

ICU - A smart optical Sensor for direct Robot Control

Jörg Kaiser and Peter Schaeffer

Department of Computer Structures,
University of Ulm, Germany,
e-mail: kaiser@informatik.uni-ulm.de, ps1@informatik.uni-ulm.de

Abstract-- Technological advances will allow the integration of smart devices which may comprise sensor components, computational devices and a network interface. The built-in computational component enables the implementation of a well-defined high level interface that does not just provide raw transducer data, but a pre-processed, application-related set of process variables. The paper deals with two issues. Firstly, it describes an architecture which encourages the design of multi-level control hierarchies exploiting the easy availability of application related sensor information. This is based on encapsulated smart components, a well defined communication interface and the easy access to this information by a variant of a shared data space. Secondly, we describe the functions and the hardware of ICU which constitutes the prototype of an optical sensor designed for vehicle guidance operating as a smart component in such a system. The task of the sensor is to detect a guidance line and directly produce the information needed by the steering system to control the vehicle.

Index terms-- Vehicle guidance, co-operative control, low cost sensor, smart sensor, middleware, CAN-Bus interface

1 INTRODUCTION

Technological advances will allow the integration of smart devices which may comprise sensor components, computational devices and a network interface. The built-in computational component enables the implementation of a well-defined high level interface that does not just provide raw transducer data, but a pre-processed, application-related set of process variables. Consequently, the interfaces and the functions of these smart components may include functions related to overall control, supervision, and maintenance issues. In his excellent survey of vision sensors, Alireza Moini [1] makes the point that “smart sensors are information sensors, not transducers and signal processing elements”. Perhaps the most interesting and challenging property of these intelligent devices is their ability to spontaneously interact with the overall system. This enables a modular system architecture in which smart autonomous components co-operate to control physical processes reactively without the need of a central co-ordination facility. In such a system, multiple different sensors will co-operate to augment perception of the environment and autonomous actuators will co-ordinate actions to increase speed, power and quality of actuation thus forming decentralized virtual sensor and actuator networks.

As an example consider the vision tasks of an autonomous mobile robot. There may be multiple levels of image processing and recognition ranging from line tracking over obstacle avoidance to scene recognition and image understanding [2]. All these tasks need different levels of reactivity. With multiple dedicated low cost/low power sensors, the need for a fast reactive behaviour can be met without interfering with higher levels. The output of the tracking sensor can directly be used by the motor drive system to keep the vehicle on a guidance line, the obstacle detection sensor may directly stop the motors with minimal latency. Higher level analysis may use the same sources of data but usually is less predictable and responsive because it will have to relate multiple sensors in more complex planning and recognition tasks.

In the context of a project dealing with adequate models and middleware techniques for communication and co-ordination in control systems [3], we developed ICU (Intelligent Camera Unit). ICU is an optical

sensor which computes and disseminates information which is directly related to actuator control rather than deliver just the raw image for further processing.

The rest of the paper is organized as follows. Section 2 motivates a distributed control architecture composed from smart components. In 3, we describe the requirements and intentions of such an architecture and briefly sketch the main ideas. The functions and basics of ICU are covered in 4 while section 5 details the hardware architecture. Conclusions and acknowledgements are given in 6 and 7, respectively.

2 CO-OPERATING OPTICAL SENSORS

There is a large variety of optical sensors and vision sensors ranging from very simple CCD and CMOS arrays to intelligent vision chips and artificial retinas [1]. They can detect relatively simple things as distributions of light intensities or detection of marks or lines. More sophisticated sensors enable motion detection and deriving the speed of a device from optical flow analysis directly embedded in the sensor array [4]. These devices are usually specialized to a single function which they can perform faster, better and with less energy as compared to traditional vision systems which usually convert a digital image in an analogue format coming from the first days of television and then use a frame grabber easily wasting 30 Mbyte of PCI bandwidth again to create a digital image in the processor memory. The advantage of this approach is its generality, i.e. that an image, once in memory is the raw material which can be analysed by software in any desired aspect. However, high performance processors are needed resulting in an overall complex and rather power consuming system. For autonomous mobile vehicles this may be a problem. On the other hand, specialized optical devices only perform a single task. Therefore, multiple such devices may be needed to collaboratively sense and improve the perception of the environment. Special purpose sensors can operate in different spectral ranges and can be used to build robust systems with the possibility to use adaptive strategies to replace defective functions by “virtual sensors” combining the virtues of active sensors, probably with lower precision, timeliness or resolution. To combine these sensors in an effective and robust way and enable direct actuator control, distributed middleware is needed to support easy diffusion of the information to each entity which may be interested in the data, to support spontaneous generation of information and allow temporal constraints to be specified on information propagation. Moreover, it is desirable to have a low configuration effort when integrating new sensors.

