Open middleware for robotics

Molaletsa Namoshe[1*], N S Tlale[1], C M Kumile[2], G. Bright[3]

[1]Department of Material Science and Manufacturing, CSIR, Pretoria, South Africa,

mnamoshe@csir.co.za, ntlale@csir.co.za

[2]Department of Mechanical Engineering, Tshwane University of Technology, Pretoria, South Africa

kumilecm@tut.ac.za

[3]Department of Mechanical Engineering, University of KwaZulu-Natal, Durban, South Africa

brightg@ukzn.ac.za

***Abstract*-Despite advances in recent years, autonomous multi-robot systems remain classed as complex systems, because control and coordination of these systems remain a challenging task. Autonomous mobile robot houses heterogeneous sets of connected modular devices and are expected to communicate both synchronously and asynchronously. Robot complexities make the development of components for robot applications non-trivial and failure prone exercise. In trying to find a solution to the problem efficient modular interaction, robot software "Middleware" emerged. Middleware is software layer that provides an infrastructure for integration of applications and data in distributed systems domain.**

This article discusses freely available middleware for robotics and their technologies within the field of multi-robot systems to ease the difficulty of realizing robot applications. And lastly, an example of algorithm development for multi-robot co-operation using one of the discussed software architecture is presented.

Keywords – autonomous mobile robot, middleware, reusability, software architecture, robot application

I. INTRODUCTION

Despite considerable gains in mobile robotics in recent years, control and coordination of autonomous multi robot system remains a challenge. The difficulty in robotics system stems due to the following attributes: (i) Rapid change in sensors, actuators and computer technologies lead to increased investigation into new robot capabilities, e.g. new sensor and actuator systems imply that sophisticated signal processing algorithms are possible (ii)Robotic systems are inherently distributed, i.e. sensors and actuators are distributed over interconnected subsystems, and in multi-robot system the distribution scales up , (iii) Components or Processes need to interact in an efficient way, (vi) Autonomous robot system requires its sensing, action and processing ability to work in accord and (v) autonomous systems incorporate the use of algorithms such SLAM, Obstacle Avoidance, Navigation primitives, Vision processing algorithms just to name a few. These algorithms are angled towards solving problem faced by a robot operating in a physical environment, i.e. a robot capable of sophisticated decision-making as single system or in a team working to accomplish a task. The time and work required to come up with such a robot system is quite enormous. A robot programmer needs to be well informed in a number of engineering fields. Robotics domain covers various fields like signal processing, computer science, Vision, Electrical, artificial intelligence. This is not always possible, and leads to a scenario where individuals and institutions around the world concentrate on specific topic of robot system, and present their results and move to the next project.

The current trend in robotics research is working to change all that by the use of open robotic software/ *Middleware*. This is connectivity software that consists of a set of enabling services that allow multiple modular processes (modules/ components) running on board a robot or off board to interact across a network [1]. The software is a platform for components *reuse,* i.e. reusability of modules accepted as solved problem in robotics e.g. Extended Kalman filters. This 'plug and play' concept will open up new area in robotics where new standard need to be met by research.

Integrating this modules into one system is not an easy matter though, because these components are developed individually and have different network technologies, but need to communicate both asynchronous and synchronously. Middleware manages the iteration between robotic modules by abstract interfaces and transparent communication protocols to processes computing off board. The software allows programmers to concentrate on building sophisticated and smarter modular components and incorporate these into the system without upsetting the existing modules. This will lead to increased robot intelligence or multi robot system able to perform complex tasks.

The rest of the paper is structured as follows. Section II covers related work, i.e. looking at other robot software in robotic with similar attributes to middleware discussed in section III. Section IV discusses the advantages of using robot software. Section V discusses the criteria used to choose appropriate software. Section VI shows simulation results, followed by conclusion and future work in section 7. Acknowledgement and reference are in section 8 and 9 respectively.

II. RELATED WORK

TABLE I

ROBOT SOFTWARE CLASS

Middleware		**Discussed**
	Player	√
	Orca	√
	Miro	√
	OpenRDK	×
	Marie	√
	Smartsoft	×
	Orocos	√
Control architecture		
	ADE	×
	MCA	×
	ADE	×
Development Toolkit		
	TCA (IPC & TDL)	×
	CARMEN	√
	YARP	√

There are many robotic middleware, control architectures and development tool kits in use toady. Table I above shows freely available robotic software and classes them according to their distinct categories. The ticks indicate that software is briefly discussed in this article.

