A FLEXIBLE ROBOTICS AND AUTOMATION SYSTEM
Parallel Visual Processing, Realtime Actuator Control and Task Automation for Limp Object Handling

Thomas Müller, Binh An Tran and Alois Knoll
Robotics and Embedded Systems, Faculty of Informatics, Technische Universität München
Blotzmannstr. 3, 85748 Garching, Germany

Keywords: Flexible automation, Parallel processing, Realtime actuator control, Limp object handling.

Abstract: In this paper, an intrinsically parallel framework striving for increased flexibility in development of robotic, computer vision, and machine intelligence applications is introduced. The framework comprises a generic set of tools for realtime data acquisition, robot control, integration of external software components and task automation. The primary goal is to provide a developer- and user-friendly, but yet efficient base architecture for complex AI system implementations, be it for research, educational, or industrial purposes. The system therefore combines promising ideas of recent neuroscientific research with a blackboard information storage mechanism, an implementation of the multi-agent paradigm, and graphical user interaction.
Furthermore, the paper elaborates on how the framework's building blocks can be composed to applications of increasing complexity. The final target application includes parallel image processing, actuator control, and reasoning to handle limp objects and automate handling-tasks within dynamic scenarios.

1 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 accuracy, and are able to work in environments not suitable for humans. Still, todays industrial robots are often limited to very specific repetitive tasks, as they must be statically programmed ("teached") in advance. But industrial applications nowadays tend to require greater flexibility, as the manufactured products become highly customized and thus the production scenarios become more complex. Also, automation engineers strive to advance to production fields like handling of limp or deformable objects, that have not been considered before. In this context sensorimotor integration, visual servoing, and adaptive control are some of the most prominent buzz-words being investigated in the recent past by academics.

The flexible robotics and automation framework presented in Section 3 refers to these topics and provides a generic, configurable, and interactive; but nevertheless sound and efficient foundation for such tasks. Section 4 then shows, how the proposed framework can be used to build an application for robot-assisted, semi-automated limp object handling.

2 RELATED WORK

This section elaborates on how the proposed framework fits into findings / results of existing recent research in related fields. Furthermore, common robotics frameworks are mentioned and their relation to the presented system is discussed.

2.1 Related Research

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 framework from which intelligence may arise. Multiple systems have been proposed to fulfill this requirement, including Act-R (Newell, 1994; Anderson, 2007) and Soar (Lehman et al., 2006). 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

Müller T., An Tran B. and Knoll A. (2010).
A FLEXIBLE ROBOTICS AND AUTOMATION SYSTEM - Parallel Visual Processing, Realtime Actuator Control and Task Automation for Limp Object Handling.
In *Proceedings of the 7th International Conference on Informatics in Control, Automation and Robotics*, pages 289-294
DOI: 10.5220/0002951002890294

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 this scope in mind. Furhtermore, additional requirements like robustness and repeatability, generic interfacing, user-friendlyness and graphical interaction have to be taken into account with high priority. Still, we are impressed by the incredible performance of biological cognitive systems and, although we do not propose a cognitive architecture, try to integrate certain aspects where we find them 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 (Erman et al., 1980). 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 3.3). Nevertheless, a drawback with traditional blackboard systems is the single expert, i.e., a processing thread, that is activated by a control unit (Corkill, 2003). There is no strategy for concurrent data access in parallel scenarios. Futhermore, there is no space 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 3.2 and 3.3.

Finally, a **multi-agent system** (MAS) is a system composed of a group of agents. According to a common definition by Wooldridge (Wooldridge, 2009) 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 (Bellifemine et al., 2003)) use a communication paradigm based on FIPA's agent communication language (FIPA, 2002), which is designed to exchange text messages, not complex data items. Thus we instead implement the blackboard paradigm which is capable of maintenance of complex items. 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 these MAS paradigms.

2.2 Related Systems

The following paragraphs discuss some of the most widespread robotics frameworks with respect to applicability for vision-based limp object handling and automation in the target scenario.

