A Generic Approach to Realtime Robot Control and Parallel Processing for Industrial Scenarios

Thomas Müller and Alois Knoll
Robotics and Embedded Systems
Technische Universität München
Blotzmannstr. 3, 85748 Garching, Germany
{muelleth,knoll}@cs.tum.edu

***Abstract*—In this paper, we introduce an intrinsically parallel framework striving for increased flexibility in development of robotic, computer vision, and machine intelligence applications. The primary goal is to provide a sound and easy-to-use, but yet efficient base architecture for complex sensor-based robotic systems with focus on industrial scenarios.**

The framework combines promising ideas of recent neuroscientific research with a blackboard information storage mechanism and an implementation of the multi-agent paradigm. Additionally, a generic set of tools for realtime data acquisition and robot control, integration of external software components, and user interaction is provided.

The paper is completed with a tutorial section showing how the building blocks afore described can be composed to applications of increasing complexity.

I. Introduction

These days, robots are common in industrial production setups. They accompany assembly lines all over the world, as they have interesting properties for production processes: they never tire, provide high accurarcy, and are able to work in environments not suitable for humans. Still, todays industrial robots are often limited to very specific tasks, as they have to be statically programmed ("teached") in advance.

But times change and as the production scenarios become more complex, industrial application tend to require greater flexibility. In the recent past prominent buzz-words in this context are *sensorimotor integration*, *visual servoing*, and *adaptive control*, to name but a view being investigated by academics. But how can industrial automation engineers apply all these promising new strategies, while still allowing for traditional approaches and moreover provide for convenient user interaction?

This is where the proposed flexible robotics framework, *FlexRF*, comes into play. The framework introduced in the following sections provides a generic, configurable, and interactive; but nevertheless sound and efficient foundation for such tasks.

Design principles, i.e., modularity and extendability, are combined with unique features like automatic widget generation and auto-parallelization. This allows for rapid development and deployment of complex applications for industrial realtime robotics. Furthermore, the framework's generic interfacing facilities enable support and integration of existing hardware and software components.

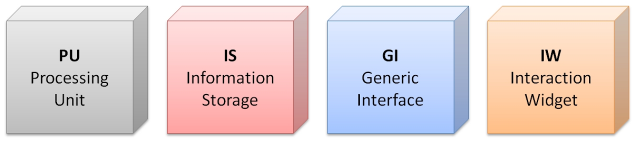

Fig. 1. Building blocks of the flexible robotics framework

Despite its simplistic design (i.e., the building-block structure, see Figure 1), it is still possible to map most recent state-of-the-art techniques (e.g., evolutionary mechanisms, parallel active perception systems, multi-agent approaches, and the like) onto the proposed architecture.

II. Related Work

This section investigates how the proposed framework fits into findings / results of existing recent research in related fields. Here, we find relevant sophisticated approaches primarily within the area of cognitive and blackboard architectures, or multi-agent systems.

Cognitive architectures originate from psychology and by definition try to integrate *all* findings from cognitve sciences into a general computational framework from which intelligence may arise. Multiple systems have been proposed to fulfill this requirement, including Act-R [1][2] and Soar [3].

Although these approaches may be biologically plausible and have the potential to form the foundation of some applications in reality, they all face the problem of being a mere scientific approach to cognition. We argue that, in order to develop applications also suitable for industrial automation, a framework for intelligent robotics and sensorimotor integration has to be designed keeping the application scope in mind. Thus, certain additional requirements emerge in the conceptual phase with high priority. Ease-of-use, generic interfacing and graphical interaction, as well as robustness and repeatability have to be taken into account here.

Still, the performance of biological cognitive systems is outstanding and thus, although we do not propose a cognitive architecture, the framework tries to integrate certain aspects where found useful and appropriate.

The principle theory considering **blackboard architectures** is based on the assumption, that a common database of knowledge, the "blackboard", is filled with such by a group of experts [4]. The goal is to solve a problem using contributions of multiple specialists. We adopt the basic ideas of this concept in the implementation of the information storage of the proposed framework (see Section V).

