AIAA-94-1282-CP

SAVA III:
A Testbed for Integration and Control of Visual Processes

James L. Crowley
LIFIA - IMAG
46 Ave Félix Viallet
38031 Grenoble, France

Henrik Christensen
Laboratory of Image Analysis, Aalborg University
Fr. Bajers Vej 7 D
DK-9220 Aalborg, Denmark

1 Introduction

During the last few years, there has been a growing interest in the use of active control of image formation to simplify and accelerate scene understanding. Basic ideas which were suggested by [Bajcsy 88] and [Aloimonos et al. 88] has been extended by several groups including [Ballard 91], and [Eklundh 92]. This trend has grown from several observations. For example, Aloimonos and others observed that vision cannot be performed in isolation. Vision should serve a purpose, and in particular should permit an agent to perceive its environment. This leads to a view of a vision system which operates continuously and which must furnish results within a fixed delay. Rather than obtain a maximum of information from any one image, the camera is an active sensor giving signals which provide only limited information about the scene.

Bajcsy observed that many traditional vision problems, such as stereo matching, could be solved with low complexity algorithms by using controlled sensor motion. Examples of such processes were presented by Krotkov [Krotkov 90]. Ballard and Brown [Brown 90] demonstrated this principle for the case of stereo matching by restricting matching to a short range of disparities close to zero, and then varying the camera vergence angles. The development of robotic camera heads has lead to the possibility of exploiting controlled sensor motion and control of processing to construct continuously operating real time vision systems.

At the same time, research in applying artificial intelligence techniques to machine vision led to an emphasis on the use of declarative knowledge to control the perceptual process. Systems such as the Schema System [Draper et. al. 89] developed a black-board architecture in which multiple independent knowledge sources attempted to segment and interpret an image. A major problem in such systems is control of perception. Such systems emphasise explicit representation of goals and goal directed processing which direct the focus of attention to accomplish system tasks. It has not been obvious how such a knowledge based approach to control of attention could be married to a real time continuously operating system.

In July 1989, the European Commission funded a consortium of six laboratories to investigate control of perception in a continuously operating vision system[1]. The consortium partners set out to build a test-bed vision system for experiments in control and integration. An experimental test-bed system was constructed which integrates a 12 axis robotic stereo camera head mounted on a mobile robot, dedicated computer boards for real-time image acquisition and processing, and a distributed system for image description. The distributed system includes independent modules for 2-D tracking and description, 3-D reconstruction, object recognition, and control. On March 18 1992, a fully integrated continuously operating vision system was demonstrated to the European Commission using this test-bed. This paper reports on the development of this system and the research which the system makes possible in control of a real-time vision system. A more complete description of the results of the project may be found in the book [Crowley–Christensen 93].

1.1 The Project Vision as Process

The starting point for the project "Vision as Process" was the demonstration of an integrated vision system capable of continuous real-time operation. It was quickly realised that such an ambition raises two problems:

1) The technical problem of integrating processes which model the environment in terms of descriptions which are qualitatively different.

2) The problem of controlling the "attention" and processing of a continuously operating system.

Concerning the first problem, different robotic tasks require different kinds of descriptions of a scene. Such descriptions can include 2D image description, 3D scene descriptions and symbolic labelling of the components of a scene. Such processes are complementary and mutually supportive. A framework is required which would permit the integration of multiple vision processes. This can be considered an "engineering" problem.

[1]The Partners in project ESPRIT "VIsion as Process" are Aalborg University (DK), University of Surrey (UK), KTH, The Royal Institute of Technology (S), University of Linkoping (S), University of Genoa (I), LTIRF-INPG (F) and LIFIA-INPG (F).

The second problem is both subtle and fundamental. Most of the algorithms used in vision have computational costs which depend on the quantity of data. In the best cases the relation is linear, but in many cases it is quadratic, cubic, or even exponential. Real time response requires that the processing time for any part of the system is limited. This requires that the amount of data considered during each processing cycle be bounded which raises the problem of which subset of the available data the system should attend during each cycle. This is part of the larger problem of controlling perception. General purpose real time vision system requires a solution to this problem.

These observations let the consortium to develop a long term work plan with both an engineering and a scientific goal. These are:

Engineering Goal:Develop techniques for integrating cyclic real time processes for description of a scene in terms of 2D images, 3D structure and labelled objects using active control of a camera head.

Scientific Goal: Develop methods (and a theory) of control of attention in perceptual processes.