We are looking at open robotic middleware capable of implementing multi mobile robot system. This constraint eliminates Orocos middleware [2]. The Orocos project was developed specifically for industrial robots as platform for building modular framework for robots and machine control. The middleware is organized in to four libraries- i. the real-time toolkit (RTT), ii. Kinematic and Dynamics library (KDL), iii. The Bayesian Filtering library (BFL), iv. Orocos Component library (OCL). These libraries are implemented in C++.

Another important toolkit is CARMEN [3], it is one of the most extensively used software in robotic research. It offers distributed collection of modules organized as three layered architecture. The hardware management layer/ base layer govern components interaction and control as well as presenting abstract interfaces to base and sensor systems. Navigation primitives are handled by the Navigation layer while the top layer is for high level task implementation. Modules communicate with each other over IPC (Inter-process communication) protocol. This tool did not make the cut because it tailored for single robot systems.

Humanoid robot uses YARP (Yet Another Robot Platform), is a multi-platform open-source framework that supports distributed computation. The main design focus is on robot control and efficiency [4]. It provides a set of protocols and a C++ implementation for inter-process communication on a local network.

III. WHY OPEN MIDLEWARE?

Developing a middleware from scratch is an enormous task, because of the level of expertise required to come up with a flexible robotic software system. Therefore, a sensible alternative is to adopt one of the already existing open middleware for building robot applications. Making choice of which to use can be influenced by a number of factors. And these include things like robot software and existing hardware compatibility issues. Which languages are supported? What communication mechanism(s) are used? Are existing sensors and actuators supported? Is the software extendable? Ease of use, how easy is it to use and understand? Is real time implementation possible? What types of modules can be implemented using the middleware? etc.

Robot systems are too complex for one person to build and maintain software and hardware, therefore modules or components need to be modular. Modularity in robotics describes the loose tightness of coupling between components, and hence has the following advantages

- ***Extensibility***- Addition of newly developed or modification of components need not interfere with the existing processes**.**
- ***Reliability***- self contained components fail safe - if one module fail the rest of the system is spared.
- ***Network support***- Some system need high computational power to process a task, therefore a more logical thing is to distribute the load using off board computers, i.e. distributed computing.
- ***Maintainability*** – components are easy to improve because the changes do not affect the rest of the system.

IV. OPEN ROBOTIC FRAMEWORKS

A. Miro

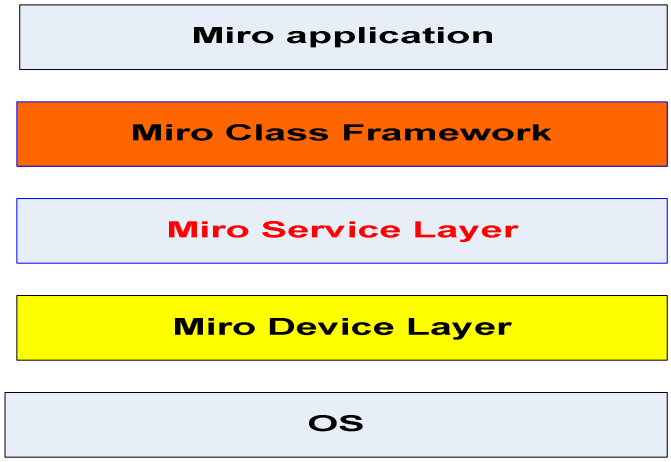

Figure 1. Miro architecture.

Miro [5] is an open, object-oriented robotic framework consisting of three layers as shown in figure 1 above. It has an underlying middleware called CORBA. CORBA standards allow for inter-process and cross-platform communication, thereby facilitating for distributed control architectures.

CORBA [11] has two main layers: ACE (Adaptive Communication Environment) and TAO (The ACE ORB). The ACE layer is linked to Miro Device layer to provide abstract interfaces to specific sensors and actuators of a robot. While the TAO CORBA and Miro Service Layer combination provides service abstraction to hardware devices via CORBA IDL (Interface Description Language). The third layer is the Class Framework which houses numerous robot control functional modules like mapping, localizer, Planner, and logging. All core functionalities are coded in C++ and hence achieve high runtime efficiency.