Nevertheless, one drawback with traditional blackboard systems is the single expert, i.e., a processing thread, that is activated by a control unit [5]. There is no strategy for concurrent data access in parallel scenarios. Futhermore, there is no means for training an expert over time, e.g., applying machine learning techniques, or even exchanging a contributor with another one in an evolutionary fashion. We deal with these shortcomings within the proposed framework and present our approach in Section IV and V.

Finally, a **multi-agent system** (MAS) is a system composed of a group of agents. According to a common definition by Wooldridge [6] an agent is a software entity being *autonomous*, acting without intervention from other agents or a human; *social*, interacting and communicating with other agents; *reactive*, perceiving the environment and acting on changes concerning the agent's task; and *proactive*, taking the initiative to reach a goal.

Most existing implementations (e.g., JADE [8]) use a communication paradigm based on FIPA's agent communication language [7], which is designed to send / broadcast text messages, but not complex data items. Thus we instead implement the blackboard paradigm for information exchange. Still, we acknowledge the above definition and the fact, that agents may concurrently work on a task and run in parallel. The processing units of our framework are hence implemented according to the MAS paradigms.

III. Creating Applications

From the perspective of modularity the proposed framework consists of the four building blocks from Figure 1). Any application developed with FlexRF is composed incrementally from these building blocks.

A typical FlexRF application (see Figure 2) is a loosely coupled group (depicted as a green cloud) of processing units (Section IV) exchanging data via the information storage (Section V). These units may or may not utilize generic interfaces (Section VI) to access hardware or external software components. Automatically, widgets allow for PU interaction with the user and visualization of feedback and processing results (Section VII).

Some challenging application examples and showcases that can now be easily implemented using the above building blocks are expatiated on in Section VIII.

IV. Processing Units

It was already in the late 19th centry, when Cajal discovered small processing entities in his groundbreaking research on microscopic brain structures [9]. These processing entities we find in the brain are called neurons and the human brain is physically enabled to perform enormously complex information processing since it contains a large net of such neurons. The neurons have two important properties: they work in parallel and they exchange information with each other.

The proposed framework is being inspired by this layout and thus provides entities that mimic these properties. These entities are called *Processing Units* (PUs). Each PU implements an thread-based interface, which instantly enables them to exploit the parallelization capabilities of the framework. The PU interface also provides an automatic gui-based configuration and feedback facility (see Section VII) and a possibility to share its information with others (see Section V).

As mentioned above, typically an application is a set of PUs using a common storage and user interface. But apart from a mere pool of units, any kind of inter-unit dependency or hierarchical structure, e.g., a chain, a tree-like structure, a fully connected or directed graph, or even layers of PU-sets can be introduced implicitly.

In a standard application the dependency structure is introduced by the programmer at compile-time. Conveniently, in this case a programmer only needs to register the data types that shall be exchanged, implement the PU's `run()` method accordingly, and call it when the PU is intended to start.

But, considering more advanced scenarios, like within the field of autonomous robotics, parallel vision or machine learning, higher flexibility may be useful. Thus the framework provides for some interesting runtime features: PUs need not neccessarily be known at compile-time nor the data types to be exchanged with other PUs. On the contrary, through user interaction or triggered by some other PU, the application is able to extend itself, at any time and according to the specific task requirements. Therefore a unit structure or dependency graph can also emerge from the system during runtime.

In this fashion, units can for example implement machine learning techniques, e.g., reinforcement learning [14], support vector machines [15], or evolutionary algorithms, to perform on a non-trivial cognitive task, such as controlling a robot or classifiying objects in a vision system.

Below, Section VIII will elaborate on this in greater detail and describe a demo application

V. Information Storage

PUs almost inevitably need to exchange data with each other, be it to distribute processing results, to share information about processing tasks, create a common knowledge base or to perform cooperative / competitive computations. Thus, as a second building block, FlexRF defines a generic *Information Storage* (IS) component.

