MobotWare – A Plug-in Based Framework for Mobile Robots

Anders B. Beck*,, Nils Axel Andersen*, Jens Christian Andersen* and Ole Ravn***

**Department of Electrical Engineering, Technical University of Denmark, Lyngby, Denmark*
(e-mails: abb@elektro.dtu.dk, naa@elektro.dtu.dk, or@elektro.dtu.dk, jca@elektro.dtu.dk)
***Centre for Robot Technology, Danish Technological Institute, Odense, Denmark,*

Abstract: This paper describes a plug-in based software framework developed at Automation and Control, DTU Electrical Engineering. The software has been used for education and research in mobile robotics for the last decade. Important design criteria have been real-time performance of the control level, easy integration of sensors, fast porting to new robots and core system stability and maintainability in an undisciplined programming environment.

Real-time performance is assured by using RTAI-Linux; core stability is obtained by using plug-ins for user developed modules. The plug-in based module structure combined with inter-module communication based on TCP/IP sockets and human readable XML-protocol makes it easy to use the system on a wide range of hardware platforms, configurations and computer platform distributions. The framework has until now been interfaced to 7 different hardware platforms and has enabled many application i.e. robust navigation in an orchard with an autonomous tractor (Andersen,2010). Furthermore by providing a simple scripting robot control language the system also supports use by non-technicians.

Keywords: Architectures, Autonomous mobile robots, Multisensor integration, Hierarchical systems, Real-time systems, Robot control, Reusable robotic software, Robotic framework

1. INTRODUCTION

Writing a software framework for autonomous mobile robot requires an effort that goes beyond one project. This means that it is important to control development over a longer period. The software development at universities is mostly based on bachelor, master and PhD projects along with short term externally funded projects. This is a very dynamic environment in which it is difficult to enforce strict software development methods and rules. Often this means that the software written in a project is abandoned when the student leaves after finishing his or her project, leading to rewriting the same functionality again and again not only in different places in the world but even at the same department.

Automation and Control, DTU Electrical Engineering have researched and built mobile robots since the early 1980's. In 1999, for the purpose of teaching and supporting thesis-based research, a Small Mobile Robot (SMR) was developed and built in 12 units. The SMR is a differential driven platform; using a standard small PC motherboard connected to custom motor control and sensor hardware. The system runs standard Linux and connects to the institute network through WiFi wireless network.

The foremost task for the SMR was teaching in real-time control of hardware systems, where engineering students competed in sensor signal conditioning, odometry calibration and finally navigating the SMR through a dynamic obstacle course.

To motivate students and research staff to avoid writing mobile robot software from scratch repeatedly a real-time motion control, SMRdemo and an elaborate scripting language, the SMR Control Language (Andersen et al., 2004) was developed. After nearly a decade of development, the remarkably sticky name SMRdemo was changed to Mobile Robot Controller (MRC).

Through years of research projects, the MobotWare architecture evolved by introducing the Automation Robot Servers (AURS), a plug-in based server framework for processing complex sensors, such as Laser scanners and cameras in a more comprehensive soft real-time environment. AURS also hosts planning algorithms and high-level mission management systems.

This paper introduces the MobotWare framework and introduces how the unique combination of hard real-time safe temporal and functional hierarchical decomposition has formed a portable, flexible and lightweight framework that provides stability and performance for research in robot systems solutions and integration enabling real applications on mobile robotics to be developed.

2. RELATED WORK

Research in mobile robot architectural frameworks received much attention in the days of the pioneering work of Brooks (1986) and Albus (1989), but is still an active research area. Particularly within systems integration and debugging there is much left to be done (Coste-Maniere et al., 2000).

