ARS ADVANCED ROBOTIC SYSTEMS INTERNATIONAL

VIP - A Framework-Based Approach to Robot Vision

Hans Utz[1]; Ulrich Kaufmann[1] ; Gerd Mayer[1] & Gerhard K. Kraetzschmar[2]
[1] University of Ulm, Neuroinformatics, Ulm, Germany, {utz|kaufmann|mayer}@informatik.uni-ulm.de
[2] Fraunhofer AIS, Sankt Augustin, Germany, gerhard.kraetzschmar@ais.fraunhofer.de

Abstract: *For robot perception, video cameras are very valuable sensors, but the computer vision methods applied to extract information from camera images are usually computationally expensive. Integrating computer vision methods into a robot control architecture requires to balance exploitation of camera images with the need to preserve reactivity and robustness. We claim that better software support is needed in order to facilitate and simplify the application of computer vision and image processing methods on autonomous mobile robots. In particular, such support must address a simplified specification of image processing architectures, control and synchronization issues of image processing steps, and the integration of the image processing machinery into the overall robot control architecture. This paper introduces the video image processing (VIP) framework, a software framework for multithreaded control flow modeling in robot vision.*
Keywords: *robot vision, robot middleware, real-time support for computer vision, resource adaptivity*

1. Introduction

Perception of the environment is a key component for autonomous mobile robots, which relies on various kinds of sensors providing data that allow to extract relevant information. Video cameras are nowadays low-cost components yet a very rich source of information, but the computer vision methods necessary to extract the desired information are often computationally expensive. Integrating computer vision methods into a robot control architecture requires great care and substantial effort by the programmer, especially in highly dynamic environments which require an adequate level of reactivity. System designers must carefully balance their desire to apply computationally costly image processing methods for maximally exploiting the information contained in camera images with the necessity to maintain timely reaction to critical sensor events and provide reasonable degree of robustness.

Better software support is needed in order to facilitate and simplify the application of computer vision and image processing methods on autonomous mobile robots. In particular, a framework for robot vision must address a simplified specification of image processing architectures, control and synchronization issues of image processing steps, and the integration of the image processing architecture into the overall robot control architecture.

This paper introduces the video image processing (VIP) framework, a software framework for multithreaded control flow modeling in robot vision and discusses its support for the development of robot vision applications. It is particularly well-suited for robot vision applications in highly dynamic environments, such as robot soccer playing like in the RoboCup middle-size robot league (Utz, H. & Kaufmann, U. & Mayer, G., 2005).

The remainder of this paper is organized as follows: The next section discusses challenges for robot vision and related work. The VIP framework is introduced in Section 3 and a discussion of the provided support for the development process of robot vision applications is presented in Section 4. The application of the framework is illustrated by a short example in Section 5, and the paper concludes with application examples, conclusions, and possible future work.

2. Image Processing on Mobile Robots

Vision systems for autonomous mobile robots must unify the requirements and demands of two very challenging disciplines: *i)* computer vision and image processing, and *ii)* robotics and embedded system. While the state of the art in computer vision algorithms is quite advanced, many computer vision methods are intrinsically computationally expensive. Even an efficient implementation of such methods cannot fix this problem. Therefore, the resource demands of computer vision methods are in conflict with the requirements posed by robotics and embedded systems, which demand very short execution cycles for the control loops which read out and process sensor data, interpret and fuse them, and determine appropriate actions for the actuators. Particularly the real-time requirements of robotics seem to rule out most sophisticated computer vision methods, which is one of the reasons why some computer vision experts get discouraged

to work on robot vision. As a consequence, robot vision software systems are often inflexible and hard to maintain, because they tend to contain hard-coded quick hacks, which for efficiency reasons try to exploit microoptimizations like performing multiple operations simultaneously, or because they are heavily model-based or incorporate application-specific heuristics.
In order to mediate between the partially contradictory requirements of advanced vision processing in a real-time constraint environment, proper conceptual support for vision processing architectures is required. Such conceptual support should encapsulate the vision application within its application domain. For a better understanding of the different requirements that need to be supported, we briefly review some characteristics of the two problem domains.