A. Asynchronous Communication

Since the framework covers intrinsically parallel scenarios, one must generally consider either synchronous or asynchronous communication strategies for data exchange between processes or threads [16]. With synchronous communication the partners wait for confirmation of sent data items, so this strategy has its main applications where the correct transmission of data is essential. Though being robust, due to its blocking nature a synchronous approach can cause problems

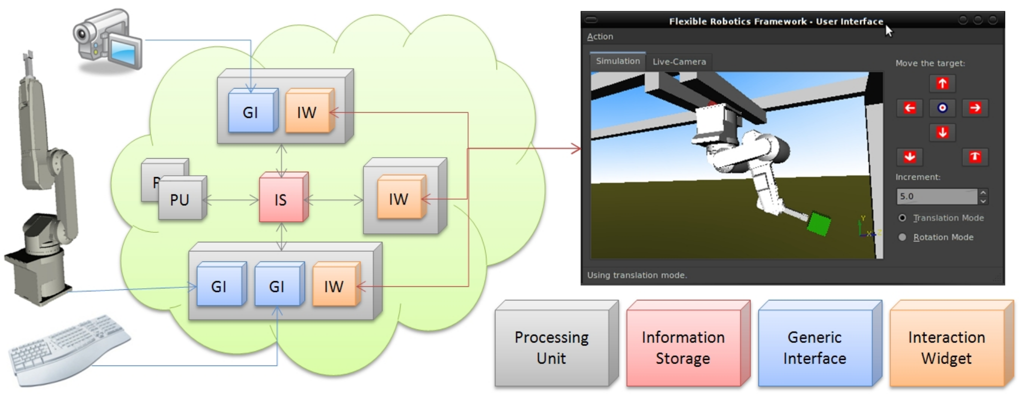

Fig. 2. A moderately complex sample application composed from the building blocks

especially for realtime systems where immediate responses have to be guaranteed.

For this case asynchronous "non-blocking" communication mechanisms (ACM) have been proposed. With ACMs information or data is dropped when capacities exceed – which is acceptable as long as the system does not block.

Non-blocking algorithms can be distinguished as being *lock-free* and *wait-free* [17]. Lock-free implementations guarantee at least one process to continue at any time – though starvation is a risk, because an operation may never finish due to the progress of others. On the other hand wait-free implementations avoid starvation, as they guarantee completion of a task in a limited number of steps [18]. Generally one can claim it essential for systems utilizing an ACM to remain in a responsive state, not to guarantee data transmission.

B. *Event Driven Communication*

The information storage implemented for the proposed framework takes the advantages of an ACM and combines them with an event driven signaling mechanism borrowed from Nokia's Qt framework [12].

In the Qt framework, communication is enabled by signaling a new event within the application. A component, e.g., a user interface, may define method stubs as being a `signal`. The signal can then be emitted whenever an event relevant for other components occurs, e.g., a mouse-click in the GUI.

Other communication partners (components of the application) implement listeners as they register themselves for this signal by defining a `slot`. The slot must have the same signature as the signal. The event passing and integrity checking is then performed by Qt and the only thing remaining to do for a developer, is implementing the reaction on the event in the slot function.

C. *A Combined Approach*

While the data storage requests are equal for both communication strategies, considering greatest possible flexibility, FlexRF's information storage supports both modalities for providing data for PUs: purely asynchronous retrieval and signal subscription.

Figure 3 shows the workflow for data registration, storage, and retrieval modalities as a diagram.

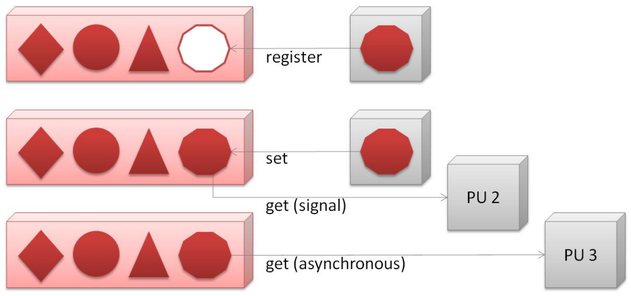

Fig. 3. Dataflow for a data item in the information storage

There is one disadvantage for implementations of the ascynchronous retrieval that has not been mentioned before: PUs might request data from the information storage, but there is no relevant data at the moment. In this case the storage still delivers an item, the NULL item, so units that perform asynchronous requests have to deal with these situations as well.

