REMOTELY CONTROLLED ARTICULATED ROBOT “ROBCO” UNDER ROS/UBUNTU

Nayden Chivarov, Ivaylo Genchev, Nedko Shivarov, Roman Zahariev, Daniel Radev and Vladimir Vladimirov

Institute of Systems Engineering and Robotics,
Block II Acad. Bonchev str., Sofia 1113 Bulgaria
e-mails: nshivarov@code.bg, ivailo_genchev@abv.bg, nedko@code.bg, roman@clmi.bas.bg, djradev@gmail.com and vl_vladimirov@yahoo.com

Abstract: This article describes an application using open-source meta-operating system/platform ROS for remotely controlled articulated robotic arm. As a base for our development and research for Remotely Controlled Articulated Robot under ROS/Ubuntu we are using multipurpose articulated educational minirobot “ROBCO”. The robot can be controlled either locally via an immediate console or remotely using a remote terminal. The immediate control is implemented by means of a computer terminal. The remote connection can be made over any TCP/IP enabled network such as the Internet. Software architecture of the remotely controlled articulated robotic arm consists of two major parts – ROS part and remote interface. The Remote control consists of two parts – Server and Client. The Client accepts user input and transmits the corresponding commands remotely to the Server via TCP/IP connection over a network (i.e. the Internet). The Server listens for Client connections and accepts commands from the connected Client.

Keywords: ROS, Linux, Ubuntu, Articulated Robotic Arm, TCP/IP, ATMega8L, RS232

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 the platform ROS is installed over that operating system. For the moment Linux [1] distribution Ubuntu [2] is the operating system that supports fully ROS. That distribution has a 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 an application using this meta-operating system/platform for remotely controlled articulated robotic arm “ROBCO”.

2. Articulated Robot “ROBCO”

As a base for our development and research for Remotely Controlled Articulated Robot under ROS/Ubuntu we are using multipurpose articulated educational minirobot “ROBCO” (see Figure 1).

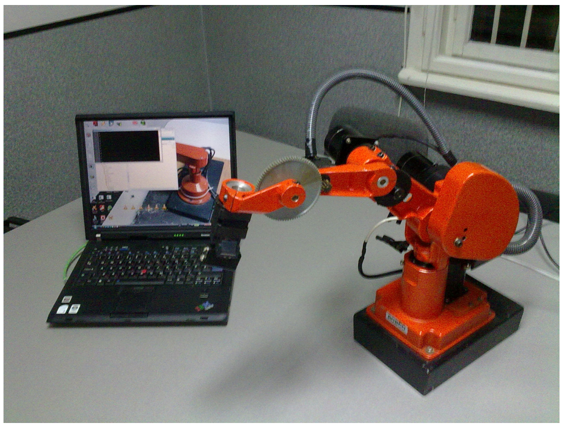


Fig.1 Multipurpose Articulated Educational Minirobot Robot – “ROBCO”

The Articulated Robot ROBCO is working in Anthropomorphic Coordinate System and has 5 Degrees of freedom. The mechanical and electrical design is a complex task, involving a variety of differing mechanisms, and a host of conflicting constraints [3]. Robot design must be guided by a thorough understanding of the underlying principles of robotics, combined with knowledge of the current technology. When multipurpose articulated educational minirobot "ROBCO" was developed, in the mechanical design was taken into consideration the following assumptions:

- Keep the weight toward the base to reduce the inertia and the load on the actuators;
- Keep the geometry simple and eliminate interaction between the joints in order to simplify the control algorithms;
- Try to balance the forces in the joints to reduce the load on the actuators;
- Keep the structure stiff and light in order to keep the resonant frequency high and to reduce the load on the bearings and actuators;
- Pay attention to gears backlash, bearings play, and manufacturing tolerances in order to increase accuracy and repeatability;
- Provide protection for transducers, cables and hoses.

The Articulated Robot ROBCO uses stepper motors. These are motors driven by a train of electrical pulses. The stator is wound as two separate coils, which produce magnetic fields offset angularly by half a rotor pole. These coils are pulsed alternately to produce a rotation magnetic field. Each pulse turns the rotor through a fixed angle (one step) and consequently, angular position change is proportional to the number of pulses. Accurate open-loop control of the rotational velocity is achieved by controlling the pulse rate. Stepper motors can slip during rapid acceleration of high inertia loads, thus, for accurate open-loop control, the pulse frequency has to be varied during times of acceleration.

3. Control System Overview

The common block diagram of the Robot is shown on Figure 2. All electronic control modules are interconnected by a system bus. That gives an extreme flexibility and scalability to the whole architecture.

