17th International Conference
ENGINEERING MECHANICS 2011
Svratka, Czech Republic, 9 – 12 May 2011

MODULAR CONTROL SYSTEM ARCHITECTURE FOR A MOBILE ROBOT

J. Hrbáček[*], R. Hrbáček[**], S. Věchet[***]

Abstract: *The paper presents both hardware and software means used to develop a scalable low/middle level control system for the autonomous mobile robot Advee. Brief description of chosen hardware resources is given and use of a Linux-based operating system as a foundation of a soft real-time control system is discussed as well as the choice of other vital software components, e.g. inter-process communication mechanism. Described system has been fully implemented and successfully exploited in operation.*

Keywords: Robotics, real-time systems, inter-process communication.

1. Introduction

Developing any new technical work brings many conceptual decisions – especially when the work is an autonomous mobile robot. But only a few of these dilemmas can be solved at the beginning of development – and design of the "brain", the control system, is not an exception. It is therefore very important to find the right trade-off between abstraction and concreteness in design that determines proportion between the ease of initial development and possibility of future modifications. Modularity mentioned in paper title is one of parameters that strongly influence adaptability of the system to unexpected eventualities. In the case of described control system it is achieved in terms of both hardware and software means – hardware components are interconnected using shared busses that allow for adding and replacing individual components as long as defined communication interface is abided. The principle of software modularity is exactly the same – the modules can be added or reimplemented on condition of maintaining of specified interfaces.

A rich variety of control system designs can be found in the field of mobile robotics – ranging from simpler libraries aimed mainly on communication among individual components of the system to complex frameworks including a lot more. To the former group we can count for example IPC (Simmons & James, 2001) or LCM (Huang et al., 2010) that provide purely module interconnection and do not constrain any other aspect of the system architecture. On the other side, frameworks like Player/Stage (Gerkey et al., 2003) or MRDS (Microsoft, 2011) offer much more than just communication (e.g. a complete simulation environment including an artificial 3D world model) but also force the system to more predetermined shape. The system designers must then decide whether the complex frameworks bring more benefits than limitations and eventually exploit one of simpler packages – as this paper shows.

2. Hardware systems

The described control platform hardware consists of two main parts – the master single-board computer (SBC) and the slave components including sensors, actuators and auxiliary devices (power units etc.). A basic structure of hardware means in conjunction with software modules is in Fig. 1.

[*] Bc. Jan Hrbáček: Institute of Solid Mechanics, Mechatronics and Biomechanics, Faculty of Mechanical Engineering, Brno University of Technology, Technická 2896/2, 616 69 Brno; CZ, e-mail: jan@hrbacek.info

[**] Radek Hrbáček: Department of Telecommunications, Faculty of Electrical Engineering and Communication, Brno University of Technology, Purkyňova 118, 612 00 Brno; CZ, e-mail: radek@hrbacek.info

[***] Ing. Stanislav Věchet Ph.D.: Institute of Automation and Computer Science, Faculty of Mechanical Engineering, Brno University of Technology, Technická 2896/2, 616 69 Brno; CZ, e-mail: vechet.s@fme.vutbr.cz

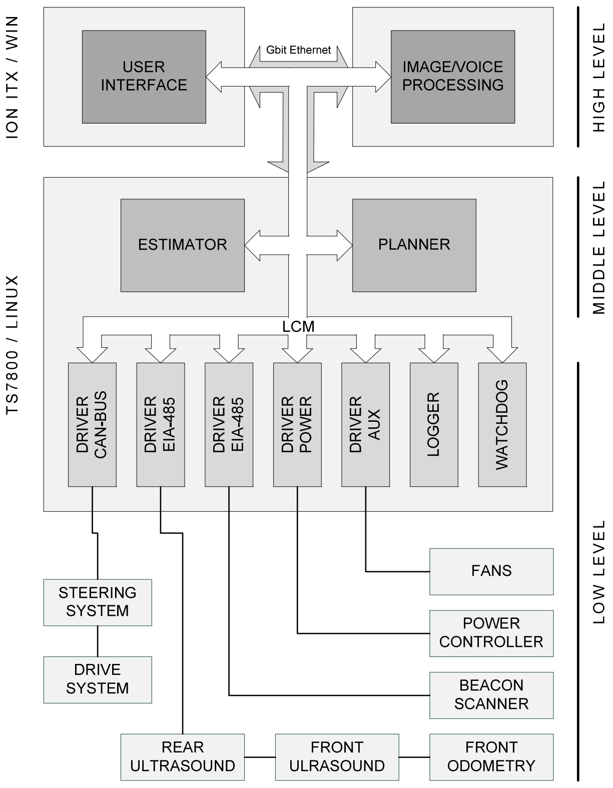

Fig. 1: Simplified hardware and software modules scheme.

