The Halcon Vision System: An Example for Flexible Software Architecture

Wolfgang Eckstein, Carsten Steger
MVTec Software GmbH
Orleansstr. 34, D-81667 München, Germany
{eckstein|stegerc}@mvtec.com

This paper presents a software architecture that is capable of easy extension and maintenance and enables the user to develop applications rapidly and in a flexible manner. This is achieved by using an object-oriented design, both for the data structures and the operators which process this data. The granularity of the operators is chosen such that they easily can be combined to solve various kinds of vision problems but on the other hand have an appropriate level of abstraction so that the user does not have to worry about low level vision. The data structures are designed such that they are easy to use but allow a high performance implementation. Based on these operators and data structures an interactive tool for rapid program development is realized. It helps the user in selecting appropriate operators in many ways. For example, the system provides context sensitive selection of possible alternative operators as well as suitable successors and required predecessors. For the task of choosing appropriate parameters several alternatives exist. For example, the system provides default values as well as lists of useful values for all parameters of each operator. To achieve this, a knowledge base containing facts about the operators and their parameters is used.

1 Introduction

Many computer vision problems can be solved by using a library of image processing algorithms in combination with simple control structures [1-5]. Examples for this class of problems are chip inspection, counting and measuring objects, and quality assessment. Many industrial vision tasks fall into this category of programs. In this domain it is essential that new applications can be developed in the shortest possible time. Also, for complex applications like aerial image interpretation or active vision, rapid prototyping is a very convenient option as a first step towards the solution. This motivates us to propose an integrated programming environment that fulfills the needs for state of the art program development. This environment is based on the HALCON image processing library which implements the operators in combination with a knowledge base containing various informations about the operators. Based on the library an interactive tool for program development called HDevelop is presented which makes use of this knowledge to help the user to rapidly implement a broad range of vision tasks.

2 Overview over the HALCON System

In this section we will give a brief overview of the HALCON image processing system that describes the structure of the operators.

2.1 System Architecture

The HALCON system is a toolbox that can be used to solve typical image analysis tasks in a problem-oriented manner in all levels of processing, ranging from image preprocessing to the final interpretation. Its main focus is the analysis of 2-dimensional images. It consists of a library containing roughly 800 image processing operators in the areas of preprocessing, segmentation, morphology, feature extraction, classification, visualization, etc., and a knowledge base that contains a high level description of each operator (see Section 2.2).

The operators are implemented in C for efficiency reasons, and have a well defined interface that takes care of correct parameter passing from the host language to the core library. Two types of parameters can be distinguished: iconic and numerical data. They can be further divided into input and output parameters, thus yielding four parameter classes. The library also serves as a database that keeps track of the iconic parameters (objects). This database is object-oriented in the sense that it presents the user with an opaque representation of the iconic objects, giving access to the actual data only through a predefined interface. The user neither has to deal with implementation details nor with data management. The image analysis operators work completely functionally, i.e., the output objects and parameters are calculated without side effects to the input objects and parameters. The database is kept in main memory to facilitate efficient access to the objects' data. Currently, the system supports a number of different object types:

- *Images* are a (sub-)set of pixels of a rectangular area. They can be interpreted as a matrix with a corresponding region that describes where the pixel values are defined. This region of interest is also used to improve the execution time of filters and segmentation procedures. Images may be of various types, including byte, short, long, float, and complex.
- *Regions* are arbitrary subsets of the discrete 2-dimensional space. They are not restricted to the domain of an image. This is achieved by implementing them using run length encoding. A

great advantage gained by this is that all morphological operations can be implemented very efficiently with no artifacts at the image borders.

- *Contours* are a collection of points that are obtained by accumulating the output of a segmentation, e.g., an edge detector, into a sequence of connected points. They can have sub-pixel accuracy.

All of these objects can be accumulated into sets of objects of one type. For example, the result of a segmentation operation is typically a set of regions, while the output of a line finder is normally a set of contours.

For numerical parameters the HALCON library uses tuples of integers, floats, and strings. Each member of a tuple can have one of these types independently of the other members of the tuple.

2.2 The Operator Knowledge Base

An integral part of the HALCON system is the operator database. The existing database describes each of the operators. Its size at present is roughly 4.5MB.

All facts about an operator are collected in one operator description. It is used for two basic tasks. The first basic task is the automatic generation of interfaces between the C library and several host languages. This means that the library can be used in very different systems. In addition, it is always state-of-the-art and can be enlarged at will.