3 REQUIREMENTS FOR THE SYSTEM CONTROL ARCHITECTURE

Traditional control systems usually center around a single control unit (CU) which has a sophisticated real world interface. The CU is the last link in a chain of transducers, converting a physical process variable, like a temperature, a pressure or an optical signal via a electrical signal to a digital representation. Signal conditioning as shaping analogue values, debouncing digital inputs and improving an image received from a camera all are performed by this CU. The resulting digital information is a raw, conditioned representation of a single physical entity. Subsequent processing, fusion of multiple sensors information, and generation of control signals for the actuators also is performed by the same processor. Because all these tasks have widely differing performance requirements and temporal constraints, complex planning and scheduling schemes have to guarantee that these tasks can be properly coordinated in the temporal domain. Simply moving to multiprocessors usually worsens the scheduling problem [5].

Using decentralized smart sensors and actuators puts computing power to the place where it is needed. These components are autonomous systems which perform a dedicated complex task beyond signal conditioning. Interconnected by a communication network, they encapsulate a certain functionality and provide meaningful application related information at their network interface. Kopetz [6] highlights the importance of an adequate interface to support functional and temporal encapsulation of smart components to support the composability of an application.

There are a number of goals which we want to reach for the control system architecture:

1. Components of the network are autonomous. Autonomy means that each component is in its own sphere of control and no control signal should cross the boundary of a component. Hence, components only interact via shared information as e.g. proposed in the data field architecture in ADS (Autonomous Decentralized Systems)[7] or the tuple space in Linda [8]. As a consequence, interfaces can be created which do not rely on any, possibly time critical control transfers. This supports easy extensibility, reliability and robustness and encourages a component-based design.
2. The architecture should support many-to-many communication patterns. A typical situation is that the information gained from a sensor can be used and analyzed in more than one place, e.g., the output of a our ICU optical sensor on a mobile robot is interesting for reactive motor control implemented on a small micro-controller as well as for long term navigation strategies implemented on a more powerful device. Another typical example is the situation in which control commands issued from a controller address a number of identical actuators; e.g., all motors have to stop in case of emergency.
3. Communication is spontaneous because control systems have to react to external events. These external events are recognized at the sensor interface of an embedded system at arbitrary points in time and lead to communication activities to disseminate the information. This is best captured in a generative, event-based communication model [9, 10, 11].
4. Communication is anonymous. Consider again the example of stopping a set of motors. When issuing a stop command, it is not of interest to address a specific motor, rather it must be ensured that all relevant motors receive the command. Similarly, when reacting to a stop command, it is not of interest which controller has issued that command. On a more abstract level, a sensor object triggered by the progression of time or the occurrence of an event spontaneously generates the respective information and distributes it to the system. Thus, it is considered as a producer. The corresponding consumer objects have mechanisms to determine whether this information is useful for them. This interaction leads to a model of anonymous communication in which the producer does not know which consumers will use its information and, vice versa, the consumers only know which information they need independently from which source they receive it. Furthermore, anonymous communication supports the extensibility and the reliability of the system because objects can be added or be replaced easily without changing address information maintained in the other objects.

To meet these requirements, we adopted a publisher/subscriber model of communication in which producers (publishers) and consumers (subscribers) of data are connected via event channels [7, 8]. In contrast to other forms of a shared information space [9, 10], the semantic of event channels integrates the notification of consumers when an event occurs and thus support the temporal relation of the event occurrence and the notification of the subscribers. Event channels support content-based communication by relating an event channel to a certain class of information rather than to a source or destination of a message. Thus, a message is routed by its content which is dynamically bound to a network related address. A more detailed description of this architecture can be found in [3]. What mainly distinguishes our publisher/subscriber scheme from existing schemes like the event service in Jini[11] is its anonymity and compared to the event service in CORBA[13] the distributed implementation.

