Knowledge-Based Control of Vision Systems

C.Shekhar[1] S.Moisan[2] R.Vincent[3] P.Burlina[1] R.Chellappa[1]

1: Center for Automation Research
University of Maryland
College Park, MD 20742

2: INRIA Sophia-Antipolis
2004 Route des Lucioles
Sophia-Antipolis Cedex, France

3: Computer Science Dept.,
University of Massachusetts
Amherst MA 01003-4610

Abstract

We propose a framework for the development of vision systems that incorporate, along with the executable computer algorithms, the problem-solving knowledge required to obtain optimal performance from them. In this approach, the user provides the input data, specifies the vision task to be performed, and then provides feedback in the form of qualitative evaluations of the results obtained. These assessments are interpreted in a knowledge-based framework to automatically select algorithms and set parameters until results of the desired quality are obtained. This approach is illustrated on two real applications, and examples from the knowledge bases developed are discussed in detail.

Keywords:

vision system
knowledge-based control
self-tuning
algorithm selection
parameter tuning

1 Introduction

Vision systems used in challenging operational environments should satisfy the conflicting requirements of flexibility and convenience. Flexibility is the ability to accommodate variations in operating conditions. Convenience pertains to the ease of operation of the system by a user who is not familiar with the technical details of the algorithms employed.

Variations in image characteristics are caused by a number of factors such as weather, lighting conditions and image acquisition parameters. A vision system should accommodate a reasonable amount of such variation, and should degrade gracefully as the image characteristics deviate from the ideal. One can allow for such variations by providing alternative algorithms for each task, as well as tuning parameters for each algorithm. In most cases, a judicious choice of algorithms and parameters provides results of acceptable quality under a wide range of operating conditions.

Vision systems in the real world are often utilized by users who, while competent in the visual analysis of images, may not be familiar with the technical details of the algorithms they employ. It is not reasonable to expect the user functioning in an operational situation to select and tune the algorithms for the task (s)he is required to perform. This function is best left to the designer, the vision specialist, who may not be available during the system's operation. It is thus obvious that a vision system that provides *flexibility* in the choice of algorithms and parameter values may not be very *convenient* for the user to utilize.

In order to achieve the conflicting goals of flexibility and convenience, we propose a knowledge-based framework to partially or fully automate the reasoning employed by the vision specialist in obtaining satisfactory results from the system. The proposed framework is implemented using the PEGASE platform [22, 21]. The original vision algorithms are integrated into this framework. The integrated system, shown schematically in Fig. 1, can then be made available to the user. This type of system is capable of self-tuning, i.e. adapting to changes in data characteristics and performance requirements with minimal external intervention. Any interaction with the user is in terms of qualitative evaluation of results, and not in terms of algorithms and parameters. In many situations, the same processing task is performed on a large data set consisting of hundreds or even thousands of images. In such cases, the system can be interactively tuned on some representative images, and once satisfactory performance is achieved, can then be used with fixed settings for batch processing of the remaining images in the data set.

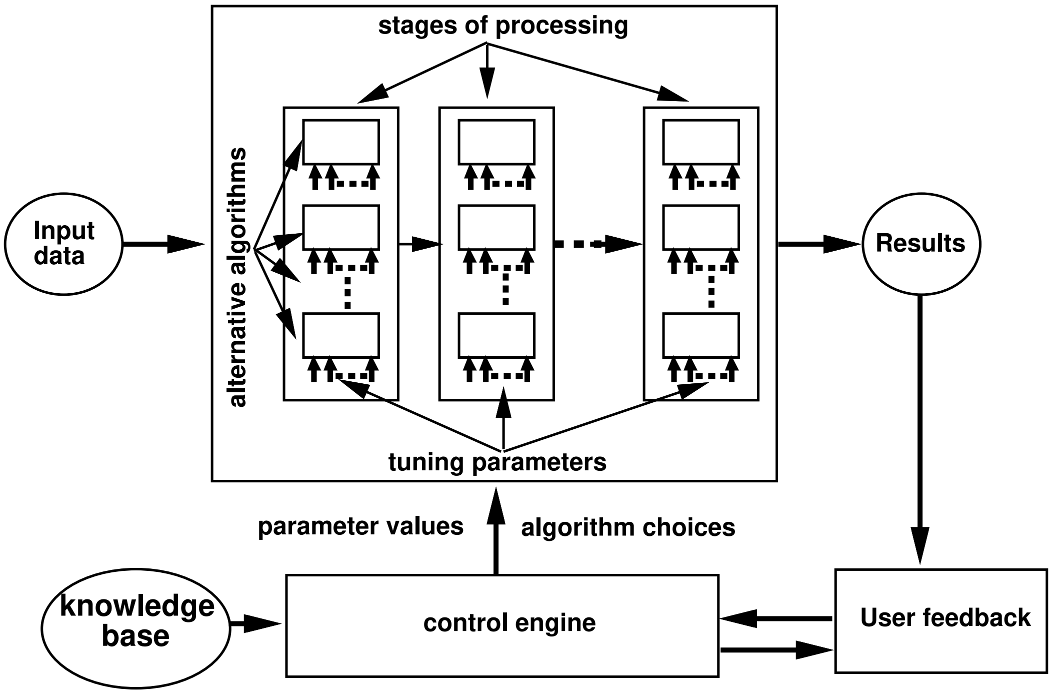


Figure 1: Architecture of a self-tuning vision system. The vision system has a number of stages of processing, with different possible algorithm choices at each step. Each algorithm may have one or more tunable parameters. The user evaluates the result(s) of processing, and the control module uses this feedback to change algorithms or parameters in order to improve the results.

The organization of this paper is as follows. Section 2 briefly reviews related work. Section 3 discusses the basic concepts of knowledge-based control. Section 4 presents the PEGASE platform. In Sections 5 and 6, results of applying our methodology to two candidate applications (Synthetic Aperture Radar (SAR) image analysis and vehicle detection in aerial imagery) are presented. These applications, developed at the University of Maryland, were selected firstly because they address interesting and non-trivial problems, and secondly because their developers were available to provide the required expertise, indispensable for constructing the knowledge bases. The final section contains the conclusions resulting from our work, as well as a list of areas for further investigation.

2 Previous work

As vision algorithms and systems have grown in power and complexity during the past few decades, there has been a corresponding growth in software platforms tailored to vision system development. Traditional methods range from graphical script generators, to vision toolkits, to object-oriented protocols for data and program interchange. The emphasis is on abstracting the types of objects and computational geometries used in image understanding into useful programming constructs [13]. One of the objectives is to enable the rapid design of algorithms using a tool-box of pre-existing constructs, and prototyping of longer processing chains by linking simpler elements together. The user is often provided with a visual programming environment (VPE), which enables him or her to mix-and-match between the available methods. For the most part, these systems offer only *syntactic integration* of vision algorithms. They provide a means to integrate code and usage syntaxes, but do not provide a means to incorporate further knowledge about the algorithms.

Knowledge-based techniques can be used in various ways in the development of vision systems. For an excellent survey of this topic, see [5]. Knowledge-based systems, also known as expert systems, have been traditionally used for the high-level interpretation of images, and for specific vision tasks such as segmentation (e.g. [14]). They incorporate mechanisms for the spatial and temporal reasoning that is characteristic of intermediate-and high-level image understanding. These knowledge-based systems are tailored towards specific tasks, and are usually not generalizable to other tasks. The work presented in this paper is different in that it proposes a general-purpose framework for knowledge-based control of vision systems. The approach can be used to develop knowledge-based autonomous or semi-autonomous systems for any vision application. This paper builds on work previously reported in [4, 18].

Some of the early work with a similar motivation is reported in [9, 15, 19, 12, 6]. More recently, this problem has been addressed in the context of the VIDIMUS project [1], with the aim of developing an intelligent vision environment for industrial inspection. A knowledge-based system (VSDE) was built within this environment, which can automatically configure a vision system for a given inspection problem. The automatic generation of an image processing script based on a user request and a knowledge-based model of an application domain is addressed in [3]. In [7] vision algorithm control is modeled as a Markov decision problem. This model is used to automatically assemble object recognition algorithms from existing vision modules. In [16, 17] a context-based vision paradigm is proposed, where the basic aim is to use contextual information to select methods and parameters in a vision application. The authors emphasize the need for explicitly encoding semantic knowledge about vision algorithms such as assumptions about their use and their inherent limitations.

The work reported in [8] is very close in spirit to the approach presented in this paper. The authors' motivation is to make it easier for non-specialists to use the SPIDER [15] image processing library. They propose a three-layered knowledge-based system: the first layer contains general knowledge about problem-solving and the use of software packages, the second contains knowledge about image processing and the third contains knowledge about the SPIDER library. The problem-solving strategy consists of problem classification, division of the problem into sub-problems, selection of appropriate methods and invocation of the appropriate SPIDER routines. Routines are automatically parameterized and executed. Intermediate results are evaluated to decide whether alternative ways of processing are preferable. They provide a backtrack mechanism to handle failures. The main limitation of their method is their reliance on the characteristics of the SPIDER library, which has been designed in a specific fashion, using principles of modularity, homogeneity and software reuse. Unlike the approach presented here, their method is not easily extensible to other algorithm libraries. Another limitation is their use of Prolog to represent all types of knowledge, whereas certain types of knowledge, such as the structure of algorithms, are better represented using frames.