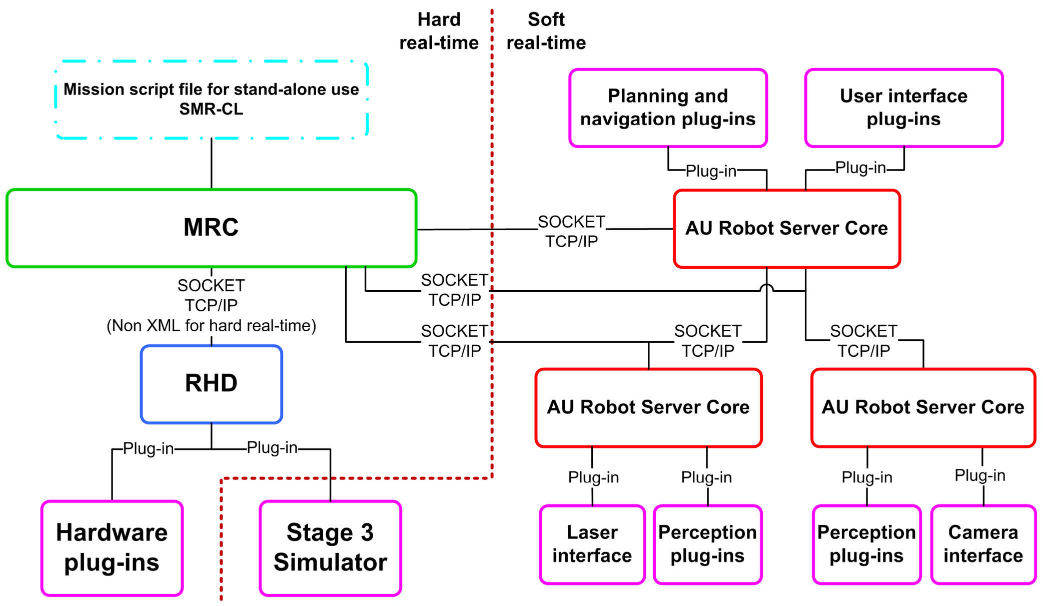


Fig. 1. Overview of the MobotWare mobile robot control framework.

MobotWare is focused towards research on integration of the framework architecture and developing a functional framework for different platforms and applications.

In the last decade, several research groups have tried to address the problem of collaboration in mobile robot research through open source projects e.g. the Player project (Vaughan et al.,2003), the CARMEN toolkit (Montemerlo et al.,2003), the OROCOS project (Bruyninckx et al.,2001), and recently followed by the ROS project (Quigley et al., 2009). Common for the projects is that they focus on comprehensive hardware and device abstraction, but leave the control architecture open for the users, thus stimulating community collaboration on low-level signal processing algorithms, but not system architectures for solving the complex tasks of mobile robots.

With exception of the ROS project, the international robotics community has recently turned its focus back towards development and standards for robotic systems architectures, where the most visible effort recently has been the Joint Architecture for Unmanned Systems (JAUS) project. The release of the NASA origined CLARAty architecture to open source (Nesnas et al.,2006), is another evident sign that research again addresses the challenges of layered architectures to solve the remaining complex challenges of mobile robots.

CLARAty follows the modern architectural doctrine of decomposing hierarchically by functional abstraction. The approach is comparable to the popular three-tiered architecture (3T) (Bonasso et al.,1997), that features a planning layer, a behavioural execution layer and a intermediate mediation layer. The decomposition of the MobotWare architecture is also somewhat similar, but a distinctive difference is that MobotWare follows two dimensions of decomposition: temporal and functional.

The temporal dimension divides MobotWare in two sections, a hard and a soft real-time constrained section (Fig. 1). Secondly a functional decomposition decomposes the architecture in levels of increasing abstraction, from hardware abstraction layer to reactive execution layer to deliberative perception and planning layers. This decomposition has lead to interesting research in the area of crossing the hierarchical boundary between hard and soft real-time without compromising hard real-time performance (Beck et al., 2009)

3. SYSTEM OVERVIEW

The MobotWare framework has three core modules:

- *Robot Hardware Daemon (RHD)* Flexible hardware abstraction layer for real-time critical sensors
- *Mobile Robot Controller (MRC)* Real-time Closed-loop controller of robot motion and mission execution
- *Automation Robot Servers (AURS)* Advanced framework for processing of complex sensors and non real-time mission planning and management.

The modular architecture defines a role for each module in the architecture. It forces developers to solve problems at the levels where they origin and belong, as all data-reduction must be handled at the level where the data is locally available, rather than where a developer finds it most convenient at a given time.

Modules are configured through XML files, which make reconfiguring to changed hardware setups or new robots quick and manageable.

Core components are connected through low latency TCP/IP connections, that makes is possible to distribute the MobotWare components across multiple computer platforms, without the need of complex 3rd party communication libraries. This could be a low-power platform for real-time control and a high-performance platform for laser or vision processing, or it could be a development module conveniently running on the development machine, while all other modules are performing on the actual robot platform.

