UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Rapports de Recherche

N°2000

Programme 3 et 4

Intelligence artificielle,
Systèmes cognitifs et
Interaction homme-machine
et
Robotique, Image et Vision

SUPERVISION OF PERCEPTION TASKS FOR AUTONOMOUS SYSTEMS: THE OCAPI APPROACH

Monique THONNAT
Véronique CLEMENT
John VAN DEN ELST

June 1993

Institut National
de Recherche
en Informatique
et en Automatique

2004 route des Lucioles
B.P. 93
06902 Sophia-Antipolis
France

Supervision of Perception Tasks for Autonomous Systems: the OCAPI approach

Monique Thonnat Veronique Clément John van den Elst
INRIA Sophia Antipolis
BP 93, F-06902 Sophia Antipolis CEDEX, France
Fax: (33) 93-65-76-43
e-mail: thonnat@sophia.inria.fr vclement@sophia.inria.fr elst@sophia.inria.fr

Abstract: This paper deals with the supervision of perception tasks, as a component of an autonomous system. First, the role of this component in an autonomous system is presented. Secondly, a model of supervision of perception tasks is proposed: needed control mechanisms as well as knowledge modeling are detailed. Then, an implementation of this model, designed as an expert system shell, named OCAPI, is presented. The facilities provided by OCAPI are shown, through an example: the supervision of an object detection process in road scenes. More precisely, the expert system, developed with OCAPI, automates planning (selection of a sequence of programs) and control of execution (parameter initialization and adjustment, as well as result evaluation). The contents of the knowledge base and a particular utilization of this knowledge base are described.

Keywords: perception tasks, supervision, expert systems, planning, control of execution.

Supervision de tâches de perception pour les systèmes autonomes : l'approche OCAPI

Résumé : Ce rapport concerne la supervision des tâches de perception, vue comme composant d'un système autonome. Nous présentons tout d'abord le rôle d'un tel composant dans un système autonome. Nous proposons ensuite un modèle de supervision de tâches de perception et détaillons les mécanismes de contrôle nécessaires ainsi que les connaissances à modéliser. Une implémentation de ce modèle, nommée OCAPI et conçue comme un noyau de systèmes experts, est présentée. Les facilités fournies par OCAPI sont montrées à travers un exemple concernant la supervision d'un traitement de détection d'objects dans des scènes routières. Plus précisement, le système expert développé avec OCAPI automatise la planification (sélection d'une séquence de programmes) et le contrôle d'exécution (initialisation et ajustement des paramètres, évaluation des résultats) dans une application routière de détection d'obstacles. Le contenu de la base de connaissance et un exemple particulier d'utilisation de celle-ci sont décrits.

Mots clés : tâches de perception, supervision, systèmes experts, planification, contrôle d'exécution.

1 Introduction

Needs for autonomous systems have grown extensively during the last decade. As examples, we can cite navigation [FHR89], autonomous vehicles [Eur91, DMC90], or spatial robotics. For such systems, both perception and action capabilities have to be flexible in order to face changing environments: robots are moving in their environment (different observation points, object emergence or vanishing...); and the environment is dynamic (other moving objects, light changes...).

In the design of systems able to act autonomously in real environments, three modules are required: perception, situation assessment and action. We propose an architecture of an autonomous system in figure 1. On the left side of the figure, the component **Perception** processes the information provided by the sensors. Perception performs low level image processing as well as higher level processing such as object recognition. The component **Perception Supervision** defines and optimizes the perception tasks because of changing environments. This component gives the flexibility to the perception system. The component **Situation Assessment** has charge of interpreting the situation; it may be a complex component if many objects of different kinds are present, moving, and interacting. On the right side of the figure, the component **Control of Actions** controls the actuators depending on the present situation; in changing conditions, feedforward and feedback controls, or a combination of both models are usually necessary.

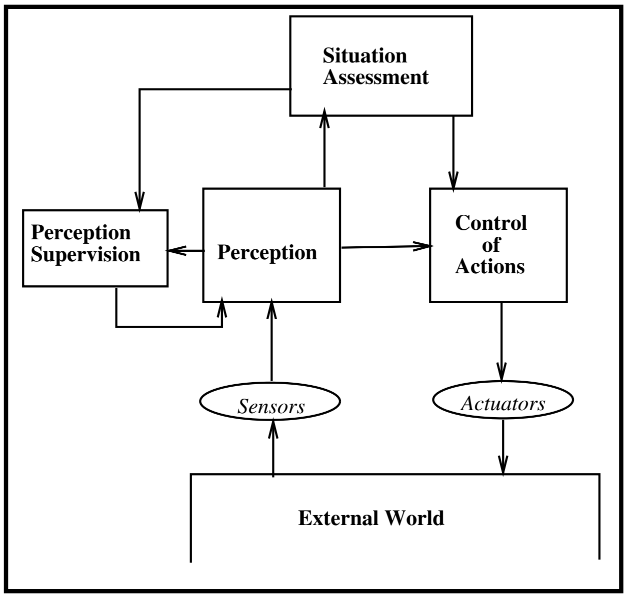

Figure 1: An architecture of an autonomous system

In this paper, we focus on the perception supervision component, and propose a supervision tool to optimize the perception of changing environments. In fact, the challenge is to automate the use of specialized programs which are often very sensitive. This problem is close to the problem concerning support for the use of existing libraries of image processing programs, that has been addressed for instance in [TIY87, Joh87, TS88, SKT88]. For a detailed description of these systems see [CT93]. In addition to the facilities needed for the use of existing libraries of image processing programs, supervision of perception tasks implies the complete automation of the decisions.

In the following sections, we present a model of supervision of perception tasks. Then, we present OCAPI, an implementation of such a model. Finally, we describe an application for the supervision of an object detection process in road scenes.

2 Supervision of perception tasks

In this section, we present the control mechanisms and the knowledge which have to be taken into account for supervision of perception or vision tasks. Within a specific context (sensors, environment...), according to a goal to reach on given data, the role of the supervision module is to define and optimize the perception tasks. More precisely, by defining the perception tasks we mean to select the vision modules and to schedule them; by optimizing the perception tasks, we mean to compute their parameters, to run the modules and to control their execution. So two main kinds of reasonings (planning and control of execution) have to be performed in the sense of generic tasks introduced by Chandrasekaran [Cha87]; for each of these reasonings, several techniques are available.

2.1 Planning for supervision

In this section, the question is: which planning technique is adequate for supervision of perception tasks?

In problem solving, planning is to find a plan for achieving a goal according to a given state of the world. A plan is defined as an ordered list of actions to perform. Let us notice that, in this context, the actions are no more the robotics actions of figure 1, but the perception tasks (for example data filtering, primitives extraction, object recognition....). Three main techniques have been developed in planning: non hierarchical, hierarchical and skeletal planning [BCF82]. While the two first methods concentrate on the reduction of conflicts in the plan steps, the last one emphasizes the expertise of the specific domain. The third method, also called script-based planning, assumes that abstract or skeletal plans exist. A skeletal plan contains the basic steps to solve a problem. First, the best skeleton is selected, then refined.

