БЪЛГАРСКА АКАДЕМИЯ НА НАУКИТЕ • BULGARIAN ACADEMY OF SCIENCES

ПРОБЛЕМИ НА ТЕХНИЧЕСКАТА КИБЕРНЕТИКА И РОБОТИКАТА, **62**
PROBLEMS OF ENGINEERING CYBERNETICS AND ROBOTICS, **62**

София • 2010 • Sofia

Robot Arm Control under ROS/Ubuntu

Nayden Chivarov

Institute of Systems Engineering and Robotics, 1113 Sofia
E-mail: nshivarov@code.bg

1. Introduction

ROS is an open-source meta-operating system. It is a platform for applications in the field of robotics. The term meta-operating system means that there is an operating system installed on the hardware already, and ROS platform is installed over this operating system. For the moment Linux distribution Ubuntu is the operating system that supports fully ROS. This distribution has great future related to porting and integration into embedded world. That gives ROS an excellent perspective for development and use of robot-related applications in systems, varying greatly in scale and complexity.

This article describes such an application of this meta-operating system/platform – control of a robotic arm from a terminal running application under ROS/Ubuntu.

2. Short survey of ROS features

ROS is an applications platform very convenient for development and use of applications in the field of robotics. There are large numbers of drivers for various types of sensors and control of different mechanisms, as well as packages for robot vision, navigation and others for ROS.

Some of the basic ROS concepts are nodes, topics and services. They help the creation of the skeleton of every ROS application.

Nodes are processes implementing a given function. They are the basic unit of execution in ROS. All functions and the whole logic of a ROS application are implemented in nodes.

Topics are tools of IPC (Inter-Process Communication), a way to exchange messages of a defined format. They allow detaching between the message sender and receiver because both sides “know” the name of the topic only, and not of each other. This allows implementing extremely scalable and flexible software and lets the developer focus on the very logic and creativeness of the application at hand.

The services give very convenient tools of two-way message communications and easy implementation of client-server relations between nodes.

ROS implements a very convenient format of defining services and messages passed between the nodes and this format is language-independent. Thus there can be multiple nodes written in different programming languages (i.e., C/C^{++}, Python) and communicating with each other implementing a complete and integrate system.

The communications between different nodes can be implemented over a network TCP/IP link that makes possible the cooperation between nodes running on different computers over Internet and logic/load distribution between multiple machines.

3. Common architecture

The control is implemented by means of a computer terminal and can be manual or automatic (Fig. 1). In manual control mode the operator controls every movement of the robot by pressing the corresponding button on the keyboard. Each of robot’s joints has two corresponding keys that drive it in a clockwise or counterclockwise direction. In automatic control mode the user chooses a script file containing a sequence of commands that are sent to the robot arm and implement a whole program of movements.

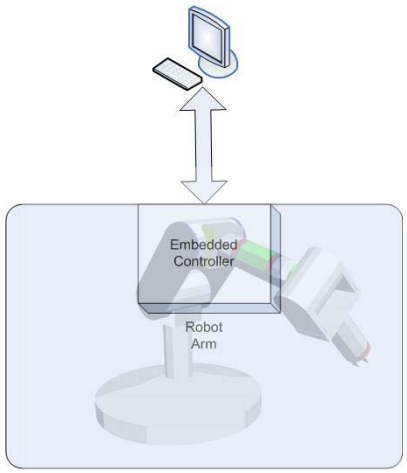

Fig. 1. Common arhitecture black diagram

That sequence, once started is carried on without user interference. The corresponding commands are received and processed by a node working under ROS, and transmitting it to the robot arm through a serial interface. Inside the robot arm there is an embedded controller that receives the commands and drives the corresponding robot mechanisms implementing the given commands.

4. Software architecture

The following graphic is generated using the rxgraph tool of the ROS system. It displays the node architecture of the application (Fig. 2).

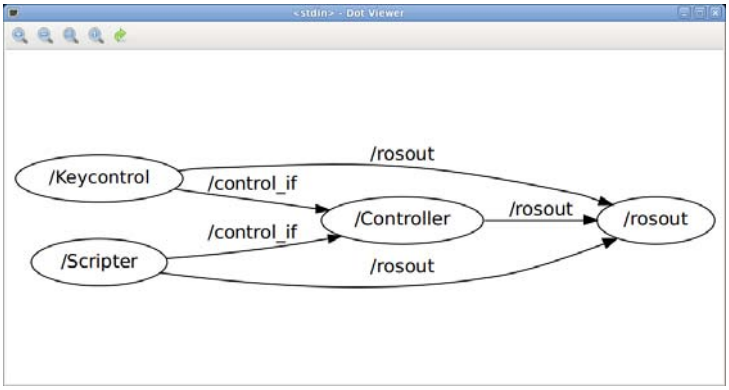

Fig. 2. Node architecture of the application

A basic software is called *roscore* and it is a part of every standard ROS application. It can be started from a shell console using command *roscore* (Fig. 3).