Orca[1] adopts a component-based software engineering approach without applying any additional architectural constraints. The framework is open-source, but uses the commercial Internet Communications Engine (ICE)[2] for (distributed) communication and definition of programming language independent interfaces. ICE provides a Java-based graphical user interaction/control instance, but capabilities to incrementally compose applications by connecting the executable nodes at runtime is very limited. One may start/stop nodes, but the communication is established using a publish/subscribe mechanism implicitly by implementing the corresponding interfaces. So while providing facilities for efficient distribution of tasks on multiple computers, Orca lacks the required flexibility when trying to automate tasks without time-consuming reprogramming (runtime reconfiguration).

The **Robot Operating System**[3] (ROS) is another open-source software framework striving to provide a (robot-)platform independent operating system. The framework uses peer-to-peer technology to connect multiple executables (nodes) at runtime in a publish/subscribe way. A master module is instantiated for the required node-lookup. An interesting feature of ROS is the logging and playback functionality (Quigley et al., 2009) allowing for replication of recorded data-streams for later usage, e.g., in a simulation environment. Still, the framework lacks a facility to record an application configuration in order to conveniently replicate an application setup for a specific task, e.g., load a pick- and place task description and execute it on various robotic setups. **Microsoft's Robotics Studio**[4], released in 2007, is a Microsoft Windows specific .NET-based software library for robotic software applications. The framwork utilizes the Coordination and Concurrency Runtime (CCR) and the Decentralized Software Services Protocol (DSSP) for message processing and passing. It includes a simulation environment and, most notably, a convenient facility to develop applications in a user-friendly graphical way. Still, it is closed source, and in addition to that, "not an ideal platform for real-time systems" (Jackson, 2007), since there is no guarantee, that a service is not interrupted.

[1] http://orca-robotics.sourceforge.net
[2] http://www.zeroc.com
[3] http://ros.sourceforge.net
[4] http://www.microsoft.com/Robotics

The open-source **Player/Stage**[5] project provides another popular and widespread framework for robotics. Here, a system typically comprises a "player", which implements a TCP-server on the robot, and "clients", the user control programs, such as a joystick controller or a keyboard device, that may connect to the server, send messages, and control the robot (Vaughan and Gerkey, 2007). Although being powerful, in the eye of the authors, Player/Stage is rather complex to setup and application development is not user-friendly. I.e., one needs to implement a character device and a interface/driver model atop the Player Abstract Device Interface (PADI), combine it with the Player Protocol and finally implement the application semantics, even for a simple pick- and place task. The flexible robotics and automation system introduced in this paper strives to overcome the drawbacks of the aforementioned systems. Furthermore, it incorporates the useful features of the above architectures to form a foundation for user-friendly limp object handling applications.

3 SYSTEM OVERVIEW

From a software engineering perspective, an application built atop of the proposed system is built from three building blocks shown in Figure 1.

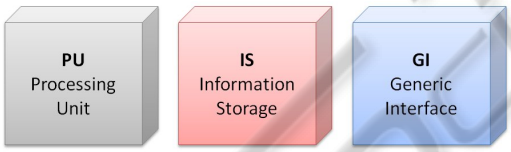

Figure 1: Building blocks of the flexible automation system.

These building blocks, namely the *information storage*, *processing units*, and *generic interfaces*, are introduced briefly in the following paragraphs.

3.1 Generic Interfaces (GI)

The first building block provides an easy-to-use interface abstraction for accessing external hardware components such as the realtime connectors for robot control, grippers and servos, or image sensors, user IO-devices (mouse, keyboard, etc.), or other sensory devices, e.g. force-torque sensors (FTS).

But implementing a generic interface is not limited to IO-devices, but indeed 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, the framework already provides predefined interfaces to the robust model-based realtime tracking library *OpenTL* (Panin et al., 2008) and the library underlying the efficient EET (exploring / exploiting tree) planner (Rickert et al., 2008) for advanced industrial robot control. Additionally, in order to support seamless integration with external applications, interfaces for accessing socket connectors for remote control, data exchange, and remote procedure calls are supplied by the framework, as well as an interface for running arbitrary executables.