3 Knowledge-based control

In a typical vision application, a number of stages of processing are involved in going from the raw input data to the final result, as shown in Fig. 1. Typically, at each stage of processing a number of alternative algorithms can be employed. Each of these algorithms, in turn, may have one or more tunable parameters. These parameters may be continuously variable, or may take discrete sets of values. Often, due to uncertainty in the data and in the problem model, it is not possible to predict beforehand if a given algorithm sequence will produce the desired result for a certain parameter setting. It may be necessary to start with a rough guess of the parameter values, execute the algorithm sequence, examine the results, and if necessary, modify the parameter values or the selection of algorithms, and repeat the procedure until results of the desired quality are obtained.

In this section, we examine the types of knowledge used by a vision specialist (and therefore required for knowledge-based control), and the implications for the design of self-tuning vision systems.

3.1 Smart modules

From a problem-solving perspective, the task of solving a vision problem using a given set of algorithms involves three types of knowledge: syntax, semantics and strategy. Syntactic knowledge is information about input and output data types and formats, input parameters, command-line arguments, etc. Information such as memory required may also be considered as

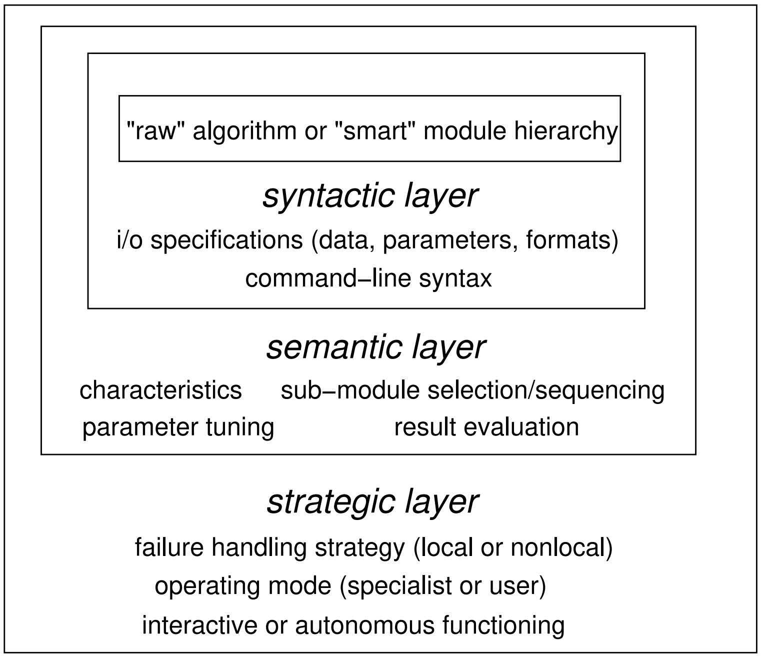


Figure 2: Packaging a vision algorithm into a "smart" module. A "raw" algorithm (in the form of an executable computer program) is not useful to anyone except the specialist who designed and implemented it. In order for another person, particularly a non-specialist, to use the algorithm, it must be accompanied by some knowledge about how to run it, how to evaluate its results, how to tune it, etc. Smart modules can themselves be connected in various ways and re-packaged for more complex tasks.

syntactic knowledge. Semantic knowledge is the vision specialist's expertise about the characteristics of the algorithms, result evaluation, assembly of algorithms for a given task, etc. Strategic knowledge pertains to the high-level decisions that should be made about result evaluation (by the system, by the user, or not at all) and failure handling (repair failure at the current module or at another place in the chain of processing, repair failure by parameter tuning or by algorithm reselection, etc.).

The objective of knowledge-based control is to provide a suitable framework for "packaging" algorithms using the three types of knowledge described above. Conceptually, as shown in Fig. 2, raw algorithms are wrapped with layers of syntactic, semantic and strategic knowledge to form "smart" modules. This can be done in a recursive fashion for more complex tasks—smart modules can be connected sequentially, or in parallel, or selectively, and the ensemble can then be re-wrapped.

3.2 Self-tuning

Self-tuning is the ability of the vision system to select the appropriate algorithms and parameter values based on the input data, contextual information, and feedback from the user in the form of result evaluations. In many cases, algorithms can be selected based on contextual information alone [16], before any data processing has taken place. Parameter tuning, on the other hand, is dependent on the characteristics of the specific data set. It has to be interleaved with the processing of the data, based on the operator's evaluation of the results. In the cases where no satisfactory result is obtained even after tuning the parameters, one or more algorithms may have to be replaced by more suitable ones.

Consider a vision system A composed of m stages, with n_i parameters in the ith stage:

$$A = A_1 A_2 \ldots A_i \ldots A_m$$

where each A_i is of the form

$$A_i = A_i(p_{i1}, p_{i2}, \ldots, p_{in_i})$$

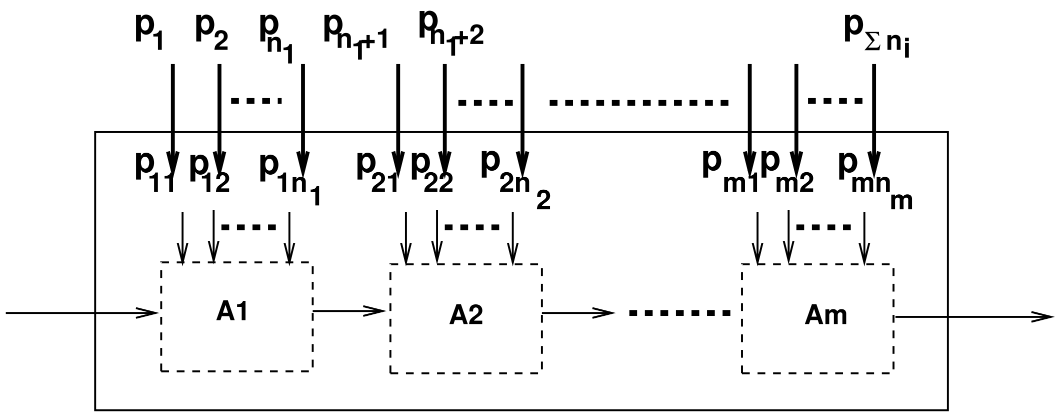


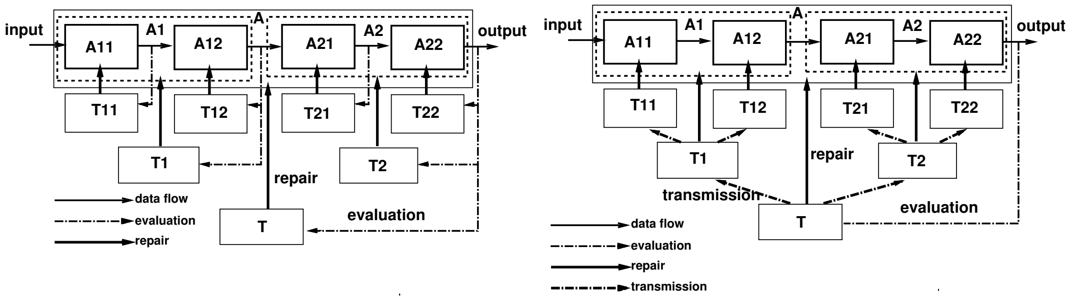
Figure 3: A simplified black-box model of a vision system consisting of stages A_1–A_m. The parameters of the individual modules become the parameters of the black box.

As shown in Fig. 3 the entire vision system can be regarded as consisting of a single algorithm $A(p_1, p_2, \ldots, p_N)$. The N parameters that tune this "black box" algorithm are the $\sum_{i=1}^{m} n_i$ parameters of the m individual stages. We refer to a specific setting of these N parameters as an operating point (OP). We define an *acceptable* operating point as a parameter setting that yields satisfactory performance for the given data set. Our basic assumption is that there exists at least one acceptable operating point (or, more generally, to account for continuous-valued parameters, operating region). In general, the default parameter setting will not be an acceptable one. The objective, then, is to be able to find an acceptable OP for the given data set in an efficient manner. In the case of discrete parameters, if each parameter p_i has k_i possible values, the total number of parameter settings is $\prod_i k_i$. An exhaustive search of the N-dimensional parameter space may therefore be too computationally expensive. The situation is even worse with continuously-valued parameters, since there are an infinite number of potential parameter settings. Further, it is bound to be tiresome for the user who has to evaluate the results at each step and provide feedback. This is where it is advantageous to use a knowledge-based approach. A vision specialist employs a problem-solving technique consisting of heuristics, rules of thumb, etc., which effectively help him or her find a short cut ("tuning path") to an acceptable operating point, without having to explore the entire parameter space. If there are multiple operating regions we need to be able to find a tuning path to an acceptable operating point lying in any one of them. Using a knowledge-based approach, the technique employed by the specialist can be integrated into the vision system to give it a self-tuning capability.