In our system, an event channel is related to a certain class of information, like e.g. certain classes of tracking information. Publishers may be smart cameras, infrared sensors and alike all pushing tracking information in the respective event channels. Multiple subscribers e.g. motor control or the navigation system receive this information and can locally filter and use it. The publisher/subscriber architecture was particularly designed to enable interoperation of embedded systems which usually have stringent performance and bandwidth constraints with more powerful networks and processors. At the moment we provide interoperability between a CAN-Bus [14] for low level reactive control and IP-based protocols. This allows a robot in a co-operating team to seamlessly access remote sensor information of another robot e.g. via a wireless IP connection.

Autonomous components encourage the design of layered intelligent systems as they were proposed by [2], [15], [16]. Most authors separate the reactive layer from higher system levels which realize deliberative behavior. Reactive behavior maps to the low level control architecture in which resources have to be provided to support fast and timely behavior with respect to safety critical actions like avoiding obstacles, recognizing landmarks or coordinating actions of multiple actuator devices. Deliberative behaviors include to a large extent activities related to planning like map building, path planning, global navigation and action co-ordination. Reactive and deliberative behaviors may be based on the same set of sensor information, however with a different degree of relating sensor information and different temporal attributes. This even may include co-ordination of different robots each equipped with a set of different sensors co-operating to jointly explore an unknown environment.

A prototype of our publisher/subscriber architecture is implemented under Linux and RT-Linux. Linux handles the non-critical communication over IP while RT-Linux is used for the time critical communication on the CAN-Bus. A gateway connects the two networks [17]. The testbed is a KURT II autonomous robot which besides various distance and odometric sensors is equipped with ICU (Fig.1).



Fig.1 Kurt-2 robots, robot in front carries ICU (courtesy of Michael Mock, Institute of Autonomous intelligent Systems (AiS))

4 FUNCTIONS OF THE ICU

ICU was primarily designed for vehicle guidance in an experimental traffic scenario. In our first application ICU tracks a guidance line (G-line) on the floor which may include embedded coloured marks. ICU disseminates the position and orientation of the G-line and the colour of the embedded marks. The information delivered by ICU is then directly used by the motor drive system to reactively keep the vehicle to follow this line, to slow down or to stop if a certain coloured mark is detected. Fig. 2 shows the principles of detecting a G-line (the white bar in the figure). The orientation of the G-line is calculated from three scan lines (S-lines). Because of the special colour representation (see Fig. 4) of the image sensor an S-line is represented by 4 pixel-lines.

ICU tries to detect the white G-line on a darker background. To do this, the image is transferred to the micro-controller which calculates the colour and brightness values of every pixel on the S-line. The maximal value of brightness is then used to detect the G-line. Therefore, the G-line must be a white line with optionally embedded coloured marks (see Fig. 2 and 3).

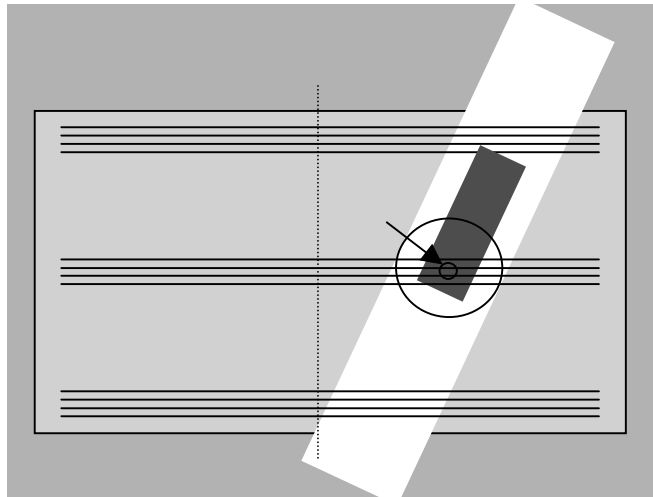


Fig.2 Principle of line detection

The problem encountered in determining the overall brightness is the colour representation. The vision sensor uses a Bayer scheme [18] as depicted in Fig. 4. Hence, the colour of an individual pixel has to be inferred from the relative brightness of the adjacent pixels. Having determined the colour of a pixel, its brightness is calculated using the weighted colour values as described below. The Bayer colour encoding uses 4 pixels to represent the values of red, green and blue.