All communication protocols, except the RHD real-time sensor interface, are implemented in human-readable XML, which is efficient for debugging and testing, yet powerful for inter-process communications due to efficient and readily available parsers. By design decision, sensor data must be processed at the source and only reduced data are distributed, in opposition black board database architectures. It makes the development doctrine slightly strict, but improves stability in a university development environment.

Development and research of new functionality within the framework, is done through a plug-in interface on all system levels. This ensures that students and researchers have a well documented entry-point into the framework that minimizes time spent on learning a complex system. Another strong advantage is that core stability of the MobotWare framework is always ensured, as failed or uncompleted projects can easily be removed – and successful projects can be saved!

4. ROBOT HARDWARE DAEMON

The Robot Hardware Daemon (RHD), a real-time device server, has been developed to cope with the increasing diversity of the supported mobile robot platforms. In opposition to related initiatives such as Player or CARMEN, RHD is limited to be a lightweight hard real-time hardware interface. Signal processing at RHD level are limited to protocol interaction and simple security functions as emergency stops.

RHD is a real-time synchronized variable database. The variable database structure provides great flexibility, but it also helps to enforce a clean cut interface between the various hardware formats and a user-manageable API without enforcing specific device abstractions. RHD is also the primary real-time control scheduler in the MobotWare framework, so much effort was used to analyze the behaviour of scheduling mechanisms in Linux, and to create a robust, lightweight and flexible real-time safe implementation.

There is a balance of keeping compatibility to a traditional desktop Linux distribution and meeting the requirement of hard real-time performance. As described in the scheduler section, both objectives have been achieved.

RHD consists of a set of core components, and a range of specific hardware driver plug-ins. The core components create the basis for a variable database, TCP/IP server and the real-time scheduler. Plug-ins creates the support of hardware devices, by managing hardware I/O, pre/post-processing and database interface.

All setup of RHD is based on a XML configuration file, which contains the setup parameters for the core components and plug-ins to accommodate any supported hardware configuration.

4.1 Variable Database

The variable database defines the functionality of RHD. Upon initialization, all plug-in modules create the I/O variables they need; subsequently the database is locked when going into hard real-time mode. The database itself is based on a variable symbol table and an associated data area.

- The symbol table defines the static information regarding the variables and bookkeeping information.
- The data area is one continuous memory pool, containing the dynamic data and a time stamp.

4.2 TCP/IP Server

As the only service connection in MobotWare, the RHD protocol is binary to improve performance in the real-time environment. To minimize the data-flow, the symbol table is only transferred to a client in a initial hand-shake. After the handshake, only dynamic data is exchanged.

Only one master client is able to write to RHD variables, but multiple read-clients are supported to give data access for the non real-time layers and operator interface.

4.3 Real-time Scheduler

Besides being a networked variable database, RHD is also the main real-time scheduler for low level robot control applications, such as MRC. The MobotWare framework is expanding to support new robots and as some of these hardware platforms are big, powerful, and heavy, fault tolerance and especially real-time performance are critical.
Hard real-time performance is not generically supported by the Linux kernel, as there is no support for kernel preemption. In projects, as RTAI (Mantegazza et al., 2000) and RT-Linux (Bruyninckx, 2002), this is solved by patching the kernel with a second scheduler, which allows tasks to work in kernel priority levels and with full kernel preemption. Our RT-implementation of choice is the Linux Real-Time Application Interface (RTAI). Linux itself, provide some means of achieving soft real-time performance. RHD is capable of operating satisfactory on a standard Linux distribution and with enhanced stability and real-time performance using the RTAI scheduler.

All scheduling in Linux happen with the period of the scheduler, the so-called *jiffy* (Abbott, 2003). Previously in Linux, this frequency was normally set to 100Hz, but seemingly the default of the vanilla kernel is now 250 Hz. This creates a period of 4 ms and makes it impossible to obtain a steady 10 ms control cycle. Fig. 2 show the timing of a Linux Interval-timer interrupt, set at a 10 ms target period on an X86 VIA C-3 platform and on an Atmel 130 MHz AVR32 NGW100 Linux evaluation board, both with a 250 Hz kernel scheduling frequency.

