# Autonomous robot running Linux for the Eurobot 2007 competition

**Tran Duy Khanh, Zidek Martin, Benda Jan, Kubias Jiri, Sojka Michal**
Czech Technical University in Prague, Faculty of Electrical Engineering
Department of Control Engineering
Karlovo namesti 13, 121 35, Praha 2
{trandk1, zidekm, sojkam1}@fel.cvut.cz
http://rtime.felk.cvut.cz/dragons/

**Abstract**

This paper presents technical solutions for an autonomous robot running *GNU/Linux*. While the majority of article focused on the software implementation, we will slightly describe some of our mechanical and electrical solutions as well. The paper outlines several interesting software built for embedded real-time application. Later we will take a look at the robot localization using a laser beacon, passive reflectors and an efficient algorithm for mobile robot localization, called Monte Carlo Localization.

## 1 Introduction

Eurobot is an international amateur robotics contest opened to teams of young people, organized either in student projects or in independent clubs. Eurobot was formed from a competition founded in France, now it takes place in Europe but also welcomes countries from other continents.

This year, the topic of the competition was *Robot Recycling Rally*. The robots were supposed to sort waste. There are three kinds of waste to find: bottles, cans and batteries. The matches involve two teams and last 90 seconds.

Each team is associated with a color, red or blue. There are two bins for each team: one for bottles and one for cans. In addition, there is one, shared, basket for the batteries. Each robot finds some garbage on the table, transports it to the correct bin, deposits it, and returns to look for some more garbage. The robot, which sorts more waste into correct bins will be the winner.

Our team, *CTU Dragons*, participated this year in the Eurobot competition for the first time. We represent *Department of Control Engineering [7]* of *Czech Technical University in Prague [6]*. During construction and implementation of robot we were using knowledge and technologies developed in our department or those commonly used in control engineering and in industry.

The rest of this paper is organized as follows. Section 2 describes mechanics and electronics of our robot and sections 3 through 7 describes various software parts of our control system. We give a conclusion and directions for future development in section 8.

This work was supported by the Ministry of Education of the Czech Republic under project 1M0567 (CAK) and by European Social Fund under project CZ.04.3.07/4.2.01.1/0045 (CEPOT).

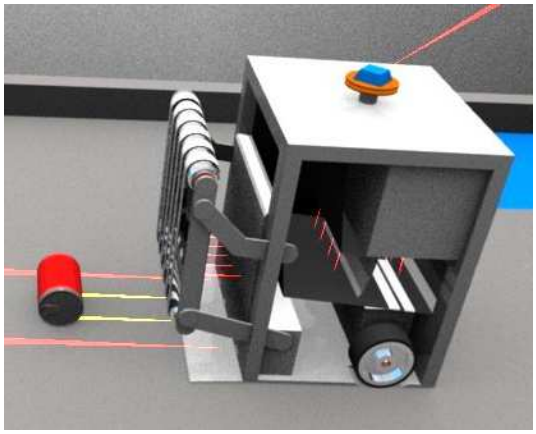

**FIGURE 1:** *3D model of the robot*

## 2 Hardware System Structure

### 2.1 Mechanical parts

Collecting mechanism consists of two rotating cylinders, rubber belts and a lifting door. Robot can hold

four waste peaces in his built-in stack. Depositing mechanism consists of a conveyor belt and a back door. For the waste deposition, robot goes to the baskets situated in the corner of the playground. Afterwards, it uses the conveyor belt or the back door in order to eject waste from the stack depending on whether it is a bottle or a can. See the 3D model of the robot in Figure 1.

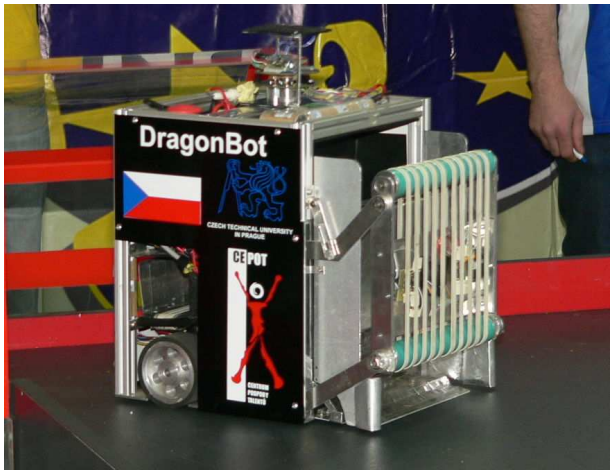


**FIGURE 2:** *The robot*

## 2.2 Electronic parts

The electronic system of the robot (see Figure 3) consists of four subsystems.