With these twin goals, the consortium has developed a long term plan leading to the demonstration of methods for the construction of integrated continuously operating vision systems, and the elaboration of a theory for the control of such systems.

1.2 System Integration and Control

When the VAP project was conceived in 1988, only a small number of vision systems were capable of performing symbolic interpretation, and they were designed for interpretation of single (static) images. The well known examples included VISIONS [Hanson-Riseman 78], ACRONYM [Brooks 81], and 3DPO [Bolles-Horaud 84].

Most work on analysis of image sequences had been carried out on pre-recorded images and the level of description was almost entirely parametric. i.e., systems could describe regions or features with independent motion in terms of their image or 3D-velocity. A review of the state of the art is provided by Huang [Huang 83]. Continuous and real-time observation of a dynamically changing scene involves more than motion interpretation. A continuously operating vision process must be able to limit processing to a small subset of the data available from visual sensors, and to adapt its processing mode dynamically in response to events in the scene and requirements of the task.

1.3 The VAP Hypotheses

From its earliest meetings, the VAP consortium agreed that vision should be studied in the context of its purpose, i.e. its use by other processes. Without dedicating to any specific application, this implies that visual processing can be controlled to concentrate on the subset of visual information which is considered relevant to the current goal as defined by a user process. In addition, the consortium recognised the ability to exploit coherence in the dynamic evolution in a scene. In a continuously operating system, temporal context permits changes in the scene to be predicted and computational resources to be directed to confirm expectations. This implies that tracking is basic operation within a continuously operating system.

The goal of the VAP project is to demonstrate that a vision system must be designed as a continuously operating "process". To demonstrate this principle, the consortium has designed a six-year research program to develop techniques to interpret a dynamically changing, quasi-structured environment. These techniques exploit goal directed focus of attention involving controlled sensor motion and control of processing. Processing is directed by goals which change dynamically in reaction to the needs of the perceptual tasks and to events in the scene.

The following section reviews some previous approaches to integration and control. This previous work establishes the set of concepts and "prior art" from which the design of the VAP skeleton system draws.

2 Systems Architectures for Integration and Control

A suitable system architecture is required for experiments in integration and control of a continuously operating system. In this chapter, we review previous approaches to architectures of vision systems. From this review, we argue that flexible integration may be achieved through use of a standard module architecture, replicated at each of the levels in the system. Such a standard module architecture is described in more detail in chapter 3.

2.1 The Reconstruction Approach

A popular approach for structuring a vision system has been proposed by Marr [Marr 82]. Marr argues for a system defined around a hierarchy of representations: images, The primal sketch, the 2.5-D sketch (viewer centred depth map), 3-D map and symbolic description. In this model, processing is organised as sequential processes in which information flows up through the levels. Processing is data-driven, in the sense that recognition and description are based on descriptions constructed at the lower levels in the system. This model is computationally demanding and it has proven difficult (if not impossible) to provide image descriptors that are sufficiently robust to allow characterisation of all phenomena in a natural environment.

The Marr processing model may be termed a *reconstruction approach* as it aims at a full reconstruction of the environment. The processing model is purely data driven, and thus poses a problem in terms of computational resources. The Marr model

assumes all processing may be carried out as a sequential process. This implies that a module uses the representation(s) at the level just below as a basis for its processing and the result is stored in the next higher representational level. The interfaces are consequently well defined. A simplified model of the processing is shown in figure 1.

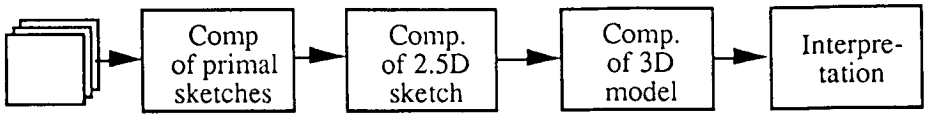

Fig. 1 A processing model for the reconstruction approach.

In terms of representations, this processing model implies that information needed to perform recognition and interpretation of settings must be available as part of the 3D model. i.e., a diverse set of descriptors must be tagged onto the 3-D model representation to facilitate recognition and description. This duplication of information up through the system and the unavailability of pixel level primitives can pose a problem in terms of model size and maintenance over time.

2.2 The Non-Committal Approach

In the VISIONS system [Hanson-Riseman 1978] data are stored on a blackboard, or common storage area and processing is performed in parallel by a number of "knowledge sources". All modules in the system can access information at any of the representational levels. This implies that information does not have to be replicated up through the system to be made available for recognition procedures. A simplified model of the non-committal approach is illustrated in figure 2.