All commands to the Robot, as well as data from step counters to the operator are passed and processed by the control module *Robot Controller.* The immediate control of the stepper motors is implemented by separate intelligent electronic modules M_1 - M_N. The distribution of different queries and commands to different modules, self-test and detection of system's configuration, as well as the whole robot intelligence at high level is performed by the embedded software of the controller and communicated to the modules by the System Bus [4].

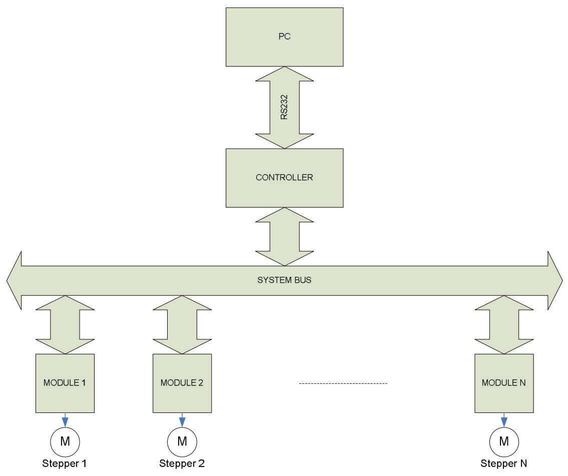


Fig.2 Control System of Educational Robots ROBCO

INTELLIGENT MODULES

The intelligent modules implement direct control of the stepper motors. They are built using a simple 8-bit microcontroller ATMega8L and driver ICs that provide the necessary levels of electrical control signals for the proper stepper motor operation. They also track the motor's relative position by counting up and down the steps that the motor performs.

ROBOT CONTROLLER

The Controller performs the following tasks:

- Communication with the host computer on which ROS/Ubuntu runs using a serial connection (RS232);
- Discovery and diagnostics of the system configuration – present intelligent modules;

- Medium-level control of stepper motors and collecting information about their current position using the System Bus and a specially designed protocol.

The Robot Controller consists of three main blocks – the Control Unit CU, the Communications Controller CC and Bus Interface module BU[5], as shown on figure 3.

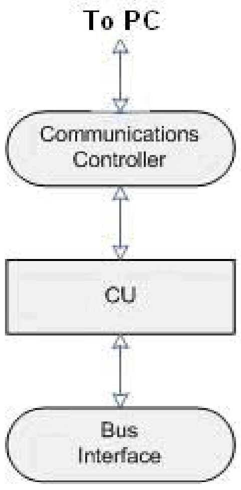


Fig. 3 Robot Controller

The Communications Controller implements receiving commands from the host computer running ROS/Ubuntu, and sending back data for the robot status. It is implemented using MAX232 integrated circuit.

The Interface Module is responsible for connecting the Robot Controller to the System Bus. It is in fact built into the main CU microcontroller.

The Control Unit is built around an ATMega32 microcontroller from Atmel’s AVR microcontroller family [6] and contains the Robot’s firmware implementing the internal intelligence of the Robot.

4. Introduction to ROS Features

ROS is an applications platform very convenient for development and use of applications in the field of robotics. There is large number 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 create the skeleton of every ROS application [7].

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 a means of IPC (Inter-Process Communication), a way to exchange *messages* of a defined format. They allow detaching between sender and receiver of a message 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.

Services give a very comfortable means 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 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 cooperation between nodes running on different computers over the Internet and logic/load distribution between multiple machines.

5. Common Architecture

The robot can be controlled either locally via an immediate console or remotely using a remote terminal (Fig.4).

The immediate control is implemented by means of a computer terminal. The remote connection can be made over any TCP/IP enabled network such as the Internet. 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 clockwise or counterclockwise direction. The corresponding commands are received and processed by a node working under ROS, and sending 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.

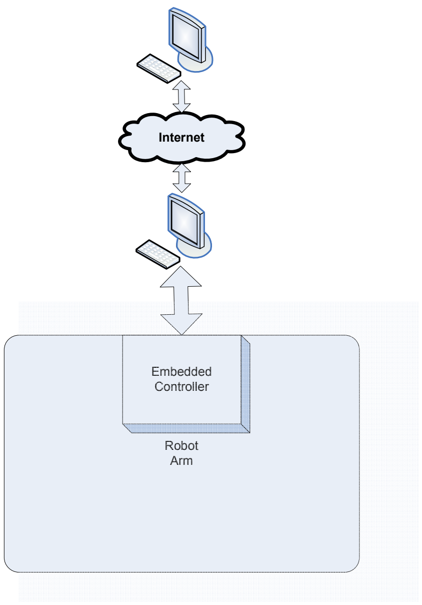


Fig.4 Common Arhitecture Black Diagram

6. Software Architecture

The application consists of two major parts – ROS part and remote interface.