B. Marie

Marie [12] is a framework for integrating existing robotic components. The software offer decentralized integration management, and advocates for the ideas of code reuse of components from other developers, as well as integrating locally developed applications.

In order to handle distinct components developed independently, Marie follows a three layers abstraction approach.

- Core layer- consists of tools for communications, data handling, and OS related issues
- Component layer – It is also a management layer and built on top of the Core layer, consists of frameworks used to implement new components.
- Application layer –Is made up of necessary interaction tools to build applications using available components.

Marie architecturally adopts Mediator Design Pattern (MDP) depicted by figure 2 below. This is a centralized control unit which handles the arbitration between applications and interacts with these modules on direct basis. Communication protocols/ data management is decouples form components functionalities, implying that specialized components can be design irrespective of how data is sent or received.

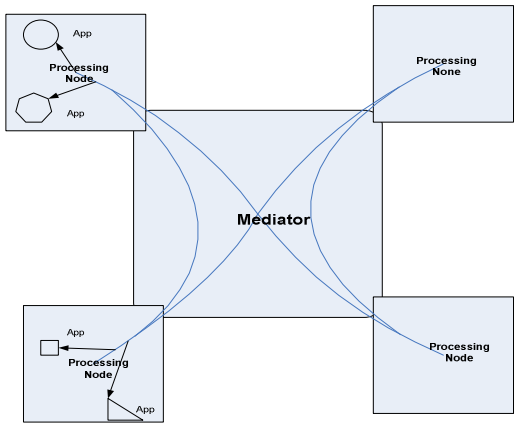

Figure. 2. Mediator design pattern for distributed system

C. Orca

Orca [8] is an open source framework (not architecture) for developing components based robotic systems. Components here are stand-alone process interacting with other components using clearly-defined interfaces. The framework does not impose any architectural constraints on the robot system. Orca is now on the second generation, called Orca 2 and is hosted by SourceForge [9]. The difference with the earlier version is the communication middleware. The first version used several transport mechanism like CORBA and some custom made communication libraries. These were ditched due to CORBA middleware complexity and some limitations presented by custom transport mechanism. The current version uses *Ice* (Internet Communication Engine) [10], [11]. It is a contract-based middleware similar in many respects to CORBA middleware. CORBA uses IDL for Interface implementation, while Ice relies on a similar specification language called *Slice*. The language is used to define communication contracts (interfaces) between components at runtime, i.e. an explicit description of services to be provided by other components to others. Ice components can communicate with each other regardless of the language of implementation. Supported languages include PHP, C++, Python, Java, C# etc.

D. Player

Player project [12], [13] is an open source software project tailored for robotic research. It provides an infrastructure for distributed access to a number of popular robotic hardware devices. Player run on many UNIX-like platforms, and is released as Free Software under the GNU General Public License. The software is implemented in C++ and uses POSIX threads or pthread interface for multi-threading.

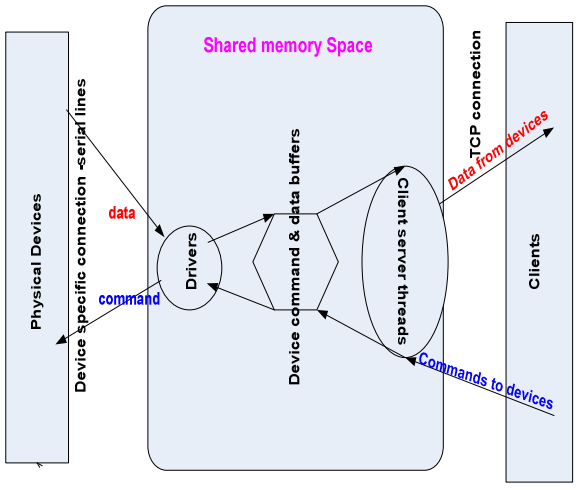

Figure 3. Player device server

Figure 3 above depicts the internal structure of player. When a client wishes to access a specific device, a command is sent to the command buffer. The command 'sit' waiting for the driver to read it form command buffer before a target device can be manipulated. In a similar vein, data from a device is written to a data buffer, where it waits for read command from a client thread. The client can run on an onboard computer or any computer that has network connectivity to the robot running player, i.e. distributed computing. Drivers have their own thread of execution and associated command and data buffer to channel information.