ICU tries to detect the white G-line on a darker background. To do this, the image is transferred to the micro-controller which calculates the colour and brightness values of every pixel on the S-line. The maximal value of brightness is then used to detect the G-line. Therefore, the G-line must be a white line with optionally embedded coloured marks (see Fig. 2 and 3).

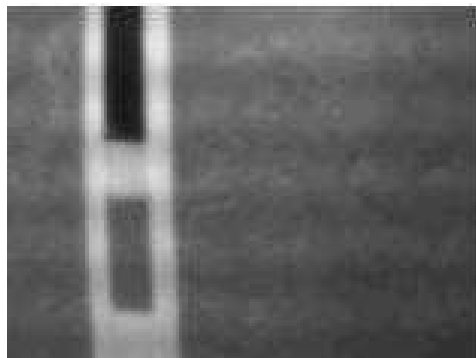


Fig.3 A line as ICU sees it

The problem encountered in determining the overall brightness is the colour representation.

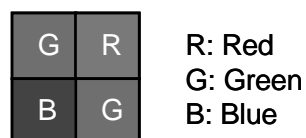


Fig. 4 Bayer Scheme of colouring

The vision sensor uses a Bayer scheme [18] as depicted in Fig. 4. Hence, the colour of an individual pixel has to be inferred from the relative brightness of the adjacent pixels. Having determined the colour of a pixel, its brightness is calculated using the weighted colour values as described below. The Bayer colour encoding uses 4 pixels to represent the values of red, green and blue. Therefore, taking the straightforward approach to directly use these values to represent a single coloured pixel would decrease the resolution of the image by a factor of 4 which was not acceptable. Hence, we used a standard colour encoding scheme which exploits the brightness values of a 3x3 neighbourhood as depicted in Fig. 5. Fig. 5 shows how the colour of a (single) pixel is calculated. Even though the spectral sensitivity of a single pixel is confined to a certain colour, an RGB-value will be assigned to it by considering the adjacent pixels. The RGB-value is calculated according to the weights of the adjacent pixels as depicted in Fig. 5. Subsequently, the brightness of every pixel is calculated on the basis of the RGB-values by the equation:

$$\text{Brightness } B = (R*38+G*75+B*15)/128$$

Thus we maintain the full resolution of the image in spite of the Bayer colour representation, however, the algorithm has the effect of the low pass filter. As a consequence, the image inevitably loses sharpness.

The next step is detecting the G-line. For this, the S-line is scanned from both sides to find the respective transition of contrast. The algorithm is rather straightforward for the moment and only checks whether the edges detected from the left-to-right and right-to-left scan have a certain distance from each other corresponding to the assumed width of the G-line. If not, a fault value is returned. All three S-lines are evaluated as depicted in Fig. 2 to determine the position and orientation of the tracking line.

The last step is to determine the colour of the marks embedded in the G-line. As Fig. 3 shows it calculates the position of the centre of the G-line and takes the colour and the brightness of the pixel at this position. If the G-line is not detected properly, ICU interpolates the respective centre position from the positions of the G-line in the upper and the lower S-line. Subsequently, the colour of the pixel at this interpolated position is selected. This approach is also useful for intersections or junctions of G-lines.