When trying to achieve a period, that is not divisible with the scheduler period, the timer will simply alternate between the two closest scheduler ticks and thus create an accurate average timing period. This might be reasonable, if the timer period is much larger than the scheduler period, but for real-time, high frequency robot control it is highly undesired.

When compiling the 2.6 Linux kernel, it is possible to set the scheduler frequency for 100, 250, 300 and 1000 Hz. To run RHD using the Linux scheduler, the scheduler loop period must be carefully selected as a multiple of the Linux scheduling period.

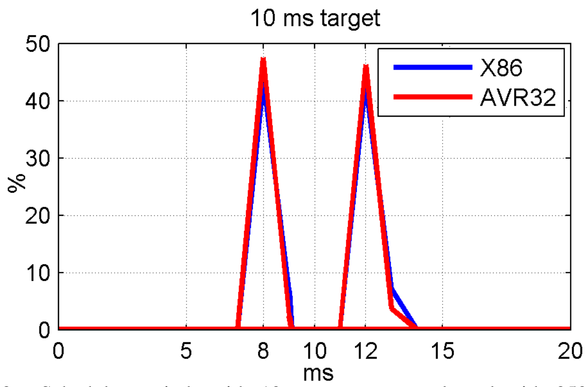


Fig 2. Scheduler periods with 10 ms target on a kernel with 250 Hz scheduling frequency. Timer periods alternate between 8 and 12 ms to obtain an average 10 ms period.

For hard real-time performance RHD uses the LXRT module and RT-FIFOs from RTAI to obtain real-time scheduling in user space.

4.4 Plug-in architecture

Specific hardware drivers are implemented through a plug-in structure where each plug-in creates the I/O control variables in the variable database. Control variables are used to transfer data from and to hardware devices, but also offer the possibility for configuring the driver plug-in dynamically.

4.4.1 AuSerial plug-in

An important is feature of RHD is a simple and easy configuration of new low level sensor devices. To make it simple to add new devices to the interface bus, and easy to do simple reconfigurations, the AuSerial plug-in was developed. Using AuSerial, newly developed devices can be introduced in the entire MobotWare architecture and configured on SMR RS-485 bus by a XML configuration and without writing a single line of C-code.

4.4.2 Other hardware plug-ins

Besides the AuSerial plug-in, a number of hardware plug-ins has been written for RHD to support a range of hardware and a simulator.

- Stage Simulator 2.1.1
- iRobot r-Flex interface
- RTK and NMEA GPS interface
- Crossbow IMU-400 and Fiber Optic Gyro
- HAKO tractor CAN-bus control interface
- Claas Axion tractor CAN-bus control interface

5. MOBILE ROBOT CONTROLLER

The mobile robot controller (MRC) is providing the basic low level real-time control of the mobile robot platform. It uses RHD (Robot Hardware Demon) as an interface to the actual robot hardware. MRC has the following features

- Odometry
- Motion controller
- SMR-CL interpreter (Andersen, 2004)
- Socket interface to high level controllers
- XML-based socket interface to sensor servers
- Socket interface to RHD
- XML-based configuration file
- Calibration support for odometry, line-sensors and distance sensors.

The odometry is based on wheel measurements and inertial sensors e.g. gyros. It is configurable using an XML-based configuration file for several kinematics: differential drive, Ackerman steering and vehicles described with linear and angular velocity (v, ω).
The motion controller provides the basic path control based on odometry as well as sensor based movements for the following SMR-CL commands:

- *fwd* point to point forward motion
- *turn* turn around center
- *drive* continuous linear motion
- *turnr* circular movement with given turning radius
- *stop*
- *followline* follow a line on the floor (painted or buried wire depending on sensors)
- *followwall* follow a wall using e.g. infrared distance sensors or laser scanner

Small Mobile Robot Control Language, SMR-CL (Andersen, 2004), (Jørgensen et al., 2008) is an interpreted language intended for control of mobile robots. The language supports sequences of basic robot actions with multiple criterions for seamless motion gluing as well as standard mathematical expressions. The system has two kinds of variables, system variables that reflect the state of the vehicle and user variables that are created the first time they are used in an assignment clause. The language is intended to be used in the tactical layer just over the control layer i.e. it is fast enough to shift between control strategies and react to state changes in real time. SMR-CL provides the bridge between the hard real-time demands of the control layer and the soft real-time demands of the planning (strategic) layer. The language may be used in two ways either as scripts runs directly from text files or as a command language through a socket interface.