3.3 Modes of operation

Any vision task A can be hierarchically decomposed into a set of subtasks A_1–A_m, each of which (say A_i) may be decomposed further into subtasks A_{i1}–A_{im_i} and so on. For instance, a typical vision system may consist of a top-level module A, consisting of sub-modules A_1 and A_2, and these may be composed of elementary subtasks A_{11}, A_{12}, A_{21}, and A_{22}. In our model, attached to each (sub)task $A_{\ldots}$ at any level in the hierarchy is a "control method" $T_{\ldots}$, an abstract entity which represents the ensemble of the specialist's semantic and strategic knowledge about the module: how/when it should be used, how to evaluate its performance, and what should be done if improved performance is required. Depending on the control methods available, this type of a system can function in one of two modes: the specialist mode or the user mode.

In the specialist mode, shown in Fig. 4(a), all the control methods are available at every level of the hierarchy. In other words, results at every stage, including the intermediate ones, are available for evaluation by the specialist. This is applicable to the test phase when the specialist is in the process of testing the functioning of the system. In the user mode, shown in Fig. 4(b), only the top-level control method T is directly available and only the results of the final stage are available for evaluation by the user. This mode is designed for the operational phase. From a control-theoretic viewpoint, a self-tuning vision system can be regarded as a closed-loop system where the observer is the user, who provides feedback in the form of result evaluations. This feedback can be either at the highest level (corresponding to the user mode), or at all levels (corresponding to the specialist mode).



(a) Specialist mode (b) User mode

Figure 4: Tuning modes for a three-level vision system. (a) The results of the modules A, A_1, A_2 and A_{11}–A_{22} and the corresponding control methods T, T_1, T_2 and T_{11}–T_{22} are available. (b) Only the results of the top-level module A and the corresponding control method T are directly available. The control methods T_1, T_2 and T_{11}–T_{22} are available only indirectly through message transmission.

In the specialist mode, the availability of intermediate results enables linear planning for the fine-grain local optimization of algorithms. The user mode, however, necessitates the use of more complex reasoning, since only the final result is available, based on which any algorithm may have to be retuned at any stage of the processing.

3.4 Result evaluation

In the user mode, the vision system is tuned based on the evaluation of results by the user. We can define two types of result evaluation: general and specific. General evaluation consists of qualitative global judgments about algorithms and results ("too many false alarms", "too many missed detections at road intersections", etc.). Specific evaluation, on the other hand, pertains to particular objects or regions in the result ("this airplane is a false alarm", "this portion of the image contains too many false alarms", etc.). It can be qualitative or quantitative in nature. An example of each type of

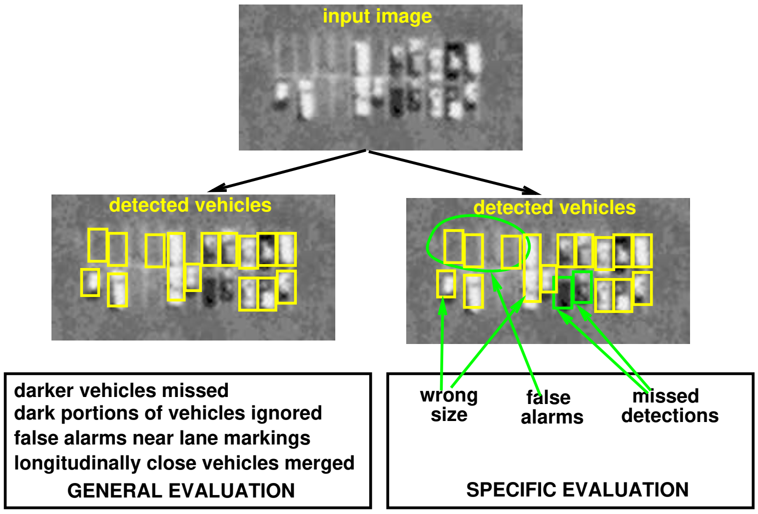


Figure 5: Example of the two types of result evaluation. The application here is the detection of vehicles in an aerial image. The user can provide feedback either in the form of general remarks about the overall result, or in more specific terms.

evaluation is shown in Fig. 5. Both types of evaluations have their advantages and drawbacks. Specific evaluations are simple, intuitive, and precise. The user is not expected to make any detailed analysis of the results. Further, for certain applications such as target detection, specific evaluations provide a very rich form of feedback to the control engine. However, they are difficult to deal with in a general-purpose framework, since they require the software platform to have special features (e.g. GUI components) that are application-dependent. This makes separation of the platform architecture from the application more difficult. General evaluations are of a more complex nature than specific evaluations, since they require more detailed feedback from the user on the types of errors present in the output. However, they are more suitable for a general-purpose framework, since they require only generic, application-independent mechanisms. In this paper, we deal primarily with general evaluations.

4 The PEGASE platform

PEGASE, which is a successor of OCAPI [4], is a general-purpose platform for algorithm control (also termed "program supervision"), consisting of a kernel library, a knowledge base description language called YAKL, verification and validation facilities, and other tools. A GUI is provided for examining knowledge bases as well as for monitoring applications during their execution. A complete description of this platform is beyond the scope of this paper; the interested reader is referred to [22, 21]. A number of control engines with different characteristics are possible in this framework. In this work, we use a control engine that provides hierarchical and skeletal based planning interleaved with control of execution [22]. It has been previously used to model the use of astronomical image processing algorithms [23].

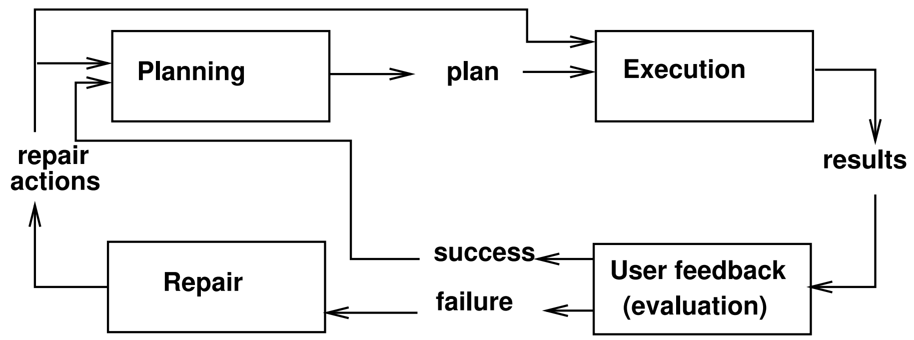
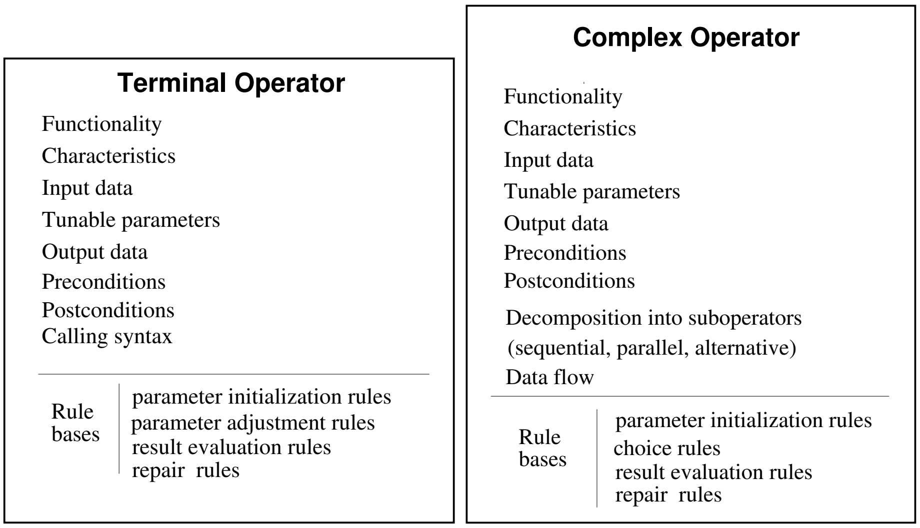


Figure 6: Functioning of the control engine

The functioning of the control engine (Fig. 6) can be decomposed into a number of phases: *planning* and *execution* of algorithms, *evaluation* of the results, and *repair*. The planning step first builds a plan, or part of a plan, which is then executed. The results of execution are then assessed in the evaluation step. If the assessments are positive, the planning process continues. If failures are detected, the repair step invokes the appropriate remedial measures which may either result in re-execution or re-planning of any part of the hierarchy.

Vision applications typically consist of a number of executable algorithms



(a) Terminal operator (b) Complex operator

Figure 7: Structure of operators in PEGASE. There are two types of operators, terminal and complex. (a)Terminal operators are "packaged" forms of executable computer algorithms, while (b) complex operators have an associated decomposition.

with associated syntactic, semantic and strategic knowledge, as described in Section 3.1. The mechanisms for planning, evaluation and repair provided by the control engine enable us to package a given vision system into an ensemble of smart modules, enabling the user to obtain optimal performance with minimal intervention. The planning, evaluation and repair steps are discussed in greater detail in the following sections.