The first subsystem is the main controller. As a main controller for the robot was chosen embedded board *BOA 5200* from Analogue Micro. This board is based on MPC5200, a *PowerPC* processor by Freescale. The board features among others 10/100TX Ethernet controller, 2 x CAN controllers, serial interface, I2C interface and SPI interface. Especially presence of *CAN controllers* was very important for choosing this board, because we decided to go for CAN bus as a main bus for interconnecting individual subsystems in our robot. CAN is a differential bus used especially in automotive industry.

The second subsystem represents the sensoric system. All the sensors are connected to a board with Hitachi/Renesas H8S2638 processor. The processor collects data from all the sensors and sends the values to the main controller via the CAN bus.

The third subsystem is the drive controller board. This board is also based on the H8S2638 processor and contains bridges for driving the motors. Robot is powered with *two brush-less DC engines by MAXON*.

The last subsystem performs collecting measurements from the laser beacon, used for robot's localization. Electronic is based on a modulated laser emitter and a detector with band-pass. Data are captured and sent to a CAN bus using a board again with the H8S2638 processor.

Robot is equipped with several tens of IR sensors for waste detection, waste recognition, opponent detection and obstacle detection.

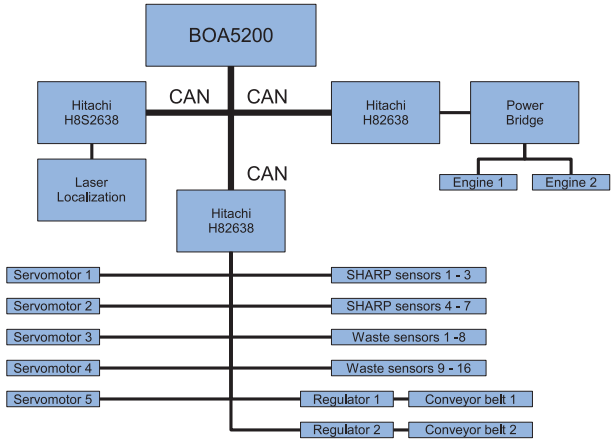


**FIGURE 3:** *Hardware structure of the robot*

## 2.3 Main control unit

As was already mentioned above, the main control unit is based on a MPC5200 board with a PowerPC processor. The board runs *Linux Kernel version 2.6.18*. For the root file system we use the Journalling Flash File System version 2 or *JFFS2*. It is a log-structured file system for use in flash memory devices. JFFS2 supports NAND flash devices and in comparison to JFFS, it supports hard links, compression and better performance.

Since a small size of integrated flash memory (16MB), we use an efficient software called *BusyBox*. It is a software application which combines tiny versions of many common UNIX utilities into a single small executable. It provides replacements for most of the utilities which can be usually found in GNU fileutils, shellutils, etc. The utilities in BusyBox generally have fewer options than their full-featured GNU versions. However, the options that are included provide the expected functionality and behave very much like their GNU counterparts.

Size of the whole root file system is less than 5MB. *Redboot*, a famous boot loader by eCos, is installed on the board to provide means for simple kernel and file system replacement. Communication with the board is realized through serial line or onboard integrated Ethernet controller to which WiFi access point is connected.

# 3 Main Control Application

The program development was carried out mostly in *C/C++* using free software tools and libraries. Components are designed with respect to the versatility. Main control program of the robot has been built on *finite state machine* architecture. We have developed a sophisticated and easy to use API to implement the state machines. By now we are using four state machine running simultaneously in separate threads. The main program consists of the main state machine for game strategy, state machine of the robot's motion, state machine of the waste collection and the last is used for the localization.

Robot motion is controlled by a layered architecture. The lowest layer comprises of *PID controllers* run in the driver board. On top of this, there are several layers run in the main controller. The *trajectory planner* layer prepares a smooth trajectory from a set of way-points and *trajectory controller* tries to keep the robot on that trajectory. Finally, the *path planner* layer finds the optimal obstacle-free path connecting two points.

During the development, we have used several components which have simplified our work in many aspects. Some of these systems will be described further in this paper.

# 4 OMK Make System

OMK [2] (*Ocera Make System*) is a Make system developed by our department under the *OCERA project* [5]. The main objectives of the OMK system is to simplify compilation of components on the host machine as well as cross compilation for the target. In addition the system brings a better directory and file structure. Make system allows to build out of sources tree and store results of build in a separate directory structure to simplify testing and program install.

A key solution is to have a central Makefile with compilation rules for most of sub-components and components. This solution allows faster and smoother change on the system, such as the kernel update. Having most rules in a central file, Makefiles in sources directories can be very simple.

The OMK system gives us a possibility to develop robot's control program and libraries on high-capacity desktops along with cross-compiling final program for the PowerPC board.