2.1. Computer Vision and Image Processing
The basic concept of computer vision is the application of operators to image data, such as logical and arithmetical operations (conjunctions, multiplications), color conversions, morphological functions (erosion, dilation), filtering functions (Gaussian filters, convolutions), or linear transforms (Fourier or wavelet transforms). Often operations transform more than one input image into a new output image. For example, the Canny edge detector (Canny, J. F., 1986) needs two images which are obtained by convolving a horizontal and a vertical Sobel operator respectively. Other operators, such as optical flow require as input a sequence of images obtained at different instances of time (Horn, B. K. P. & Schunck, B. G., 1980). In principle, an image operation can be viewed as a mapping of one or more input images into a new one. More sophisticated operations cover more general input/output mappings, i.e. the result of an image processing operation does not have to be another image, but may be any other kind of data structure, such as a color histogram, a similarity measure between two images, or any other statistic measure on the image. In this case, the definition of a filter is extended to a mapping of one or multiple input images into a new image, or one or multiple classification values associated with the image.

Sequences of such image operators reveal features within the image that can be used to identify regions of interest (ROIs). Some filters do not work on the whole image, but only on parts of it. ROIs are used either to speed up the processing loop, or to make sure that the result of successive operations is not influenced or noisified by image areas which are already known to be irrelevant (e.g. in object classification). Various kinds of feedback loops, such as integration over time, can speed up processing and improve classification results (Mayer, G. & Melchert, J. & Utz, H. & Kraetzschmar, G. & Palm, G., 2005). Because regions of interest can change between individual processing steps, they are associated to images just like the above mentioned classification values.

2.2. Robot Vision
Performing operations such as described in the previous section on the video image stream supplied by one or more cameras on an autonomous mobile robot imposes further constraints on the processing model.

Timeliness Constraints. Robots are situated in a physical world. For tasks like obstacle detection and object tracking the image processing operations must be calculated many times per second and preferably at full frame rate, which is typically 30Hz. The system designer needs to repeatedly assess the performance of the vision system and to ensure its efficiency. Whenever possible, image processing operations should be executed in parallel in order to fully exploit the available resources, such as dual-CPU boards and hyperthreading and multicore processor technologies. More complex image operations, which need not be applied at full frame-rate, should be executed asynchronously in order to ensure that the performance of other image evaluations is not jeopardized. Adequate processing models are required to support such designs.

Fixed Frame Rate Image Streams. New images usually arrive at a fixed frame rate. As the value of the obtained information rapidly decreases in a dynamic environment, a sensor-triggered evaluation model is required.

Parameterization. Most image operators need to be parameterized in order to tune the quality of their results.
Examples are the width of a Gaussian filter or the number of buckets for an orientation histogram. The calibration and optimization of parameters is an important part of the development process. Also, the configuration of the filter graph has to be altered frequently during development, which can be significantly facilitated by a flexible and configurable development environment for robot vision systems.
Development Model. Robot vision is performed on live image streams, which are recorded by a moving robot platform. This must be addressed in the development model. For many vision applications, development starts on sets of test images and recorded test image streams. If the application domain implies nondeterminism, or if the robot's actions affect the quality of sensor data by inducing effects like motion blur, the vision system needs to be tested extensively in the real-world environment. This requires effective and stringent support for the test-debug cycle, for inspection, and for adaptation of parameters in the running application.

2.3. Related Work
The widely known vision-related architectures and the associated relevant literature can be roughly divided into three categories: subroutine libraries, command languages, and visual programming languages.

Subroutine libraries are most widespread. They concentrate on the efficient implementation of image operators. Therefore, they typically provide a set of functions, each of which implements a different image processing operation. Well-known examples are the SPIDER system (Tamura, H. & Sakane, S. & Tomita, F. & Yokoya, N., 1983) and NAG's IPAL package (Carter, M. K. & Crennell, K. M. & Golton, E. & Maybury, R. & Bartlett, A. & Hammarling, S. & Oldfield, R., 1989), which are written in C or Fortran. More recent libraries include the LTI-Lib (ltilib) or VXL (vxl), which are both open-source, written in C++, and provide a wide range of image operations covering image processing methods, visualization tools, and I/O functions. The commercial Intel Performance Primitives (ipp) are an example for highly (MMX and SSE) optimized processing routines with a C-API. What all these libraries have in common is their lack of adequate support for flow control. Aside of yet another collection of mutex or semaphore helper classes and possibly some kind of thread abstraction, there is no special flow control support available.