For perception, we have to handle real world data, and to deal with unreliable or incomplete information; the state of the world (initial, final or intermediate) can not be described exhaustively. So we have to use a technique based on the description of the actions to be performed, and not on the different possible states of the world. On the other hand, we have few sensitive conflicting subgoals; so the two first techniques are not necessary. As we hold knowledge of typical processing methods, at several levels of abstraction, a skeletal technique providing a refinement mechanism is appropriate. Starting from predefined coarse skeletons, the plan is elaborated both by adaptation to the current context (by replacement or insertion of actions), and by successive hierarchical refinement.

2.2 Control of execution for supervision

We are interested, here, in defining the type of control of execution which is the best adapted for the supervision of programs.

Four main types of control of execution [HTD90] have been developed: i) prediction of failures, and planning of additional actions which wait until the world corresponds to what is expected; ii) prediction of failures, and integration in the plan of tests and correcting actions; iii) prediction of failures, and integration in the plan of tests and alternatives; iv) integration of the planning process, and control of the execution process.

In our context, for perception, we do not handle secure actions: predicting exactly the behaviour of a perception task on an image is not possible; a trial-and-error strategy is needed. Moreover as the results of actions can not be predicted, the selection of a precise method at a given step needs the results of the previous steps. So the interleaving of planning and control of execution processes is necessary for perception task supervision.

2.3 Knowledge for supervision

With [Vog86, Mat89], we can point out which kinds of knowledge are required for supervision of perception tasks: assessment of image quality, selection of appropriate operators, determination of optimal parameters, combination of primitive operators, trial-and-error experiments, and evaluation of analysis results. Let us be more precise on the knowledge needed for supervision of perception tasks, and propose a typology of the knowledge. First we introduce several concepts: goal, operator, argument, context, data description and request. Then we present the notions of choice, evaluation, initialization, and adjustment.

Goals: When we process an image, we always have in mind an objective or a goal to reach. In fact, a **goal** represents an image processing functionality. This functionality or objective can be reached by different methods which can be primitive or complex. We can note that programs in a library implement image processing functionalities or tasks to perform, but they **are not** image processing functionalities.

Operators: The abstract notion of goal has to be distinguished from that of available methods, also called **operators**. Operators are actions which can be performed; they contain specific information to reach a given goal. An operator can be a particular program (it will be referred to as **primitive operator**) or a particular sequence of processings (it will be referred to as **complex operator**). Every primitive or complex operator has a set of **characteristics** which correspond to its typical properties.

Arguments: Operators and goals have a list of input and output **arguments**. We distinguish two classes among the arguments: **data** and **parameter** arguments. Data arguments have fixed values which are set (input data) or computed (results). Parameters are tunable arguments; they have adjustable values and are always input arguments; in fact, each operator uses its input parameters in a specific way (meaning, format, range, sensitivity...).

Context: The **context** corresponds to the general and global information concerning a certain application. It describes for instance the data acquisition conditions or the application domain.

Data Descriptions: The information that is local to a data argument can be called **data description.** Two parts must be considered: the **semantical description** and the **implementation description**. The semantical description corresponds to the information about the contents of a data argument, while the implementation description corresponds to the information about the data argument itself.

Requests: A **request** states which goal has to be reached, the data of the particular case to be processed and the eventual constraints on the results to be obtained such as the required quality; this can be structured in terms of preconstraints and postconstraints. A request is thus a query to solve a goal for a specific case. A complex operator is a set of processings, which correspond in fact to a **set of requests** (note that it is not a set of programs, neither a set of goals). So each step is a request to solve a goal in a special context, with preconstraints on the input, and eventual postconstraints on the results. Several types of temporal links are necessary between the requests, representing sequentiality, parallelism.

In addition to these concepts, we need four kinds of knowledge. Knowledge to describe how to choose among methods, how to initialize input arguments, how to evaluate results, and eventually how to modify the processing with the determination of new input values for programs or selection of other programs. So we introduce four kinds of criteria: criteria for choice of operators, for evaluation of results, for initialization of parameters, and for adjustment of parameters.

Criteria for Choice: we refer here to the knowledge of how to select, among all the available operators, the operator(s) which is (are) the most pertinent; this is depending, of course, on the goal to be reached, on the characteristics of the operators associated with this goal, on the data to be processed (preconstraints), on the desired result quality (postconstraints) and on the context.

Criteria for Evaluation: we refer here to the knowledge of how to assess the results provided by the selected operator after its execution; this must essentially take into account the postconstraints expressed in the request as well as the context.

Criteria for Initialization: we refer here to the knowledge of how to initialize values of input arguments, more particularly of input parameters. Various mechanisms are required as well as various kinds of knowledge, like default values or computation methods depending on the values of other arguments.

Criteria for Adjustment: we refer here to the knowledge of how to modify values of parameters after a negative evaluation. This is depending on the evaluation and of course on the specific operator: the sensitivity of the various parameters depends highly on a particular algorithm and implementation, or on the particular decomposition into subrequests for a complex operator.

2.4 Conclusion

We have shown that for supervision of perception tasks two kinds of reasoning are needed (planning and control of execution); then we have proposed a typology of the knowledge required for such tasks. Let us now precise the links between the reasoning mechanisms of planning and control of execution, and the knowledge concepts and criteria.

Goals, operators and requests are necessary concepts for **planning** based on the description of the actions to be performed; they correspond to different hierarchical levels of knowledge about the actions. Furthermore, the concept of complex operator, linking requests together, permits to express knowledge related to predefined skeletons. In such a model, choice among available operators is essential for the planning steps, and is based on the operator characteristics and the current context.

The **control of execution** is concerned with execution of actions as well as with failures; the trial-and-error strategy is based on tests and alternatives to correct the execution. Knowledge describing operators is useful for their execution: how to run them, how to initialize their arguments and parameters using concepts such as data description, arguments, preconstraints or context and criteria for initialization. On the other hand, tests and alternatives are concerned with the same concepts, and also with postconstraints as well as criteria for evaluation and adjustment.

We can see here once more that planning and control of execution must be interleaved, because planning can not be done in one stage (at a given step, the choice of an operator depends on the results of the previous steps), and control of execution has to be performed at various levels of refinement (necessary knowledge can be available at a different level).

3 OCAPI: a task supervision tool

We have implemented supervision of perception tasks within an expert system shell architecture, named OCAPI [CT93]. It has been developed in a lisp dialect: Le_Lisp, and its object oriented facilities. Among its features, we want to point out its utilization in two modes (automatic or interactive [for development]), and the possibility of monitoring procedures written in C, fortran or lisp.