Clearly, these situations never occur, if all PUs within an application only implement listeners on the new-data-signal. Still there are situations, where signaling might not be desired, as it decreases overall application performance and thus the asynchronous retrieval seems more efficient. Consider for example a vision application, where a PU connected to the camera stores images in realtime. A complex vision PU being signaled every single image might be slower than the camera PU and it may not be appropriate to process every single frame, but in deed always the most current one. In that case, the vision PU obviously needs to utilize the asynchronous retrieval mechanism for the image frames.

D. *Practical Issues*

To conclude this section, we have to discuss instantiation, error handling, and thread safety of the proposed information storage in a user application.

As mentioned above, the information storage is generic in the sense of accepting arbitrary data item after having the data type registered. To facilitate the initialization and provide convenient access within all components of the framework, the IS is based on the *singleton* design pattern [19]. Organizing the singleton instance in a threadsafe manner concerning read and write accesses then ensures integrity and consistency. Therefore, after registration of a data type on first usage (read or write access), the item is embedded in a thread safe container, which encapsulates all access requests in a cascaded lock-unlock mechanism (using mutual exclusions).

Error handling is implemented straight forward, as the storage delivers the NULL data item whenever an erroneous request was received, i.e., the data type was not registered or no suitable data item could be found. Note, that this error handling approach is wait-free, because it completes in a limited number of steps [20] and thus generally avoids the occurrence of dead locks.

VI. Generic Interfaces

The third base component of FlexRF, the *Generic Interface* (GI) abstraction, is deduced from the following considerations: PUs are generally designed to operate or compute on data in some way. But how does data pour into the system, i.e., how can data acquisition or execution of external processes be realized and how can data be transformed into real-world actions?

The proposed framework therefore provides an easy-to-use interface abstraction. The interfaces enable convenient access to external hardware components such as image sensors, user input devices (mouse, keyboard, etc.), or other sensory devices, e.g. force-torque sensors.

There is more a developer can do with the generic interfacing mechanism: most important, implementing a generic interface is not limited to accessing input devices, but in deed one can write an interface to virtually any external component, be it software or hardware. For instance considering software libraries, at the time of publication, FlexRF already interfaces to the robust model-based realtime tracking library *OpenTL* [10] and the library underlying the efficient EET (exploring / exploiting tree) planner [11] for advanced industrial robot control.

Furthermore, it is relevant for most today's real-world applications in a robotic scenario, to access output devices such as grippers or servos; realtime connectors for industrial robot control; or socket connectors for remote control, data exchange, and remote procedure calls. Interfaces for these scenarios are thus supplied by the framework, as well as an interface for running arbitrary executables. Reading the output of those is facilitated in order to enable PUs to seamlessly integrate with external applications.

Another powerful feature of generic interfaces is their integration with PUs and the automatic configuration facilities described in the following.

VII. Interaction Widgets

As a fourth building block, FlexRF provides a graphical user interaction and visualization facility. As each PU and GI may define a corresponding *Interaction Widget* (IW), a high-level user interface can be composed easily from these widgets. This is possible, because IWs inherit from `QWidget`, which is part of the Qt framework.

IWs enable the user to textually or graphically inspect results the PUs or GIs produce and to alter their parametrization. For processing units the proposed framework provides a main window. The main window handles the initialization of Qt (it essentially creates a customized `QApplication`). Then, to display the units configuration parameters and feedback values, the PU only has to be added to the main window. Also, a menu bar entry is created automatically, if specified so.

The automatic widget generation mechanism is possible, because PUs and GIs both implement an abstract parameter container. This parameter container may store arbitrary data. In general three types of parameters exist in a container: *feedback*, *configuration* and *runtime* parameters.

Feedback parameters state the result of operations, e.g., joint-feedback of a robot, status of gripper, or an image from a camera interface.