Command languages for image processing are commonly implemented as scriptable command line tools. In case of the imlib3d package (imlib3d), the image processing operators can be called from the Unix command line. The CVIPtools (Umbaugh, S. E., 1998) are delivered with an extended Tcl command language. Both packages provide programming constructs for conditionals and iteration. While the programmer has complete control over the system in a very flexible way, she also carries full liability over the dynamics of the image processing cycle. Additionally, the scripting-based approach does not make it any easier to meet the required performance constraints of our typical application domains.

Visual programming languages currently present probably the most sophisticated solution approach. They allow the user to connect a flowchart of the intended processing pipeline using the mouse. They combine the expressiveness and the flexibility of both libraries and command languages. Often, they provide not only a wide spectrum of image processing functions and statistical tools, but also a complete integrated development environment. Many of the available systems are commercial products, with Visi-Quest (formerly known as Khoros/Cantata) being one of the most advanced ones. According to the information available from their web site, it supports distributed computing capabilities for deploying applications across a heterogeneous network, data transport abstractions (file, mmap, stream, shared memory) for efficient data movement, and some basic utilities for memory allocation and data structure I/O.

To the best of our knowledge there is no image processing framework that combines all of the features described above, like processing parts of the filter tree on demand and in a flexible yet powerful way. Such a capability would make the system suitable for a wider range of image processing tasks, like active vision problems on autonomous mobile robots.

3. Framework Based Solution Approach

An essential step towards easier and faster development of mobile robot software is the reuse of solutions for often encountered tasks. Frameworks are second generation programming abstractions. While libraries group related algorithms into pre-built packages, frameworks do not only foster the reuse of code, but also the adaption of a design (Johnson, R. E., 1997).

We propose a framework-based design for addressing the integration challenges incurred by robotics applications. The framework allows to encapsulate general challenges of the domain and separate them from the specific solution of a particular problem in an application subdomain. A framework-based design offers a set of features that allows to address the problems of robot vision in a natural way.

Inversion of control flow. Frameworks simplify significantly the programmer's task to define the program logic and to specify the control flow of the application by providing adequate control flow abstractions. The system designer only needs to implement the individual functionality of the target application. Using this functionality and the available control flow abstractions, the system designer then specifies the application on a functional level. The framework then executes the implemented code as the execution logic implies. By providing different models of parallel or interleaved execution, the framework can add significant flexibility and address issues like reactivity and scalability.

The VIP framework introduces a sensor-triggered evaluation model of image operations. The basic processing unit is called a filter. This denotes not only a (non-)linear image transformation function like a Sobel operator, but any kind of input/output mapping, including e.g. neural classificators on image features. The control flow is organized as a filter tree, which is evaluated in depth-first order. VIP only processes filters (and their predecessors), actually subscribed by other modules of the application, ensuring that only the minimally required filter tree is evaluated. For parallel execution of image operations, the framework also allows to define multiple processing trees. These can be individually priorized to separate highly reactive vision tasks from slower processing as discussed in more detail in (Utz, H. & Kaufmann, U. & Mayer, G., 2005).

Management of data flow. Frameworks often organize data flow as well. Data flow management can shield an application programmer from various subtle problems, like e.g. locking in multithreaded applications. Also, a poorly conceived organization of data flow can easily lead to memory leaks, which represent a subtle and hard-to-trace

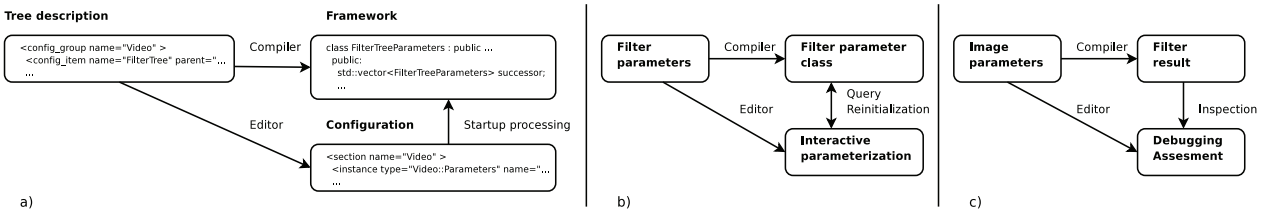

Fig. 1. Applications of the parameter and configurations management toolkit

category of programming errors that compromises the overall stability of an application.

While the control flow is evaluated in a tree in depth first order, the data flow can be much more flexibly organized as a directed acyclic graph (DAG) of filters. That is, each filter can use data from multiple input filters and its output can be used by multiple successor filters. The data handled by each filter may consist of image buffers that adhere to a broad set of common memory formats for images, but may also include meta-information, such as a list of ROIs, histogram values, etc., which can also be passed through the filter DAG. This data is passed between filters as reference counted memory objects, allowing for zero-copying.