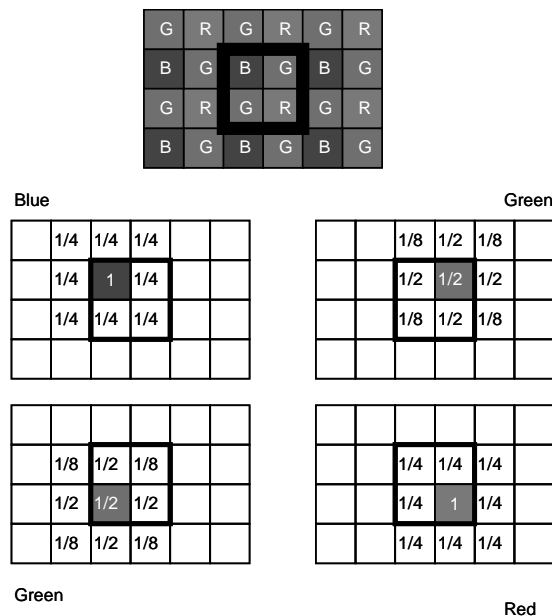


Fig. 5 Colour encoding

The sensor is rather sensitive to the precise focus of the lens. Currently, because the sensor always has a fixed distance to the G-line, focussing has only been performed once and therefore is done by hand. Because ICU usually does not provide an image for inspection to adjust the focus, we provided a focussing aid based on an automatic detection of a maximum contrast transition of an appropriate black to white edge. We only use green pixels for the mechanism because colour is not needed and green pixels are dense on the sensor (see Fig. 4 and 5) whereas lines of blue or red pixels always exhibit gaps. The green pixels are arranged in a zick-zack pattern which, however, does not have a major influence on the detection of a black-to-white transition.

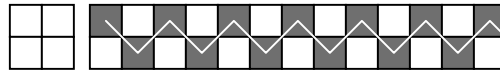


Fig. 5 Scanning for focussing control

If the lens is out of focus, the contrast edge is blurred meaning that the values of adjacent pixel do not exhibit a sharp transition. By adjusting the focus precisely, adjacent pixels indicate a sharp transition which in turn triggers an LED for external inspection. The condition for the LED is a maximal brightness difference that is above a certain threshold between any two adjacent pixels.

5 THE HARDWARE OF ICU

ICU was intended as a low cost sensor for simple imaging tasks which can directly be used for control applications. ICU should be adaptable to various simple vision tasks. Therefore, we took a camera/processor approach and put emphasis on an easy and fairly universal processor interface. Fig. 6 depicts the basic components of ICU. We connect the sensor to an asynchronous external port, which is not the most efficient way but eases the use of different micro-controllers. In fact, we used the front end sensor and the interface logic even with a simple micro-controller as Motorola 68HC11 for educational purposes. In the current version, a 16-Bit controller C167 from Siemens [19] (now provided by Infineon) is used featuring a minimal instruction execution time of 100ns and a peak rate of 10Mips.

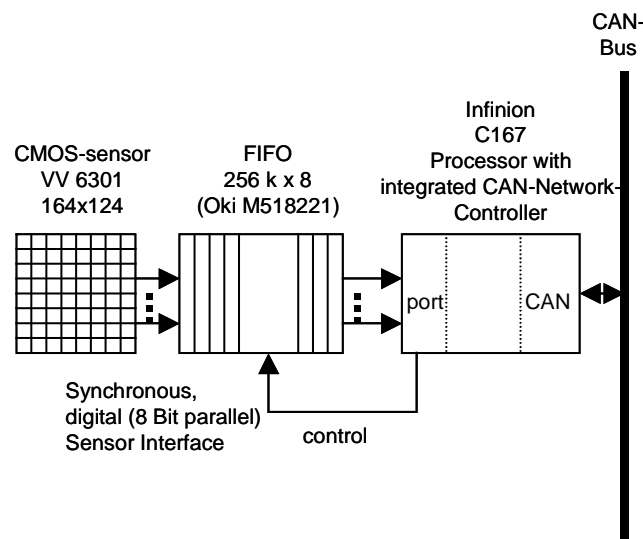


Fig. 6 The main components of ICU

The heart of the interface logic is a FIFO which decouples the fast synchronous operation of the sensor from the slower asynchronously operating processor interface. The only control line from the sensor to synchronize the processor with the sensor is a “start-of-frame” signal. The FIFO can buffer up to 13 frames which are delivered by the sensor with a maximum rate of 30 frames/sec. Currently, we only use 15 frames/sec, partly because of the port interface to the processor which is rather slow in copying data from the FIFO to the processor memory.

ICU has a CAN-Bus [14] network interface which is popular in automotive and industrial control systems and allows transfer rates up to 1 Mbit/sec with a protocol efficiency of about 50% payload. The maximal length of a single CAN-Bus message is 8 Bytes. The CAN-Bus was designed to reliably operate in environments exhibiting high electric noise, but obviously not for high speed data transfer. Therefore it would be impossible to transfer raw image data which would need around 2,3 Mbit/sec (160*120 pixels, 1 byte/pixel, 15 frames) even for the low resolution sensor. However, the pre-processed complete information to control the movement of the vehicle fits into a single CAN message and puts no significant load on the communication system.