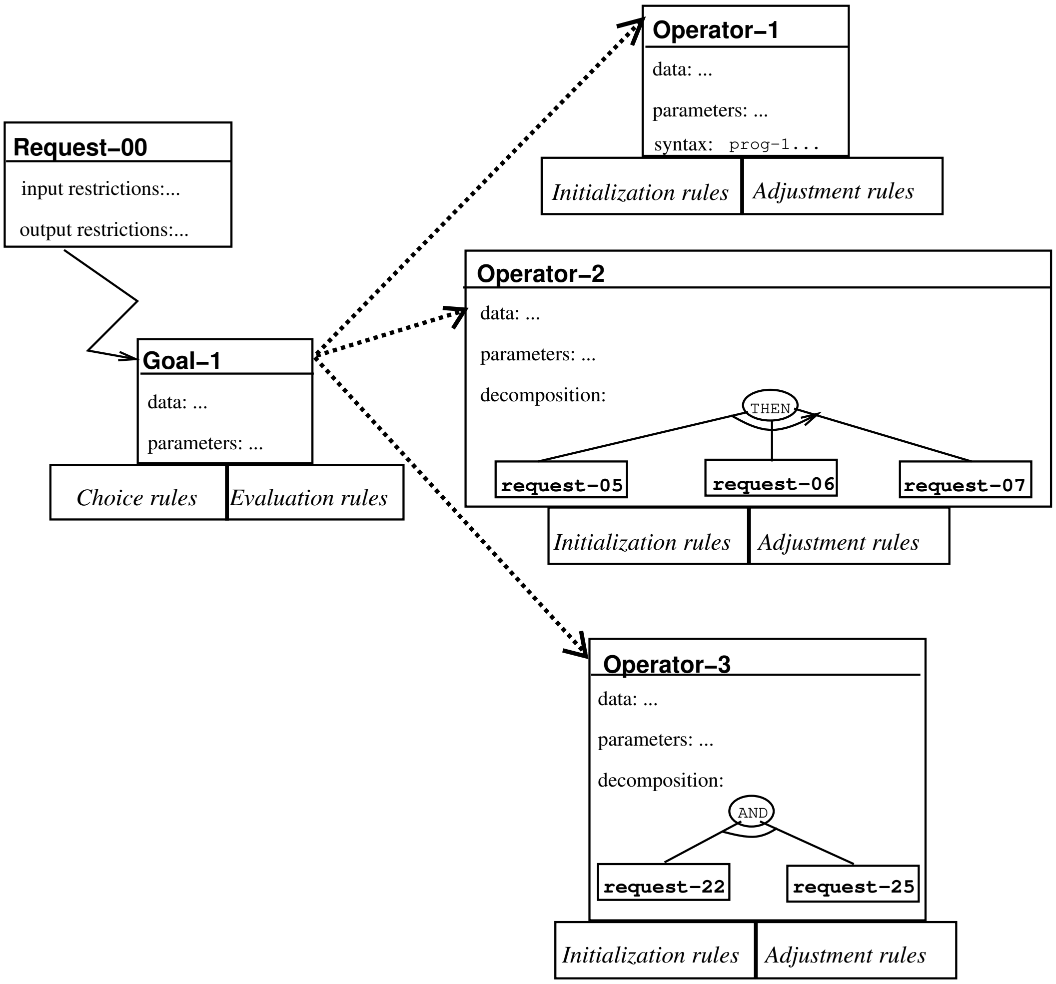

Figure 2: Relations between the objects, and the rule bases in OCAPI

3.1 Knowledge model implementation

The static or descriptive knowledge is implemented with frames, i.e. goals, operators, arguments, data descriptions, requests, and contexts. The heuristic criteria are implemented with production rules; they describe the knowledge concerning choice, evaluation, initialization and adjustment. Strong relations exist between the various concepts we have introduced; these relations structure the knowledge base in order to facilitate the expression of the knowledge and also its utilization. To each goal and operator small bases of rules specialized in choice, evaluation, initialization and adjustment are associated. Figure 2 shows the relations between the frames, and the rule bases. A request (for instance *Request-00*) is related to a particular goal (here *Goal-1*); this goal can be solved by several operators (*Operator-1, Operator-2,* and *Operator-3* in this case). A decomposition of a complex operator (*Operator-2,* or *Operator-3*) is a tree of requests to other sub-goals. Each goal, and each operator have two private rule bases: one rule base on choice, and one on evaluation for a goal; one rule base on initialization, and one on adjustment for an operator.

Structuring the knowledge base in this way greatly improves the efficiency and facilitates the development of knowledge bases at different levels of abstraction.

3.2 Reasoning model implementation

The control mechanisms of OCAPI are built around two reasonings: planning and control of execution, which are interleaved. The control structure has five components: an execution controller, a planner, a parameter settler, a general rule interpreter, and an interface to activate external programs. The execution controller has the role of a supervisor and of a controller: it controls the execution of a tree of requests, and decides to call other components for specific tasks. The algorithm presented in Figure 3 controls the execution of a tree of requests. Of course, to process the initial request, this algorithm works on a tree reduced to one request. In a first step, a global matching verifies that there exists at least one operator to solve the goals of the requests. Then the next request to process is determined. For this request, the available operators are sorted (using the choice rules and the operator characteristics); invalid operators are eliminated. Then one operator is chosen. Before its execution, the parameter settler determines the values of its parameters: using initialization rules for the first attempt, using adjustment rules for the next ones. If the operator is a primitive one, the execution is done through the interface with programs; in the case of a complex one, this algorithm is recursively called to control the execution of the tree of requests of its decomposition. Then, according to the request, the evaluation of the results is achieved automatically using the evaluation rules, or by graphic presentation of the results to the user who evaluates them interactively. In case of unsatisfactory results, another execution is performed after adjustment of the input parameters of the operator, or after choice of another operator (updating of the sort of the operators).

Handling sequences of data: by sequences of data, we mean temporal sequences of data provided by a given sensor when observing a scene. To process these data, the

-1- *global matching*

while not all requests have been processed

-2- *selection of the request to be processed*

-3- *classification of the operators (using choice rules and operator characteristics)*

while the request is not satisfied

-3b- *re-classification of the operators*

-4- *selection of the best operator*

-5- *execution of the operator (using initialization or adjustment rules)*

-6- *evaluation of the results (using evaluation rules)*

Figure 3: The algorithm for the execution of a tree of requests

goal and the context are the same; so the decisions (choice of the operators, and setting of the values of the parameters) which have been made for one data are used to initialize the process of the following one. Of course even if the contents of the data are globally the same, slight changes may occur from one data to another; so the evaluation and adjustment mechanisms are still provided.

4 Supervision of object detection in road scenes

In this section, we present an example of an expert system built with OCAPI. It is developed to perform the detection of objects (such as the road, cars, and posts) in road scenes and urban scenes using stereovision data. We show how the various kinds of knowledge are expressed in the knowledge base.

This example has been developed within the framework of the Eureka project Prometheus. The data are taken from two cameras which are fixed on the top of a moving car. A pyramidal stereovision algorithm based on contour chain points [MTB90] is used to reconstruct the 3D environment in front of the vehicle. Obstacles are detected using the 3D as well as 2D information.