VIP allows for multiple processing trees that may run in parallel within the dataflow graph. VIP ensures the correct evaluation order of synchronously connected filters and proper synchronization with input buffers from asynchronous processing trees, e.g. ensuring that successive filters, which are linked to the same asynchronous processing tree, actually process images originating from the same input image. This effectively shields the vision programmer from dealing with the subtle locking issues associated with an asynchronous processing model.

Standard functionality. A framework may provide building blocks of standard functionality for an application domain. The reuse of the design by the application programmer ensures the matching of the interfaces and helps to eliminate the need for glue code that moderates between the prebuilt components and their new application.

The VIP framework models the various cameras supported by it as source filters in and provides implementations for different video devices, such as frame grabber cards and digital camera connections via IEEE 1394 or USB. A set of basic image operations such as color conversions are provided as well. The reuse of framework components actually presented the basis for a rapidly growing filter library, which is used for multiple robotics projects targeting different scenarios. This library utilizes the IPP library for providing highly efficient implementations of image operations.

Middleware integration. Integrating a framework into robotics middleware allows to reuse existing infrastructure already provided for the robotics domain. The VIP framework is integrated into the MIRO middleware (Utz, H. & Sablatnög, S. & Enderle, S. & Kraetzschmar, G., 2002), which features a sophisticated CORBA-based communication environment and extensive support for configuration and parameter management.

In essence, support for solving the problems intrinsic to robotic vision is supplied on the basis of the configurability and adaptivity of the framework and its associated execution logic, by providing special purpose filters, and also by additional development tools. The CORBA infrastructure is exploited to communicate results of high-level filters such as object classifications to client applications, but also to allow remote inspection of filter results for debugging purposes. VIP is implemented as a C++ whitebox framework for Linux platforms.

4. Support of the Development Process

Robot vision applications require extensive testing and tuning of filter configurations. The VIP framework utilizes the infrastructure available in MIRO for extensive support of the development process. VIP features include generic inspectability for every filter output image within the data flow graph. These image data are either communicated locally by shared memory, thereby allowing for synchronized queries of multiple images in a processing tree, or may be requested by remote machines anywhere in the network.

VIP utilizes the parameter and configuration management toolkit provided by MIRO for various purposes within the framework architecture. A basic property of the toolkit is to define parameters in an XML-based description language in so-called parameter files. The description language allows the definition of user-defined data types, supporting nesting, aggregation, and single inheritance. Additionally, default values and a verbatim description can be provided for each parameter. The parameter files are used for two purposes. First, a compiler translates them into classes of a target programming language (currently C++). Second, they are utilized by an interpreter to provide generic GUI-based editing facilities. Configurations of parameter instances are stored in so-called configuration files that are parsed on startup of an application. Note, that the compiler also provides methods that implement the parsing of parameters from configuration files (see Figure 1a). This infrastructure allows to specify the complete configuration of the filter graph and its associated filters in a single configuration file. Thereby, even extensive reconfigurations of the filter

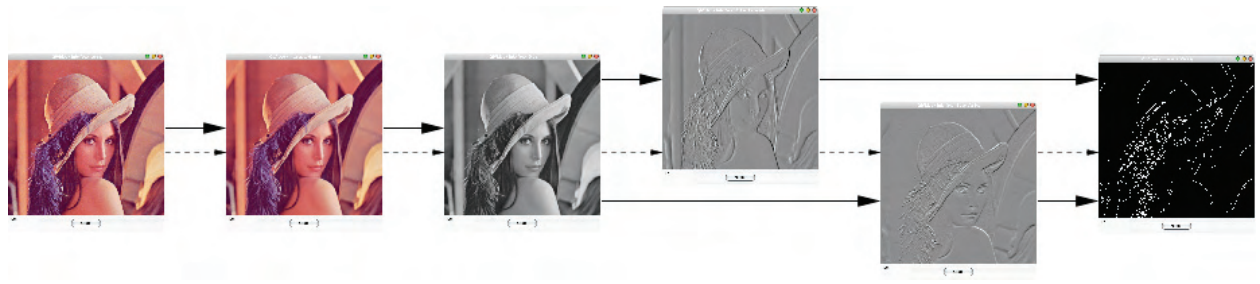

Fig. 2. Original image, intermediate processing steps (blurred, grayed and convolved images) and resulting edge detection. The thick, solid lines denote the data flow, the thinner, dashed lines the control flow

graph can be performed without recompiling a single line of code.