[5] http://playerstage.sourceforge.net

3.2 Processing Units (PU)

The base class for processing data, the PU, provides a configuration, control and feedback facility and a possibility to share information with others. Furthermore, each processing unit is designed as a thread and supplies a description of the action it performs to the automation system (see Section 3.4).

Typically, an application comprises a set of PUs, where PUs perform their action in parallel, either continuously or they are triggered by a start event. While most hardware interfaces need a cyclic, continuous update / retrieval (such as the robot joint values or the camera interface), higher-level actions wrapped into a PU, e.g. moving the end-effector from A to B, handing a workpiece over to the next robot, or finding a grasping point in a visual scene, most commonly need a trigger-event in an assembly workflow.

3.3 Information Storage (IS)

In order to map a complex assembly workflow, exchanging data between PUs is essential. For example the input device unit maps to the target pose generation unit, which again maps to the joint values corresponding to a robotic end-effector.

The singleton information storage supplied by the framework is the building block designed for this purpose. Figure 2 shows the workflow for data registration, storage, and retrieval modalities as a diagram.

As shown in the figure, after registering a data-item, synchronous and asynchronous data access is possible, i.e. PUs can either listen for the event generated, when a data item has changed in the storage, or poll the data only when needed.

3.4 Task Automation

Tasks are generally described as a set of connected actions, where each processing unit defines a such action. In order to combine actions to a complex task,

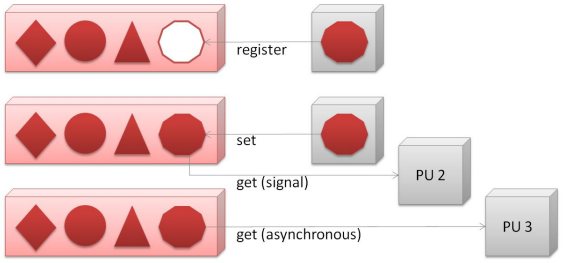

Figure 2: Dataflow for a data item in the information storage.

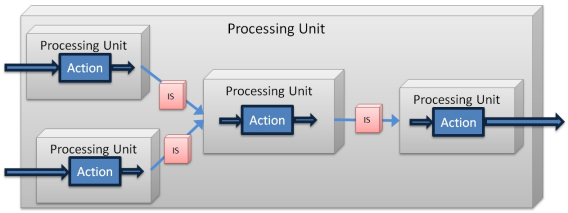

Figure 3: Processing units can be generated online to combine a set of arbitrary actions into a more complex one for automation.

a generic action description comprising an identifier, IO and configuration parameters, etc., was developed. Utilizing this action description, it is possible to specify a complex task by combination of primitive actions (e.g., operations like “grasp”, “screw”, “move-to”).

Furthermore, the framework supplies an automation unit, which enables a user to transform a task record into an action and generate a new processing unit (see Figure 3) from it. Increasingly complex tasks can be automated, as these composite actions are provided to the user interface for further combination. Thus it is easy to quickly and conveniently adapt, e.g., a production system to a novel product variant requiring different assembly steps.

4 TOWARDS LIMP OBJECT HANDLING

The following paragraphs show step-by-step, how the application for limp object handling is implemented using the framework’s building blocks.

4.1 A Simple Example

In a very basic, single threaded application example, the PU shown in Figure 4 is designed for processing data from and sending data to the robot and interaction with a PC keyboard.

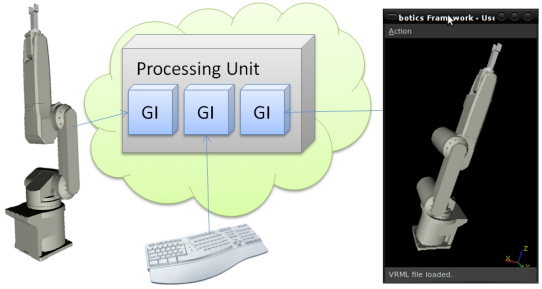

Figure 4: Application scheme for a simple keyboard controlled robot.

The actual hardware components are accessed utilizing the generic interfaces. The PU polls the robot interface and the keyboard interface continuously in an 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. Finally, a widget provides for feedback visualization of robot movements. This widget is statically embedded in a Qt-based graphical user interface, which is updated cyclically by the PU.