4.1 Planning

A vision application developed using PEGASE consists of a set of pre-existing algorithms (in the form of computer programs), a knowledge base (KB) on the use of these algorithms, and a control engine. A vision *functionality* is the abstract representation of a vision task. It is realized in a concrete form by one or more *operators* corresponding to it. An operator may be either terminal, corresponding to an executable algorithm, or complex, represented by a predefined skeletal plan. A skeletal plan describes a network of connections between operators (choice, sequence, repetition, etc.) for realizing a given functionality. The description of an operator contains information about its arguments (name, type, range, default values, etc. of the input data, output data and parameters), semantic information about its applicability (in the form of pre- and post-conditions), as well as criteria for parameter initialization, result evaluation, etc. For operators corresponding to real executable algorithms the calling syntax is also provided. The structure of operators is shown in Fig. 7.

The basic planning mechanism used in PEGASE is similar to the block schematics often used to represent signal processing algorithms, but is different in two ways:

(a) Plan hierarchy: each "box" in the plan can be a complex vision task, with its own plan,

(b) Plan abstraction: the components of the plan represent abstract vision tasks, and not specific algorithms. During execution, *choice rules* are used for selection of the operator best suited for the task at hand. They are of the form:

```
if
            the data have property x
            AND context field f has value y
            AND the request has a constraint z
then
            choose an operator with characteristic b
            AND do not choose operator w
```

The above example shows only some of the possible types of premises and actions in a choice rule. Many other types of premises and actions are possible, for choice rules as well as for the other kinds of rules described later.

Once an operator is selected, it is initialized using *initialization rules*, of the form:

if

    the data have property x
    AND context field c_f has value y

then

    set parameter p to value $f_x(y)$

4.2 Evaluation

The performance of an operator is analyzed using *evaluation rules*, which are of the form:

```
if
            the output does not satisfy criterion c
then
            judge plan entity P to have quality q
            declare failure
```

The "plan entity" could be the current operator, a result of processing, or in general, any plan element in the hierarchy. Many other types of evaluation are possible, using information about the previous evaluation, about the number of times the operator has been (unsuccessfully) executed, about whether the result is better or worse than before, etc.

4.3 Repair

Failure handling is performed either locally inside the operator, or nonlocally by message transmission to another part of the plan.

Local failures are handled by parameter tuning using *repair* and *adjustment rules*. A repair rule for local failure-handling is usually very simple, of the form:

```
if
            true
then
            re-execute
```

When this rule is applied, the operator is re-executed, invoking one or more adjustment rules of the form:

```
if
            result r has quality q
then
            adjust parameter p using method m
```

When the failure of the current operator is caused by the (undetected) failure of another, the repair mechanism employs message transmission from the current operator to the offending one. The repair rule in this case is of the form:

```
if
            result r has quality q
then
            send message m to operator o
```

The control then switches to operator o, whose failure-handling mechanism is then activated, based on the message m that has been sent to it.

4.4 Integrating an application into PEGASE

The following two subsections describe how to build a knowledge base (KB) for an application, and how to run the integrated application.

4.4.1 Building a knowledge base

The first step is the construction of a set of skeletal plans for the application. A plan consists of an operator hierarchy, where the top-level operator solves a given vision task. The task performed by this operator is then recursively decomposed into sub-tasks, each with an associated complex or terminal operator, until the lowest levels of the operator hierarchy contain only terminal operators. Complex operators have an operator decomposition associated with them, whereas terminal operators specify the calling syntax of the associated algorithm. A goal instance is created corresponding to the top-level operator functionality.

The next step is the creation of the rules which express the approach used by the specialist to execute a plan. The knowledge structures provided by PEGASE enable the systematic expression of the "rules of thumb" and "approximate reasoning" which are crucial to successful problem-solving. This is done by means of the five types of rules discussed earlier. A mixture of numeric and symbolic reasoning may be required in formulating them. Choice rules, being the simplest, are defined first. They use the values of context fields, data definitions, constraints in the request, etc. to select an operator from among the choices available. Initialization rules are defined next. These may be somewhat more difficult, in that they have to formalize the "rough initial guesses" that the specialist makes before starting an vision task. Adjustment rules are then defined for operators which have adjustable parameters. Step sizes for parameters have to be carefully chosen so that the change in the behavior of the algorithm is neither too sudden nor too gradual. Then evaluation rules are defined. This is a rather difficult task, since the question "Are the results good enough?" is often subjective, and appropriate quality measures may not be readily available. However, a close examination of the specialist's problem-solving technique often reveals hidden reasoning capable of being expressed in concrete terms as evaluation rules. Finally, the repair rules are defined. These rules determine the overall failure-handling mechanism, and are crucial to the success of the application.

4.4.2 Running an application

The solving of a problem starts with the creation of a request, which states the functionality, the data on which this is to be achieved, and the context in which the problem is being solved. Using choice rules, the available operators for the functionality are rank-ordered. The best operator is selected, and executed on the given data after the operator's initialization rules have been applied. If the operator is a terminal one, this corresponds to the execution of the corresponding algorithm. If it is a complex one, its children are pushed onto an execution stack, to be examined in turn and executed depending on their sequencing. If the mode of execution control calls for it, the results of executing an operator are judged using evaluation rules, and in the event of a failure, either the same operator is re-executed after its parameters have been adjusted using the relevant rules, or the next best operator is applied. In the event of successful execution, the next operator in the sequence is selected for execution. This continues until the execution stack is empty. It is to be noted that during this whole process, the knowledge base is not modified at all, i.e. there is no learning taking place. Only the choice of algorithms and parameters is modified. The knowledge base is created off-line, and provided at the outset.

The next two sections illustrate our approach using two real applications. The first application, synthetic aperture radar (SAR) image analysis, illustrates the specialist mode of self-tuning, while the second application, vehicle detection, illustrates the user mode.

5 Example: SAR Image Analysis

This application is based on the work of Kuttikkad and Chellappa [10]. Its goal is to analyze a SAR image to identify objects such as targets, buildings, roads, and trees, and to segment the remaining parts of the image into various categories such as grass, water and bare ground. The first step is the detection of regions of high backscatter using a constant false alarm rate (CFAR) technique [10]. In the next step non-target pixels in the image are classified as grass, tree, bare ground, road or shadow using a maximum

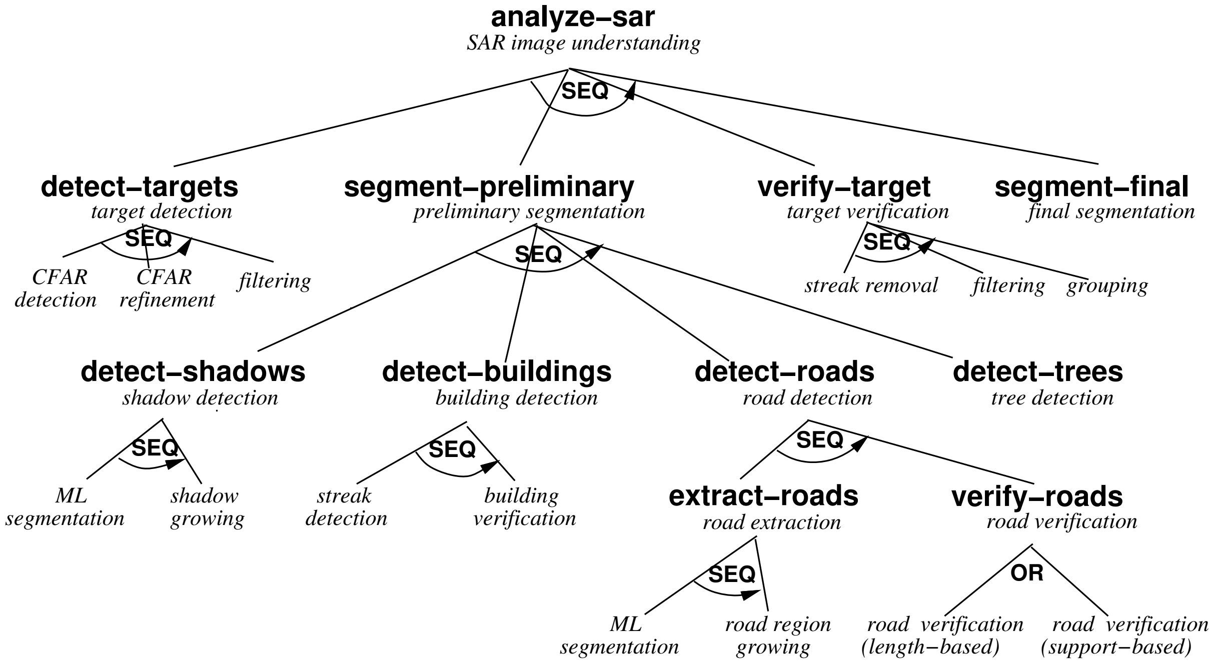


Figure 8: Operator hierarchy for SAR image analysis. The functionality corresponding to each operator is shown in italics. To avoid clutter, not all the operator names are shown.

likelihood (ML) approach. Training data obtained from other images of similar scenes is employed. This is a preliminary classification, using no high-level information whatsoever. A large percentage of the pixels are likely to be misclassified.