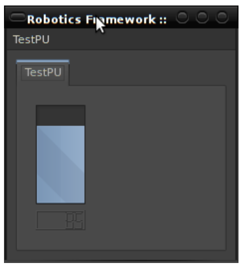

Fig. 4. A widget with display components for an integer feedback parameter.

At the time of development the parameters are added to a local parameter vector and the feedback values are automatically connected to the interaction widget at start-up. Whenever the value of a parameter changes, the IW displays these changes. Figure 4 shows the IW automatically generated for a single integer parameter.

Configuration parameters are loaded from an XML-file at system start up and a PU or GI can register on a certain value, e.g., a camera-interface might register on the input image size, color format or camera device. The framework then automatically creates controls in the corresponding widget for these parameters.

Specific to configuration parameters is, that they may only be altered, if a device (PU or GI) has not been initialized yet. Whenever the configuration is changed, the registered PUs or GIs have to be re-initialized to keep the system in a consistent state.

Similar to the mechanism for configuration parameters, the framework produces controls for **runtime parameters** automatically. But opposed to those, runtime parameter are the basis of direct interaction with the user.

Left to say is, that in spite of control generation being automated, a developer of course may introduce new custom or adjust the existing controls according to application specific requirements.

VIII. Application Examples and Showcases

A. A Simple Example

In a very basic, single threaded application example, the PU shown in Figure 5 is designed for processing data from and sending data to the robot and interaction with a PC keyboard.

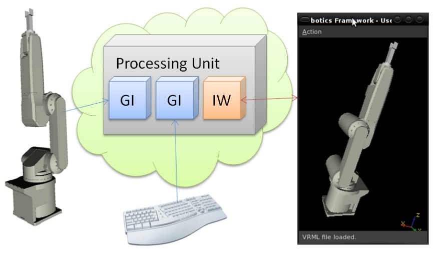

Fig. 5. Application scheme for a simple keyboard controlled robot

The actual hardware components are accessed utilizing the generic interfaces. The PU requests feedback from the robot interface and the keyboard interface in a endless while-loop. Then, corresponding to the key pressed by the user, a robot action is computed and sent via the robot interface to the actual hardware.

No communication with other PUs is implemented here, so the application does not need to initialize an information storage. A widget provides for feedback visualization of robot movements. This widget is statically embedded in a Qt-based graphical user interface, the application main window discussed above.

Clearly, integration of all hardware interfaces and computations into a single PU is non-optimal considering load distribution and parallelization. Therefore, the next example shows, how performance of such an application can be improved using multiple PUs and the information storage.

B. Multiple PUs and Information Sharing

Using the IS to share and exchange processing data enables the application developer to decompose a problem into semantically independent tasks and distribute them to multiple task-specific PUs.

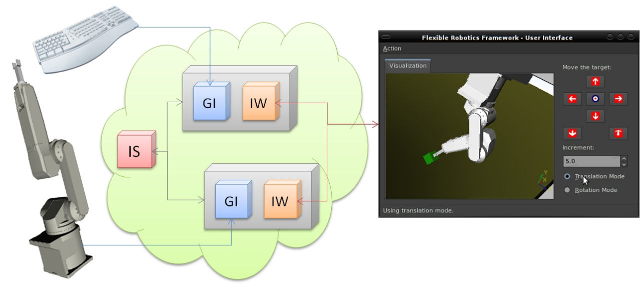

Fig. 6. Multithreaded application for keyboard control of a robot

Figure 6 shows a control system, where keyboard / mouse events are handled by the GI of the upper PU, also offering some configuration options in the interaction widget (shown in the right part of the screenshot). The PU in this case posts a data item containing the user's selection for the new target position of the robot's end-effector. The robot control unit implements a listener on that new target position. It computes a trajectory to move the robot to the target and sends the corresponding commands via the robot GI.

Note, that in the example the listener is not implemented event based (i.e., it does not implement a slot to the new target signal), but in an asynchronous manner. This ensures, that the robot smoothly completes a motion. Only after the movement has finished the next trajectory may be computed.

C. A Complex Visual Servoing System

Figure 7 shows a rather complex sensorimotor example application, where most previously described concepts and building-blocks of the flexible robotics framework are applied.