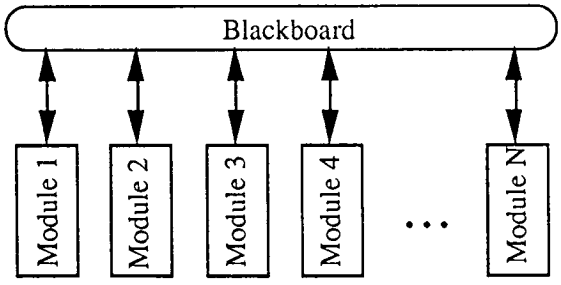

Fig. 2. Architecture for the non-committal approach.

In this architecture, each of the modules in the system has control information that specify the information that must be available before the module may carry out its task. In addition, it has control information that specifies the information which may be provided by the module. Through use of a control executive, it is possible to perform both goal directed and data driven processing through use of this control information.

This processing model imposes few constraints on the representations used in the systems and it is simple to add new modules to the system. The common blackboard is a potential problem for a continuously operating system. All information generated and used by the system passes through the blackboard. It might thus pose a problem with respect to information bandwidth. To indicate the amount of information, which may be generated directly from images one may look at the Image Understanding Architecture, described by Weems et al. [Weems 1989]. In that system, the storage reserved for intermediate representations is 4 GB and the system is only aimed at analysis of single images. Such an architecture will require extensive use of special purpose hardware, in particular when applied in the temporal context.

2.3 The Purposive Approach

Introduction of goal directed operation and use in a limited and well defined domain of application allow synthesis of a vision system which is composed of a set of specifically engineered modules. Such modules may be designed to be computationally well behaved, in the sense that the computational complexity is bounded and often robust representations can be provided. This approach to construction of vision system has been promoted by Bajcsy [Bajcsy 88], Ballard [Ballard 91] and [Aloimonos et al. 88]. Although the approach exploits specific vision modules, Bajcsy has tried to enumerate a set of modules that might form a general purpose system. The use of dedicated modules is a way to provide robust information and computationally tractable techniques. Well known examples of dedicated modules used for robot navigation are optical flow modules that can compute the position of the focus of expansion for the optical flow field, and modules which can compute the time to contact from motion in an image sequence.

This approach to system construction is termed *the purposive* or *the animate* approach. It is envisaged that the construction and analysis of specific modules will gradually provide insight that will allow definition of modules applicable in general vision systems. The convergence towards a standard set of modules through analysis of diverse application domains might provide valuable insight, but it is not obvious that convergence will be achievable.

In the purposive approach, the exploitation of information is task driven and may be very different from one task to the next. The basic system architecture should thus be flexible and facilitate dynamic change of the information flow. In practical systems a number of modules may exploit the same representation and once a system has been defined an

analysis of the representational requirements may point to the definition of a set of standard representations. Given present state of the art no such general representations are known.

A purely purposive approach to vision rejects most of the established techniques. Modules are to be built from scratch whenever a new type of information/representation is required. There is consequently a concern that from this approach little insight will be gained in terms the general vision problem.

2.4 The Vision as Process Architecture

During the first year, the consortium "Vision as Process" addressed the problem of design of an architecture which would meet the following criteria:

1) Continuously operating.

2) Integrating software contributions from geographically dispersed laboratories

3) Integrating description of the environment with 2D measurements, 3D models and recognition of objects.

4) Capable of supporting diverse experiments in gaze control, visual servoing, navigation and object surveillance.

5) Dynamically reconfigurable as the task changes.

The result was the design of a distributed test-bed system composed of independent modules. Modules may communicate by message passing over a central message server, or by dedicated "high-band width" channels. Systems can be composed from sub-sets of the available modules for individual experiments.

The architecture adopted by the consortium is shown in figure 3. The system has a data flow part, which is similar to the Marr processing model. It should be noted that the data flow is not only bottom-up but may also be top-down. Top-down expectations (derived from the present set of goals and contextual information) can be used to direct/control processing at lower levels, while detected event at the same time can drive a reconstructive mode of processing. The VAP architecture contains a common communication channel that allow communication between any pair of modules in the system. This communication channel may be used both for investigation of a non-committal processing model, and for investigation of purposive systems, as the component processes in the system in figure 3 may either general purpose or dedicated.