Player supports two kinds of communication mechanism, which are client- server communication and Device-Device communication. Inter-device communication is possible within a player server or between servers by pass-through device. Inter-client communication is possible but it is left to the programmer to figure it out.

V. CHOOSING A SUITABLE MIDDLEWARE

The choice of middleware to use in our robot system was influenced by factors mentioned in section 3. Considering this issues, Miro, Orca and Player jumps out of the list as one of the possible software to consider. Selecting which software best suit our need, we needed to look at the robot platforms already in our lab. We have a couple of ER1 robots [14] from Evolution robotics, Robotino robot [15] from Festo, and about two in house developed robots. Given these robot platforms, Player make a sensible choice because it is portable to both ER1 and Robotino robots, reducing the time needed to port the software.

Table 2 below shows a comparison between open robotic middleware discussed in this article.

TABLE II

SUMMARY OF OPEN ROBOTIC FRAMEWORKS

Middle-ware	**Arbitration Model**	**Components information sharing**	**Additional Tools**
Player	Threads, process	Client/ server	Logging, Remote inspection
Orocos	Call backs, threads	CORBA	Stage & Gazebo simulators
Carmen	Processes	IPC	Visualization, logging
Marie	Processes	Handles many	configuration GUI
Orca	Processes	ICE	Logging, remote inspection
Miro	Processes	CORBA	Logging
Yarp	Processes, Threads	Client, server	logging

As shown the table Miro uses CORBA for transport mechanism while Player and Orca rely on client/server and ICE model respectively.

CORBA is one of the most widely used communication software in robotics because it offers flexible inter-module and inter-robot communication. The main drawback with CORBA is that it is large and complex software. The complexity associated with this communication middleware violates one of the criteria mentioned in section 3 which is ease of use. Though, CORBA and Miro middleware combination seems to form a very stable system.

ICE is a modern implementation software similar many respects to CORBA. The difference being that ICE is a much smaller and has consistent API. Communication tasks in ICE are managed by a core library using a protocol that has options for compressing messages. Compressed data stream offer fast data transmission than uncompressed one. Furthermore there is also support for UDP (User Datagram Protocol) and TCP (Transmission Control Protocol). ICE can be implemented in a number of operating system flavors namely Linux, Windows, and MacOS X.

Player on the other hand uses client/server communication model. In this model, threads communicate through shared memory space. The space organized such that each device has a specific command and a data buffer. These buffers present an infrastructure for asynchronous communication channel between the read/write client threads and the device threads (drivers). When a client wishes to control a device through player, the following sequence is followed; first client reader thread receives a command from a client, then it writes the command into the command buffer for that specific device. The command will 'sit' in the command buffer until a device thread is ready to receive new command. The driver then reads the command from the command buffer and passes it to the target device. Data from devices follows a similar process but in this instant a device thread writes it into data buffer in the shared Global address space. This data is sent to clients by default at 10Hz but a programmer can changes the setting to meet the demands of clients.

Players' applications information sharing is not optimal because of the 'sitting' of data/ command in the buffers. The server also does not implement device logging mechanism, implying that multiple clients can concurrently write commands to a single command buffer. Since there is no queuing of commands, each time a device thread is ready to read it 'picks' what it finds in the command buffer irrespective of whether its old or new command. This presents a disadvantage since there is no guarantee that a command will reach target device since new command overwrites old command.

Looking at the concurrency models of the three frameworks one can conclude that ICE offers a better inter-module or inter-robot communication protocol. Orca though, has few readily developed robotic components and hardware and sensor support. Due to this lack, Orca is not selected. Player

then takes the nod because it supports a wide variety of sensors, actuators and robot platforms available in the market today. It is actively developed by both the public and developers, creating a good platform for components reuse. Player project in addition has got two simulation tools called Stage and Gazebo which are 2D and 3D environments respectively.

VI. PLAYER/ STAGE SIMMULATION RESULTS

The paper present simulation results of an ongoing project in multi-robot system. In the project a handful of robots explore the environment for hidden target. Control of robots in the system is achieved through the use of perception-action units, or behaviors. Each behavior is activated depending on feedback information from carefully selected sensor readings or message from a team member. Behaviors such as *ObstacleAvoidance, GoalsearchMode, GoalFoundMode, TargetHomingMode, Mapbuilding, CommunicationMode, pathplanning, and Pathfollowing* are used. *GoalsearchMode* mode has two other sub modes, *Initializer* and *Gobbler*. *Initializer* mode requires all robots in a team to giveaway their position and communication channel to other robots in the environment. *Gobbler* mode is used when robots are initially close knit, and it causes robots to spread quickly way from each other before normal search begins.