The second basic task is the provision of knowledge about the use of the operators. This knowledge is available at system runtime and callable by the host language. In the phase of program development this means while working with an interactive development tool or environment the focus of attention lies on supporting the developer in her search for promising operator sequences and suitable parameter values.

3 Concepts for Faciliating Rapid Program Development

In this section the concepts that are necessary to fulfill rapid prototyping will be presented.

3.1 Rapid prototyping and reduction of programming time

In order to reduce the amount of time that is spent to compose a program, the effort to edit it has to be minimized. There are a number of means by which this can be achieved. First of all, the user must be able to select the operator he wants to use with minimal use of the keyboard. Therefore, the system should have menus which present the operators in a structured manner to the user. This means that operators have to be grouped into meaningful chapters and sections. Of course, every user has a different notion of meaningful chapters. Therefore, the system should allow different structuring criteria. Once the user has acquired some knowledge about the system and knows the names of the operators that are most useful to him, it is often more convenient for him to enter the name of the operator rather than to select it from the menu.

Another concept to reduce the programming time is that the system should provide reasonable names for input and output variables by default and provide the user with a list of all variable names that have been used so far, so he can quickly enter variable names into a parameter field. Furthermore, the system has to provide useful default values for each operator. Additionally, the system should automatically lay out the program text in a manner in which the structure of the program is readily apparent.

One last important point is that the system and the language it uses should be self contained and have minimal overhead.

3.2 Support in all aspects of programming

To facilitate rapid program development, the user should receive maximum support and help by the system. This means that the system has to provide useful suggestions for the operators that can be used. It must provide suggestions for possible preceding and succeding operations. For example, if the user wants to apply a dynamic threshold operation to an image, the system should list some smoothing operations, like a mean or a Gaussian mean, as a predecessor. Also, the system has to suggest possible alternatives for the currently selected operator if the user is not satisfied with the results it yields.

Once the user has selected an appropriate operator, the system should display (upon request) a list of useful values for each parameter. This serves two purposes. If the user has little knowledge about the operator, he can get an impression which values might be appropriate and can try values contained in the list as a starting point for optimal parameter selection. Furthermore, if the values that the system suggests are well chosen, the user will rarely have to try more than two or three values from the list before finding the best value for his application. This further facilitates rapid program development.