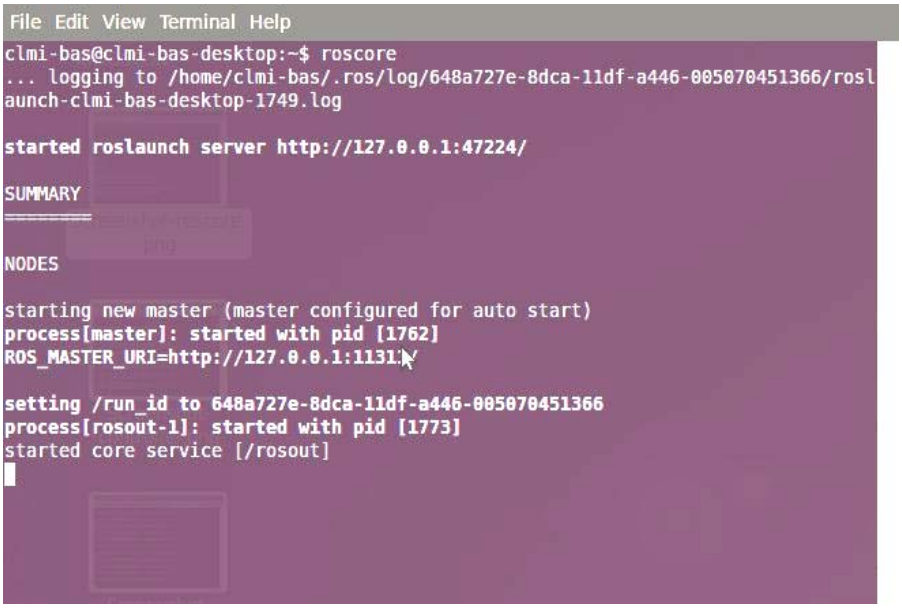

Fig. 3. Rosout node

The node rosout (Fig. 3) is a standard node for almost every application under ROS and it implements visualization of various system messages – information and error logs and other changes of the state of application. It takes messages from all of the rest of nodes that belong to the application and displays them in the console.

The application itself consists of the nodes:

– *Controller,*

– *Keycontrol,*

– *Scripter.*

The *Controller* node (Fig. 4) takes commands from the other two modules and implements their validation, formatting and transmitting through the serial link to the embedded controller of the robot arm. It is accomplished by registering for receiving messages on the topic *control_if.*