2.1. Master computer

There were several aspects essential in the master computer selection process:

- industrial-grade *ruggedness* needed to lay solid foundations of overall control system reliability and robustness in harsh conditions,
- *communication busses* support necessary to interface peripheral devices (CAN-bus, EIA-485 etc.),
- sufficient *computational power* to run simple low-level data manipulation modules as well as more complex middle-level data processing.

All these features are implemented by the Technologic Systems' TS-7800 single-board computer which has been chosen to host low- and middle-level software equipment.

2.2. Sensors and actuators, auxiliary systems

Hardware components other than computers are a mixture of industrially used devices and custom developed modules. The sensory equipment is formed by sixteen ultrasonic range finders providing information on near surrounding of the robot and incremental rotary encoders for odometry. Sensors are polled for data using EIA-485 communication bus. Motion of the robot is handled by a driving motor and a steering motor, both industrially graded and controlled over CAN-bus. Last but not least, there is a system of power supply and switch modules that provide power to individual components of the robot. Control over these hardware modules (and thereby boot-up and halting sequence of the whole robot) is taken by a separate module communicating with the Linux SBC through EIA-422 link.

3. Low- and middle-level software means

Overall software equipment of the robot is divided into three layers:

- the *low level* provides interaction with hardware devices described in previous chapters – basically it translates device-specific data into standard inter-modular messages and vice versa,
- the *middle level* implements robot state estimation and path planning,
- the *high level* cares handles user interaction.

This paper deals with the two lower layers. Both these layers are hosted on the Linux single-board computer although thanks to used inter-module communication mechanism can be run on separate computers and even under different operating systems without any change of communication-related code. The use of Linux-based operating system was clear from the beginning – due to its high modularity it can be run in a variety of forms ranging from small embedded applications to multiprocessor supercomputers. The TS-7800 SBC is primarily designed to operate in conjunction with Linux and the source code of almost all custom hardware drivers is published by the manufacturer which leads to simple adaptability to concrete needs of the application and feasibility of fixing potential errors in implementation (own experience – problems with 9-bit serial port mode).

Every piece of developed software functionality is a part of some *module* – an independent process communicating with other modules using hereinafter described mechanisms.

3.1. Inter-module communication

Inter-module communication mechanism (or generally inter-process communication, IPC) belongs to the most important parts of the system. There are several roles that such mechanism should be playing:

– define *unambiguous interface* between modules

– provide *high-throughput* and *minimal-latency* communication channel

– support *multiple platforms* – at least Linux and Windows, C and C#.NET

A number of systems that solve these requirements exist – differing in the basic paradigm (client-server vs. publisher-subscriber), complexity (encapsulating more IPC methods, data marshalling[1] etc.) or operating system/programming language support. A good comparison of several popular IPC libraries targeted for use in robotics is provided in (Huang et al., 2009).

As already stated, whole software equipment of the robot is spread over minimally two computers. This fact disqualifies many traditional IPC methods as for example shared memory, files or pipes. The computers are interconnected through Ethernet and solution based on the standard TCP/IP protocol suite seems to fit well. The next decision after filtering out unsuitable mechanisms is conceptual – choosing the paradigm. Considering that most of the data flowing through the robot is of periodic nature, the publisher-subscriber model suits better. It can be implemented as “push” or “pull” – the “push” mechanism delivers data instantly while the “pull” method uses clients asking periodically for any new data. To ensure lowest possible latency, “push” method has to be used. Unambiguity of the communication between different interface implementations is best achievable using dedicated message definition files and programming language specific generators of message-related code (marshalling, un-marshalling, publishing etc.).