This architecture imposes few constraints on the design of component modules and it provides flexibility for the investigation of system level control issues. Initially it was envisaged that the main flow of data would exploit the communication links between adjacent modules, while only control information would be communicated through the common channel. During the execution of the project it was realised that a more flexible processing model was needed to make computations both efficient and robust.

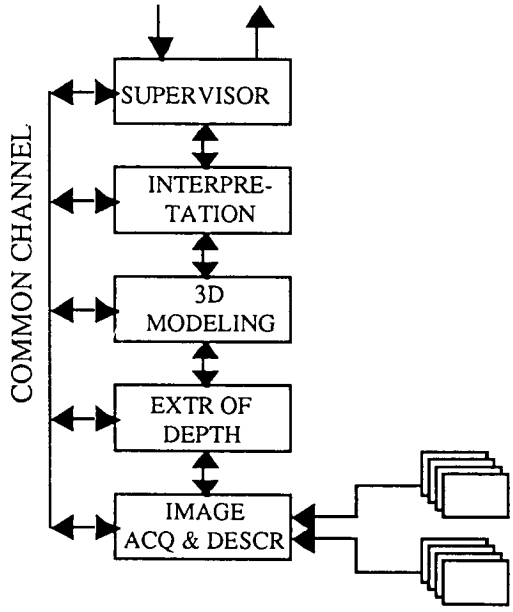

Fig. 3. The VAP system architecture

Having selected a distributed architecture composed of modules, the consortium turned its attention to the design of a common component for each module. At the very least, this standard module must provide the communications interface. It was soon observed that scheduling was a basic to continuous operation and a cyclic scheduler was provided which calls the procedures which implement each phase of computation. The phases of operation included phases for integration of new data and phases for control of processing.

In order to obtain temporal context, the consortium drew on previous results in image tracking. A tracking architecture was defined composed of the phases predict-match-update [Crowley et al. 88], [Granum-Christensen 88] based on techniques used in the control community since the early sixties [Kalman 60]. The architecture is shown in figure 4.

The *analysis* block, in figure 4, is responsible for the frame by frame analysis, which generated a set of geometric primitives (or tokens). The correspondence with information in the temporal context is performed in the *match* block. To simplify matching the information in the temporal context is used in a prediction of the expected content of the next frame. Once correspondence has been established the information contained in the internal models must be updated to reflect the new information contained in the new frame. Once updating has been carried out the cycle may start all over again.

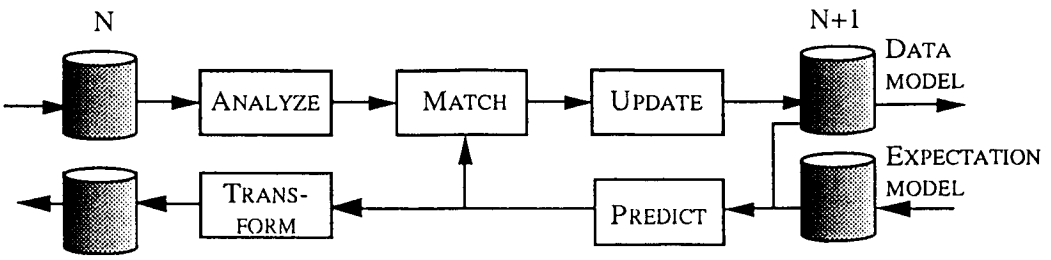

Figure 4. Basic Predict-Match-Update cycle for the module architecture.

As a model at level N+1 is used for prediction of primitives in the next frame, the predictor may also be given other types of input which can be used for guidance of processing. Introduction of goal derived information into the model at level N+1 will consequently allow top-down/attention based control of processing. A prediction may be transformed into a representation that is compatible with the one used the level below, so that it may drive processing at the level below. This flexibility facilitates investigation of different control strategies.

2.5 Control Issues

Construction of an operational system includes issues in control to ensure satisfaction of user defined goals. Goals are widely recognised as a fundamental component of intelligent systems. The consortium initially defined a set of general goal commands :

search(X): Is X present in the scene?

find(X): Where is X, given X has been identified earlier?

relate(X,Y): What is the spatial relation between X and Y?

describe(X,Y): Determine property Y for object X

watch(X): Allocate resources for notification of changes for X

track(X): Maintain a dynamic description of object X.

This set of goals defines the user level interface to the system. Based on the potential goals, the system must be able to allocate its resources for optimal satisfaction of the concurrent goal(s).