Notably, no communication with other PUs is implemented here, so the application does not need to initialize an information storage.

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 this can be improved using multiple PUs and the information storage.

4.2 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.

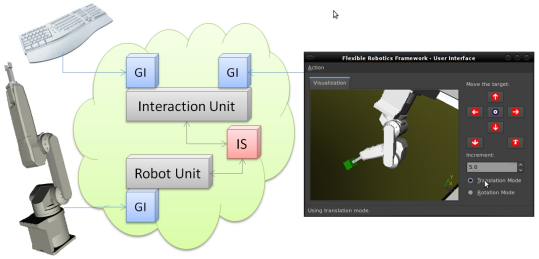

Figure 5: Multithreaded application for keyboard control of a robot.

Figure 5 shows a control system, where keyboard / mouse events are handled by the GI of the upper PU,

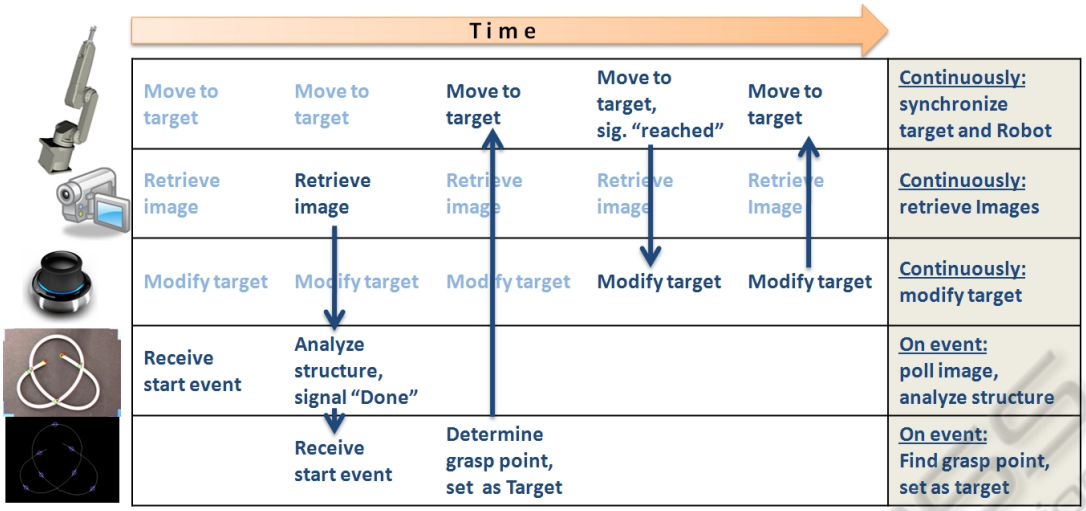

Figure 6: A workflow detail for automatically grasping a limp object from the table and handing it over at a pose defined through user-interaction.

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 thus be computed.

4.3 Limp Object Handling

Concluding the application section, the following paragraphs describe a rather complex limp object handling application. Here, all the proposed concepts from Section 3 are applied.

The application features a 6 DOF mouse-device PU and a realtime robot control unit sending new position commands every seventh millisecond and simultaneously sharing feedback information (joint angles and Cartesian pose of the end-effector). A separate GUI-unit provides for visual feedback, additional manual robot control through buttons and display of the live-image and (semi-)processed images from the IEEE1394 camera. In the application, a focus of attention (FoA) unit implements the attention condensation mechanism (Müller and Knoll, 2009). 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.

A detail of the workflow for manipulation of a limp object in the workspace of the robot is scetched in Figure 6.

The figure shows three PUs performing continuous actions, the robot control unit, the camera unit and the mouse unit. Furthermore, two event-based units are shown, one for analyzing the object structure (ends and intersections) and one for determining a suitable point for grasping the object (see Figure 7 for a detail of the process).

