ros-pkg.googlecode.com/svn/stacks/serial_communication/trunk
$ mv trunk serial_communication
$ rosmake serial_communication
```

- Install the TraxBot driver stack:

```
$ svn co http://isr-uc-ros-pkg.googlecode.com/svn/stacks/mrl_traxbot/trunk
$ mv trunk mrl_traxbot
$ rosmake mrl_traxbot
```

1.4. Download and Install the Arduino IDE

It is neccessary to install the Arduino IDE to upload the firmware to the TraxBot's Arduino:



```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
 */

void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);   // set the LED on
```

Done compiling.

```
Binary sketch size: 1026 bytes (of a 30720 byte maximum)
```

1 Arduino Duemilanove w/ ATmega328 on /dev/ttyUSB0

- Install the Arduino IDE in Ubuntu:

```
$ sudo apt-get install arduino
```

Verify if the installed IDE version is v1.0 or higher. Otherwise download it from: http://arduino.cc/en/Main/Software.

1.5. Upload firmware to Arduino UNO

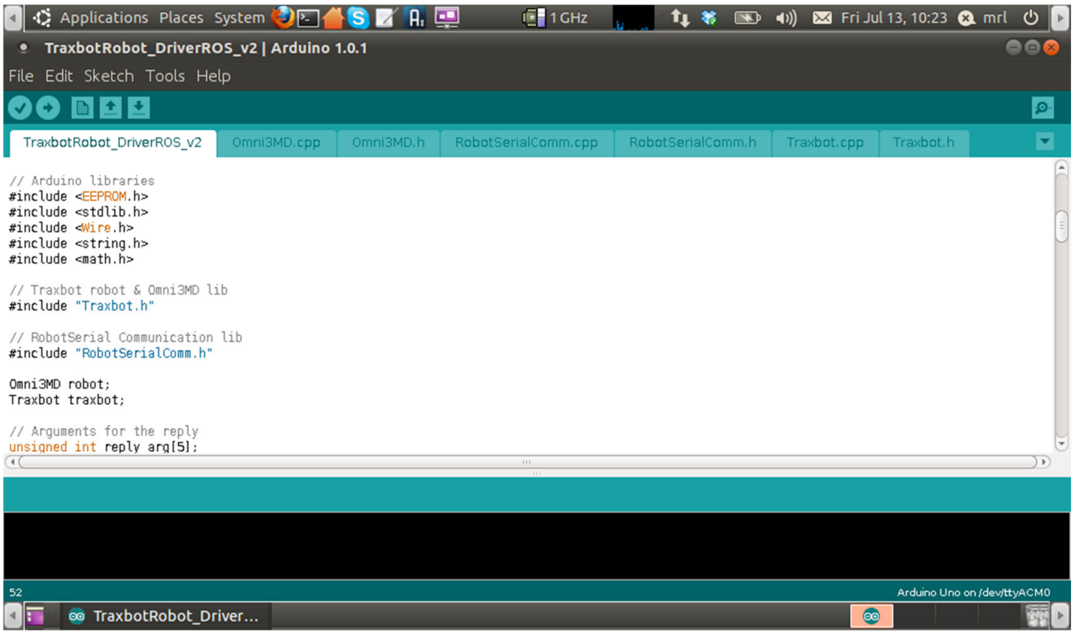
- After the installation, run the Arduino IDE:

```
$ arduino
```

Open the driver firmware in the Arduino IDE, go to ***File*** -> ***Open*** and insert the following path folder:

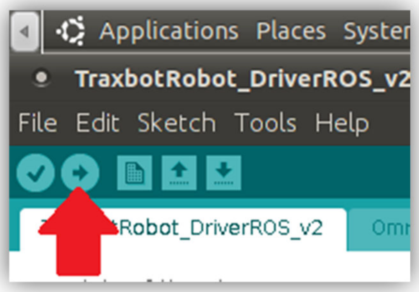
```
/home/USER/stacks/mrl_robots/traxbot_robot/upload_arduino
```

Double-click in "*TrabotRobot_DriverROSv2.ino*". If these steps were successful, you will see the firmware code in the Arduino IDE:


Applications Places System
1 GHz
Fri Jul 13, 10:23 mrl
TraxbotRobot_DriverROS_v2 | Arduino 1.0.1
File Edit Sketch Tools Help
TraxbotRobot_DriverROS_v2
Omni3MD.cpp
Omni3MD.h
RobotSerialComm.cpp
RobotSerialComm.h
Traxbot.cpp
Traxbot.h
52
Arduino Uno on /dev/ttyACM0
TraxbotRobot_Driver...