A number of approach to goal directed processing have been reported in the vision literature. Most of these efforts include use of a cyclic process, in which data received are matched against expectations. Depending on success or failure in the matching process an updating or event detection process is used to drive the next cycle of processing. An example of such a cyclic process is described by [Tsotsos 87] for the ALVEN system.

When the VAP effort was initiated the use of production systems and reasoning under uncertainty appeared the most promising in terms of providing insight into the problem of the cycle of planning, sensing and interpretation. These tools have been incorporated into the system. In system level control, externally defined goal commands are translated into actions by rule based planning. Planning generates the sequence of state transitions (actions and their parameters) expected to allow completion of a goal. These actions are then executed by one or more system modules. The internal handling of such actions is an issue that is resolved for each of the modules. A skeleton system constructed by the consortium provides the framework for experiments in control and coordination of visual processes.

3. The SAVA III Skeleton System

In order to perform experiments in control and integration of a continuously operating vision system, the VAP consortium constructed an empty "skeleton" system. This skeleton was then provided to partners so that they could "fill in" the functional parts needed for their experiments[2]. This system was named SAVA, for the french acronym "Squelette d'Application pour la Vision Active". The SAVA Skeleton system provides a standard module with communication and interface components that permit an experimenter to construct and run distributed real time vision system.

The structure of SAVA has evolved with our understanding of the problems of integration and control. The original SAVA system was released at month 12 of VAP-I. Experiences during the second year of VAP-I brought out a number of shortcomings in the design. A team composed of people from AUC, KTH and LIFIA designed a revised version, SAVA II, which was released at the month 24 milestone. An intense integration effort was performed in preparation of the month 33 integrated demonstration, with software and hardware contributions from all VAP partners integrated into SAVA II. Modifications in communications and interface design, as well as a large number of small improvements, led to release of SAVA 2.4 in March 1992.

Experience with SAVA II has shown the importance of demons for combining purposive and event driven control of perception. This led to the desire for an interpreter for demon functions. In addition,

[2]The incompatibility of successive releases of MOTIF have created problems for portability and have cost the consortium considerable time and money.

programming control experiments in SAVA II was a somewhat difficult task. Control knowledge was embedded in procedural code and thus hard to understand or change. It was decided to design a control system based on an interpreter for declarative expressions of the control logic. From these two needs emerged the idea of using the CLIPS 5.1 rule interpreter for the control component and the demon interpreter within each module. CLIPS 5.1 is written in C and is provided with the full source code. As a result, it was extremely easy to integrate the SAVA modules into the CLIPS environment.

A new version of the skeleton system, SAVA III, has been created based on the principle of interpreting control information. In SAVA III, most of the procedures for processing and communication are written as C procedures and explicitly declared to the CLIPS 5.1 rule interpreter. Rules and functions are then written using these procedures. The basic processing cycle is built as a sequence of states with transitions managed by rules. The processing performed within a state can be easily changed based on either perceptual events or external commands. Because the control rules are interpreted, the control sequence for a module may be changed dynamically, without re-compiling a module. It is even possible for a module to send another module function definitions as ASCII messages, using the CLIPS deffunction facility. The rule based scheduler is particularly useful for the implementation of demons. Demons may be programmed as rules which react to the contents of the model as well as to external messages.

In addition to the changes in the control part, a major effort has been made to add the possibility of synchronised operation to the modules. SAVA is a soft real-time system, distributed over a set of workstations operating under UNIX. At some time in the future, we intend to port SAVA onto dedicated hardware running under a real time programming environment. However such systems are relatively difficult to program and debug. The use of UNIX and distributed processing permits the VAP-II project to perform experiments with a reasonable effort.

A synchronisation system has been built into SAVA III in order to compensate for uncertainties in communication and execution time for distributed modules. A synchronisation module provides other modules with a universal time reference. In this way, all information that is processed or communicated is time-stamped, permitting an estimate of dynamic processes to be observed or controlled.

The following sections present a detailed description of the components of the SAVA III system. It first gives a brief overview of the components of the skeleton system and its standard module. It then describes processes for interpreting messages using a rule based interpreter, and the design of "demon" processes that perform pre-attentive detection of events. A description of the rule based control of a module is presented, followed by a description of the synchronisation of modules.

3.1 Overview of the SAVA III Software Skeleton

The SAVA skeleton system is composed of the following components:

1) A launcher program that permits the user to assign modules to processors and to initiate operation.

2) A distributed mailbox system that is launched on the different processors to establish a communications system and to launch the component processes.

3) A library of communication procedures for modules. This library include procedures for communication by message as well as procedures for dedicated high band-width communication between processes.