6. AUTOMATION ROBOT SERVERS

Sensors with high data volume – like cameras and laser scanners – usually require time consuming data processing. This is not easily compatible with the real time requirements of the motion control.

Each sensor is often used for more than one purpose, e.g. the laser scanner may be used for detecting obstacles, for wall following, for human detection and emergency stop. An architecture, which supports sharing of sensor data, is therefore essential.

More than one processor unit is often needed when processing data from these high volume sensors. A standardised communication interface is therefore beneficial. In a university environment new functionality is often developed by students. It is therefore important, that the development can be done in independent groups and within a limited time frame. The architecture and programming interface must therefore be reasonably simple, and it must be easy to add (and remove) the developed modules without influencing other parts of the software.

The MobotWare solution to these challenges is a set of servers called AURS. A typical AURS configuration consists of a camera server, a laser scanner server and a mission management server.

Each server consists of a server core that provides a number of general services to the functional plug-ins.
These services include:

- Management for loading, unloading and configuring plug-ins dynamically
- Communication services (socket based)
- Event handling (generation and implementation)
- Server wide shared plug-in variable tree
- Plug-in to plug-in communication
- Direct access to base plug-ins (like laser sensor plug-in and pose history)
- API for common tools (like linear algebra, coordinate conversion and image algorithms from openCV.
- Data logging and replay services

As an example the perception laser scanner server could be configured as shown in Fig. 3. There is a plug-in to maintain the recent robot pose in each of the maintained coordinate systems (odometry and map coordinates), these are capable of providing the robot pose at any given (sensor) time. The laser scanner plug-in provides (raw) laser scanner range data and the laser scanner pose – relative to the robot. The other plug-ins uses the sensor data to create and maintain a specific perception model.

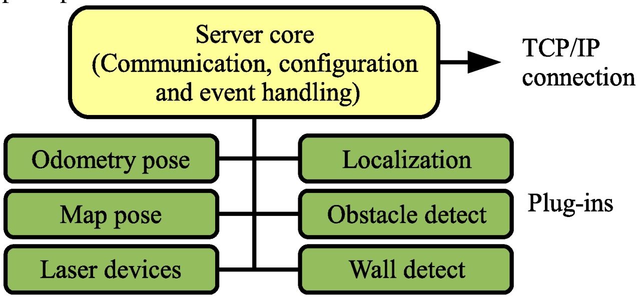


Fig 3. The AURS structure, with server core and functional plug-ins

The communication to and from a plug-in (apart from the internal plug-in to plug-in function calls) is handled by the server core using XML based messages. Each plug-in handle commands formatted into one or more XML tag types, and thus can share the same server provided socket connection.

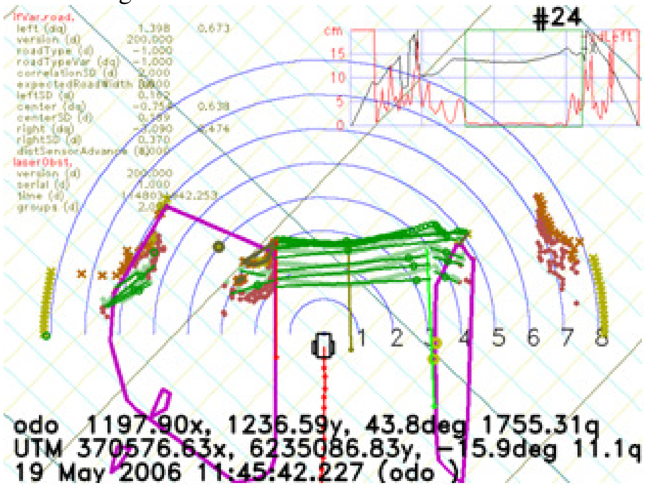
A plug-in can also trigger an event – an event could be that a new lasers scan is available – the server may then be configured to react by sending an XML command to other plug-ins, e.g. to update the wall and obstacle model. Update of the obstacle model could trigger a further event, e.g. to start a new behaviour generation.
As described here, both sensor processing and part of the perception model is handled in one server. This is intentional, as the high volume data flow from the sensor plug-in to the perception plug-in in this way is kept internally in the server. The reduced data flow from the generated model is communicated to other servers only.