```
// Arduino libraries
#include <EEPROM.h>
#include <stdlib.h>
#include <Wire.h>
#include <string.h>
#include <math.h>

// Traxbot robot & Omni3MD lib
#include "Traxbot.h"

// RobotSerial Communication lib
#include "RobotSerialComm.h"

Omni3MD robot;
Traxbot traxbot;

// Arguments for the reply
unsigned int reply arg[5];
```

Next, connect the USB cable from the computer, taking into attention Section 1.6.2 and upload the firmware code as shown in the next figure,


Applications Places Syster
TraxbotRobot_DriverROS_v2
File Edit Sketch Tools Help
Robot_DriverROS_v2
Omr

You will know when the code was successfully uploaded, when you see the confirmation message - "*Done Uploading*".

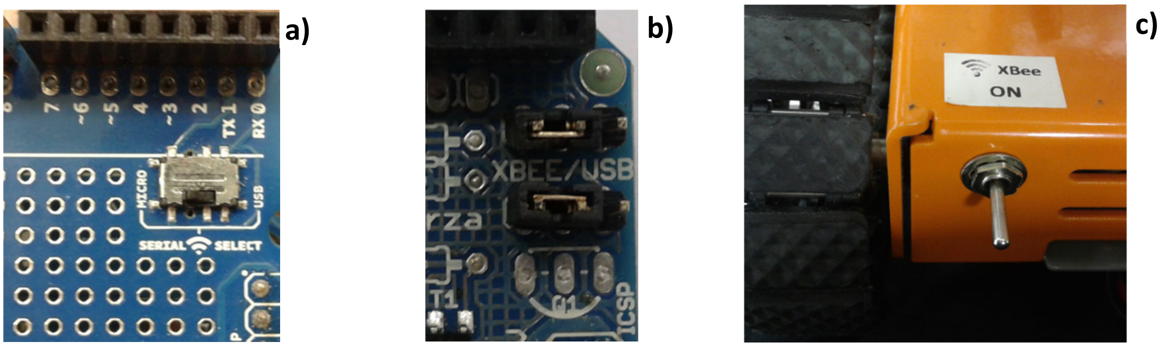
1.6. Important Notes

1.6.1. Hardware startup

Before plugging the USB cable to the Arduino Uno (e.g., code upload or simply sending commands), make sure that the traxBot's power switch is ON. Otherwise the robot will not respond, because the Arduino Uno uses the first initialized power source (battery or USB power).

1.6.2. Uploading code to Arduino

If the XBee shield is mounted on top of the Arduino Uno board, its switch should be in USB mode in order to upload code, otherwise it will not be possible.



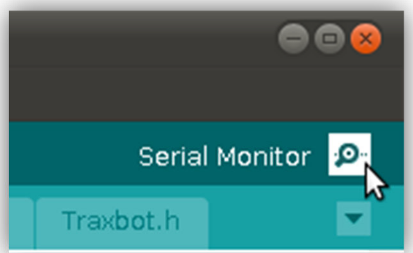
Available switches: **a)** XBee shield Serie 1 **b)** XBee shield Serie 2 **c)** External switch

Chapter 2 -

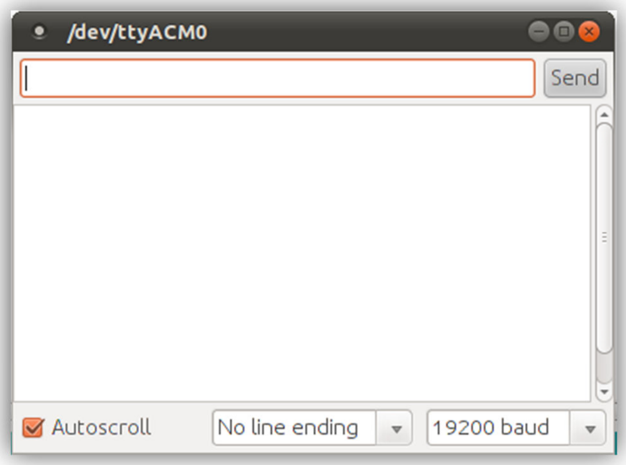
User Manual

2.1. Testing firmware

To test the firmware, switch the TraxBot on, as refered in section 1.6.1, open the Arduino IDE and then open the serial terminal:

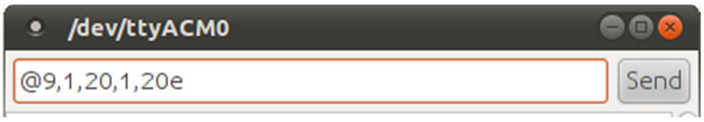


The serial console window will pop up:



Attention: the baud rate should be define to "***19200 baud***".

To perform a linear move with the TraxBot motors, as a testing firmware example, type in the serial console: "***@9,1,20,1,20e***", followed by the Enter button.



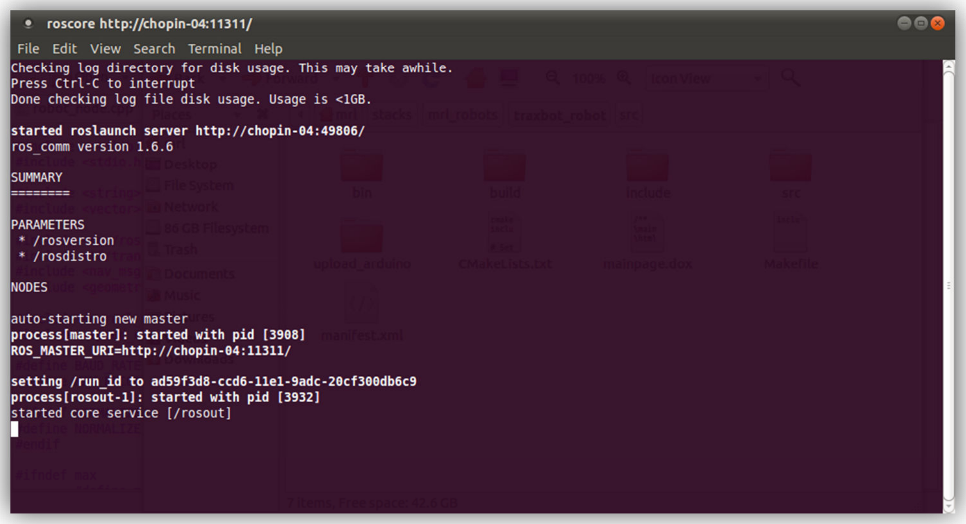
```
@9,1,20,1,20e
```

If all the procedure was done correctçy, the TraxBot will start to move forward in a straight linear motion. To stop the motors, simply send the command "***@11e***"

2.2. Running TraxBot driver in ROS

- First run the roscore Master, in an Ubuntu console:

```
$ roscore
```



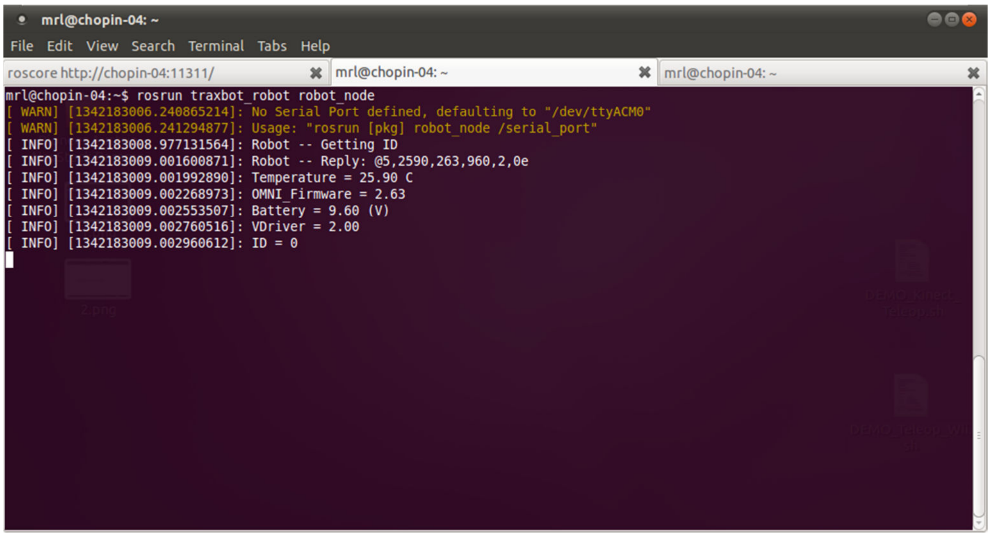
roscore http://chopin-04:11311/
File Edit View Search Terminal Help