4) A skeleton module structure built around a scheduler.

5) A set of graphical man-machine interfaces.

The SAVA system provides mailbox communication for data, control and acknowledgements, as well as a procedures for dedicated high-band-width channels between modules. Messages include formatting information that permits the message passing system to pack and unpack messages.

Visual perception is performed within processes imbedded in copies of the SAVA "standard module". The SAVA III standard module is shown in Figure 5. The standard module is composed of a number of procedures (shown as rectangles) that are called in sequence by a scheduling process.

A SAVA module repeatedly executes a cycle in which it:

1) Acquires new data.
2) Transforms this data into an internal representation.
3) Makes predictions from its internal model.
4) Matches the predictions with the transformed data.
5) Uses the match results to update the internal model.
6) Executes demons to detect perceptual events within the internal model.

This cyclic process is executed by a rule-interpreter. Each phase corresponds to a state in which a particular operation is performed. At each state transition new messages which have arrived on the mail box channel are read and processed. Such messages may change the procedures that are used in the process, change the parameters that are used by the procedures, or interrogate the current contents of the description that is being maintained.

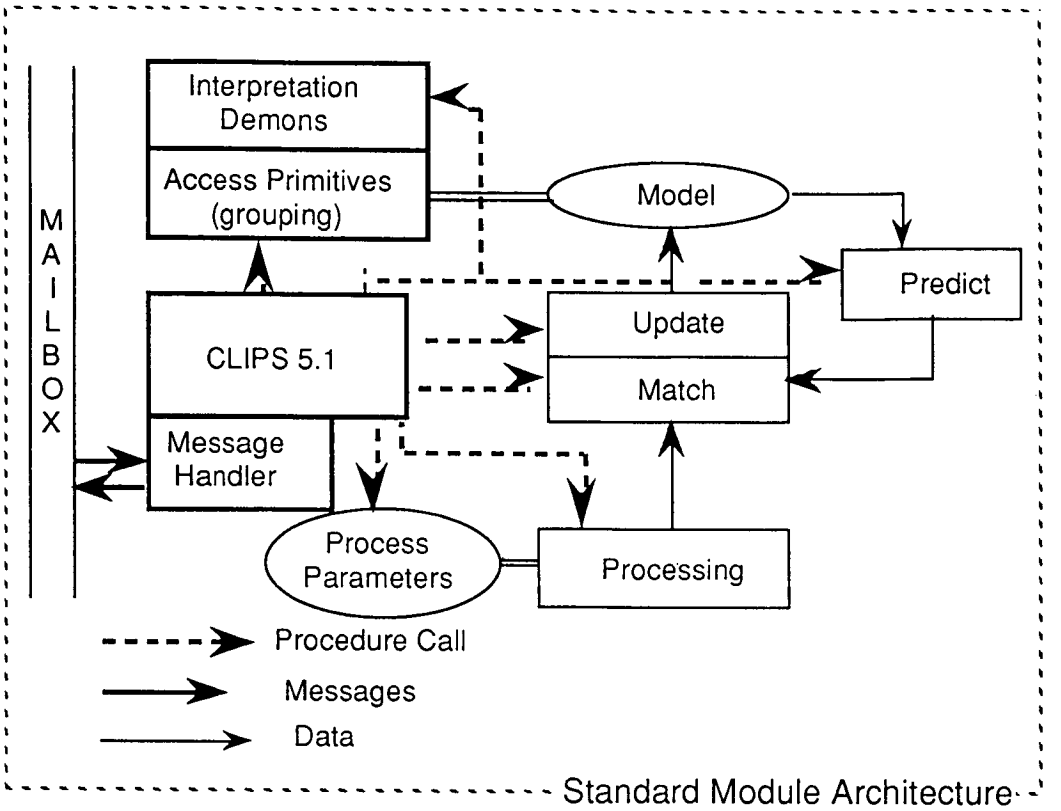

Figure 5 Architecture of a Standard Module in SAVA III

The cyclic process within a module is managed by a control token placed on the working memory. This control token is a simple list in which the first atom is the word "phase" and the second is the name of one of the phases: {get-data, transform, predict, match, update, messages, demons}. The definition of theses phases is as follows:

get-data Acquire a new observation

transform Transform the data to the internal representation

predict Predict the contents of the observation

match Match the prediction to the observation

update Update the model using the correspondence of the prediction and the observation

demons Execute a set of automatic procedures for event detection.