As for this project we are interested in an autonomous obstacle detection system, the process has to be supervised. Knowledge on the use of this process is needed for robustness, and above all to manage the great variety of the scenes. More precisely a lot of contextual values in the scenes vary, like luminosity (depending on the weather conditions and on the moment in the day), complexity of the scene (depending on the number of objects in front of the car), and velocity (depending on the location of the scene: highway, countryside road or urban street). Since the scenes may be quite various, it is not possible to fix the values of the different parameters involved in the processing.

The initial request is the processing of a particular pair of images (taken by two cameras) for object detection. Figure 4 shows an example of such images in the case of an urban scene.

Figure 4: Stereo images corresponding to an urban scene (s8p10l.ext and s8p10r.ext)

First a 3D reconstruction is made of the scene using a pyramidal stereovision process. The pyramidal stereovision process works at several resolutions (to be precise, four resolutions). For each resolution, first an extraction of the primitives and then a matching of these primitives is performed. For each image (the left and the right one), extraction of primitives consists of contour detection, computation of precise orientation, thresholding by hysteresis, and contour chaining. Matching results obtained at a given resolution are interpolated, and used to reduce the search at the immediately higher resolution. Then a polygonal approximation is performed on the contour chains to obtain only straight line segments. Finally the 3D reconstruction together with 2D information (the straight line segments), are used to detect the road, the cars, and the posts.

4.1 The PROMETHEUS knowledge base

In this section, we give examples of the various objects and rules in the knowledge base.

4.1.1 Goals

The PROMETHEUS knowledge base contains several goals, like e.g.: *object-detection, pyramidal-stereovision, primitives-extractions, primitives-extraction, stereo-matching, contour-detection, orientation, thresholding, chaining, polygonal-approximation, 2D-3D-cooperation.*

For instance, the goal *object-detection* [Figure 5] has two input arguments (the pair of images containing the road scene pictures), and three output arguments (three images containing respectively the detected road, cars and posts). No choice rules, nor evaluation rules are attached to this goal.

Goal	**object-detection**
input data :	imleft
	imright
output data :	imroad
	imcars
	imposts

Figure 5: The goal *object-detection*

Figure 6 shows the goal *stereo-matching* corresponding to the actual matching of two images (as part of the pyramidal stereovision process). Six input arguments (three pairs of images, with respectively the contour chains, the gradients, and the gradient orientations) are listed. The output arguments are an image containing the matched primitives (3D information), and an image containing disparity information. Four choice rules and one evaluation rule are attached to this goal.

Goal	**stereo-matching**
input data :	lchains
	rchains
	lgradients
	rgradients
	lorient
	rorient
output data :	xyz
	outdisps
choice-rules :	C1, C2, C3, C4
evaluation-rules :	E1

Figure 6: The goal *stereo-matching*

In table 1 we show how the argument *lgradients*, containing the gradient information of the left image, is described in the knowledge base. It contains the following fields: the name, the functionality (currently expressed as comment), the type, the default value, and the semantic and implementation data description (will be explained later).

INPUT DATA ARGUMENT **lgradients**	
Name:	lgradients
Comment:	"Left image gradients"
Type:	image
Default:	()
Semantic description:	...
Implementation description:	...

Table 1: Structure of input argument *lgradients*

4.1.2 Operators

Several operators have been defined in the knowledge base: each of them corresponds to a particular goal. Some examples are shown in Table 2.

goal	**operators**
object-detection	O-detect-objects
primitives-extractions	O-extract-prims-left-right
primitives-extraction	O-extract-primitives
pyramidal-stereovision	O-stereo-pyr1, O-stereo-pyr2
stereo-matching	O-ma-stereo-match, O-meygret-stereo-match
contour-detection	O-deriche
orientation	O-ori-grad
thresholding	O-thres-hysteresis
chaining	O-chaining-gg1, O-chaining-gg2
polygonal-approximation	O-approx-poly
2D-3Dcooperation	O-2D-3D-cooperation
road-detection	O-detect-road
car-detection	O-detect-cars
post-detection	O-detect-posts
...	

Table 2: Some goals and their corresponding operators

As we saw before there are two types of operators, primitive operators and complex operators. The operator *O-detect-objects* [Figure 7] is an example of a complex operator.

Operator	**O-detect-objects**
goal	object-detection
input data :	imleft
	imright
output data :	imroad
	imcars
	imposts
decomposition :	request r-pyramidal-stereovision
	then request r-polygonal-approximation
	then request r-2D-3D-cooperation

Figure 7: The complex operator *O-detect-objects*

This operator corresponds to the goal *object-detection*, and has the same input and output arguments as this goal. It is decomposed into several substeps (requests to other goals): first pyramidal stereovision, then polygonal approximation and finally object detection using both 2D and 3D information. Figure 8 shows the details of the first request, which is a request to the goal *pyramidal-stereovision* (the pyramidal stereovision process). Control of the data flow of input and output data of the operator, and of the subrequests is expressed directly in the request.

Request	**r-pyramidal-stereovision**
goal	pyramidal-stereovision
restrictions :	**input data** :
	img $\leftarrow$*imleft of OPERATOR O-detect-objects*
	imd $\leftarrow$*imright of OPERATOR O-detect-objects*

Figure 8: Description of the first subrequest of the operator *O-detect-objects*

An example of a primitive operator is the operator *O-meygret-stereo-match* [Figure 9] that corresponds to the goal *stereo-matching*. Besides the input and output arguments of its corresponding goal, it also has some additional parameter arguments. The parameters *thr-m* and *thr-o* are two threshold values, respectively on the magnitude and the orientation of the gradient. Table 3 shows how the argument *thr-m* is described in the knowledge base. The parameter *zoom* contains the image reduction factor. Finally, this operator also contains three initialization rules and three adjustment rules.

4.1.3 Context

The object **context** describes the general information concerning a certain application in a structured way. The information in the context is used during the reasoning process by

Operator	**O-meygret-stereo-match**
goal	stereo-matching
characteristics	average-quality-match, high-adaptability
input data :	lchains
	rchains
	lgradients
	rgradients
	lorient
	rorient
parameters :	thr-m
	thr-o
	zoom
output data :	xyz
	outdisps
syntax :	*match-dzv* lchains rchains lgradients rgradients lorient rorient xyz outdisps *-sl* thr-m *-sa* thr-o *-fact* zoom
initialization-rules :	I1, I2, I3
adjustment-rules :	A1, A2, A3

Figure 9: The primitive operator *O-meygret-stereo-match*

PARAMETER ARGUMENT **thr-m**	
Name:	thr-m
Comment:	"Threshold on the gradient magnitude"
Type:	real
Default:	.15
Range:	[0,1]

Table 3: Structure of parameter argument *thr-m*

production rules (e.g. choice rules to decide pertinent operators in function of the context and initialization rules to adapt initial values of the parameters to the current context).