3.3 Minimization of programming errors and debugging tools

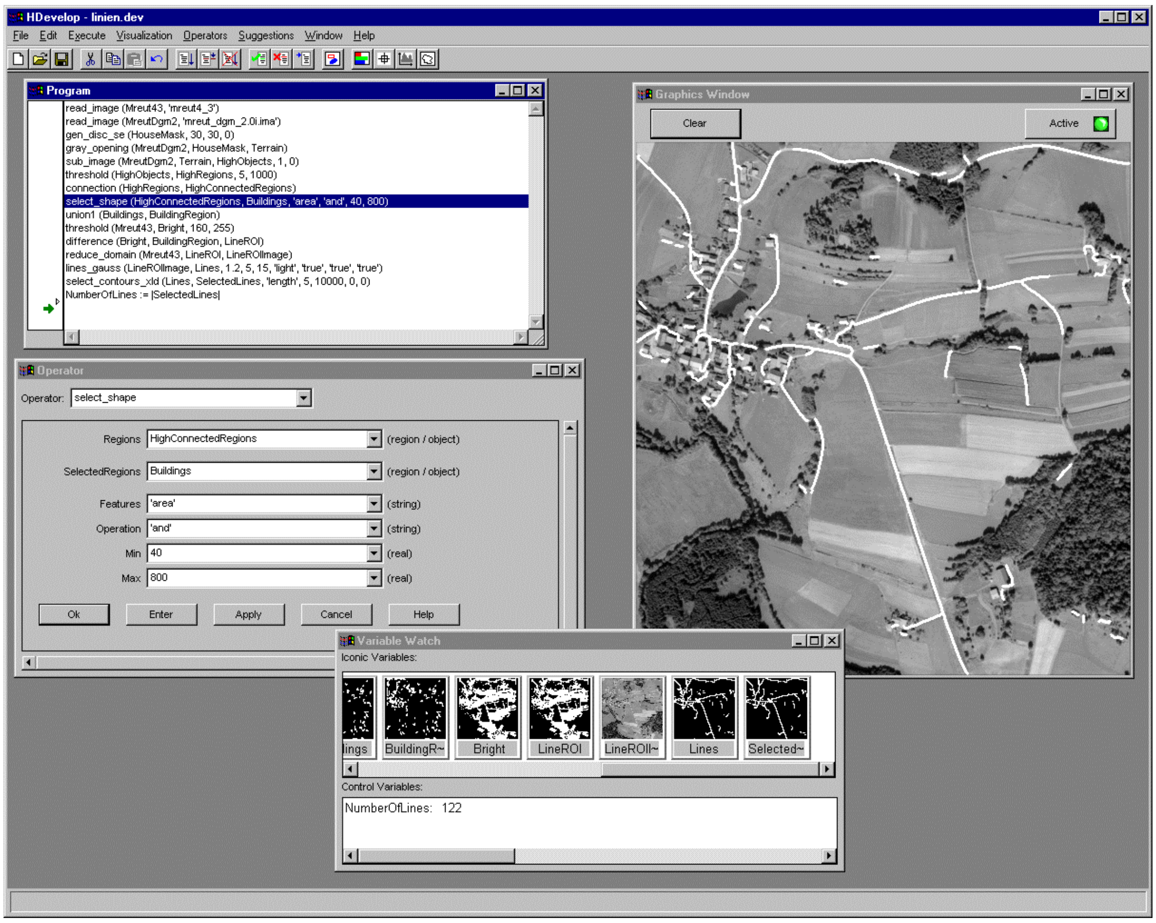
In order to minimize programming errors, two aspects are of importance. First of all, the operators should be selectable by graphical elements like menus or lists. This prohibits the user from selecting operators that do not exist. Furthermore, the parameters that the user has to enter must be presented in a structured manner. This means, for example, that each parameter should have its own data entry field and that variables and values can be selected and placed into each field through the use of graphical elements, like menus. The system then has to take care of syntactical elements like parentheses, spaces, and commas for the presentation of the program. This prohibits the user from forgetting parentheses or inadvertedly deleting a comma while editing.

To help the user to detect programming errors as quickly as possible, there has to be a direct link between the editing of the program and its execution. This means that once the user has fully parameterized an operator and finished editing, the system should immediately execute the operator and display the results of the computation.

Furthermore, in order to debug complex programs, the system must provide most of the capabilities of a good symbolic debugging tool. This means that the user must be able to set the program counter to any position he desires. Also, there has to be an execution mode in which the user steps through the program in a single step manner and a mode in which the program runs continuously. The latter mode has to be able to handle breakpoints, so the user can stop the program at certain locations. In all cases, the user must be able to change thecontents of variables and the program itself, even while it is being executed.

3.4 Development of complex applications and integration into other tools

In order to enable the development of complex applications, the language that is used must contain a complete set of execution control commands, i.e., branch constructs like *if ... then ... else* or loop constructs like *for ... endfor* or *while ... endwhile*. However, the completeness of the language used has to be taken into account. Since the user might want to execute calls to



```
read_image (Mreut43, 'mreut4_3')
read_image (MreutDgm2, 'mreut_dgm_2.0i.ima')
gen_disc_se (HouseMask, 30, 30, 0)
gray_opening (MreutDgm2, HouseMask, Terrain)
sub_image (MreutDgm2, Terrain, HighObjects, 1, 0)
threshold (HighObjects, HighRegions, 5, 1000)
connection (HighRegions, HighConnectedRegions)
select_shape (HighConnectedRegions, Buildings, 'area', 'and', 40, 800)
union1 (Buildings, BuildingRegion)
threshold (Mreut43, Bright, 160, 255)
difference (Bright, BuildingRegion, LineROI)
reduce_domain (Mreut43, LineROI, LineROIImage)
lines_gauss (LineROIImage, Lines, 1.2, 5, 15, 'light', 'true', 'true', 'true')
select_contours_xld (Lines, SelectedLines, 'length', 5, 10000, 0, 0)
NumberOfLines := |SelectedLines|
```

Operator: select_shape

Regions	HighConnectedRegions	(region / object)
SelectedRegions	Buildings	(region / object)
Features	'area'	(string)
Operation	'and'	(string)
Min	40	(real)
Max	800	(real)