The following graphic is generated using the *rxgraph* tool of the ROS system. It displays the node architecture of the ROS part of the application (Figure 5).

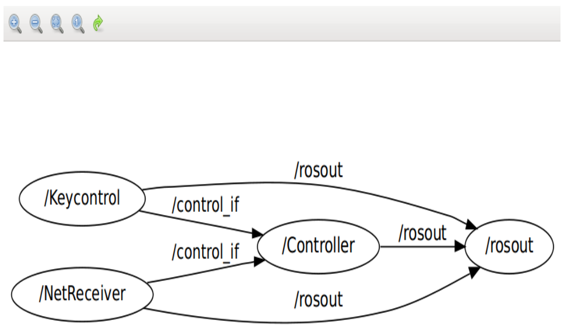
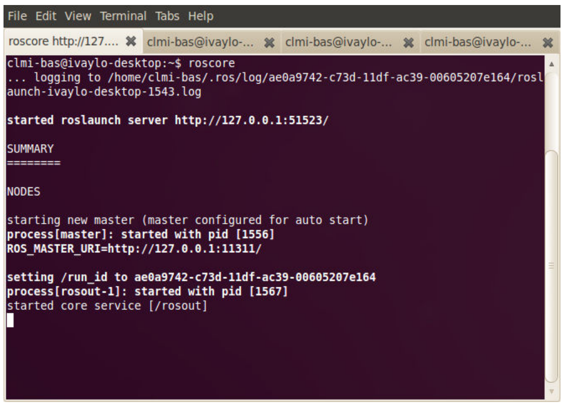


Fig.5 Node Architecture of the Application

A basic software module that is not shown here 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. 6).



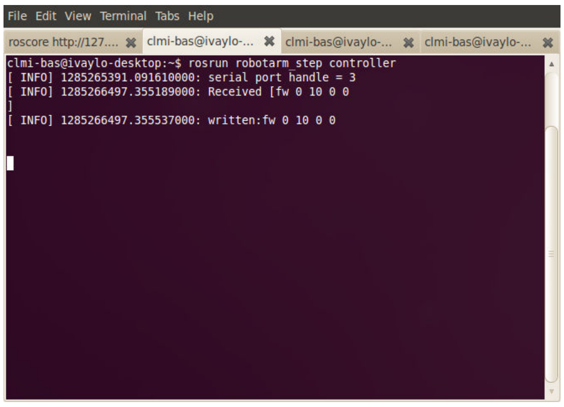
```
clmi-bas@ivaylo-desktop:~$ roscore
... logging to /home/clmi-bas/.ros/log/ae0a9742-c73d-11df-ac39-00605207e164/rosl
aunch-ivaylo-desktop-1543.log

started roslaunch server http://127.0.0.1:51523/

SUMMARY
========

NODES

starting new master (master configured for auto start)
process[master]: started with pid [1556]
ROS_MASTER_URI=http://127.0.0.1:11311/

setting /run_id to ae0a9742-c73d-11df-ac39-00605207e164
process[rosout-1]: started with pid [1567]
started core service [/rosout]
```

Fig.6 Roscore

The node rosout 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 ROS part of the application consists of the nodes *Controller, Keycontrol* and *NetReceiver.*

The *Controller* node (Fig. 7) 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.*

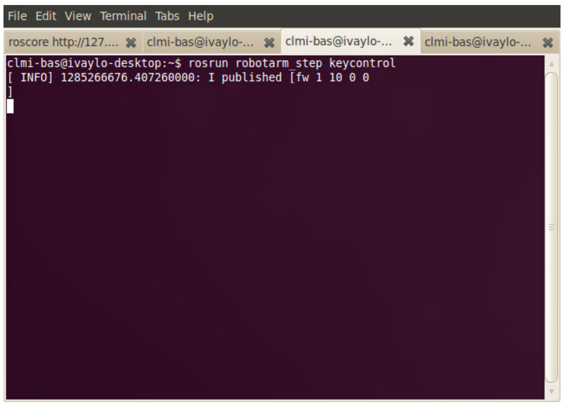


```
clmi-bas@ivaylo-desktop:~$ rosrun robotarm_step controller
[ INFO] 1285265391.091610000: serial port handle = 3
[ INFO] 1285266497.355189000: Received [fw 0 10 0 0
]
[ INFO] 1285266497.355537000: written:fw 0 10 0 0
```

Fig.7 Controller Node

The other two nodes transmit commands for the local and remote mode of operation correspondingly.

The *Keycontrol* node (Fig. 8) reads commands from the local user input (a local terminal console), generates appropriate commands, formats them into messages and publishes them on *control_if* topic [8].