The definition of the object context in the PROMETHEUS knowledge base is presented in table 4. Presently, the production rules use various information of the context, like information about the physical environment: the *type of camera*, the distance from the camera to the front of the car (*depth_min*), and whether there is *motion* or not (motion of the cameras and/or motion of the objects). Some other information in the context concerns the *user constraints*. Herein the level of *details* that must be extracted from the image (compared to the amount of information in the scene) are specified, or the desired *stereo matching adaptability* and *quality*. The *mode* specifies if the processing has to be performed in a completely automatic way (without possibility of asking a human user to validate the results) or in an interactive way.

CONTEXT ITEM		POSSIBLE VALUES	COMMENTS
camera	.type	*ccd*	type
	.depth_min	*[0, -10]*	distance from camera to front of car
motion		*present, absent*	presence or not of moving cameras or objects
user-constraints	.mode	*interactive, automatic*	interaction with user or not
	.extraction.details	*none, few, many, all*	need for more or less details
	.matching .adaptability	*high, low*	desired matching adaptability
	.quality	*high, average*	desired matching quality

Table 4: Definition of the context in the PROMETHEUS base

4.1.4 Data description

The **data description** is used to describe information that is local to a data argument. It contains a **semantic description** and an **implementation description**. Data descriptions can be defined for several types of arguments, like image, text, symbol, list, integer, real, etc. In the PROMETHEUS knowledge base the most important data arguments are those of the type image. The information about the contents of an image is stored in the semantic description, and the technical information about the image itself is stored in the implementation description. Table 5 shows a subpart of a semantic description of an image, and table 6 of an implementation description of an image. The presented information gives a good indication how the concept of data description is used. In the semantic description of an image the *nature* of the contents of the image can be stored, or the *number of primitives* the image contains (e.g. the number of contour chain points). In the implementation description of an image we store information about the image itself, like the image *size* and the *minimum* and *maximum* pixel value.

SEMANTIC DESCRIPTION OF DATA TYPE **image**		
ITEM	POSSIBLE VALUES	COMMENTS
nature	*road-scene-picture, gradients, contour-chains, disparities, 3D-reconstruction, ...*	nature of the contents of the image
primitives.number	*[0,→]*	number of primitives
noise	*absent, average, much, unknown*	noise level
...		

Table 5: Subpart of the semantic description of an image in the PROMETHEUS base

IMPLEMENTATION DESCRIPTION OF DATA TYPE **image**		
ITEM	POSSIBLE VALUES	COMMENTS
format	*inrimage, visilog*	image format
minimum	*[0,→]*	minimum pixel value
maximum	*[0,→]*	maximum pixel value
size .x	*[0,→]*	image size in x direction
.y	*[0,→]*	image size in y direction
...		

Table 6: Subpart of the implementation description of an image in the PROMETHEUS base

4.1.5 Choice rules

When several operators are available to solve the same goal, knowledge is needed about how to select the pertinent operator according to the context and the operator characteristics.

Several stereo matching operators are available to satisfy the goal *stereo-matching*. In the PROMETHEUS knowledge base there are two. Each of them has its own typical properties. The operator *O-ma-stereo-match* is an operator that produces a high quality match (without much errors). The quality of the match is not very dependent of the parameter values. The operator *O-meygret-stereo-match* produces a lower quality match, but here the quality can be influenced by the parameters (changing the parameter values also changes the quality of the match). These typical properties are stored in the knowledge base as the *operator characteristics*. Based on these characteristics and the information in the context, the most appropriate operator can be chosen. The following rule is an example of a choice rule for the goal *stereo-matching:*

if *context.user-constraints.matching.quality* *high*
then :use-operator-with-characteristic *high-quality-match*
comment "high quality match desired"

As another example, we look at the two operators *O-stereo-pyr1* and *O-stereo-pyr2* that correspond to the goal *pyramidal-stereovision.* These two operators perform stereovision using a pyramidal technique, and are based on contour chain points. The first operator directly implements this algorithm, while the second one begins with the separation of the two interlaced acquisition frames by doing a sampling of the lines. The second operator is thus well-adapted when motion is present, and when two consecutive lines are not taken consecutively, but at time t and $t + T/2$ (if t is the time, and T is the acquisition period). The rule presented below implements such a choice criterion; it is attached to the goal *pyramidal-stereovision:*

if *context.motion present*
then :use-operator-with-characteristic *with-sampling-for-motion*
comment "only 1 of the 2 interlaced frames if motion"

4.1.6 Evaluation rules

Most of the time it is very difficult to express evaluation criteria to assess the results of low level goals (like thresholding) since, for example, results are matrices of pixels. But for higher level goals which provide descriptions or object interpretations as results, it is easier to find a criterion of evaluation. For example, the intermediate level goal *stereo-matching* provides as output matched pairs of primitives which can be counted and compared to a required value. Here we show an automatic evaluation rule of the goal *stereo-matching*:

if *xyz.primitives.number* < *imchains.primitives.number / 3*
then :assess *eval-number-primitives insufficient*
:impasse
comment "insufficient matched pairs"

It tests the ratio between the number of matched primitives (that are stored in the data description of the image *xyz* of the goal *stereo-matching*) to the number of input primitives (stored in the data description of the argument *imchains*). If this ratio is below the value of 1/3 the rule qualifies the number of matched pairs of primitives as insufficient, and a failure (impasse) is decided.

4.1.7 Initialization rules

Initialization of the parameters of an operator depends highly on the context. For instance, the operator *O-thres-hysteresis* which performs a thresholding by hysteresis (as part of the primitive extraction process), needs the initialization of two input parameters: an upper threshold (*thrmax*) and a lower threshold (*thrmin*). These thresholds are defined as numerical parameter arguments.

The initialization of these values is usually performed in two reasoning phases. In a first phase only **symbolical information** is used. The system reasons for example that if the user only wants *few* details to be extracted, the threshold should be *high,* as shown in the rule:

if	*context.user-constraints.details few*
then	:set-local *threshold high*
comment	"few details implies high threshold"

In a second reasoning phase the symbolical values are translated to real **numbers**, as in the rule:

if	*threshold average*
then	:set-local *heuristic-value 25*
comment	"average threshold implies 25 as heuristic value"

This second phase of translation into real numbers can be done by selecting a static heuristic value (as shown in the previous rule) or by computing dynamically a value using a calculation method (e.g. probability calculations based on the histogram).

In some cases the initialization is much simpler. For example the initialization of the parameter *thr-m* (a threshold on the magnitude of the gradient) of the operator *O-meygret-stereo-match* (see figure 9 and table 3). If no rules initialize this parameter argument, the system uses the default value, which is stored in the argument structure, as initial value.

4.1.8 Adjustment rules

When an operator fails because the results are not satisfactory, knowledge is needed to modify its parameters in function of the type of the failure.