Ok Enter Apply Cancel Help

NumberOfLines: 122

Figure 1: Main components of the HDevelop system

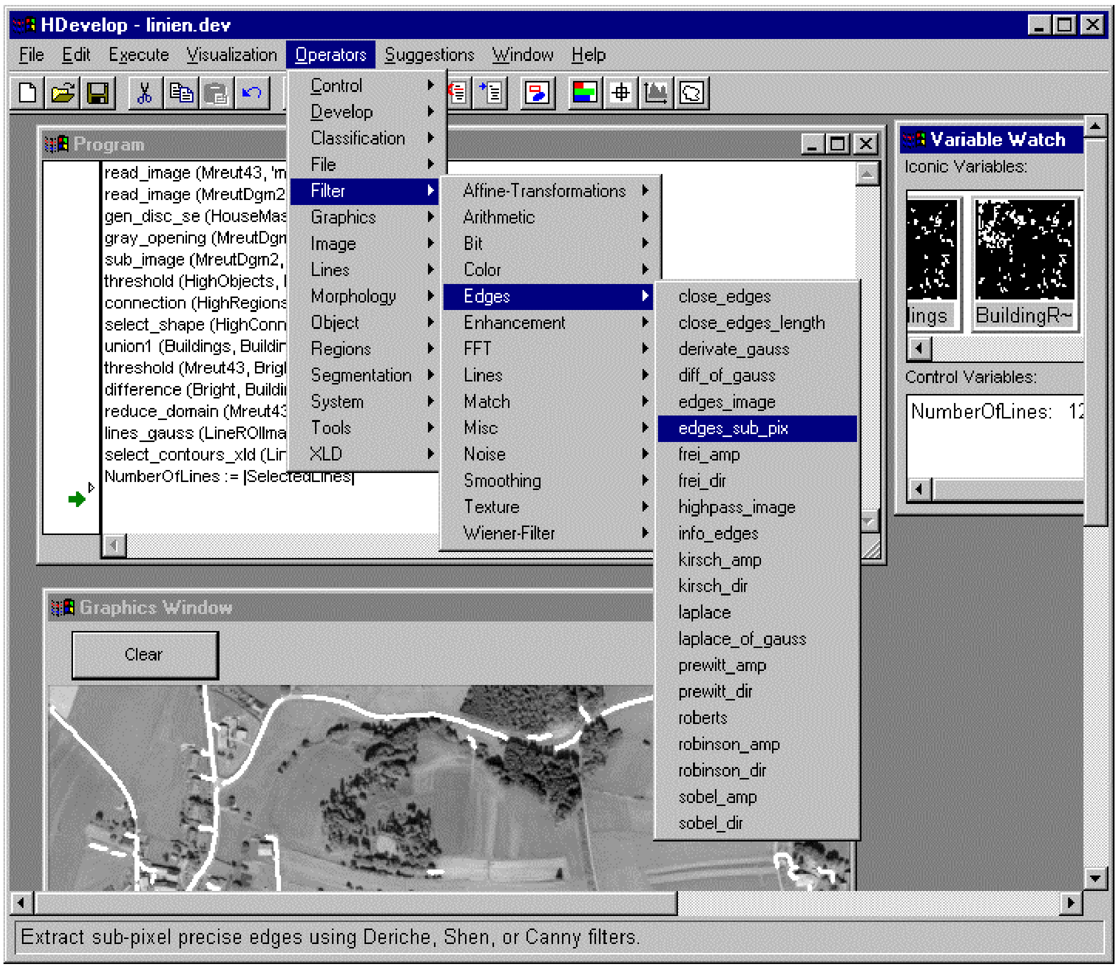
the underlying operating system or another library he is accustomed to, the desired level of completeness is very hard to achieve. Also, the user may want to integrate the program he has developed into a larger application as a subtask. Therefore the system should allow the user to save the program he has developed in the programming language of his choice.

4 Example Application

Figure 1 shows the general appearance of the system. In this case HDevelop was used to develop a program that detects roads in aerial images with a ground resolution of 2m. In this resolution roads can be modeled as lines that are brighter than their surroundings. However, buildings should not be detected by the program. From these considerations a program was developed that first extracts objects that are higher than their surroundings from a digital terrain model of the area. In the second part of the program lines that are brighter than their surroundings are extracted. The final result of the process can be seen in the visualization window in Figure 1.

The HDevelop system consists of four major windows: the program and the operator window for program development, a window for management of iconic and numerical variables, and one or more windows for visualization.