Shadow regions are detected and then eroded and grown using morphological operations. The same is done for pixels classified as road. Very small regions of either class are eliminated, and the rest are grouped into homogeneous regions. Shadow regions that are adjacent to a bright streak (in the CFAR output) towards the sensor are classified as building shadows. Roads are verified using a shape/size criterion. ML segmentation is then repeated for pixels previously misclassified as road and shadow, this time classifying them as grass, bare ground or trees. Tree regions are grown using morphological operations, and verified using a size argument, as well as by the presence of adjoining shadow regions away from the sensor. In the CFAR output, streaks corresponding to buildings are eliminated, and the remaining target pixels are grouped into clusters. In the final step, ML segmentation is repeated, and based on the previous steps, pixels misclassified as shadow, road or tree are re-classified into grass or bare ground.

Sample results obtained by the approach are shown in Fig. 9.

Knowledge base

The hierarchy of operators for this application is shown in Fig. 8. The knowledge base consists of the following major components: 21 operators (13 terminal and 7 complex), 18 sequential and 2 choice links, and a total of 29 production rules (2 choice, 10 initialization, 6 evaluation, 6 adjustment and 5 repair). Some simple examples from this KB are presented to give the reader a feel for the kinds of objects and reasoning involved in a real application.

An example of a complex operator is shown in Fig. 10. The functionality is road verification, which verifies road hypotheses obtained by pixel classification and region growing. This operator has a decomposition into a choice between two terminal operators, one of which is shown in Fig. 11. The choice rules for deciding which operator to select are shown in Fig. 12.

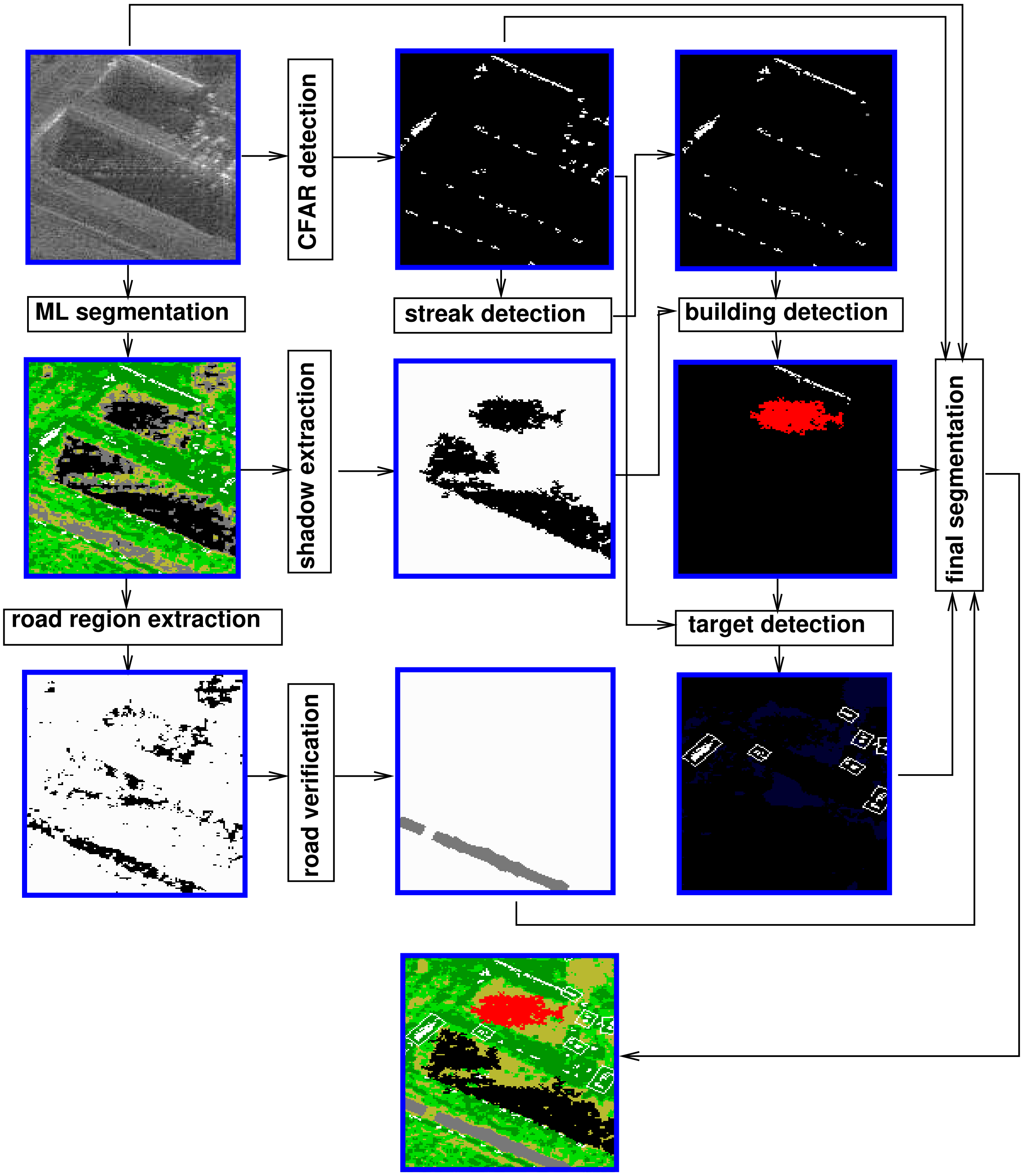


Figure 9: Sample results of SAR image analysis.

```
Complex Operator
        name :  verify-roads
     comment :  verify road hypotheses obtained from the previous steps

Functionality :  road-verification

Input Data
PGM-Image name :  road-region-image
   comment :  Binary PGM image file with road areas in black

Output Data
PGM-Image name :  road-image
   comment :  Output PGM image file for storing detected shadows

Preconditions
   valid  road-region-image

Postconditions
  valid  road-image

Body
 verify-road-1 OR  verify-road-2 ;

Choice criteria
...

Data Flow
 verify-roads.road-region-image/ verify-road-1.road-region-image
 verify-roads.road-region-image/ verify-road-2.road-region-image
 verify-roads.road-image/ verify-road-2.road-image
 verify-roads.road-image/ verify-road-1.road-image
```

Figure 10: Example of a complex operator. In this and other examples from the KB, the following font convention is used: constructs from the KB definition language (YAKL) are shown in normal typewriter font, objects defined in the KB in boldface, and comments and literals in italics.

The reasoning is as follows: in rural areas, roads are likely to have longer unbroken stretches, whereas roads in urban areas have many intersections. Hence an operator for verifying road hypotheses in rural areas should use length as a criterion.

The rules for initializing the parameter "PFA" (probability of false alarm) for the operator "detect-targets" are shown in Fig. 13. This operator is used to detect targets in a SAR image. The parameter PFA is a threshold which determines the number of bright pixels that are classified as target pixels. The higher the PFA, the more likely it is that a given pixel is classified as a target pixel. The rules for evaluating this operator are shown in Fig. 14. The user is asked to judge if the results of target detection are satisfactory. If not, appropriate action is taken via adjustment rules, such as shown in Fig. 15. An example of the entire chain of reasoning is shown in Fig. 16.

6 Example: Vehicle detection

The methodology discussed in Section 4 has also been applied to the detection of vehicles in aerial imagery. We have used a simplified version of the vehicle detector developed at the University of Maryland [2], which detects and approximately localizes vehicles of a specified size and orientation. Our objective here is to demonstrate the application of the approach presented in this paper to a simple existing vision application. We do not attempt to improve on the underlying application itself in any way. The main stages of processing, shown in Fig. 17, are as follows.