Having all these demands in mind, minimalistic yet powerful LCM (Huang et al., 2009; Huang et al., 2010) has been chosen as an optimal mechanism for message passing in Advee.

3.1.1. LCM: Lightweight Communications and Marshalling

LCM is a set of libraries developed originally by the MIT DARPA Urban Challenge Team for their autonomous vehicle Talos. It covered all requirements except for the .NET framework support – this minor flaw was solved by creating the *lcm-dotnet* port which has been added to the project. There is now a binding for C/C++, Java, Python, MATLAB and C# what makes it extremely flexible – users can for example test production code in C/C++ or Java in MATLAB simulation environment. LCM is composed of three main functional blocks:

– *message definition* using a C-like definition language that can be compiled into language-native message support files using the *lcm-gen* tool

– *message marshalling/un-marshalling* – conversion between native and byte representation with defined endianness and runtime type safety checking

– *communication itself* based on UDP multicast

[1] marshalling means encoding the message into a byte array sendable over the network

3.1.2. Standard module interface

On the top of LCM there are a few standard messages defined that each module has to implement – an error reporting message and module state setting and reporting messages. State of the module can take one of globally defined states (INIT, RUN, PAUSE, STOP, ERROR and TERMINATE) that form a state machine common to all modules. This allows using a watchdog module that guards other modules from misbehavior (it can e.g. restart an unresponsive module).

3.2. Remote diagnostics and simulation support

Inherent feature of the LCM's transport layer (UDP multicast) is that virtually unlimited number of recipients can subscribe to messages from outside of the system without any negative impact upon the system. It is then easy to implement a non-obtrusive diagnostic tool that can monitor system modules and visualize important data. Currently there are three separate diagnostic applications: for Linux console, Android OS and Windows/.NET, each with slightly different usage. A communication logger is also implemented simply by listening to all messages and writing them to a file.

It is possible to replace any module by its simulation equivalent with the same interface without letting know the rest of the system. This is used to emulate outer environment of the robot by replacing the low layer (both software and hardware) by a SIMLIB-based (Peringer, 2010) chassis simulator that runs a dynamic model of the robot in a given artificial world.

4. Practical experiences from robot operation

In general, presented control system architecture together with custom diagnostic tools proved to be flexible enough to allow software reorganizations during Advee's development and highly reliable in operation enabling commercial utilization of the robot. LCM as the backbone of the system fits the demands very well, although there are some difficulties related to UDP multicast implementation on Android OS (but this is related to diagnostics only). Communication latencies are imperceptible.

5. Conclusions

Unified modular architecture with shared communication mechanism brings many benefits including simple system monitoring/diagnostics, simulation and testing using unmodified production code. Scalability is given both by the ease to add more modules (for example interfacing a new sensor) and by the ability to spread individual modules to more computers without changing single line of code. Because every module runs as a separate process, the robustness of the whole system is not threatened by individual module instability. The qualities of presented approach were verified during over 100 hours of production operation.

Acknowledgements

Published results were acquired with the support of the Ministry of Education, Youth and Sports of the Czech Republic, research plan MSM 0021630518 "Simulation modeling of mechatronic systems".

References

Gerkey, B.P., Vaughan R.T. & Howard, A. (2003) The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. Proceedings of the International Conference on Advanced Robotics (ICAR 2003), Coimbra, Portugal, pp. 317-323.

Huang, A.S., Olson, E. & Moore, D.C. (2010) LCM: Lightweight Communications and Marshalling. Proceedings of the International Conference on Intelligent Robots and Systems (IROS), Taipei, Taiwan.

Huang, A.S., Olson, E. & Moore, D.C. (2009) Lightweight Communications and Marshalling for Low-Latency Interprocess Communication. Technical Report MIT-CSAIL-TR-2009-041, Cambridge, USA.

Microsoft Corporation (2011). Microsoft Robotics Developer Studio. <http://www.microsoft.com/robotics/>.

Peringer, P. (2010) SIMLIB Home Page. <http://www.fit.vutbr.cz/~peringer/SIMLIB/>.

Simmons, R. & James, D. (2001) Inter-Process Communication: A Reference Manual. Pittsburgh, USA.