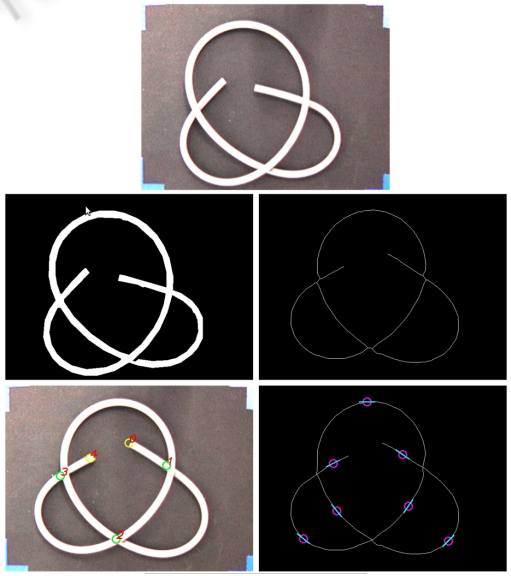

Figure 7: From top to bottom: The region of interest, background segmentation, thinning, structure analysis and possible grasping points.

The application comprises four more units, which

are not shown. These are the FoA unit and the GUI unit mentioned before, and an event-based gripper unit. Using the forth one, the automation unit, the user is able to compose an arbitrary workflow (the application task), e.g., the one shown in Figure 6, by connecting the IO-parameters of different units and then transforming the task into a new PU, e.g., a "find-and-grasp-object" unit.

5 CONCLUSIONS

The paper introduces a flexible robotics and automation system for parallel, asynchronous and decoupled processing on hardware architectures like multicore or multiprocessor systems.

A typical modular application developed with the framework 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; and multiple interfaces to hardware or external software components. Furthermore, the proposed system incorporates facilities to generate automation functions, for example from telepresent teach-in or predefined action primitives.

As an example, the paper shows, how the robotics system can be used to develop an application for limp object handling, incorporating parallel visual processing and realtime control of the actuator in a user-friendly way.

ACKNOWLEDGEMENTS

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

REFERENCES

Anderson, J. R. (2007). *How Can the Human Mind Occur in the Physical Universe?* Oxford University Press.

Bellifemine, F., Caire, G., Poggi, A., and Rimassa, G. (2003). JADE: A White Paper. Technical report, Telecom Italia Lab and Universita degli Studi di Parma.

Corkill, D. D. (2003). Collaborating Software: Blackboard and Multi-Agent Systems & the Future. In *Proc. of the International Lisp Conference*.

Erman, L. D., Hayes-Roth, F., Lesser, V. R., and Reddy, D. R. (1980). The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty. *Computing Surveys*, 12(2):213–253.

FIPA (2002). ACL Message Structure Specification. Technical report, Foundation for Intelligent Physical Agents.

Jackson, J. (2007). Microsoft Robotics Studio: A Technical Introduction. *IEEE Robotics and Automation Magazine*, 14(4):82–87.

Lehman, J. F., Laird, J., and Rosenbloom, P. (2006). 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.

Müller, T. and Knoll, A. (2009). Attention Driven Visual Processing for an Interactive Dialog Robot. In *Proc. of the 24th ACM Symposium on Applied Computing*.

Newell, A. (1994). *Unified Theories of Cognition*. Harvard University Press.

Panin, G., Lenz, C., Nair, S., Roth, E., in Wojtczyk, M., Friedlhuber, T., and Knoll, A. (2008). A Unifying Software Architecture for Model-based Visual Tracking. In *Proc. of the 20th Annual Symposium of Electronic Imaging*.

Quigley, M., Gerkey, B., Conley, K., Faust†, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., and Ng, A. (2009). ROS: an open-source Robot Operating System. In *ICRA 2009 Workshop on Open Source Software in Robotics*.

Rickert, M., Brock, O., and Knoll, A. (2008). Balancing Exploration and Exploitation in Motion Planning. In *Proc. of the IEEE International Conference on Robotics and Automation*.

Vaughan, R. T. and Gerkey, B. P. (2007). *Software Engineering for Experminetal Robotics*, chapter Reusable Robot Code and the Player / Stage Project, pages 267–289. Springer tracts on Advanced Robotics. Springer.

Wooldridge, M. J. (2009). *An Introduction to MultiAgent Systems*. John Wiley & Sons, 2nd edition.