# 5 ORTE

ORTE stands for *OCERA Real Time Ethernet* [1] and it is a part of OCERA project [5]. ORTE [1] is an open source implementation of RTPS (*Real-Time Publish-Subscribe*) communication protocol defined by Real Time Innovations. RTPS is an application layer protocol, which has two main communication modes. One is the publish-subscribe protocol for transferring the data from publisher to subscribers, and the other one is Composite State Transfer (CST) protocol, which transfers state. RTPS protocol was designed to use an unreliable underlying network protocol, such as *IP/UDP*.

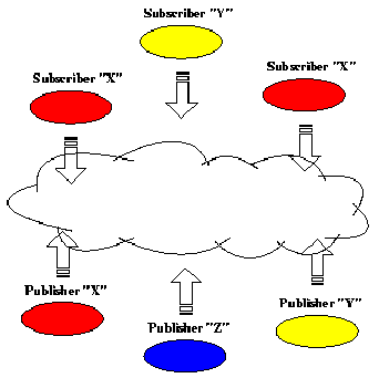


**FIGURE 4:** *Publisher-subscriber model of ORTE communication*

The publish-subscribe architecture was designed to simplify data distribution from one source to many recipients. The publisher does not have to have any knowledge of the number or location of subscribers. Also, the subscribers simply receive the data anonymously and thus they don't need to know any information about the publisher. An application can be publisher and subscriber at the same time.

The publish-subscribe architecture is best suited for distributed applications. It is scalable and the data flows can be managed easily regardless of the number of nodes (publishers and subscribers) connected to the system. When subscribing to a data flow, the application specifies only the topic of the data it wants to receive, rather than to any specific publisher.

The publish-subscribe services are typically made available to applications through middle-ware, that sits on top of the operating system network interface and presents an application programming interface.

In our case, we are using ORTE [1] to publish data from an application, which intercepts data from CAN bus. Then the data are subscribed in the main controlling application running on the main controller board, but also in a graphical application

running on a standard desktop or portable computer, which we use for visualizing all the data from the robot. Thus we can see on-line all the sensors readings, information about the position of the robot and so on. The controlling application publishes all the commands for controlling the motion of the robot to ORTE [1] also, and the data are subscribed in CAN bus communication application, which transforms them into appropriate CAN messages and sends them to the motor driver board. This approach makes it possible to control the robot as from the controlling application as from the graphical application on a separate computer in the same manner.

# 6 Localization

To navigate reliably in the playing area, a robot must know where it is. Reliable position estimation is a key problem in mobile robotics. As mentioned above, the robot localization is done using a laser beacon and three passive reflectors. The laser beacon is positioned above the robot. Passive beacons are covered with reflective tape and placed on reserved supports around the playing area. Measured angles between the reflections are used to calculate robot's position. By means of a calculated position, Monte Carlo algorithm [3] will be able to update its model of the predicted position.

*Monte Carlo Localization* makes use of a methods that were invented in the seventies and recently rediscovered independently for instance in the target-tracking, statistical or computer vision. Monte Carlo Localization method is a particle filter, where the probability density is represented by maintaining a set of samples, that are randomly drawn from it. By using a sampling-based representation we obtain a localization method that can represent arbitrary distributions. Thus, the method is able to *localize globally* a robot. Otherwise Monte Carlo Localization is able to localize a mobile robot without knowledge of its starting position, which allow robot to re-localize after a collision with obstacles or opponent robot on the playground.

In robot localization, we are interested in estimating the state of the robot at the current time. The state vector consists of the position and orientation of the robot. The probability density function is taken to represent all the knowledge we possess about the state, and from it we can estimate the current position. To localize the robot we need to recursively compute the density. This is done in two phases:

*Prediction phase:* In the first phase we use a motion model to predict the current position of the robot. In our case, the predicted position is done by the odometry. Odometry is the use of data from the rotation of wheels or tracks to estimate change in position over time.

*Update phase:* In the second phase we use a measurement model to incorporate information from sensors to correct the predicted position. In our case, the measured position is calculated using reflected angles between beacons.

The global localization capability of the Monte Carlo Localization method is illustrated in the following figures. In the first iteration, the algorithm is initialized by drawing 5000 samples representing the probability density. As the robot moves over the area, the samples are concentrated to the estimated position.