At the end of each cycle, the scheduler executes a set of demons. Demons are responsible for event detection, and play a critical role in the control of reasoning. Some of the demon procedures, such as motion detection, operate by default, and may be explicitly disabled. Most of the demons, however, are specifically designed for detecting certain types of structures. These demons are armed or disarmed by recognition procedures in the interpretation module according to the current interpretation context.

In the SAVA III system, the procedures of a module are made explicitly available to the CLIPS 5.1 rule based interpreter. This includes the original SAVA II scheduler, so that the system is upwards compatible. In addition to acting as a scheduler, the rule interpreter is also used to define the control part of demons and to interpret messages from other modules.

3.2 Communications Between Modules

Modules communicate control, data requests, reply and synchronisation information using message passing based on Unix Sockets. The SavaSend() function is used to send mailbox messages to other modules. A SavaSend command contains three fields:

Header: The destination and type of message.

Format: An ASCII description of the message format.

Body: The message including commands and parameters.

The destination is a symbolic name for another module. The types of message may be control, acknowledge or data. All message exchanges are initiated by a control message. The format string is transmitted with the message and is used both to encode and decode the message. In this way a change in message protocol may be made with a minimum of difficulty. This format string can contain conversion directives like %d, %f, and %c, based on the C language printf protocol. We have added conversion

directives for sending arrays, structures and images.

Many SAVA functions accept a variable number of arguments. Furthermore, the type of these arguments is unspecified. These functions accept a fixed number of normal arguments, followed by an arbitrary number of arguments of unknown type. The last normal argument is a format string which describes the arguments following it.

Large data structures may be communicated between modules using dedicated sockets. Communication of dedicated channels are performed by the functions SavaRead and SavaWrite. As with mail-box, messages, high band-width channel messages are encoded with an ASCII format directive which is transmitted with the message. High band-width channels in SAVA are faster than message passing because the channels provide a direct connection. No intermediate routing is necessary.

Messages passed through the mail box communication system are interpreted by the rule interpreter. New messages are transformed into working memory elements by the function "checkmessage". Checkmessage creates a list in working memory composed of the keyword "message" followed by the name of the sender, a message keyword and and ASCII string with the body of the message. Checkmessage assures upwards compatibility with message types that were defined in SAVA II which have not be transformed to SAVA III.

The checkmessage function is executed at the end of each phase of the standard module. For example, the transition from match to update is performed by the rule "update-phase" :

```
(defrule update-phase
      (declare (salience -100))
      ?p <- (phase match ?c)
=>
      (check-message)
      (retract ?p)
      (assert (phase update ?c))
)
```

The result of check-message is a list of the form:

```
(message  ?sender ?command ?body)
```

If ?command string is tested to determine how the message should be interpreted. If command corresponds to one of the CLIPS keywords "deffunction" or "defrule", then ?body is interpreted by the CLIPS function BUILD. This is permits external modules to define functions and rules using the CLIPS deffunction and defrule constructs.

If ?command corresponds to a previously defined function, then ?body is executed using the clips "eval" construct. If ?command is unknown, or the interpretation is not successful, eval returns FALSE. The result returned by eval is used by the function "reply" to send a reply to the sender. If the message evaluates to a "NIL", then reply does not send a message.

The CLIPS function "build" will interpret a string as if it has been typed to the interpreter. This may be used to interpret defrule and deffunction messages from other modules, as shown by the rule "interpret-def-commands". The command "mv-append" is used to compose a list with the desired commands.

```
(defrule interpret-def-commands
      (declare (salience 100))
      ?m <- (message ?sender ?command ?body)
      (test (member ?command
            (mv-append deffunction defrule)))
=>
      (reply ?sender (build ?body))
      (retract ?m)
)
```

Functions may be defined at initialisation or by messages from other modules. A function, encoded in an ASCII string, may be executed using the CLIPS "eval" command, as shown by the rule "interpret-function-messages"

```
(defrule interpret-function-message
      (declare (salience 100))
      ?m <- (message ?sender ?command ?body)
      (test (member ?command
            (mv-append list-deffunctions))
=>
      (reply ?sender (eval ?body))
      (remove ?m)
)
```

If the interpretation is not successful, eval returns FALSE. Unless the result is NIL, it is sent to the ?sender in a reply message.

3.3 Automatic Interpretation by Demon Processes

A demon is an automatic procedure which operates on the internal model of each module to detect events. Currently active demon procedures are executed after the update phase of each cycle. Demons are responsible for event detection, and play a critical role in the control of reasoning. Some of the demon procedures, such as motion detection, operate by default, and may be explicitly disabled. Most of the demons, however, are specifically designed for detecting certain types of structures.