For example, we have seen that the evaluation of the goal *stereo-matching* can assess that the number of matched pairs of primitives is insufficient; in such a case, it is possible to adjust the input parameters of the corresponding operator *O-meygret-stereo-match* which are a threshold on the magnitude of the gradient *(thr-m)*, and a threshold on the orientation of the gradient *(thr-o)*. We show below two adjustment rules for this operator. The first one states that the adjustment method for the parameter *(thr-o),* is a technique by percentage:

if ...
then :adjust-by percentage *thr-o*
:%-step *thr-o .3*

The second rule states that if the number of matched pairs of primitives is insufficient then the two parameters *(thr-m)* and *(thr-o)* have to be increased:

if :assessed? *eval-number-primitives insufficient*
then :increase *thr-m*
:increase *thr-o*

4.1.9 Summary

The entire PROMETHEUS knowledge base is composed of 32 goals, to which 39 operators are attached. Of these operators, 24 are primitive (programs) and 15 are complex (with decomposition into substeps). Furthermore the knowledge base contains 120 rules: 15 choice rules, and 20 evaluation rules attached to the goals; plus 64 initialization rules, and 21 adjustment rules attached to the operators.

There are not much more operators than goals, so the knowledge base could be extended by adding new alternative operators both for extraction of primitives and stereo matching. We also see that there are quite a lot of initialization rules compared to other rules. The initialization rules are important for the autonomy of the system and contain much information. In the next section we will take a closer look at the performance of the knowledge base.

4.2 PROMETHEUS knowledge base utilization

4.2.1 A request execution

In Figure 10 the initial request corresponding to the detection of objects in road scenes is presented.

This request has four main components: the name of its goal, the restrictions of the arguments of the goal (input and/or output), its context, and the data descriptions. The name of the goal is *object-detection*. The restrictions on the input arguments set the input data (the left and right images *[s8p10l.ext,* and *s8p10r.ext]*). The restrictions on the output arguments are to store the detected road data in the file *s8p10.ext.road*, the detected cars data in the file *s8p10.ext.cars* and the detected posts data in the file *s8p10.ext.posts*.

The particular context related to this request is as follows:

- the value of the user mode is *automatic*;
- *many* details must be extracted;

Request	**r-objects-10**	
goal :	object-detection	
restrictions :	**input data :**	imleft ← *s8p10l.ext*
		imright ← *s8p10r.ext*
	output data :	imroad → *s8p10.ext.road*
		imcars → *s8p10.ext.cars*
		imposts → *s8p10.ext.posts*
context :	user-contraints.mode	*automatic*
	user-contraints.extraction.details	*many*
	user-contraints.matching.adaptability	*high*
	user-contraints.matching.quality	*average*
	cameras.type	*ccd*
	cameras.depth_min	*-1.5*
data descriptions :	imleft	nature ← *road-scene-picture*
		minimum ← *0*
		maximum ← *255*
		size.x ← *512*
		size.y ← *368*
	imright	nature ← *road-scene-picture*
		minimum ← *0*
		maximum ← *255*
		size.x ← *512*
		size.y ← *368*

Figure 10: The initial request *r-objects-10*

- desired matching adaptability is *high*;
- desired matching quality is *average*;
- *ccd* type cameras are used;
- the distance from the camera to the front of the car is 1.5 meter.

The particular data descriptions for the input images are also specified in the request. Information that is known and thus specified, is for example the size of the image, the minimum and maximum pixel value, and the nature of the contents of the image. Not all information is known. Some information, like for example the number of primitives in the image or the noise level, will be determined later (during the reasoning process).

Given this initial request, the PROMETHEUS knowledge base used 16 different image processing programs to detect the road, cars and posts. First the pyramidal stereovision process was performed for both the left and the right image. On four image resolution levels, operators and parameters were chosen to extract primitives, and to perform the stereo matching. Then a polygonal approximation was performed. Finally the objects were detected; as examples, the four detected cars are shown in figure 11. Some of the operators have been used several times: for instance, the programs for the primitive extraction have been run twice (left and right image) at each of the four image resolution levels of the pyramidal stereomatching process. Furthermore, some operators are applied more than once on the same data arguments, with adapted parameter values, like for example the stereo matching. For the complete execution of the initial request, the knowledge base applies 62 primitive image processing operators.

Figure 11: Detected cars in urban scene s8p10l.ext

Figure 12 shows a subpart of the dynamic tree, that is constructed by the planning process. The root of the presented tree is the operator *O-pyr-level-match* that performs only one image resolution level of the pyramidal stereomatching process. This complex operator is decomposed into three substeps. The first step is a reduction (by the operator *o-zoom*) of the resolution of the images. Then in the second step, the primitives of the left and right images are extracted, and finally the third step is the stereo matching. The primitive extraction of an image is performed by the complex operator *O-extract-primitives*. The third step of its decomposition is the request *r-thres-hysteresis*. To execute the corresponding operator *O-thres-hysteresis*, the threshold values *thrmax* and *thrmin* are initialized. For example, on the first image resolution level, the value *thrmax* is initialized to 27 for the right image. In fact, in the context the user specified to extract many details. For this context two values are pertinent. The first one (25) is a heuristic value, and the second one (29) is obtained by a calculation method based on the image histogram. Based on this information, first a range of possible values was defined ([25-29]). Then the initial value of *thrmax* was set to 27 (in the middle of this range).