With the program and operator window, shown on the top left of Figure 1, the user can edit the program code in a textual description with the help of a graphical editor. To enter a line of code, he can select operators from the menu *Operators* or enter the operator name in a text field. The menu, shown in Figure 2, is structured in a chapter/section fashion to enable the user to select the appropriate operator quickly. By using the menu, the user is able to learn the use of the system quickly and easily. Once the user has acquired some knowledge about the system, and knows the names of the operators that are most useful to him, it is often more convenient for him to enter the name of the operator rather than to select it from the menu. In order to minimize the number of keystrokes, the system selects the appropriate operator if the user enters a substring of the name. If more than one operator name matches the name that the



```
read_image (Mreut43, 'm
read_image (MreutDgm2
gen_disc_se (HouseMas
gray_opening (MreutDgn
sub_image (MreutDgm2,
threshold (HighObjects, I
connection (HighRegions
select_shape (HighConn
union1 (Buildings, Buildir
threshold (Mreut43, Brigl
difference (Bright, Buildii
reduce_domain (Mreut43
lines_gauss (LineROIIma
select_contours_xld (Lir
NumberOfLines := |SelectedLines|
```

NumberOfLines: 1

Extract sub-pixel precise edges using Deriche, Shen, or Canny filters.

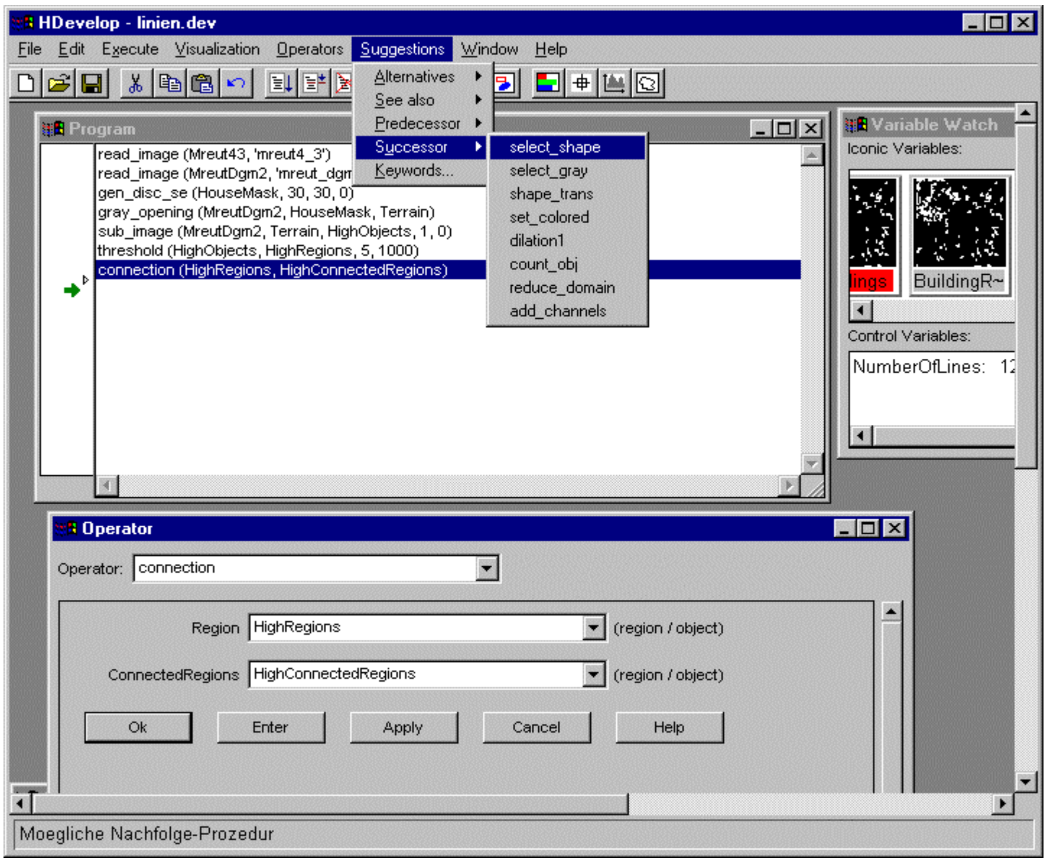
Figure 2: Chapter structure of the HALCON operators menu

user provides, he is given a list of operators that match and can select the appropriate one. Both input methods help to minimize programming errors since they prohibit the user from selecting operators that do not exist.