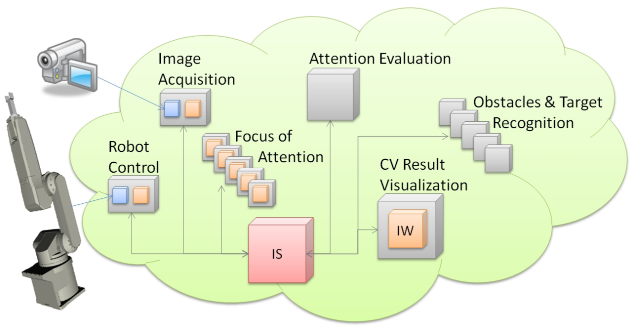

Fig. 7. Scheme for an evolutionary visual servoing application

The application features a realtime robot control unit sending new position commands every seventh millisecond and simultaneously sharing position feedback information. The interaction widget provides for visual feedback and additional manual robot control through the GUI. Another IO unit, the image acquisition unit, connects to the camera, retrieves new images, and stores them in the IS.

In the application, each of the focus of attention (FoA) units implements the attention condensation mechanism [21]. This approach speeds up visual processing by performing a relevance evaluation on the visual field as it creates regions of interest for salient areas appropriately. Saliency is expressed using a multivariate rating function, thus an evolutionary mechanism can be applied to optimize the performance, i.e., by mutating the parameters of the function.

The attention evaluation unit shown in the figure realizes a high-level cognitive component used for cloning, mutations, and deletions of FoA units. Performance of a FoA unit can be judged on by checking, how many of the created regions actually contained relevant objects, i.e., wherein the recognition units found either a target or obstacle.

The set of PUs for object recognition and tracking runs in parallel. A unit in this set asynchronically requests a region of interest from the IS for object detection. If there is no relevant region, a suspension time for a PU is calculated according the optimal back-off algorithm [22]. In this way, a recognition unit tries to optimally estimate the suspension time until the next relevant data item can be delivered.

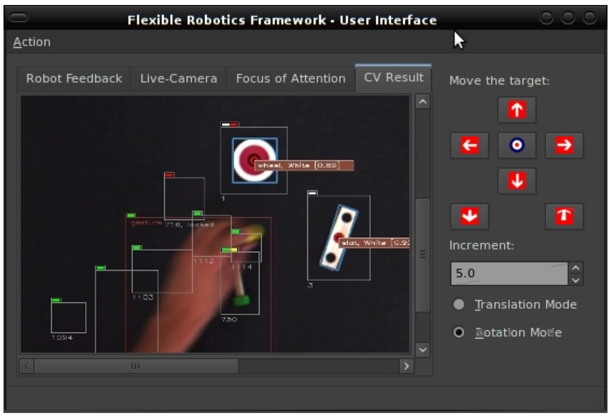

Fig. 8. The GUI for the visual servoing system

The graphical user interface for the application shown in Figure 8 is, as in the examples above, a composition of interaction widgets. The robot control buttons (red arrows) are created from runtime parameters and their location in the widget is customized. Also, the robot control PU has custom widget for the robot's joint-feedback, a 3D display of the robotic setup.

Imagine investigating the information flowing in the system on user interaction: whenever the user presses a button, the robot PU's runtime parameters are changed accordingly. The change-event then causes the PU to compute a new target pose for the robot's endeffector. As the joint-feedback of the robot interface is connected to the robot PUs feedback parameter vector, a change their causes the feedback widget to adjust the robot in the 3D view.

IX. Conclusion and Future Work

The paper introduces a flexible framework for parallel, asynchronous and decoupled processsing on recent hardware architectures like multicore or multiprocessor systems.

It is shown, that the typical modular application is an unordered set of instances of the building blocks. The set may comprise a structure of processing units; a blackboard for storage and exchange of data; a graphical interfacing and configuration machanism; and a magnitude of interfaces to hardware or external software components.