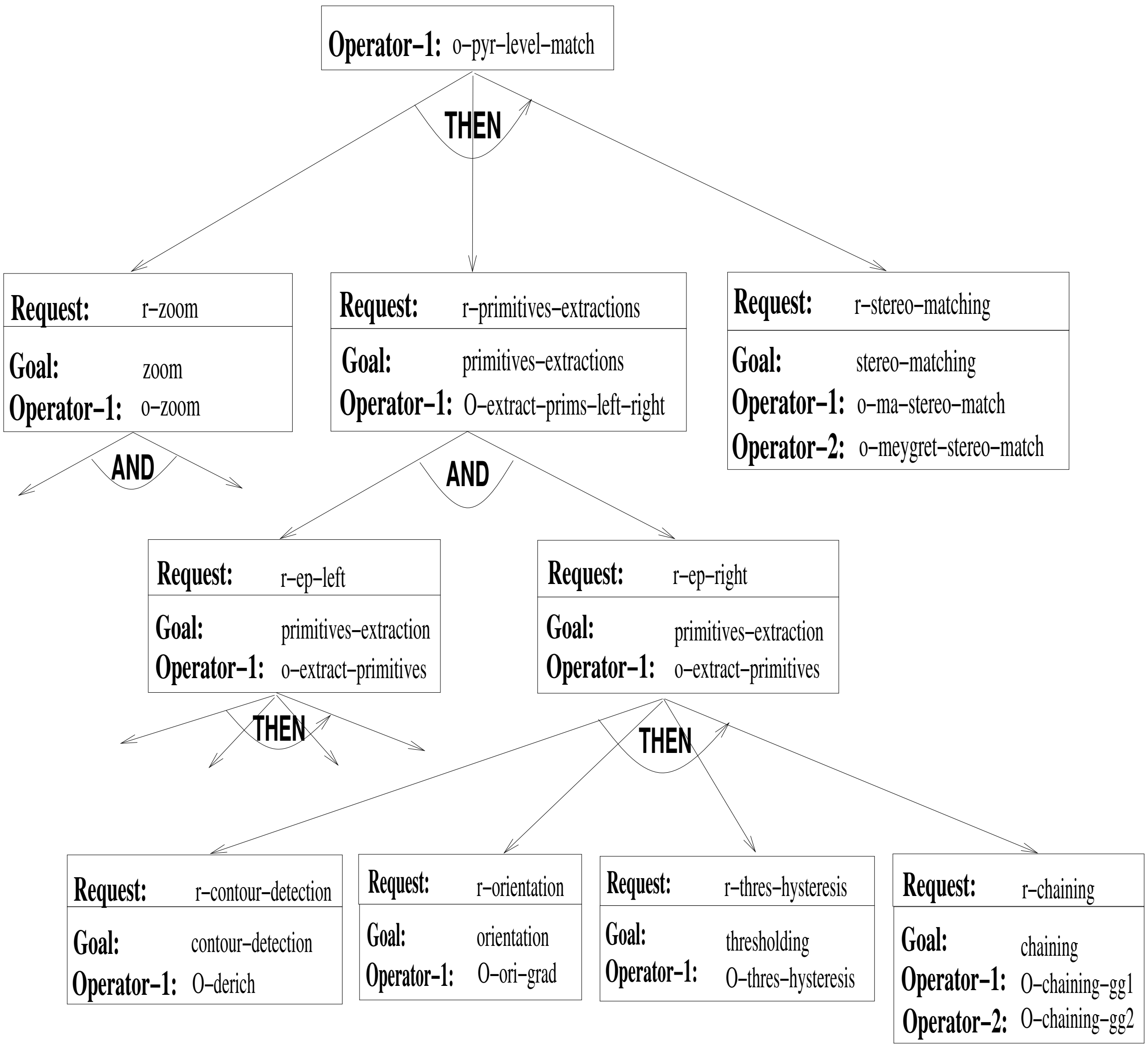

Figure 12: A part of the dynamic tree for the initial request *r-objects-10*

On every image resolution level the primitive extraction is followed by a stereo matching. Two operators are available to perform the stereo matching, *O-ma-stereo-match* and *O-meygret-stereo-match*. Based on the information in the context (high adaptability and average quality of the matching), the planning decided to apply the stereo matching operator *O-meygret-stereo-match*. On the second image resolution level, only 71 among 215 input primitives were matched. So the evaluation determined that the number of matched pairs of primitives was insufficient (less than a third of matched primitives). The matching operator has been rerun with adjusted parameters *thr-m* and *thr-o*. Then the number of matched primitives was sufficient (84).

4.2.2 Other initial requests

To test the performance of the knowledge base, other requests were executed besides the above request related to an in-town scene. In the case of PROMETHEUS as well as in the general case of autonomous systems, we are often concerned with sequences of successive images from the same scene, and indeed with images provided in various contexts. So we present the behaviour of the knowledge base on such a sequence taken from the previous in-town scene. Then we comment the performance of the knowledge base for the processing of images from 2 other scenes: a countryside scene, and another in-town scene.

First we present an experiment related to the processing of a sequence of successive pairs of images. This request is close to the request *r-objects-10* (figure 10): the scene is the same in-town scene, the input data are successive pairs of images, and the context is the same as shown in paragraph 4.2.1. The first data of the sequence have been processed as for the simple request *r-objects-10*. For the following data, the system used the decisions taken for the previous data to start its reasoning; they concern choice of operators as well as setting of values of parameters. For example for the operator *O-meygret-stereo-match* at the second image resolution level, the decisions concerning the first data were obtained after two trials; for the following data, only one trial was necessary thanks to a better initialization of the parameters. Such facilities for handling sequences of data optimize the processing time.

The knowledge base has also been tested on a countryside scene (see figure 13). Its nature is very different from the in-town scene; for instance, there is only very few objects in the scene. So even if the user constraints specified in the context were the same (mode automatic, many details to extract ...), the initial parameter values have been quite different, thanks to the large amount of initialization rules. The results obtained on this scene were of the same quality than for the in-town scene; the detected car is shown in figure 14. The knowledge base is thus well-adapted to the processing of various scenes.

Finally, we have tested the performance of the knowledge base for different context values for a second in-town scene. This scene (figure 15a) contains two kinds of obstacles: a vehicle and a cyclist. The vehicle is well contrasted but the cyclist is difficult to detect. An important step for obstacle detection is the extraction of the primitives. A human specialist interested in the complete detection of all the obstacles obtains very good results (figure 15b). Depending on the fact whether it is interesting to extract more or less information, two different contexts are defined. Images (15c) and (15d) show the contour chain points the knowledge base obtained in two different contexts (respectively for the values *many* and *all* of the context field *user-contraints.extraction.details*). For instance,

Figure 13: Countryside road scene.

Figure 14: Detected car in countryside road scene.

the first context (many details) is interesting for vehicle detection and tracking and the second context (all details) is necessary for obstacle detection.

5 Conclusion

In this paper we have addressed the problem of the supervision of perception tasks for autonomous systems. A model of supervision of perception tasks, including control mechanisms as well as knowledge modeling, has been proposed. Then, we have presented an implementation of this model designed as an expert system shell, named OCAPI. The facilities provided by OCAPI are shown, through an example: the supervision of object detection process in road scenes by the PROMETHEUS knowledge base.

We have seen with this example of supervision of object detection process, that an important set of knowledge has to be expressed. This knowledge is both numerous (especially in terms of initialization criteria) and rich (compound of several types of knowledge). The expression of this miscellaneous knowledge has been facilitated by the knowledge language provided by OCAPI; planning and trial-and-error mechanisms of OCAPI permit to have a very flexible system.

In the context of the Eureka project Prometheus, for a real time demonstrator vehicle,