At the moment ICU can be configured to either periodically publishing position and colour information on the CAN-Bus or to transmit a tracking position on demand. A small local software component, called event channel handler connects ICU to the publisher/subscriber middleware. The information delivered in a CAN message by ICU is depicted in Tab.1. ICU disseminates:

1. the position of the G-line within its window of perception,
2. the colour in separate values for red, green and blue and
3. brightness information

Byte	Content
1	x-position on upper line
2	x-position at center line
3	x-position at lower line
4	R-intensity
5	G-intensity
6	B-intensity
7	Brightness
8	Not used

Tab. 1 Payload of the CAN message

ICU is configured via the CAN-Bus. The configuration message defines the rate of the tracking information. If the configuration message is empty, i.e. it does not include any parameters, the camera returns a single message. In addition to the CAN interface, a serial interface is available for debugging purposes and for changing the flash memory of the micro-controller. Fig. 7 and Fig. 8 show the ICU prototype and the test environment depicting ICU hooked to a CAN analysis tool.

6 CONCLUSION AND FUTURE WORK

Design of control systems for complex artifacts like autonomous vehicles, driver assisting cars, airplanes and industrial plants tend to become enormously complicated. One step towards solving the problem is to provide computational power at the sensors and actuators making them smart devices encapsulating all computations needed to convert a physical process value to an application related meaningful information.

The autonomy and encapsulation properties encourage hardware/software co-design on the device level and component-based design methodologies [20] for putting the building blocks together. In the context of a project dealing with appropriate models and mechanisms to support decentralized systems composed from these components, we developed a couple of smart sensors.



Fig. 7 ICU prototype

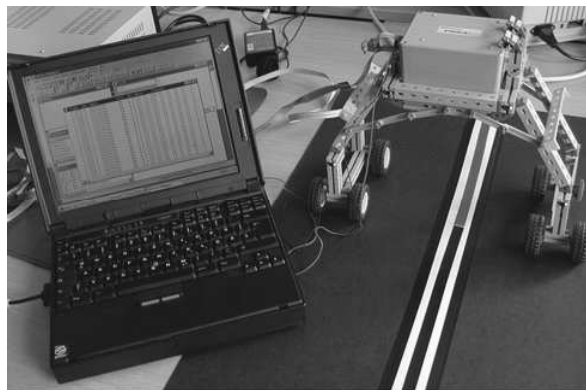


Fig. 8 Test environment for ICU

ICU is an optical sensor specialized on recognizing a tracking line for vehicle guidance. ICU provides tracking information via a CAN-Bus interface and the respective middleware, thus making it available to any entity which may exploit this information. This may be the smart motor controllers for directly steering the vehicle or a navigation system just logging the course. ICU is the first prototype of a low cost/low power optical sensor which is used in such an environment. Because ICU is adapted to a single application related function, resources like hardware, communication bandwidth and power consumption can be reduced to a minimum just as required by the application. ICU is designed for vehicle guidance based on tracking predefined lines and marks. At the moment, we use off-the-shelf processor boards [21] and only designed the glue hardware ourselves. In the future, we will be working in two directions: 1. we will design better and more robust algorithms to detect G-lines particularly detecting intersections and junctions of G-lines. 2. we will design a processor board which provides a faster interface to the optical sensor to enable more sophisticated vision tasks without substantially increasing power consumption which at the moment is around 250mA@5V for the entire ICU.

7 ACKNOWLEDGEMENTS

The work on ICU was partly sponsored by GMD AiS¹ and the University of Magdeburg. We want to thank Michael Mock (GMD) Edgar Nett (U. Magdeburg), Reiner Frings (GMD) and Martin Gergeleit (U. Magdeburg) for their co-operation and support. Particularly, we want to acknowledge Michael Wallner's effort to integrate ICU in the publisher/subscriber communication environment. This work was partly funded by the EU in the CORTEX² Project under the contract No.: IST-2000-26031.