Figure 4 below shows three robots exploring and environment in a search mode. The robots in the simulation are able to locate a target (green square) but other behaviors such as communication and map building are not yet optimal. The path taken by a robot when *TargetHomingMode* is entered is currently implemented in MATLAB. A point of interest is tracked using a camera keeping it in view all the time. Range and bearing to the goal are evaluated and appropriate speeds are calculated to enable the robot stop at target position. The speed is proportional to range and it reduces as is goal is reached. Figure 5 shows a trajectory followed by a robot from an initial position to a target position.

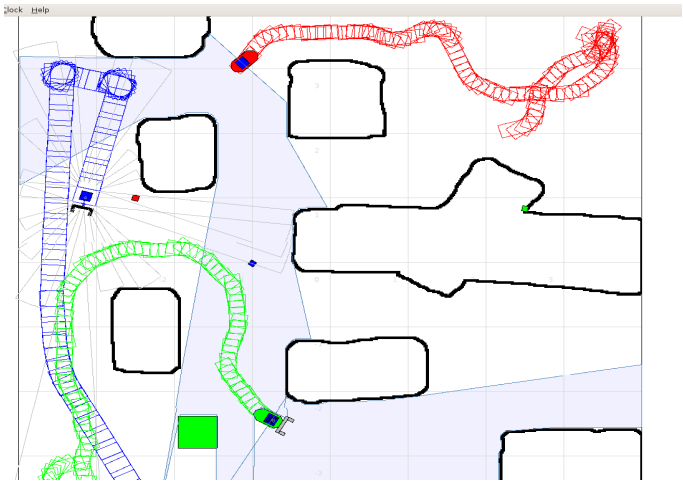

Figure 4. Three robots in GoalsearchMode

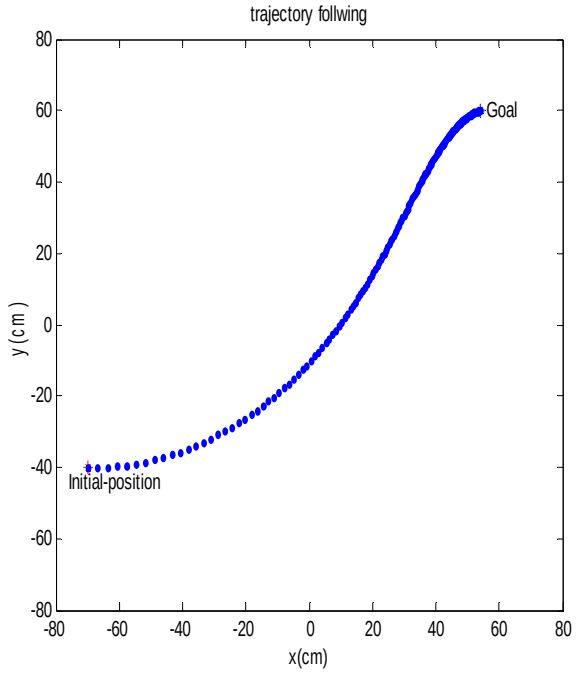

Figure 5. Trajectory following

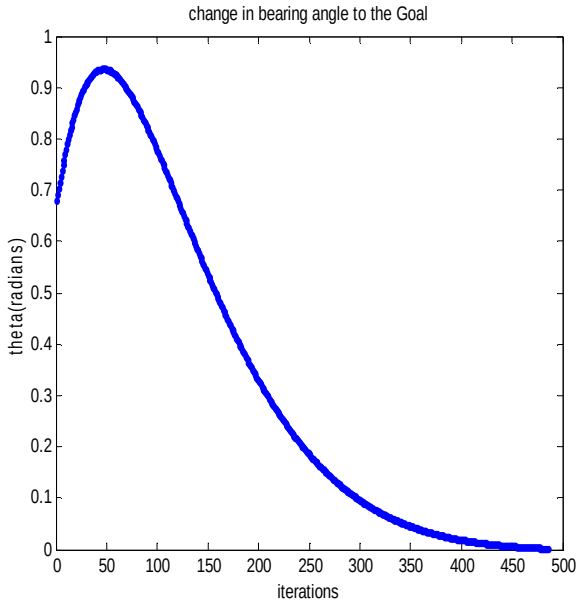

Figure 6 shows bearing angle to the goal

Figure 6 above shows how the bearing to the target changes as the robot moves towards a goal position. Initially bearing is around 0.68 radians but as the robot moves parallel to the target for a few seconds it sharply increases to above 0.9 radians. Then the robot start turning towards the goal position indicated in the figure by decline in bearing which eventually settles to zero, when robot and target are aligned.