Preprocessing: Edge pixels are extracted using the Canny edge detector. Both gradient magnitude and gradient direction are computed.
Hypothesis generation: A modified generalized Hough transform (GHT) is used to locate areas corresponding to centers of candidate vehicles. Edge pixels vote for all possible centers of vehicle contours which contain the pixel. The votes are collected in an accumulator array, and thresholded. The result is a set of hypothesized vehicle centers. Local "rubber-band" contour matching is subsequently applied to reject candidate vehicles which

```
Terminal Operator
   name :  verify-road-1
comment :  Extended fill

Functionality :  road-verification

Characteristics :  length-based

Input Data
PGM-Image name :  road-region-image
      comment :  Binary PGM image file with road areas in black

Input Parameters
Float name :  aspect
      comment :  Aspect ratio threshold for road segments
      default :  2.
Float name :  fill-ratio
      comment :  Fraction fill of bounding rectangle for valid road
      default :  .6
Integer name :  min-road-length
      comment :  min road length in pixels
      default :  25

Output Data
PGM-Image name :  road-image
      comment :  Output PGM image file for storing verified roads
     I-O relations :
       road-image.path :=  road-region-image.path,
       road-image.basename :=  road-region-image.basename,
       road-image.extension :=  .roads

Preconditions
  valid  road-region-image

Postconditions
  valid  road-image

Call
 language :  shell
 syntax :   rdtwo  road-region-image.get-filename  road-image.get-filename
               -a  aspect   -r  fill-ratio   -l  min-road-length
 program name :  rdtwo
```

Figure 11: Example of a terminal operator

```
Rule
  name :  ch-verify-road-1
  comment :  Roads in rural areas have longer unbroken stretches,
               since there are fewer intersections
if
           context.scene-type ==  rural
then
          use-operator-of-characteristic  length-based

Rule
  name :  ch-verify-road-2
if
           context.scene-type ==  urban
then
          use-operator-of-characteristic  support-based
```

Figure 12: Examples of choice rules

```
Rule
  name :  init-PFA-1
  comment :  Initialize PFA heuristically for low noise
if
           context.noise ==  low
then
           PFA  :=  .001

Rule
  name :  init-PFA2
  comment :  Initialize PFA heuristically for high noise
if
           context.noise ==  high
then
           PFA  :=  .0001
```

Figure 13: Examples of initialization rules

```
Rule
  name :  ev-target
  comment :  Ask the operator to judge if
               the number of target pixels looks ok
if
          true
then
          assess-data-by-user  cfar-image
          ( num-target-pixels-correct num-target-pixels-too-low
                                      num-target-pixels-too-high)

Rule
name :  ev-target2
  comment :  if the number of target pixels is not ok, repair
if
          not assess-data? cfar-image  num-target-pixels-correct
   then
          assess-operator failed repair
```

Figure 14: Examples of evaluation rules

```
Rule
  name :  adj-PFA-1
  comment :  If # target pixels is too low, increase PFA
if
           assess-operator?  detect-targets  detect-PFA-too-low
then
           adjustment-method  PFA percent-float,
           adjustment-step  PFA  400,
           increase  PFA

Rule
  name :  adj-PFA-2
  comment :  If # target pixels is too high, reduce PFA
if
          assess-operator?  detect-targets  detect-PFA-too-high
then
          adjustment-method  PFA percent-float,
          adjustment-step  PFA  80,
          increase  PFA
```

Figure 15: Examples of adjustment rules

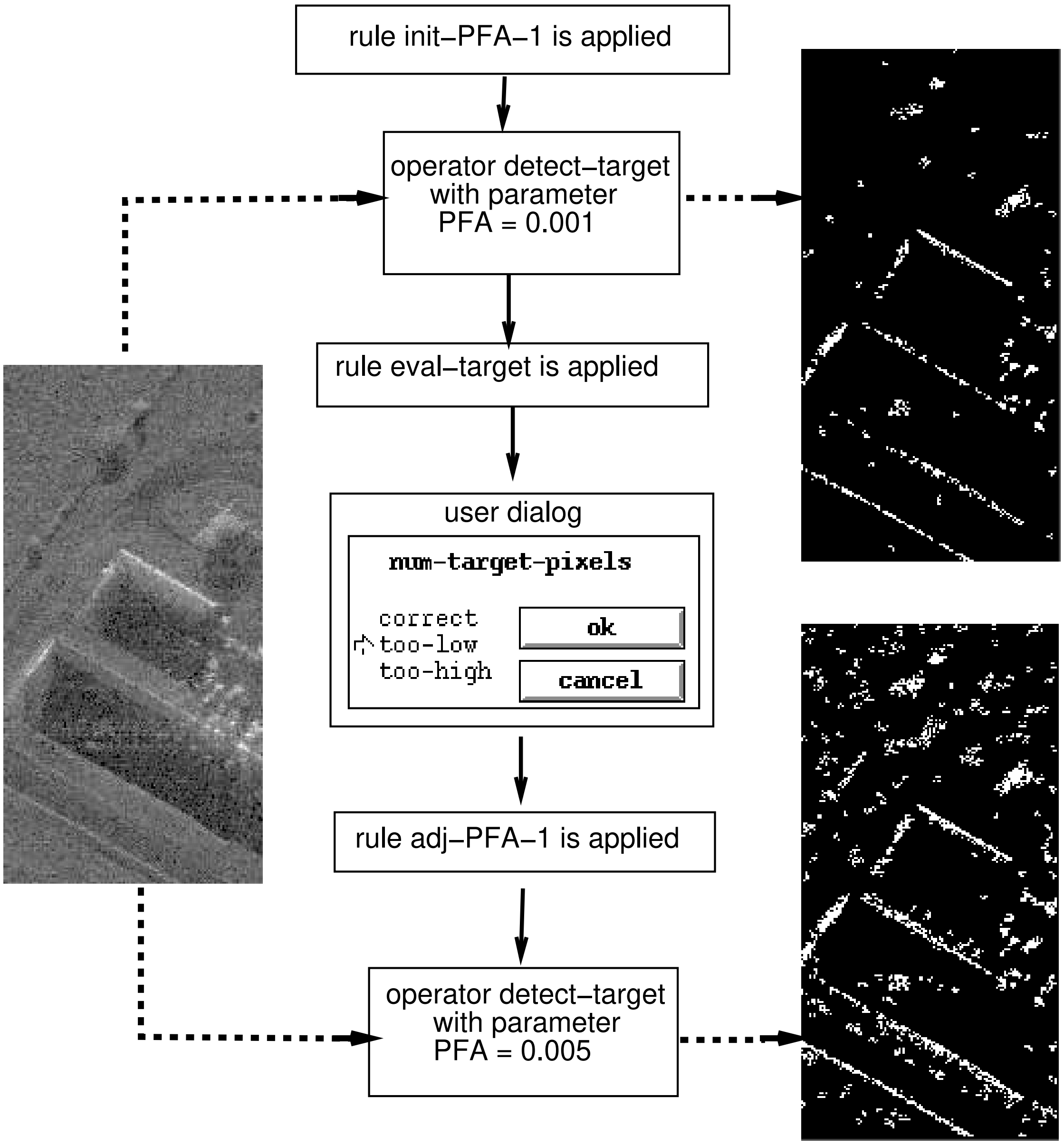


Figure 16: Example of the reasoning used by the control engine. The initial choice of the parameter PFA is 0.001. The operator "detect-targets" is executed, and the user examines the result and judges the number of target pixels to be too low. The parameter is then adjusted to 0.005, and the operator is re-executed.

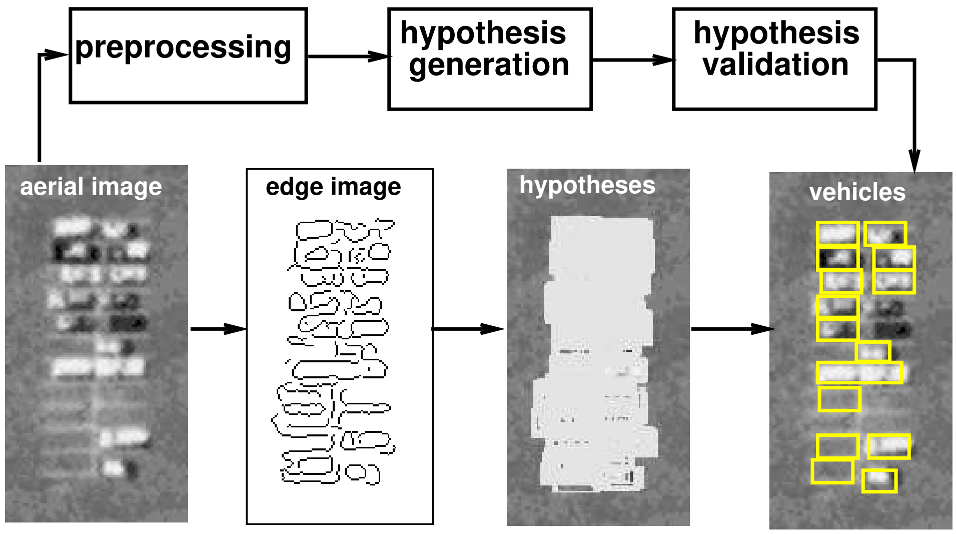


Figure 17: A simplified version of the UMD vehicle detector. The input is an aerial image. Contours are extracted, and vehicle hypotheses are generated. These hypotheses are then validated to obtain the final result.