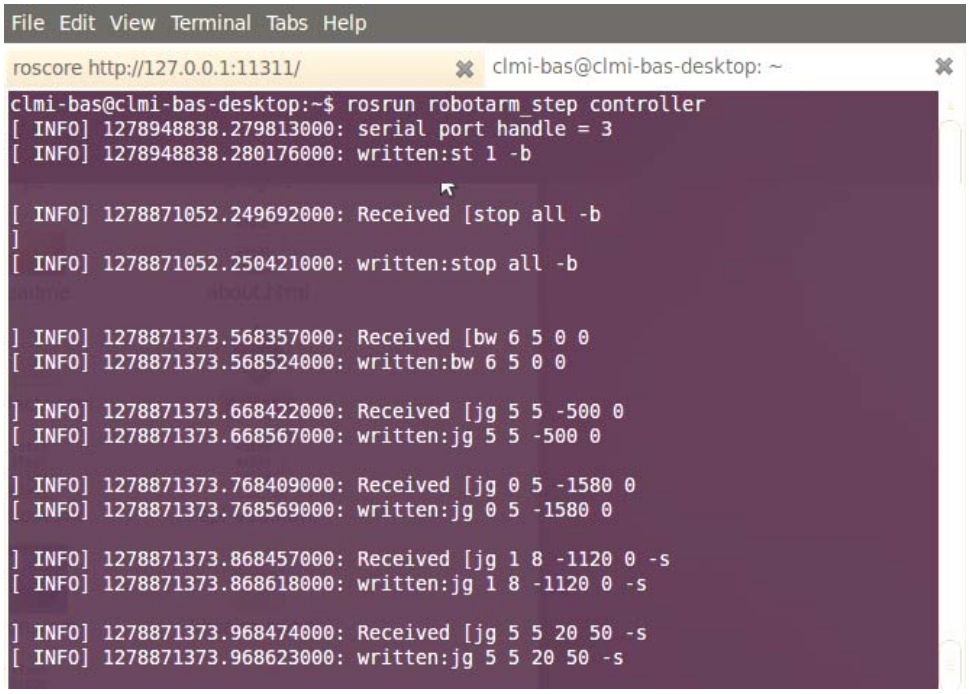

Fig. 4. Controller node

The other two nodes transmit commands for the manual and automatic mode of operation correspondingly.

The *Keycontrol* node (Fig. 5) reads commands from user input (terminal console), generates appropriate commands, formats them into messages and publishes them on *control_if* topic.

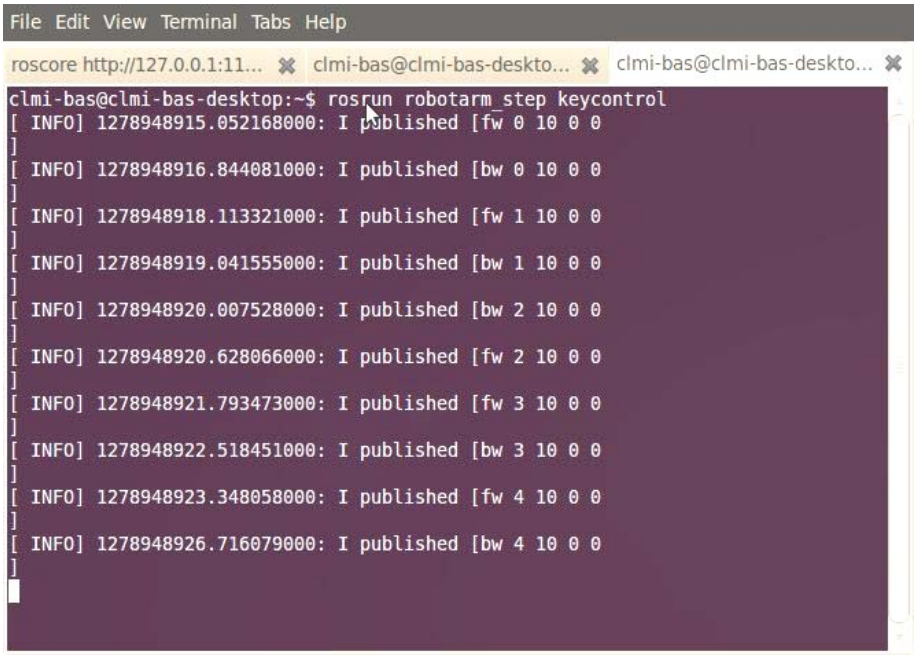

Fig 5. Keycontrol node

The *Scripter* node (Fig. 6) reads commands from a given script file (a text file, containing commands to the robot arm's embedded controller), filters and formats them and publishes them on the topic *control_if*.