VII. Conclusion & Future work

The paper briefly described a number of open source robotic framework comparing their concurrency model and their information sharing methods. ICE middleware seem attractive for implementing interfaces for components but Orca system supports a few sensory and actuatory devices. This constraint means time lost in actually building new components for robots used, than actually coding algorithms for robot application. Player uses client server model to access and control a number of popular robot bases. Its main drawback is the latency issues because commands 'sit' in their respective buffers waiting to be read. Having said that though, Player is easy to use and the client side can be implemented in any language with socket support.

The project was undertaken with no prior knowledge of how player work, hence some modules are still lagging behind in terms of implementation. The next step is to get modules mentioned in section four to function together by sharing the right information to accomplish the cooperative search task.

VIII. Acknowledgment

The author would like to thank the MMM robotic group for their questions and additions. This work was fully supported by Council for Scientific and Industrial Research (CSIR).

IX. References

[1] D. Krüger, I. Lil, N. Sünderhauf, R. Baumgartl, P. Protzel, "Using and Extending the Miro Middleware for Autonomous Robots," Towards Autonomous Robotic Systems (TAROS), Guildford, September 2006.

[2] H. Bruyninckx. "Open robot control software: the OROCOS project," *In IEEE International Conference on Robotics and Automation* (ICRA'01), vol. 3, pp 2523–2528, 2001. [Online: http: //www.orocos.org].

[3] M. Montemerlo, N. Roy, and S. Thrun. Perspectives on standardization in mobile robot programming: The Carnegie Mellon navigation (CARMEN) toolkit. In IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2436–2441, 2003. [Online: http:// carmen.sf.net].

[4] G. Metta, P. Fitzpatrick, and L. Natale. "YARP: yet another robot platform," *International Journal on Advanced Robotics Systems*, vol. 3, No. 1, pp. 43–48, 2006. [Online: http://yarp0.sf.net].

[5] Utz, H., Sablatnög, S., Enderle, S., Kraetzschmar, G., "Miro – Middleware for Mobile Robot Applications", vol. 18, No. 4, pp. 493-497, IEEE Transactions on Robotics and Automation, Aug 2002.

[6] Vinoski, S. "CORBA: integrating diverse applications within distributed heterogeneous environments," IEEE Communications vol. 35, No. 2, pp 46-55, Feb 1997.

[7] C. Cote, D. Letourneau, F. Michaud, and Y. Brosseau, "Robotic Software Integration Using MARIE," International Journal of Advanced Robotic Systems, vol. 3, No. 1, pp 55-60, March 2006[Online: http://marie.sf.net].

[8] G. Brooks, A., Kaupp, T., Makarenko, A., Oreback, A., and Williams, S. "Towards component-based robotics," In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, April 2005.

[9] Orca Project. Orca2. http://orca-robotics.sf.net.

[10] ZeroC,Inc. Ice performance, 2005. [Online: http://zeroc.com/performance].

[11] M. Henning and M. Spruiell. "Distributed Programming with Ice". 2006.

[12] Gerkey. B, Vaughan. R, & Howard. A, "The Player/Stage project: Tools for multi-robot and distributed sensor systems", *In Proceedings of the 11th international conference on advanced robotics*, pp. 317–323. Coimbra, Portugal, 2003.

[13] I. T.H.J. Collett, B.A. MacDonald, and B.P. Gerkey, "Player 2.0: Toward a practical robot programming framework". In Australasian Conference on Robotics and Automation (ACRA'05), December 2005. [Online: http:// playerstage.sf.net]..

[14] http://www.evolution.com/er1/ [accessed 29-08-2008].

[15] http://www.festo-didactic.com/int-en/learning-systems/new-robotino/ [accessed 23-08-2008].