In the early stages of the development of a visual sensor such as an object classification, this allows to exchange the actual camera-based image source for an offline image source. Providing such a dummy device, which actually re-reads stored image sequences from disk, requires only the exchange of a single filter in the graph within the configuration file.

The parameter toolkit is also utilized for the parameterization of individual filters. The filter parameters can not only be specified in the configuration file, they can also be queried and set through a network transparent interface, utilizing the XML-based serialization/deserialization capabilities generically provided by the compiler. The configuration editor can act as a client to these interfaces and allows to edit and change the filter parameters on the fly without interrupting the running application (see Figure 1b).

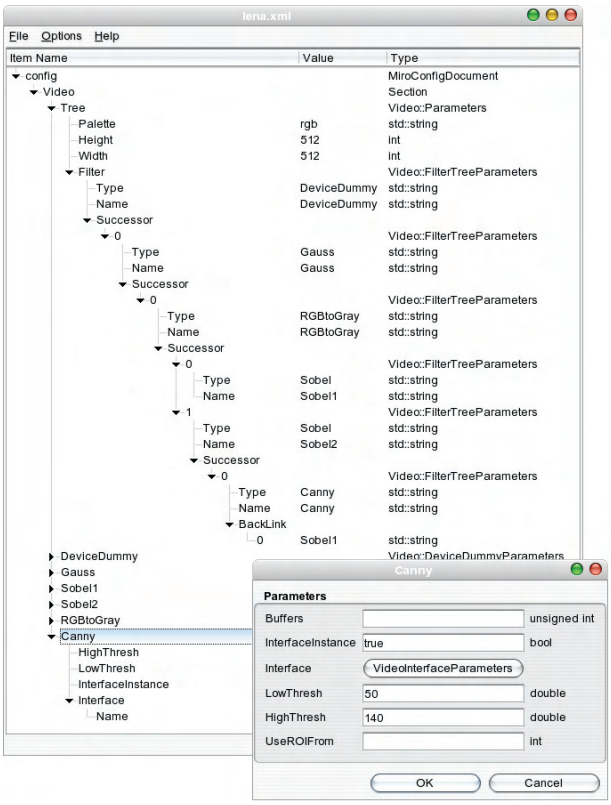

Fig. 3. Graphical user interface displaying the configuration from Fig. 2. These dialogs are automatically generated from the parameter descriptions

The third application of the parameter toolkit is found in the specification of filter meta-information, as the compiler based approach does not provide any information on run-time performance penalties for such specifications. However, they allow to generically inspect the results from various filters along with the produced image output (see Figure 1c). A utilization of this feature by the GUI-based end user tools is not yet implemented, though.

5. Example Configuration

The previously described set of features provided by the VIP framework is best illustrated by a small example. Figure 2 describes the derivation of an edge image for a standard test image of computer vision. The original image is blurred using a Gaussian and then transformed into a grey image. A horizontal and vertical Sobel operator is applied next. In the last step, the Canny operator is applied. The screenshots are taken from the generic inspection tool. Meta-information is not provided by these simple filters. Data and control flow are illustrated in Figure 2. The thick, solid lines denote the data flow, while the dashed lines illustrate the control flow.

In Figure 3, the configuration is shown in the graphical user interface for the filter graph configuration as discussed in the previous section. In addition to the control and data flow, parameters such as the threshold for the Canny operator can be edited in typesafe dialog fields. The thresholds displayed in the dialog window can be altered on the fly, while the vision application is running.

6. Application

The VIP framework is successfully applied in different robotic scenarios, such as biologically-motivated neural network learning, neural object classification in an office environment (Fay, R. & Kaufmann, U. & Schwenker, F. & Palm, G., 2004), and reliable high speed image processing in the RoboCup middle-size robot league (Mayer, G. & Kaufmann, U. & Kraetzschmar, G. & Palm, G., 2005). The application of VIP in these very different scenarios is the

source of a large library of filters shared between these projects.

The application of VIP in RoboCup consists of a dual camera setup, combining a directed camera for object detection, classification, and tracking with an omnidirectional camera for obstacle avoidance and near-distance ball tracking. The full RoboCup application consists of 66 filters with 108 connections. One of the fastest path, a simple color-based football goal detection, takes only around 4 msecs to execute, while one of the slowest paths, a full, neural network-based classification of robots, requires around 20 msecs on average when seeing one robot per image (measured on 1.4GHz Pentium M).