Future improvements include the implementation of the *Message Passing Interface* standard [23]. This extends the parallelization capabilities, as it allows for automatic load distribution on multi-platform architectures, such as computer clusters or grids. Another future feature is a possibility to priorize certain PUs in favor of others. This may eventually be useful for industrial applications where safety critical components for human-robot interaction have to be implemented.

Yet, as the design concepts of the framework are mature and especially the user application interfaces remain stable, existing applications may be easily transferred to future versions of FlexRF.

Acknowledgment

This work is supported by the German Research Foundation (DFG) within the Collaborative Research Center SFB 453 on "High-Fidelity Telepresence and Teleaction".

References

[1] A. Newell, *Unified Theories of Cognition*. Harvard University Press, 1994.
[2] J. R. Anderson, *How Can the Human Mind Occur in the Physical Universe?* Oxford University Press, 2007.
[3] J. F. Lehman, J. Laird, and P. Rosenbloom, "A Gentle Introduction to Soar, an Architecture for Human Cognition," 14-Sept-2009, http://ai.eecs.umich.edu/soar/sitemaker/docs/misc/GentleIntroduction-2006.pdf.
[4] L. D. Erman, F. Hayes-Roth, V. R. Lesser, and D. R. Reddy, "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty," *Computing Surveys*, vol. 12, no. 2, pp. 213–253, 1980.
[5] D. D. Corkill, "Collaborating Software: Blackboard and Multi-Agent Systems & the Future," in *Proc. of the International Lisp Conference*, 2003.
[6] M. J. Wooldridge, *An Introduction to MultiAgent Systems*, 2nd ed. John Wiley Sons, 2009.
[7] "FIPA ACL Message Structure Specification," Foundation for Intelligent Physical Agents, Tech. Rep., 2002.
[8] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa, "JADE: A White Paper," Telecom Italia Lab and Universita degli Studi di Parma, Tech. Rep., 2003.
[9] R. y Cajal, *Comparative Study of the Sensory Areas of the Human Cortex*. Harvard University, 1899.
[10] G. Panin, C. Lenz, S. Nair, E. Roth, M. in Wojtczyk, T. Friedlhuber, and A. Knoll, "A Unifying Software Architecture for Model-based Visual Tracking," in *Proc. of the 20th Annual Symposium of Electronic Imaging*, 2008.
[11] M. Rickert, O. Brock, and A. Knoll, "Balancing Exploration and Exploitation in Motion Planning," in *Proc. of the IEEE International Conference on Robotics and Automation*, 2008.
[12] Nokia Corporation, "Online Reference Documentation," 12-Sept-2009, http://doc.trolltech.com/.
[13] R. Pfeifer and J. Bongard, *How the Body Shapes the Way We Think*. MIT Press, 2005.
[14] L. P. Kaelbling, M. L. Litman, and A. W. Moore, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237 – 285, 1996.
[15] B. Scholkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2001.
[16] D. E. Culler, J. P. Singh, and A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishers, 1999.
[17] H. Sundell and P. Tsigas, "Fast and lock-free concurrent priority queues for multi-thread systems," in *Proc. of the International Parallel and Distributed Processing Symposium*, 2003.
[18] M. Herlihy, "Wait-free synchronization," *ACM Transactions on Programming Languages and Systems*, vol. 13, no. 1, pp. 124–149, 1991.
[19] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Orientated Software*. Addison-Wesley Professional Computing Series, 1998.
[20] M. P. Herlihy, "Impossibility and Universality Results for Wait-Free Synchronization," in *Proc. of the 7th Annual ACM Symposium on Principles of Distributed Computing*, 1988, pp. 276–290.
[21] T. Müller and A. Knoll, "Attention Driven Visual Processing for an Interactive Dialog Robot," in *Proc. of the 24th ACM Symposium on Applied Computing*, 2009.
[22] T. Müller, P. Ziaie, and A. Knoll, "A Wait-Free Realtime System for Optimal Distribution of Vision Tasks on Multicore Architectures," in *Proc. of the 5th International Conference on Informatics in Control, Automation and Robotics*, 2008.
[23] "MPI, A Message-Passing Interface Standard," University of Tennessee, Knoxville, Tennessee, Tech. Rep., June 1995.