The server structure allows both client-pull data flow – send a command and get a reply – and server-push data flow, where data is send as a response to an event. Timed events are supported directly by the server core.
A large number of plug-ins are available, these include:

- a laser scanner plug-in, supporting Sick and Hokuyo scanners,
- a camera device plug-in.
- a MRC interface plug-in for easy access to real time data and behaviour commands
- a road detection plug-in using a slightly tilted laser scanner (Andersen et al., 2006a)
- an obstacle detection plug-in using laser scanner data
- an obstacle detection plug-in using stereo camera data
- a road detection plug-in using single camera data (Andersen et al., 2006b)
- a localization plug-in combining laser scanner and a-priori map (Tjell et al., 2008)
- a plug-in that implements a rule based mission scheduler
- a mission manager plug-in
- an obstacle avoidance plug-in based on visibility graph.

Data recorded from laser scanner, camera, GPS and robot pose can be replayed based on log files from each of the relevant plug-ins and synchronized by the server core. The replay ability eases development of especially perception plug-ins significantly.

An operator interface is available to interact with all servers and all plug-ins, and display also some of the more complex data structures. An example from the operator display is shown in Fig. 4.



```
odo  1197.90x, 1236.59y, 43.8deg 1755.31q
UTM 370576.63x, 6235086.83y, -15.9deg 11.1q
19 May 2006 11:45:42.227 (odo )
```

Fig. 4. Operator interface example, illustrating traversable road (green) and obstacles (purple) after treatment of the perception plug-ins.

7. APPLICATIONS

The MobotWare framework has been used with great success in education and several research and innovation projects. Using the plug-in structure of the real-time hardware daemon (RHD) seven different hardware platforms have been interfaced ranging from an 8 kg educational robot to the 7.5

tonne 240 bhp Claas Axion 840 tractor (Fig .6). This means that systems based on CAN-bus, Profi-bus, RS232, RS485 and USB have been interfaced demonstrating the versatility of RHD.

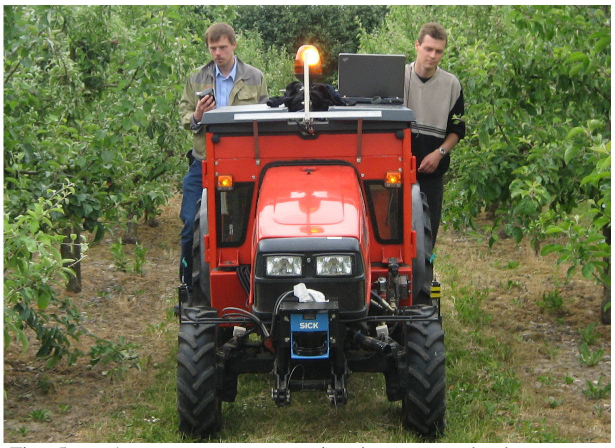


Fig. 5. Autonomous grass cutting between orchard tree rows using topological feature-based localization in tree rows through the MobotWare framework.

The scalability obtained by having modules communicating via sockets has been demonstrated by implementing a small system on an Atmel 130 MHz AVR32 NGW100 Linux evaluation board while vision intensive mobile robot systems have been implemented using several computers.

The inclusion of the robot control language in the system makes it programmable by non technicians and allows fast implementation of test solutions.
The efficiency on the language is seen in our basic robot course where the students have to make a C-solution and an SMR-CL solution to the same problem. They use 10 times as much time on the C-solution and score 30 to 40 % more points in the SMR-CL solution during the final competition.

In the innovation project "Safe + Reliable - Further Development of a Field Robot", the MobotWare framework was used to develop robust control of a field robot (Fig. 5). The system implements a localization system based on fusion of laser based tree row detection and gyro assisted odometry. Autonomous driving in an orchard with 10 rows thus including headland turns has been demonstrated. This shows that the modular plug-in structure allows implementation of complex robot control systems (Andersen et al., 2010).

Fig. 6. The Agrocom stereo vision equipped Claas AXION 840 runs MobotWare for autonomous field operations.