7. Conclusions and Future Work

The difficulties of applying advanced computer vision to autonomous mobile robots in dynamic environments are discussed. The VIP framework is introduced, which was designed to facilitate the application of computer vision in robotics by managing the additional challenges of robot vision in this domain. The middleware-based framework approach allows to support roboticists with respect to non-functional aspects like configuration, priorization, and performance assessment. Extensive development support is provided in the form of parameter management, GUI-based configuration, and generic inspection of images and meta-information.

Future work on the framework will include further support for the generic modeling of filter properties in the development process. So, for instance the specification and verification of filter requirements for their predecessors, such as the type of associated information, the supported image formats and properties, such as in-place applicability of the filter for generic optimization if allowed by the filter graph configuration.

Acknowledgments:

The work described in this paper was supported by DFG SPP 1125 within the project *Adaptivity and Learning in Teams of Cooperating Mobile Robots*. Miro, including its video image processing facilities and documentation, is available at http://smart.informatik.uni-ulm.de/MIRO/.

8. References

imlib3d. Available via http://imlib3d.sourceforge.net/, Accessed: 2005-10

ipp. Intel Performance Primitives (IPP). More information on http://www.intel.com/software/products/perflib/, Accessed: 2005-10

ltilib. LTI-Lib, Available via http://ltilib.sourceforge.net/doc/homepage/index.shtml, Accessed: 2005-10

vxl. VXL, Available via http://vxl.sourceforge.net/, Accessed : 2005-10

Canny, J. F. (1986). A computational approach to edge detection, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 679-698, 1986

Carter, M. K. & Crennell, K. M. & Golton, E. & Maybury, R. & Bartlett, A. & Hammarling, S. & Oldfield, R. (1989). The design and implementation of a portable image processing algorithms library in FORTRAN and C, Proceedings of the 3rd IEEE International Conference on Image Processing and its Applications, pp. 516-520, 1989

Fay, R. & Kaufmann, U. & Schwenker, F. & Palm, G. (2004). Learning Object Recognition in a NeuroBotic System, 3rd Workshop on SelfOrganization of AdaptiVE Behavior SOAVE, Groß, H.-M. & Debes, K. & Böhme, H.-J. (Ed.), pp. 198--209, 2004

Horn, B. K. P. & Schunck, B. G. (1980). Determining optical flow, AI Memo 572, Massachusetts Institue of Technology, 1980

Johnson, R. E. (1997). Frameworks = (components + patterns), *Commun. ACM*, Vol. 40, No. 10, 1997, pp. 39-42, ISSN 0001-0782

Mayer, G. & Kaufmann, U. & Kraetzschmar, G. & Palm, G. (2005). Neural Robot Detection in RoboCup, In: *Biomimetic Neural Learning for Intelligent Robots*, Wermter, S. & Palm, G. & Elshaw, M. (Ed.), pp. 349-361, Springer, Heidelberg

Mayer, G. & Melchert, J. & Utz, H. & Kraetzschmar, G. & Palm, G. (2005). Neural Object Classification and Tracking, 4th Chapter Conference on Applied Cybernetics, IEEE Systems, Man and Cybernetics Society, 2005

Tamura, H. & Sakane, S. & Tomita, F. & Yokoya, N. (1983). Design and implementation of SPIDER-a transportable image processing package, Computer Vision, Graphics and Image Processing, Vol. 23, No. 3, pp. 273-294, 1983

Umbaugh, S. E. (1998). *COMPUTER VISION and IMAGE PROCESSING: A Practical Approach Using CVIPtools*, Prentice Hall, June 1998

Utz, H. & Kaufmann, U. & Mayer, G. (2005). Advanced Video Image Processing on Autonomous Mobile Robots, Nineteenth International Joint Conference on Artificial Intelligence, Workshop on Agents in Real-time and Dynamic Environments, Edinburgh, Scotland, 2005

Utz, H. & Sablatnög, S. & Enderle, S. & Kraetzschmar, G. (2002). Miro -- Middleware for Mobile Robot Applications, *IEEE Transactions on Robotics and Automation, Special Issue on Object-Oriented Distributed Control Architectures*, Vol. 18, No. 4, pp. 493-497, August 2002