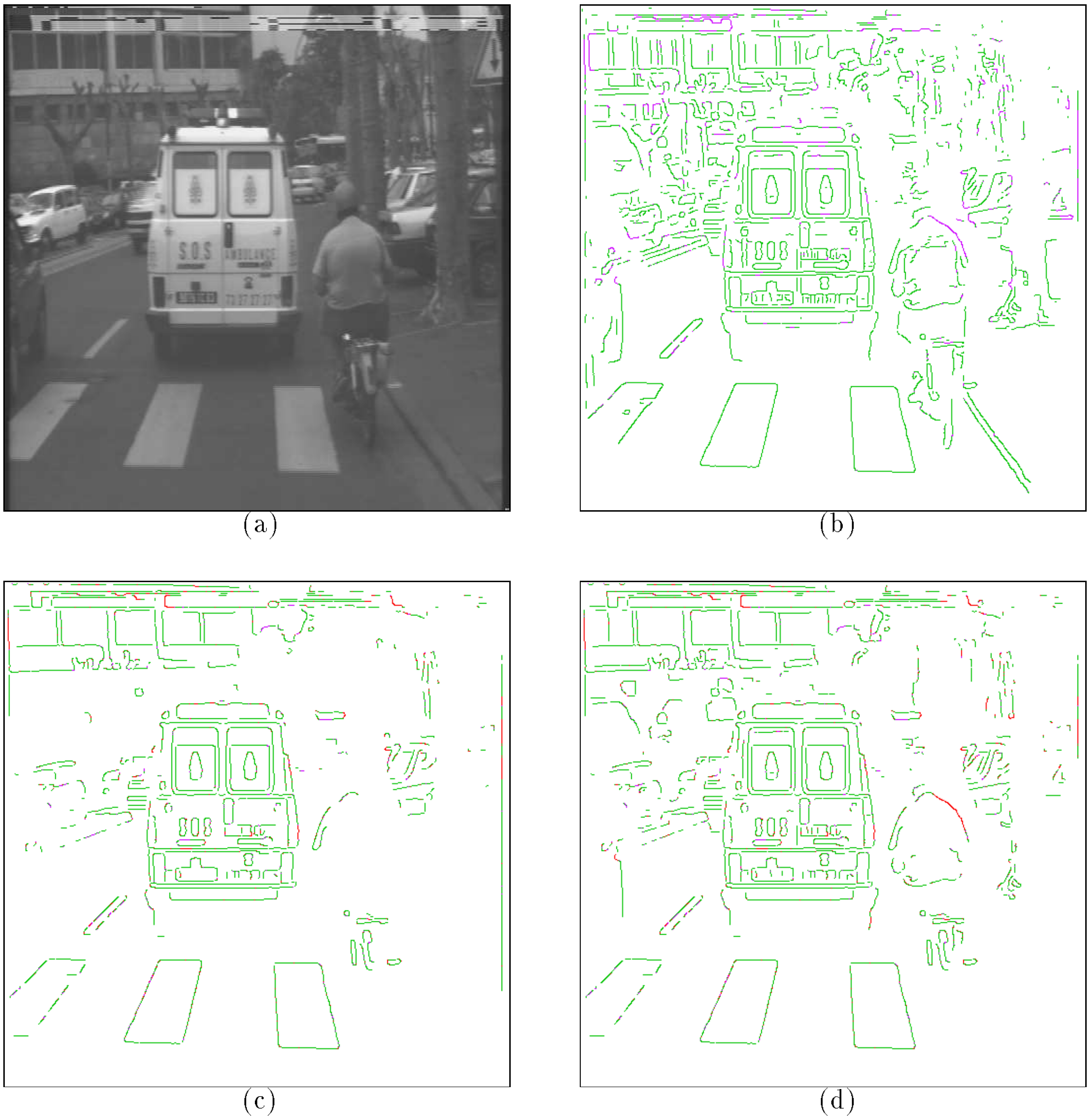

Figure 15: (a) Second in-town scene; with contour chain points extracted by (b) specialist; (c) and (d) the knowledge base in two different contexts.

named Prolab2, we are currently working on an implementation in C of this model for supervision of perception tasks and on the extension of the facilities for temporal data management.

References

[BCF82] A. Barr, P. R. Cohen, and E. A. Feigenbaum. *Handbook of Artificial Intelligence.* Pitman, 1982.

[Cha87] B. Chandrasekaran. Towards a Functional Architecture for Intelligence Based on Generic Information Processing Tasks. In *Proc. of the* 10^{th} *IJCAI*, pages 1183–1191, 1987.

[CT93] V. Clément and M. Thonnat. A Knowledge-based Approach to Integration of Image Processing Procedures. *CVGIP: Image Understanding*, 57(2):166–184, March 1993.

[DMC90] E. D. Dickmanns, B. Mysliwetz, and T. Christians. An Integrated Spatio-temporal Approach to Automatic Visual Guidance of Autonomous Vehicles. *IEEE Transactions on Systems, Man, and Cybernetics*, 20.6:1273–1284, November 1990.

[Eur91] Eureka PROMETHEUS Office. *Proceedings of the Workshops on Vision*, France, 1990-1991.

[FHR89] C. Fennema, A. Hanson, and E. Riseman. Towards Autonomous Mobile Robot Navigation. Tech.Rep. 89-104, University of Massachusetts at Amherst, Computer and Information Science, 1989.

[HTD90] J. Hendler, A. Tate, and M. Drummond. AI Planning: Systems and Techniques. *Ai Magazine*, Summer:61–77, 1990.

[Joh87] M. D. Johnston. An Expert System Approach to Astronomical Data Analysis. In *Proceedings, Goddard Conf. on Space Applications of Artificial Intelligence and Robotics*, pages 1–17, 1987.

[Mat89] T. Matsuyama. Expert Systems for Image Processing: Knowledged-based Composition of Image Analysis Processes. *Comput. Vision Graphics Image Process.*, 48:22–49, 1989.

[MTB90] A. Meygret, M. Thonnat, and M. Berthod. A Pyramidal Stereovision Algorithm Based on Contour Chain Points. In O. D. Faugeras, editor, *Lecture Notes in Computer Science* **427**, pages 83–88. Springer-Verlag, 1990.

[SKT88] H. Sato, Y. Kitamura, and H. Tamura. A Knowledge-based Approach to Vision Algorithm Design for Industrial Parts Feeder. In *Proceedings, IAPR Workshop on Computer Vision, Special hardware and industrial applications*, pages 413–416, Tokyo, 1988.

[TIY87] T. Toriu, H. Iwase, and M. Yoshida. An Expert System for Image Processing. *FUJITSU Sci.Tech.J.*, 23.2:111–118, 1987.

[TS88] T. Tanaka and N. Sueda. Knowledge Acquisition in Image Processing Expert System EXPLAIN. In *Proceedings, Int. Workshop on Artificial Intelligence for Industrial Applications*, pages 267–272, Hitachi City, 1988.

[Vog86] R. C. Vogt. Formalized Approaches to Image Algorithm Development Using Mathematical Morphology. In *Proceedings, VISION'86*, Detroit, 1986.

Contents

1 Introduction **2**

2 Supervision of perception tasks **3**

2.1 Planning for supervision 3

2.2 Control of execution for supervision 4

2.3 Knowledge for supervision 4

2.4 Conclusion 6

3 OCAPI: a task supervision tool **7**

3.1 Knowledge model implementation 7

3.2 Reasoning model implementation 8

4 Supervision of object detection in road scenes **9**

4.1 The PROMETHEUS knowledge base 10

4.1.1 Goals 11

4.1.2 Operators 12

4.1.3 Context 13

4.1.4 Data description 15

4.1.5 Choice rules 16

4.1.6 Evaluation rules 17

4.1.7 Initialization rules 18

4.1.8 Adjustment rules 18

4.1.9 Summary 19

4.2 PROMETHEUS knowledge base utilization 19

4.2.1 A request execution 19

4.2.2 Other initial requests 23

5 Conclusion **24**