8. CONCLUSION

The mobile robot software framework presented in this paper, *MobotWare*, fulfils the requirements for a stable comprehensive framework capable of utilizing and controlling a large number of robot hardware platforms and sensors. MobotWare consists of three distinct layers, the RHD as the hardware abstraction layer, the MRC as the real-time robot controller and the AURS a server structure for quasi real-time perception and behaviour planning. The framework has been developed over a number of years in a university environment and been used in a large number of courses, projects and applications.

REFERENCES

Abbott D., "*Linux for Embedded and Real-time Applications"*, Elsevier Science, 2003.

Andersen N. A. & Ravn O., A Real-Time Control Language for Mobile Robots, *CIGR International Conference* , Beijing 11-14 oct. 2004.

Andersen J. C., Blas M. R., Andersen N. A., Ravn O., Blanke M., Traversable terrain classification for outdoor autonomous robots using single 2D laser scans, In: *Integrated Computer-Aided Engineering, vol: 13(3)*, p. 223-232, IOS Press, Amsterdam., 2006

Andersen J. C., Andersen N. A., Ravn O., Vision Assisted Laser Scanner Navigation for Autonomous Robots. In proceedings of the *10th International Symposium on Experimental Robotics* 2006 (ISER '06); 39, p. 111-120 Springer-Verlag Berlin, 2006.

Andersen J. C., Ravn O., Andersen N. A., Autonomous Rule-Based Robot Navigation in Orchards, *7th Symposium on Intelligent Autonomous Vehicles,* Lecce, September , 2010

Albus, J. S.; McCain, H. G. & Lumia, R., *NASA/NBS Standard Reference Model for telerobot control system architecture (NASREM)*, U.S. Dept. of Commerce, National Institute of Standards and Technology, Gaithersburg, MD :, 1989, iv, 76 p.

Beck A. B., Andersen N. A., Ravn O., Mission Management for Mobile Robots, *Proc. of the fourth Swedish Workshop for Autonomous Robotics*, p. 76-77, Västerås, Sweden, September 2009.

Bonasso, P. R.; Firby, J. R.; Gat, E.; Kortenkamp et al., Experiences with an architecture for intelligent, reactive agents, *Journal of Experimental and Theoretical Artificial Intelligence,* 1997, Vol.9 Issue.2-3, 237-256

Brooks, R. A., A Robust Layered Control System for a Mobile *Robot*, *IEEE Journal of Robotics and Automation*, 1986, RA-2, 14-23

Bruyninckx H., Open robot control software: The OROCOS project, *Proc. of IEEE Int. conf. on Robotics and Automation*, 2001.

Bruyninckx H., *"Real-Time and Embedded Guide",* K.U.Leuven, Leuven, 2002.

Coste-Maniere, E. & Simmons, R., Architecture, the backbone of robotic systems, *IEEE International Conference on Robotics and Automation (ICRA)*, 2000, Vol. 1, 67-72

Jørgensen R.N., Nørrremark M., Sørensen C.G., Andersen N.A. , Utilising scripting lanquage for unmanned automated guided vehicles operating within row crops, *Computers and Electronics in Agriculture, vol 62(2)*, p 190-203, 2008

Montemerlo M., Roy N., Thrun S.,Perspectives on Standardization in Mobile Robot Programming: The Carnegie Mellon Navigation (CARMEN) Toolkit, *Proc. Of the IEEE/RSJ Int. Conf. On Intelligent Robots and Systems* ,Las Vegas, Nevada, Oct., 2003.

Mantegazza P., Bianchi, E. , Dozioi, L. , et al., RTAI: Real-Time Application Interface, *Linux Journal*, issue 72, p. 142-150, 2000.

Nesnas, I. A., Simmons, R., Gaines, D. ,et al., CLARAty: Challenges and Steps Toward Reusable Robotic Software, *International Journal of Advanced Robotic Systems,* 2006, Vol. 3, pp. 023-030

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T. B., Leibs, J., Wheeler, R. & Ng, A. Y., ROS: an open-source Robot Operating System *International Conference on Robotics and Automation,* 2009

Tjell P., Hansen, S.. *"Laser based navigation in orchard"*. M.Sc. Thesis Technical University of Denmark, 2008.

Vaughan R.T., Gerkey B.P., Howard, On device abstractions for portable, reusable robot code *Proc. Of the IEEE/RSJ Int. Conf. On Intelligent Robots and Systems* ,Las Vegas, Nevada, Oct., 2003.