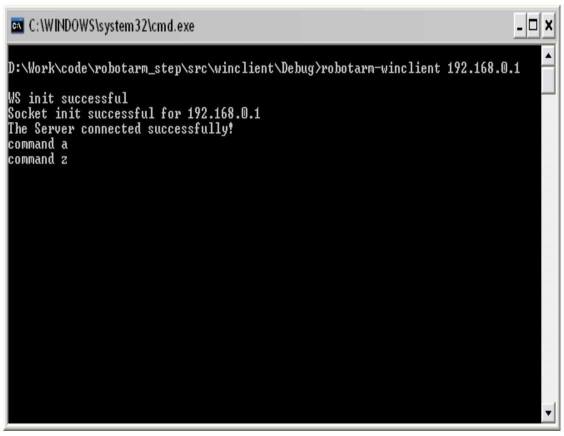
```
clmi-bas@ivaylo-desktop:~$ rosrun robotarm_step keycontrol
[ INFO] 1285266676.407260000: I published [fw 1 10 0 0
]
```

Fig.8 Keycontrol Node

7. Remote Control

The Remote control consists of two parts – Server and Client. The Client accepts user input and transmits the corresponding commands remotely to the Server via TCP/IP connection over a network (i.e. the Internet). The Server listens for Client connections and accepts commands from the connected Client. Then it publishes the commands to the *control_if* topic where they become available for the Controller node to read and relay them to the Robot.

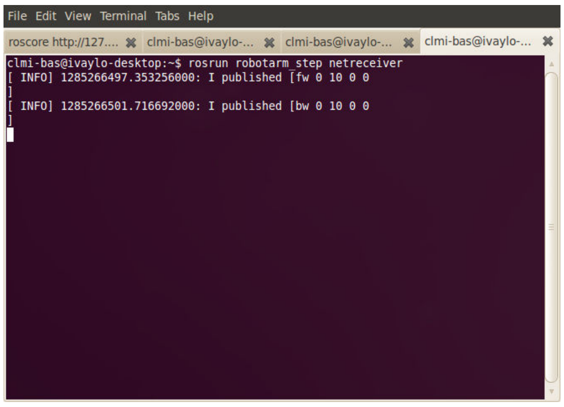
The Client is implemented for Windows and is a console application that connects to the Server at a provided IP address (Fig. 9). Then it starts to read user input, format appropriate commands and send them via the connection to the Server.



```
D:\Work\code\robotarm_step\src\winclient\Debug>robotarm-winclient 192.168.0.1

WS init successful
Socket init successful for 192.168.0.1
The Server connected successfully!
command a
command z
```

Fig.9 Windows Client Application

The Server is implemented for Linux and is both a TCP/IP server and a ROS node called *NetReceiver* (Fig. 10). It waits for commands from the remote Client console, and publishes them on the topic *control_if.*



```
clmi-bas@ivaylo-desktop:~$ rosrun robotarm_step netreceiver
[ INFO] 1285266497.353256000: I published [fw 0 10 0 0
]
[ INFO] 1285266501.716692000: I published [bw 0 10 0 0
]
```

Fig.10 NetReceiver Node - Server

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 reading commands from a file and sending them to the Controller node 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.

8. 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 for a 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 the efforts on 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 means that it introduces for distribution of an application over multiple computers allows for cooperation of nodes on an embedded system with those working on a local or remote computer terminal, and in the future eventually control through handheld computer devices and smartphones. All those features make ROS a candidate for the first choice and excellent perspective in the field of robotics and embedded control.

REFERENCES:

[1] http://www.linux.org

[2] http://www.ubunto.com

[3] R. Zahariev, N. Valchkova., Automated measuring system for analysis of the robot's condition, "Robotics and Mmechatronics 2008", Varna, 17-21 September, 2008, "Science News", ISSN1310-3946, V. 4/107, pp. 220-224.

[4] N. Chivarov, P. Kopacek and N. Shivarov; Modular Control System For The Family Of Educational Robots "ROBCO"; Robotics and Mechatronics 2009, October, Varna, Bulgaria; p23 – p. 27, ISSN1310-3946.

[5] Chivarov, N., Shivarov, N. and P. Kopacek; Mechatronics Educational Robots ROBKO Phoenix; "International Journal Automation Austria - IJAA" 2009; "International Journal Automation Austria - IJAA" 2009, Volume 17, Issue 2, ISSN 1562 – 2703, p. 68 – p. 73.

[6] Atmel; http://www.atmel.com

[7] http://www.ros.org/wiki/

[8] Nayden Chivarov; Robot Arm Control under ROS/Ubuntu; Problems of Engineering Cybernetics and Robotics, Volume 62; 2010 Sofia; ISSN 0204-9848