8 REFERENCES

- [1] Alireza Moini: Vision Chips or Seeing Silicon, Third Revision, <http://www.eleceng.adelaide.edu.au/Groups/GAAS/Bugeye/visionchips/index.html>, March 1997
- [2] R.A. Brooks: A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation*, vol. RA-2, no.1, March 1986
- [3] J. Kaiser, M. Mock : Implementing the Real-Time Publisher/Subscriber Model on the Controller Area Network (CAN), *Proceedings of the 2nd Int. Symp. on Object-oriented Real-time distributed Computing (ISORC99)*, Saint-Malo, France, May 1999
- [4] J. Tanner, C. Mead: An integrated analog optical motion sensor, in R.W. Brodersen & H.S. Moscovitz, editor, *VLSI Signal Processing, II*, pp. 59{87. IEEE, New York, 1988.
- [5] J. Stankovic: Misconceptions about Real-Time Computing, *IEEE Computer*, 1988
- [6] H. Kopetz, E. Fuchs, D. Millinger, R. Nossal: An Interface as a Design Object, *Proc. Int'l Symp. on Object-oriented Real-Time Distributed Computing (ISORC-99)*, St. Malo, France 1999
- [7] B. Oki, M. Pfluegl, A. Seigel, D. Skeen: "The information Bus®- An Architecture for Extensible Distributed Systems", *14th ACM Symposium on Operating System Principles, Asheville, NC*, Dec 1993,pp.58-68
- [8] R. Rajkumar, M. Gagliardi, L Sha: " The Real-Time Publisher/Subscribe Inter-Process Communication Model for Distributed Real-Time Systems: Design and Implementation", *IEEE Real-time Technology and Applications Symposium*, June 1995
- [9] K. Mori. Autonomous decentralized Systems: Concepts, Data Field Architectures, and Future Trends, *Int. Conference on Autonomous Decentralized Systems (ISADS93)*, 1993
- [10] N. Carriero, D. Gelernter: "Linda in Context", *Communications of the ACM*, 32, 4, April 1989
- [11] Rene Meier: State of the Art Review of Distributed Event Models, *Tech. Report TCD-CS-2000-16*, Trinity College, Dublin Ireland, March 2000
- [12] T. Harrison, D. Levine, and D. Schmidt: The design and performance of a real-time CORBA event service. In *Proceedings of the 1997 Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA)*, 1997
- [13] Sun Microsystems. Java Distributed Event Specification. *Sun Microsystems, Inc.*, July 1998, <http://www.javasoft.com/products/javaspaces/specs>
- [14] Object Management Group. CORBAservices: Common Object Services Specification - chapter 4, Event Service Specification. *Object Management Group*, March 1995. <http://www.omg.org/library/csindx.html>.
- [15] ROBERT BOSCH GmbH: "CAN Specification Version 2.0", Sep. 1991
- [16] D. J. Musliner: CIRCA: The Cooperative Intelligent Real-Time Control Architecture, *PhD-Dissertation*, University of Michigan, 1993
- [17] K. Z. Haigh: Situation-Dependent Learning for Interleaved Planning and Robot Execution, *Tech. Report CMU-CS-98-108*, CMU, February 1998
- [18] M. Wallner: Ein Publisher/Subscriber-Protokoll für heterogene Kommunikationssysteme, *Master thesis*, University of Ulm, March 2001
- [19] VLSI Vision Ltd: VV6300 Low Resolution Digital CMOS Image Sensor, Technical Description cd 34021-b.fm, UK, 1998, www.vvl.co.uk
- [20] Siemens AG: 16 Bit Micro-controllers – C167 Derivatives, User Manual 03.96 Version 2.9, 1996

¹ GMD AiS: German National Research Institute for Information Technology, Institute for Autonomous Intelligent Systems

² CORTEX: CO-operating Real-time senTient objects: Architecture and Experimental evaluation.

- [21] C. E. Pereira , L. B. Becker, C. Villela, C. Mitidieri , J. Kaiser : On Evaluating Interaction and Communication Schemes for Automation Applications based on Real-Time Distributed Objects, *Proceedings of the 2nd Int. Symp. on Object-oriented Real-time distributed Computing (ISORC2001), Magdeburg, Germany, May 2001*
- [22] Phytec: Mini-Modul 167 Hardware Manual, Phytec Messtechnik GmbH, Mainz, 1999, <http://www.phytec.de>