```
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://chopin-04:49806/
ros_comm version 1.6.6

SUMMARY
========

PARAMETERS
 * /rosversion
 * /rosdistro

NODES

auto-starting new master
process[master]: started with pid [3908]
ROS_MASTER_URI=http://chopin-04:11311/

setting /run_id to ad59f3d8-ccd6-11e1-9adc-20cf300db6c9
process[rosout-1]: started with pid [3932]
started core service [/rosout]
```

- After the installation, run the Arduino IDE:

```
$ rosrun traxbot_driver traxbot_driver
```

The driver assumes ***/dev/ttyACM0*** as the default serial port. If the robot is connected on a different serial port, e.g. ***/dev/ttyACM1***, you should run:



mrl@chopin-04: ~
File Edit View Search Terminal Tabs Help
roscore http://chopin-04:11311/
mrl@chopin-04: ~
mrl@chopin-04: ~

```
mrl@chopin-04:~$ rosrun traxbot_robot robot_node
[ WARN] [1342183006.240865214]: No Serial Port defined, defaulting to "/dev/ttyACM0"
[ WARN] [1342183006.241294877]: Usage: "rosrun [pkg] robot_node /serial_port"
[ INFO] [1342183008.977131564]: Robot -- Getting ID
[ INFO] [1342183009.001600871]: Robot -- Reply: @5,2590,263,960,2,0e
[ INFO] [1342183009.001992890]: Temperature = 25.90 C
[ INFO] [1342183009.002268973]: OMNI_Firmware = 2.63
[ INFO] [1342183009.002553507]: Battery = 9.60 (V)
[ INFO] [1342183009.002760516]: VDriver = 2.00
[ INFO] [1342183009.002960612]: ID = 0
```

At this moment, the driver is activated and ready to send velocity commands to the robot and receive any kind of information from it, depending on the desired application.

2.3. Available TraxBot driver Topics

- To verify the available topics, simply run in a console,