The *Suggestions* menu allows the user to look for other operators that might be interesting for him in the current context. It provides suggestions for possible preceding and succeeding operations. For example, if the user wants to apply a dynamic threshold operation to an image, the system lists some smoothing operations, like a mean or a Gaussian mean, as a predecessor. The system also provides possible alternatives for the currently selected operator if the user is not satisfied with the results it yields. Finally, it lists pointers to other operators that might be of interest for the user in the selected context. Figure 3 displays an example from the *Suggestions* menu. In this case the user is interested in possible successor operators to the \verb|connection| operator. These menus facilitate rapid prototyping since the user can try different operators very quickly.

Once the user has selected an operator, the parameters are presented to him in a structured manner in the parameter editing area of the main window, in which each parameter has its own data entry field. The layout of this area is determined dynamically from the knowledge about operators, including types, value lists, assertions, ranges, and default values. Consequently, no changes to the user interface have to be coded when a new operator is added. The system provides reasonable names for variables by default if possible. Furthermore, each parameter has a button associated with it, so that the user can bring up a list of variables used so far and enter or replace the current variable name. Furthermore, for most parameters list of useful values are added to the the menu. These value lists serve two purposes. If the user has little knowledge about the operator, he can get an impression which values might be appropriate and can try values contained in the list as a starting point for optimal parameter selection. Furthermore, if the values that the system suggests are well chosen, the user will rarely have to try more than two or three values from the list before finding the best value for his application. Additionally, with this scheme the user rarely has to enter data on the keyboard. Since the parameters are presented in a manner in which the user does not have to take care of the syntax of the language, programming errors that result from missing parentheses and other similar error sources are avoided. The system also takes care of the layout of the program.

Once the user has developed a program, it can be debugged through the use of the execution control



```
read_image (Mreut43, 'mreut4_3')
read_image (MreutDgm2, 'mreut_dgm
gen_disc_se (HouseMask, 30, 30, 0)
gray_opening (MreutDgm2, HouseMask, Terrain)
sub_image (MreutDgm2, Terrain, HighObjects, 1, 0)
threshold (HighObjects, HighRegions, 5, 1000)
connection (HighRegions, HighConnectedRegions)
```

NumberOfLines: 12

Operator: connection

Region: HighRegions (region / object)

ConnectedRegions: HighConnectedRegions (region / object)

Moegliche Nachfolge-Prozedur

Figure 3: Suggestions menu for the connection operator

buttons. These provide single-step and continuous execution until the user stops the program or a breakpoint is reached. Breakpoints and the program counter can be set in the area to the left of the program code window. During program execution all results are immediately visualized. Iconic data is displayed in the graphics window and in symbolic form in the variable window. Numerical data is only displayed in the variable window.

Finally, if the user is satisfied with the results of his program and needs a specialized application, e.g., a fancy user interface or a program requiring calls to the underlying operating system, he can output the program in a programming language of his choice. At the moment the system can produce Visual Basic and C++ code.

5 Conclusions

In this paper the requirements and concepts that allow the rapid development of computer vision applications have been presented. A tool HDevelop has been developed that integrates most of these concepts into a working rapid prototyping environment, and is currently in use. One additional area of improvement that will further speed up program development is the graphical input of parameters and and real-time visualization of results. For example, if the user can select the parameters of a threshold operation using sliders and the results are immediately displayed when the user moves the slider, the time to find appropriate parameters for an operator will be greatly reduced.

References

[1] Amerinex Artificial Intelligence, Inc. General support tools for image understanding. Technical report, Amerinex Artificial Intelligence, Inc., Amherst, MA, USA, 1992.

[2] Niels O. Kirkeby and Henrik I. Christensen. The vision programmers workbench (VIPWOB). In Christensen and Crowley [5], pages 195--224.

[3] Lars Olsson. Module network tool. In Christensen and Crowley [5], pages 225--252.

[4] Michael Klupsch and Wolfgang Eckstein. Object-oriented image processing in Smalltalk: Using complex operator objects. In Nagib~C. Callaos, editor, *International Conference on Information Systems Analysis and Synthesis — ISAS '96*, pages 833--839, Orlando, FL, USA, 1996. International Institute of Informatics and Systemics (IIIS).

[5] Henrik I. Christensen and James L. Crowley, editors. *Experimental Environments for Computer Vision and Image Processing*, Singapore, 1994. World Scientific Publishing.