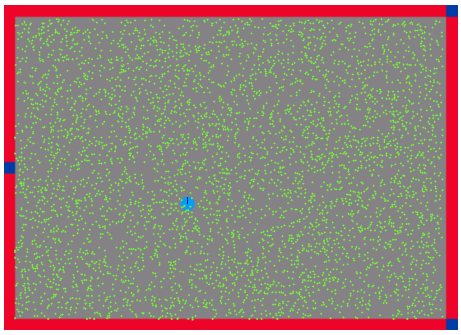

**FIGURE 5:** *Initialization of 5000 samples.*

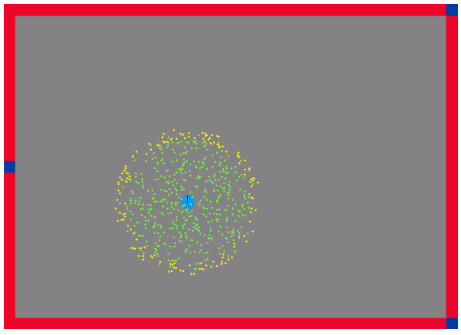

**FIGURE 6:** *Global localization after the first step.*

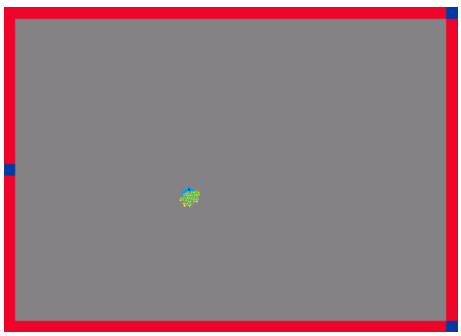

**FIGURE 7:** *Estimated position after 15 steps.*

# 7 Visualization

Within the project, we have developed an application used for visualization of the robot. The application shows sensor values, position of the robot and allows controlling of all subsystems of the robot.

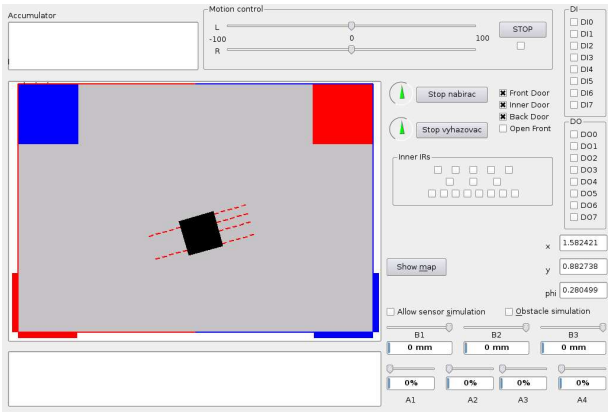


**FIGURE 8:** *Robot visualization*

In addition it makes simulation of the robot behavior possible. Thus, the simulation simplifies development of algorithms, such as path planning or localization algorithm.

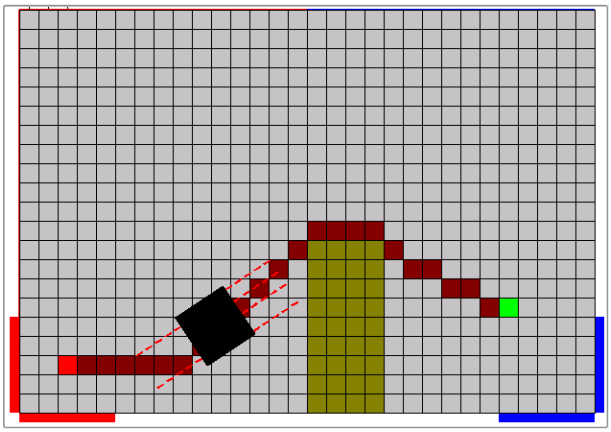

**FIGURE 9:** *Trajectory visualization*

# 8 Conclusion and Future Work

In this work we introduced a technical solution of an autonomous mobile robot for the Eurobot competition. After a year of work, we have developed an amount of hardware and software components. With the experience of this year we hope to be more successful in the upcoming competition. Recent development is concentrated on bug fixing of the application interface, robot's movement, path planning and localization part. Analyzing existing problems should be a no less important task to the success for the next year competition.


**FIGURE 10:** *CTU and CEPOT logos*

# References


[1] *ORTE Documentation – Communication Components*, 2004, Jan Krakora, Pavel Pisa, Frantisek Vacek, Zdenek Sebek, Petr Smolik, Zdenek Hanzalek *http://www.ocera.org/download/components/WP7/orte-0.3.1.html*

[2] *OMK documentation*, Pavel Pisa, Michal Sojka *http://rtime.felk.cvut.cz/hw/index.php/OMK*

[3] *Monte Carlo Localization: Efficient Position Estimation for Mobile Robots*, Dieter Fox, Wolfram Burgard, Frank Dellaert, Sebastian Thrun

[4] *Building Embedded Linux Systems*, 2003, Karim Yaghmour

[5] *OCERA Project: http://www.ocera.org*

[6] *Czech Technical University in Prague, Faculty of Electrical Engineering: http://www.feld.cvut.cz*

[7] *Department of Control Engineering: http://dce.felk.cvut.cz*

[8] *CEPOT Programme: http://www.cepot.cz*

[9] *Eurobot official homepage: http://www.eurobot.org*