```
$ rostopic list
```

The possible topics available, for this TraxBot driver are:

	Topic	Message type	Description
Publishers	/odom	nav_msgs::Odometry	Robot odometry (x,y,θ).
	/sonar_front	sensor_msgs::Range	Front sonar range in (cm).
	/sonar_right	sensor_msgs::Range	Right sonar range in (cm).
	/sonar_left	sensor_msgs::Range	Left sonar range in (cm).
	/battery_power	std_msgs::Float32	Battery tension in (V).
	/driverFirmware	std_msgs::Float32	OMNI-3MD firmware version.
	/driverTemperature	std_msgs::Float32	OMNI-3MD temperature.
	/robotID	std_msgs::Int16	Robot ID (integer).
	/encoder1	std_msgs::Int32	Encoder motor 1 in (pulses).
	/encoder2	std_msgs::Int32	Encoder motor 2 in (pulses).
Subscribers	/cmd_vel	geometry_msgs::Twist	Velocity commands (linear,angular).
	/stopMotors	std_msgs::Empty	Activate callback function to stop motors.
	/encodersReset	std_msgs::Empty	Activate callback function to reset encoders.

Appendix A - Protocol Table

A.1. Protocol data insertion

The communication frame between ROS and the Arduino follows the following protocol:

INITIAL Char	COMMAND	SEP	1st PARAMETER	SEP	2nd PARAMETER	SEP	...	n PARAMETER	FINAL Char
@	1 - 17	,	Int num	,	Int num	,	...	Int num	e

A.2. Protocol data insertion example

This protocol accepts 3 types of data: Initial configuration, Cinematic commands and sensors readings.

Example of a linear move command to the motors:

@	Command	,	Motor1 direction	,	Speed motor1	,	Motor2 direction	,	Speed motor1	e
@	9	,	1	,	40	,	1	,	40	e

A.2. Communication protocol technical info

- Serial Baud rate 19200

- All inserted values must be integers

Frame		Description
Send	Reply	
@1e	-	OMNI-3MD motor driver auto calibration.
@2,Kp,Ki,Kde	-	Define PID motor controller gains Kp,Ki and Kd (0 - 65535).
@3,enc,valuee	-	Set encoders prescaler, **enc:** enconder (1 - 2) **value:** value (0 - 4)
@4,enc,valuee	-	Set encoder value, **enc:** enconder (1 - 2) **value:** value (0 - 65535)
@5e	@5,temp,firm,bat,r_firm,r_ide	Provides robot information, **temp:** OMNI-3MD temperature **firm:** OMNI-3MD firmware **bat:** Battery power **r_firm:** Robot firmware **r_id:** Robot ID
@6e	@6,enc1,enc2e	Provides encoder readings, **enc1:** encoder 1 (Left) **enc2:** encoder 2 (Right)
@7e	@7,son1,son2,son3e	Provides sonars readings, **son1:** encoder 1(Front) **son2:** encoder 2 (Left) **son3:** encoder 2 (Right)
@8e	@8,enc1,enc2,son1,son2,son3e	Provides sonars and encoders readings, **enc1:** encoder 1 (Left) **enc2:** encoder 2 (Right) **son1:** encoder 1(Front) **son2:** encoder 2 (Left) **son3:** encoder 2 (Right)
@9,dir1,speed1,dir2,speed2e	-	Send linear move commands with PID controller, **dir1:** direction motor 1 (1- 2) **speed1:** speed motor 1 (0 - 100) **dir2:** direction motor 2 (1 - 2) **speed2:** speed motor 2 (0 - 100)
@10,dir1,speed1,dir2,speed2e	-	Send linear move, **dir1:** direction motor 1 (1- 2) **speed1:** speed motor 1 (0 - 100) **dir2:** direction motor 2 (1 - 2) **speed2:** speed motor 2 (0 - 100)
@11e	-	Stop motors.
@12e	-	Encoders reset.
@13e	(to the console) @13,”0/1”e	Check Debug mode (0-1).
@14,”0/1”e	-	Set debug mode (0-1).
@15e	@15,”0/1”e	Info stream mode (0-1).
@16e	@6,enc1,enc2e	Streaming encoder readings, **enc1:** encoder 1 (Left) **enc2:** encoder 2 (Right)
@17e	-	Stop stream.

Appendix B - ROS Cheat Sheet

ROS Cheat Sheet

Filesystem Command-line Tools

rospack/rosstack	A tool inspecting packages/stacks.
roscd	Changes directories to a package or stack.
rosls	Lists package or stack information.
roscreate-pkg	Creates a new ROS package.
roscreate-stack	Creates a new ROS stack.
rosdep	Installs ROS package system dependencies.
rosmake	Builds a ROS package.
roswtf	Displays a errors and warnings about a running ROS system or launch file.
rxdeps	Displays package structure and dependencies.

Usage:

```
$ rospack find [package]
$ roscd [package[/subdir]]
$ rosls [package[/subdir]]
$ roscreate-pkg [package_name]
$ rosmake [package]
$ rosdep install [package]
$ roswtf or roswtf [file]
$ rxdeps [options]
```

Common Command-line Tools

roscore

A collection of nodes and programs that are pre-requisites of a ROS-based system. You must have a roscore running in order for ROS nodes to communicate.

roscore is currently defined as:

```
master
parameter server
rosout
```

Usage:

```
$ roscore
```

rosmsg/rossrv

rosmsg/rossrv displays Message/Service (msg/srv) data structure definitions.

Commands:

rosmsg show	Display the fields in the msg.
rosmsg users	Search for code using the msg.
rosmsg md5	Display the msg md5 sum.
rosmsg package	List all the messages in a package.
rosnode packages	List all the packages with messages.

Examples:

Display the Pose msg:

```
$ rosmsg show Pose
```

List the messages in nav_msgs:

```
$ rosmsg package nav_msgs
```

List the files using sensor_msgs/CameraInfo:

```
$ rosmsg users sensor_msgs/CameraInfo
```

rosrun

rosrun allows you to run an executable in an arbitrary package without having to cd (or roscd) there first.

Usage:

```
$ rosrun package executable
```

Example:

Run turtlesim:

```
$ rosrun turtlesim turtlesim_node
```

rosnode

Displays debugging information about ROS nodes, including publications, subscriptions and connections.

Commands:

rosnode ping	Test connectivity to node.
rosnode list	List active nodes.
rosnode info	Print information about a node.
rosnode machine	List nodes running on a particular machine.
rosnode kill	Kills a running node.

Examples:

Kill all nodes:

```
$ rosnode kill -a
```

List nodes on a machine:

```
$ rosnode machine aqy.local
```

Ping all nodes:

```
$ rosnode ping --all
```

roslaunch

Starts ROS nodes locally and remotely via SSH, as well as setting parameters on the parameter server.

Examples:

Launch on a different port:

```
$ roslaunch -p 1234 package filename.launch
```

Launch a file in a package:

```
$ roslaunch package filename.launch
```

Launch on the local nodes:

```
$ roslaunch --local package filename.launch
```

rostopic

A tool for displaying debug information about ROS topics, including publishers, subscribers, publishing rate, and messages.

Commands:

rostopic bw	Display bandwidth used by topic.
rostopic echo	Print messages to screen.
rostopic hz	Display publishing rate of topic.
rostopic list	Print information about active topics.
rostopic pub	Publish data to topic.
rostopic type	Print topic type.
rostopic find	Find topics by type.

Examples:

Publish hello at 10 Hz:

```
$ rostopic pub -r 10 /topic_name std_msgs/String hello
```

Clear the screen after each message is published:

```
$ rostopic echo -c /topic_name
```

Display messages that match a given Python expression:

```
$ rostopic echo --filter "m.data=='foo'" /topic_name
```

Pipe the output of rostopic to rosmsg to view the msg type:

```
$ rostopic type /topic_name | rosmsg show
```

rosparam

A tool for getting and setting ROS parameters on the parameter server using YAML-encoded files.

Commands:

rosparam set	Set a parameter.
rosparam get	Get a parameter.
rosparam load	Load parameters from a file.
rosparam dump	Dump parameters to a file.
rosparam delete	Delete a parameter.
rosparam list	List parameter names.

Examples:

List all the parameters in a namespace:

```
$ rosparam list /namespace
```

Setting a list with one as a string, integer, and float:

```
$ rosparam set /foo "['1', 1, 1.0]"
```

Dump only the parameters in a specific namespace to file:

```
$ rosparam dump dump.yaml /namespace
```

rosservice

A tool for listing and querying ROS services.

Commands:

rosservice list	Print information about active services.
rosservice node	Print the name of the node providing a service.
rosservice call	Call the service with the given args.
rosservice args	List the arguments of a service.
rosservice type	Print the service type.
rosservice uri	Print the service ROSRPC uri.
rosservice find	Find services by service type.

Examples:

Call a service from the command-line:

```
$ rosservice call /add_two_ints 1 2
```

Pipe the output of rosservice to rossrv to view the srv type:

```
$ rosservice type add_two_ints | rossrv show
```

Display all services of a particular type:

```
$ rosservice find rospy_tutorials/AddTwoInts
```

Logging Command-line Tools

rosbag

This is a set of tools for recording from and playing back to ROS topics. It is intended to be high performance and avoids deserialization and reserializationof the messages.

rosbag record will generate a ".bag" file (so named for historical reasons) with the contents of all topics that you pass to it.

Examples:
  Record all topics:

```
$ rosbag record -a
```

  Record select topics:

```
$ rosbag record topic1 topic2
```

rosbag play will take the contents of one or more bag file, and play them back in a time-synchronized fashion.

Examples:
  Replay all messages without waiting:

```
$ rosbag play -a demo_log.bag
```

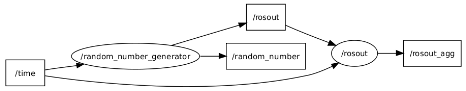
  Replay several bag files at once:

```
$ rosbag play demo1.bag demo2.bag
```

Graphical Tools

rxgraph

Displays a graph of the ROS nodes that are currently running, as well as the ROS topics that connect them.

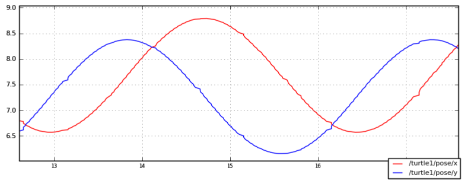


Usage:

```
$ rxgraph
```

rxplot

A tool for plotting data from one or more ROS topic fields using matplotlib.



Examples:
  To graph the data in different plots:

```
$ rxplot /topic1/field1 /topic2/field2
```

  To graph the data all on the same plot:

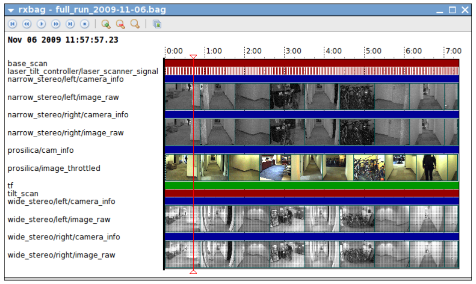
```
$ rxplot /topic1/field1,/topic2/field2
```

  To graph multiple fields of a message:

```
$ rxplot /topic1/field1:field2:field3
```

rxbag

A tool for visualizing, inspecting, and replaying histories (bag files) of ROS messages.

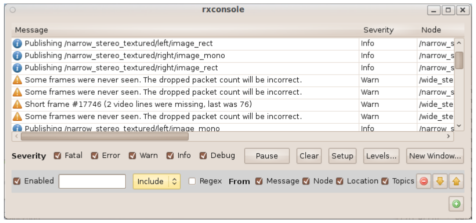

rxbag - full_run_2009-11-06.bag
Nov 06 2009 11:57:57.23
0:00
1:00
2:00
3:00
4:00
5:00
6:00
7:00
base_scan
laser_tilt_controller/laser_scanner_signal
narrow_stereo/left/camera_info
narrow_stereo/left/image_raw
narrow_stereo/right/camera_info
narrow_stereo/right/image_raw
prosilica/cam_info
prosilica/image_throttled
tf
tilt_scan
wide_stereo/left/camera_info
wide_stereo/left/image_raw
wide_stereo/right/camera_info
wide_stereo/right/image_raw

Usage:

```
$ rxbag bag_file.bag
```

rxconsole

A tool for displaying and filtering messages published on rosout.



Message	Severity	Node
Publishing /narrow_stereo_textured/left/image_rect	Info	/narrow_s
Publishing /narrow_stereo_textured/right/image_mono	Info	/narrow_s
Publishing /narrow_stereo_textured/right/image_rect	Info	/narrow_s
Some frames were never seen. The dropped packet count will be incorrect.	Warn	/wide_ste
Some frames were never seen. The dropped packet count will be incorrect.	Warn	/narrow_s
Short frame #17746 (2 video lines were missing, last was 76)	Warn	/wide_ste
Some frames were never seen. The dropped packet count will be incorrect.	Warn	/wide_ste
Publishing /narrow_stereo_textured/left/image_mono	Info	/narrow_s

Severity ☑ Fatal ☑ Error ☑ Warn ☑ Info ☑ Debug
Pause
Clear
Setup
Levels...
New Window...
☑ Enabled
Include
☐ Regex
From ☑ Message ☑ Node ☑ Location ☑ Topics

Usage:

```
$ rxconsole
```

tf Command-line Tools

tf_echo

A tool that prints the information about a particular transformation between a source_frame and a target_frame.

Usage:

```
$ rosrun tf tf_echo <source_frame> <target_frame>
```

Examples:
  To echo the transform between /map and /odom:

```
$ rosrun tf tf_echo /map /odom
```

view_frames

A tool for visualizing the full tree of coordinate transforms.

Usage:

```
$ rosrun tf view_frames
$ evince frames.pdf
```