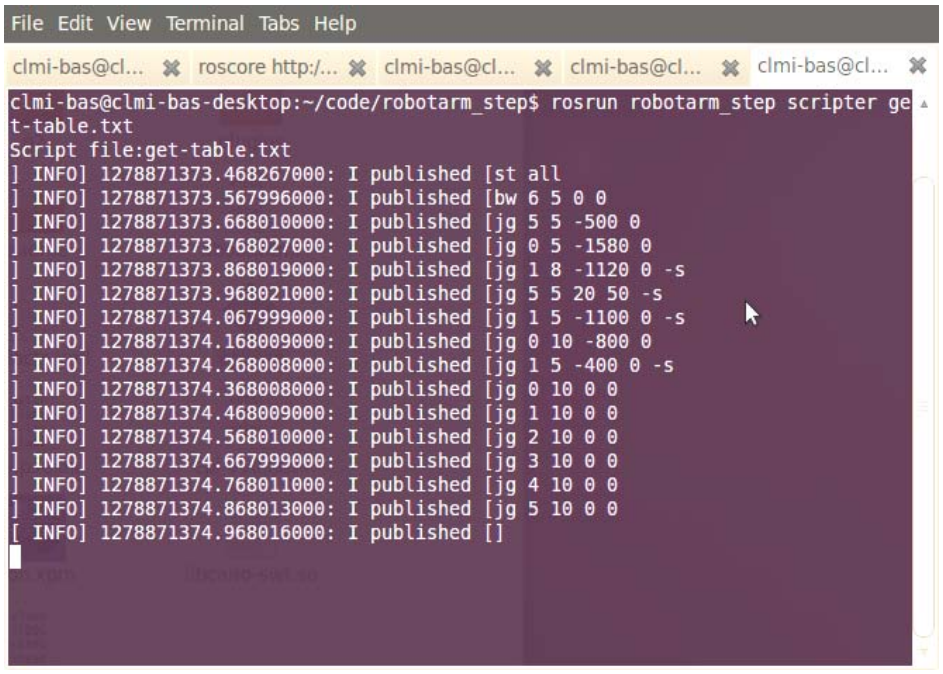

Fig. 6. Scripter node

In this application the flexibility of the ROS platform is clearly visible. Such a system becomes very scalable and easily extendable. New nodes can easily be added, for example a node for receiving commands from a remote terminal and many others. All those nodes can publish on the same topic *control_if* without changing the Controller node, as there is no need for any node to be aware of the number and functions of the others that publish on the same topic.

5. Sample source code

The source code implementing one of the nodes – the *Controller* – is shown below:

```
/*
 * listener.cpp
 *
 *  Created on: Jun 27, 2010
 *      Author: ivaylo
 */

#include <stdio.h>   /* Standard input/output definitions */
#include <string.h>  /* String function definitions */
#include <termios.h> /* POSIX terminal control definitions */
#include <unistd.h>  /* UNIX standard function definitions */
#include <fcntl.h>   /* File control definitions */

#include <ros/ros.h>
#include <std_msgs/String.h>

#include "common.h"
#include "ser.h"

int serial;

void listenCallback(const std_msgs::StringConstPtr& msg)
{
	char line[254];

	ROS_INFO("Received [%s]", msg->data.c_str());
	if (!write(serial, msg->data.c_str(), msg->data.length()))
	{
		ROS_ERROR("write failed\n");
//		close(serial);
		return;
	}

	ROS_INFO("written:%s\n", msg->data.c_str());
```

```
	// Read a line until it contains "ok" or "err"
	do
	{
		if (!readln(serial, line))
			continue;
	}
	while (!strstr(line, "ok") && !strstr(line, "err"));
}

int main(int argc, char** argv)
{
	serial = initport();
	ROS_INFO("serial port handle = %d", serial);

	ros::init(argc, argv, CTRL_NODE_NAME);
	ros::NodeHandle n;
	ros::Subscriber chatter_sub = n.subscribe(CTRL_TOPIC, 1000,
listenCallback);
	ros::spin();
```

6. Conclusion

The meta-operating system ROS makes it easy to implement applications in the fields of robotics and embedded control. Its open source architecture (flexible and scalable itself) allows the convenient development and integration of a large variety of application in this field. Using the tools and means that it supplies, the developer can focus on efforts of the creativeness and increase his/her productivity. Not only the development and integration are made easy but also future support and extension of the code too. ROS is firstly implemented under Ubuntu Linux, which makes it easy to port and integrate into embedded systems. The tools it introduces for distribution of an application over multiple computers allow the cooperation of nodes on an embedded system with those working on a local or remote computer terminal, and in future eventually the control through handheld computer devices and smartphones. All these features make ROS a candidate for the first choice and excellent perspective in the field of robotics and embedded control.

References

1. **http://www.ros.org/wiki/**
2. **http://www.care-o-bot.org/**
3. **http://www.ubunto.com**
4. **http://www.linux.org**

Управление роботом манипулятором под ROS/Ubuntu

Найден Шиваров

Институт системного инженерства и робототехники, 1113 Sofia
E-mail: nshivarov@code.bg

(Р е з ю м е)

РОС представляет собой мета-операционная система (платформа) с открытым кодом для развития и применений в области робототехники.

Мета-операционная система означает, что в определенной управляющей системе, построенной на реальных электронных компонентах, уже существует встроенная операционная система, на которой инсталирована платформа РОС.

В настоящий момент распределение (distribution) Ubuntu является операционной системой, полностью поддерживающей РОС. Такое Линукс распределение имеет большое будущее, связаное с переносом и интеграцией в малых встроенных (embedded) системах. Это дает системе РОС отличные перспективы для развития и использования роботов, связанных с применениями в различных по масштабу и комплекности робототехнических системах.

В настоящей статье описано подобное применение этой мета-операционной системы (платформы) для управления роботом-манипулятором с помощью компьютерного терминала, на котором инсталирована РОС/Ubuntu.