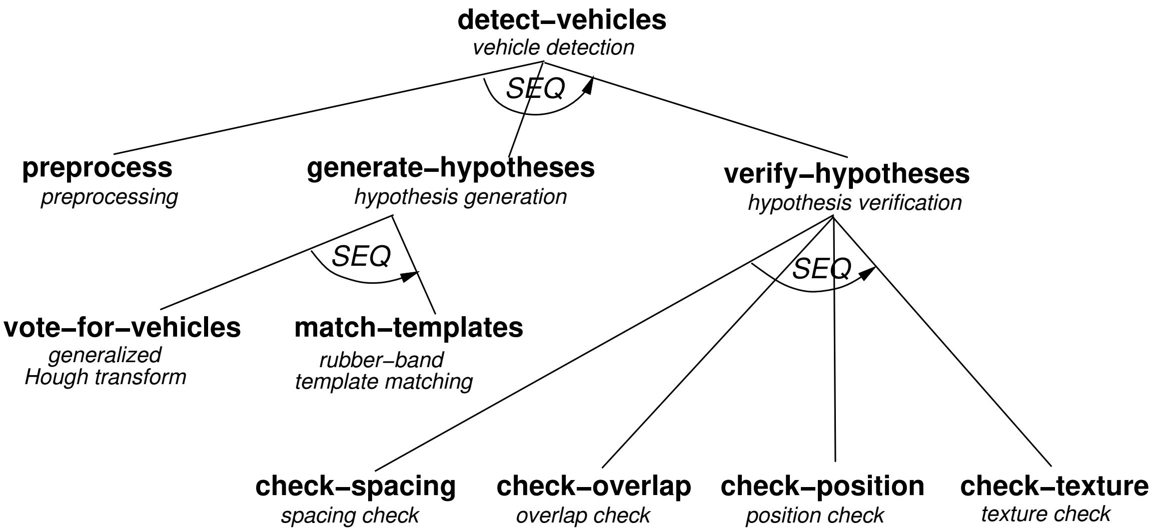


Figure 18: Operator hierarchy for the UMD vehicle detector. The functionality corresponding to each operator is shown in italics.

do not have sufficient support boundaries on both sides of the vehicle.

Hypothesis verification: This stage eliminates false matches by resolving spatial conflicts (overlaps) and by rejecting low-contrast candidate vehicles. This is done in four steps. In the first step, the conflict resolution is done purely on the basis of the distances between the centers of candidate vehicles. If two candidate vehicles are closer than a certain fraction of their width, the one with the greater boundary support is retained. The second step uses the size of the overlap area between two conflicting vehicles as a criterion for rejecting the weaker vehicle. In the third step, the longitudinal distance between adjacent vehicles lying on the same axis is used as a filtering criterion. The final step relies on the textural contrast between candidate vehicles and the background.

6.1 Knowledge base

The operator hierarchy for the vehicle detector is shown in Fig. 18. Currently, the knowledge base consists of 13 operators (4 complex and 9 terminal), 10 sequential operator links, two choice links, and a total of 37 rules (2 choice, 1 initialization, 11 assessment, 8 adjustment and 15 repair).

6.1.1 Evaluation and repair mechanisms

As in any target detection application, there are two principal types of errors: missed detections (MDs) and false alarms (FAs). The general objective is to reduce both types of errors as much as possible. In practice, some tradeoff is made between the MD rate and the FA rate. Currently, the user is asked to choose between the responses MD (too many missed detections), FA (too many false alarms) and OK (results are satisfactory). If errors of both kinds are simultaneously present, as is usually the case, the user selects the error which is more significant. If the response is not OK, the user is further queried about the type of MD or FA, as shown in Fig. 19. Currently, MDs due to the following three situations are recognized: vehicles too large or too small, vehicles having low contrast, vehicles too tightly packed. Four types of FAs are handled: multiple hits from the same vehicle, false positives at control lines (the lines used to demarcate the different parking spots), puddles/oil stains mistaken for vehicles, and FAs due to pavement texture. Extensive testing on a diverse set of aerial images will enable us to create a richer taxonomy of errors.

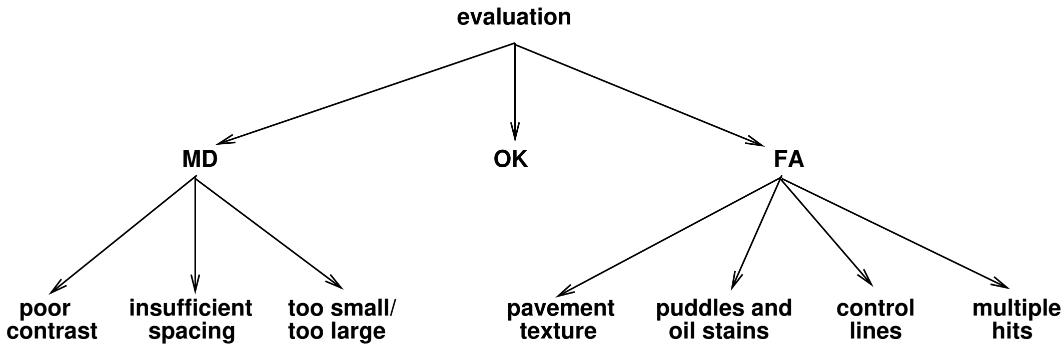


Figure 19: User evaluation of the UMD vehicle detector. The user looks at the result, and judges if there are missed detections (MDs) or false alarms (FAs), and if so, of what type.

The repair mechanism has a nested structure, and is interleaved with the evaluation mechanism. For every allowed error subtype there are one or more repair methods, which are tried one after the other until either they are exhausted or the error disappears. An example is shown in Fig. 20. The first run of the system produces the result shown in the first box in the second row. Evaluation rules are fired, asking the user to judge this result. The user indicates that there seem to be false alarms, possibly due to control lines on the parking area. This triggers repair rules, which successively transmit this judgment to the operator "verify-hypotheses" and then to "check-texture", which is responsible for filtering out such false detections. An adjustment rule in "check-texture" is then applied, adjusting a parameter "contrast-threshold" from 5 to 15. This parameter is the minimum allowable difference in mean grey level between a candidate vehicle and its background. Raising this threshold eliminates the false alarms caused by the control lines, producing the result shown in the first box of the third row. The user evaluates this result again, judging it to have too many missed detections, possibly due to the vehicles being too close together. This judgment triggers a sequence of repair and adjustment rules, resulting in the decrement of a parameter which specifies the factor by which vehicles have to be scaled before deciding that they overlap. This allows candidate vehicles spaced closer together to be judged as non-overlapping, and hence acceptable. The final result, which the user judges to be "OK", is shown in the first box on the last row.

At the end of this process, we are left with a parameter setting which is optimal for input images with characteristics similar to the one used in Fig. 20. If this parameter setting is used on a very different type of input image, we may not obtain comparable results. In Fig. 21, we show the results on another input image using this parameter setting. This input image (Fig. 21 (a)) has a different texture, and the vehicles are oriented differently. As a result, the result of the first trial (Fig. 21 (b)) is unsatisfactory. As in the previous case, the user judges that there are missed detections due to the vehicles being too close together. This triggers the same repair/adjustment sequence as before, producing the result in (Fig. 21 (c)).

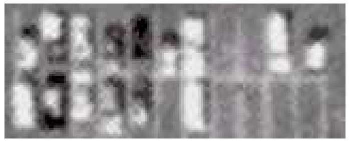
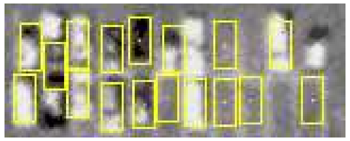
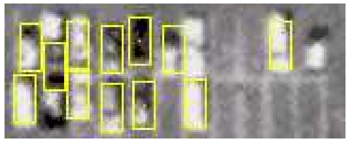
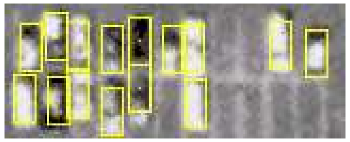
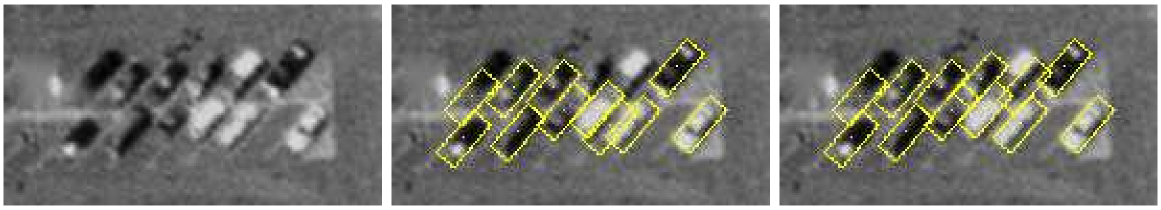
data/result	user feedback	some of the rules applied	actions taken
input image 		**choice rules** **initialization rules**	based on contextual information, appropriate operators are selected and initialized. The selected operators are executed (**result #1** is obtained)
result #1 	**false alarms** **(control lines)**	**(repair rule of operator detect–vehicles)** *if there are FAs due to control lines, inform operator* **verify–hypotheses** **(repair rule of operator verify–hypotheses)** *if there are FAs due to control lines, inform operator* **check–texture** **(adjustment rule of operator check–texture)** *if there are too many FAs, increase parameter* **contrast–threshold**	The parameter "contrast–threshold" of the operator "check–texture" is increased from 5 to 15. This makes it easier to reject low–contrast candidates. The operator "check–texture" is re–executed. (**result #2** is obtained)
result #2 	**missed detections** **(insuffcent spacing)**	**(repair rule of operator detect–vehicles)** *if there are MDs due to insufficient spacing, inform operator* **verify–hypotheses** **(repair rule of operator verify–hypotheses)** *if there are MDs due to insufficient spacing, inform operator* **check–overlap** **(adjustment rule of operator check–overlap)** *if there are too many MDs, decrease parameter* **overlap–threshold**	The parameter "overlap–threshold" of the operator "check–overlap" is decreased from 1.0 to 0.8. This allows candidate vehicles to be more densely packed Operators "check–overlap"and "check–texture" are re–executed. (**result #3** is obtained)
result #3 	**OK!**		

Figure 20: Example of the execution-evaluation-repair cycle.



(a) Input image (b) Trial 1 result (c) Trial 2 result

Figure 21: Results obtained by using the final parameter setting from the example in Fig. 20 on another input image with different characteristics.

7 Conclusions and future work

This paper has presented a methodology for adding flexibility and convenience to an existing vision system by integrating the vision specialist's knowledge into it using a knowledge-based architecture. The proposed methodology assumes that the vision system has a non-empty operating region at which it yields satisfactory results. It imitates the approach used by the specialist in reaching a point in this region from a given or default setting. The system can thus self-tune in response to the user's evaluations. This type of a system will be of immense value in making the full power of vision algorithms available to end-users. Here are some possible directions for future work:

Detecting failure: Obviously, if no combination of the tuning parameters can yield satisfactory results, neither the specialist nor the self-tuning framework will have any chance of succeeding. On the other hand, if the self-tuning mechanism does not capture the full complexity of the specialist's reasoning, it may fail in difficult cases even if a solution exists. Detecting this failure may not always be easy for the user, since the self-tuning mechanism may have loops and other complex chains of reasoning. If the system does not solve the problem in a reasonable amount of time, it should be considered as having failed. Experimentation on large and diverse data sets and constant refinement of the knowledge base will ensure that such failures do not occur too often.

Validity of the model: Closely related to the problem of detecting failures is the validation of the basic self-tuning model itself. This model assumes that based on certain "symptoms" of the result, a diagnosis can be made as to which parameter (or algorithm) is at fault. As in [7], the vision system is assumed to possess a "Markovian" property, which allows order-independent and mutually independent parameter tuning. The problem of parameter tuning is essentially ill-posed: there may be many more parameters than symptoms, and a one-to-one mapping may not be easy to establish in all cases. A more complex problem-solving model that takes this into account needs to be investigated.

Specific evaluations: Currently, result evaluations are in the form of general remarks about the results obtained, and not about specific portions or objects of the output. It will be useful to incorporate some mechanisms for handling specific evaluations. The challenge is in making these mechanisms as application-independent as possible.

Spatio-temporal reasoning: Most vision systems perform some amount of spatio-temporal reasoning, typically involving operations like windowing, localization, region growing, proximity checking, etc. This reasoning is usually embedded in the vision algorithms themselves, or else is contained in a knowledge base developed for a specific application using a expert system shell. If generic aspects of the spatio-temporal reasoning used in vision tasks can be identified, it may be possible to incorporate them into the algorithm control scheme presented in this paper. As with mechanisms for specific evaluations, the challenge lies in doing this in an application-independent manner.

Quantitative performance measures: There has been some recent work on statistical performance characterization of vision algorithms (e.g. [11]). Integrating this kind of tool into a self-tuning vision architecture, although a challenging task, is a promising area of research, because it would add a quantitative dimension to the result evaluation process.

Learning: Machine learning principles can be applied to exploit the wealth of information accumulated over a period of time as the vision system is tested on different data sets. Preliminary work is reported in [20]. Currently, the operational history of the self-tuning procedure is not directly available to the system developer. The availability of this information will enhance machine learning capabilities, and will also be useful in performance analysis.

Other applications: We propose to test the methodology on other candidate problems such as multisensor registration, and to validate it using large and diverse data sets.

Acknowledgments

We would like to thank Shyam Kuttikkad, Vasudev Parameswaran, Monique Thonnat, John van den Elst, Yasmina Chitti, Hany Tolba and Azriel Rosenfeld for their contributions to the work reported here. We would also like to thank Les Novak of MIT Lincoln Laboratories for providing the SAR image used in one of the examples.

References

[1] R. Bodington, "A software environment for the automatic configuration of inspection systems," in *First International Workshop on Knowledge-Based Systems for the (re)Use of Program Libraries*, (INRIA, Sophia Antipolis, France), Nov. 1995.

[2] R. Chellappa, X. Zhang, P. Burlina, C. L. Lin, Q. Zheng, L. S. Davis, and A. Rosenfeld, "An integrated system for site-model supported monitoring of transportation activities in aerial images," in *DARPA Image Understanding Workshop*, vol. 1, (Palm Springs, CA), pp. 275–304, Feb. 1996.

[3] S. A. Chien, "Using AI planning techniques to automatically generate image processing procedures: A preliminary report," in *Second International Conference on AI Planning Systems*, (Chicago, IL), pp. 219–224, June 1994.

[4] V. Clément and M. Thonnat, "A knowledge-based approach to the integration of image processing procedures," *CVGIP: Image Understanding*, Vol. 57, pp. 166–184, Mar. 1993.

[5] D. Crevier and R. Lepage, "Knowledge-based image understanding systems: A survey," *Computer Vision and Image Understanding*, Vol. 67, pp. 161–185, Aug. 1997.

[6] D.Charlebois *et al.*, "A case-based planner to autmate reuse of ES software for analysis of remote sensing data," in *International Geoscience and Remote Sensing Symposium (IGARSS)*, vol. 3, pp. 1851–1854, 1991.

[7] B. Draper, "Modeling object recognition as a Markov decision process," in *Proceedings of the IAPR International Conference on Pattern Recognition*, (Vienna, Austria), pp. 95–99, Aug. 1996.

[8] F.Grimm and H.Bunke, "An expert system for the selection and application of image processing subroutines," *Expert Systems*, Vol. 10, pp. 61–74, May 1993.

[9] A. R. Hanson and E. M. Riseman, "VISIONS: A computer system for interpreting scenes," in *Computer Vision Systems* (A. Hanson and E. Riseman, eds.), San Francisco, CA: Academic Press, 1978.

[10] S. Kuttikkad and R. Chellappa, "Building wide area 2D site models from high resolution polarimetric synthetic aperture radar images," Tech. Rep. CAR-TR-776, Computer Vision Laboratory, University of Maryland, College Park, MD 20742-3275, June 1995.

[11] X. Liu, T. Kanungo, and R. M. Haralick, "Statistical validation of computer vision software," in *DARPA Image Understanding Workshop*, vol. 2, (Palm Springs, CA), pp. 1533–1540, Feb. 1996.

[12] T. Matsuyama, "Expert systems for image processing: Knowledge-based composition of image analysis processes," *Computer Vision, Graphics and Image Processing*, Vol. 48, pp. 22–49, Jan. 1989.

[13] C. C. McConnell and D. T. Lawton, "IU software environments," in *DARPA Image Understanding Workshop*, pp. 666–676, DARPA, Apr. 1988.

[14] A. M. Nazif and M. D. Levine, "Low-level image segmentation: an expert system," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 6, pp. 555–577, Sept. 1984.

[15] K. Sakaue and H. Tamura, "Automatic generation of image processing programs by knowledge-based verification," in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 189–192, 1985.

[16] T. Strat, "Employing contextual information in computer vision," in *DARPA Image Understanding Workshop*, (Washington, DC), pp. 217–229, Apr. 1993.

[17] T. Strat and M. A. Fischler, "Context-based vision: recognizing objects using both 2D and 3D imagery," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 13, pp. 1050–1065, Oct. 1991.

[18] M. Thonnat and S. Moisan, "Knowledge-based systems for program supervision," in *Proceedings of First International workshop on Knowledge-Based systems for the (re)Use of Program Libraries (KBUP'95)*, 1995. URL: http://www.inria.fr/orion/.

[19] T. Toriu, H. Iwase, and M. Yoshida, "An expert system for image processing," *Fujitsu Sci. Tech. Journal*, Vol. 23.2, pp. 111–118, 1987.

[20] R. Vincent, S. Moisan, and M. Thonnat, "Learning as a means to refine a knowledge-based system," in *Proceedings of Third Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop*, (Hatoyama, Japan), pp. 17–31, Nov. 1994.

[21] R. Vincent, S. Moisan, and M. Thonnat, "A Library for Program Supervision Engines," Tech. Rep. 3011, I.N.R.I.A., Sophia Antipolis, France, 1996. URL: http://www.inria.fr/orion/.

[22] R. Vincent and M. Thonnat, "Planning, executing, controlling and replanning for IP program library," in *Proceedings of IASTED International Conference Artificial Intelligence and Soft Computing (ASC'97)*, (Banff, Canada), July-August 1997. URL: http://www.inria.fr/orion/Articles/ASC97.html.

[23] R. Vincent, M. Thonnat, and O. J., "Program Supervision for Automatic Galaxy Classification," in *International Conference on Imaging Science, Systems and Technology (CISST'97)*, (Las Vegas, Nevada), June 1997.