Demons may be invoked by other demons or by commands received from other modules, including from a human supervisor. A demon is instantiated by entering a demon token in working memory. A demon token is simply a list with three elements:

```
(demon  <name> <id>)
```

where <name> is the name of the demon and <id> is a unique identity determined by the function "gensym". Multiple copies of the same demon can be instantiated, each having its own "id". Each demon can create its own state in working memory, indexed by <id>. A demon can be removed by removing the demon token

from working memory.

The control part of a demon is encoded as rules. As an example consider a demon to find ellipses in the image:

```
(defrule ellipse-finder   "The demon for an
ellipse finder"
      (phase demons)
      (demon ellipse-finder ?id)
      (ellipse-demon-data  (id ?id)
                  (parameters ?p))
=>
      (assert (get-ellipses ?p)))
```

If we suppose that the function "get-ellipses" will instantiate a structure of type ellipse for each ellipse found, then a second rule can be written to treat each ellipse.

```
(deffrule  hypothesize-cylinder "generate
cylinder hypotheses"
      (phase demons)
      (demon cylinder-finder)
      (ellipse (id ?id) (cx ?x) (cy ?y)
            (major ?ma) (minor ?mi)
            (angle ?angle))
      (test (< 5  (abs ?angle)))
=>
      (assert (cylinder  (cx ?x) (bottom ?y)
            (radius ?ma) (ellipse ?id)))
      (assert (cylinder  (cx ?x) (top ?y)
            (radius ?ma) (ellipse ?id))))
```

Other rules can be used to detect the existence of cylinders with the same axis and to reduce cylinder hypotheses to a minimum number, or to use the hypothesis of a cylinder with several ellipses to generate the hypothesis of a cup.

Goals for the module can be entered into working memory as a three element list:

```
(goal  <name>  <priority>)
```

Goals can then have the effect of activating and dis-activating demons. An example of a goal invoking a demon is the rule "cup-demons".

```
(deffrule  cup-demons
      "invoke the cylinder finder"
      (phase demons)
      (goal find-cup ?p)
=>
      (assert (demon cylinder-finder)))
```

An example of removal of demons is the rule "remove-non-cup-demons".

```
(deffrule  remove-non-cup-demons
      "remove cup demons"
      (phase demons)
      (goal find-cup ?p)
 ?d <-(demon ?n ?id)
      (not (?n cylinder-finder))
=>
      (retract ?d))
```

Having a rule interpreter provides explicit control knowledge for demons and their control logic. It also permits the working memory to be used to create and free working memory for representing demons state. The result is a flexible, easy to use, tool for experiments in control of perception. In the following section we present an example of such control.

4 The Visual Navigation Demonstrator

This section illustrates the use of SAVA III by presenting an overview of the a visual navigation system. This system was constructed for the milestone 1 demonstration of VAP-II presented in June 1993. The structure of the demonstration system is shown in figure 6. The system is composed of processes for

1) Fixation control of the binocular head.
2) Local navigation actions for a mobile robot.
3) Image acquisition and processing.
4) Tracking and grouping a 2-D description of the contents of the image.
5) Computing and maintaining a 3-D description around a fixation point.
6) Recognition of landmarks and object.
7) System Supervisor for coordinating processing of the other system modules.

Fixation Control Unit

The fixation control unit provides a standard interface to the device controller for the VAP/SAVA binocular stereo head. This module maintains a copy of the current state of the fixation point and the component axes for the binocular head. It receives commands in the form of tasks expressed in either device or motor coordinates. Commands are communicated to the binocular head, the robot-arm (neck) or the mobile platform using such device-level control.

The fixation control unit also contains facilities for programming procedural style "perceptual actions". Such perceptual actions are reflex procedures that command the state of the binocular head at either the device level or the motor level based on measurements made from images. Examples of low level perceptual actions include ocular reflexes for servoing aperture, focus and vergence. Other examples include procedures for tracking a moving object.

Image Acquisition and Processing

The image acquisition and processing module handles all image processing requirements for the other modules, thus minimizing the communication requirements. This module is based on two computer cards constructed by the consortium. The first of these, the Pyramid card digitizes synchronised stereo images and immediately computes a 12 level binomial pyramid for the two images. Processing time for each pair of images is 40 ms. The second card extracts edge segments using Gaussian derivatives.

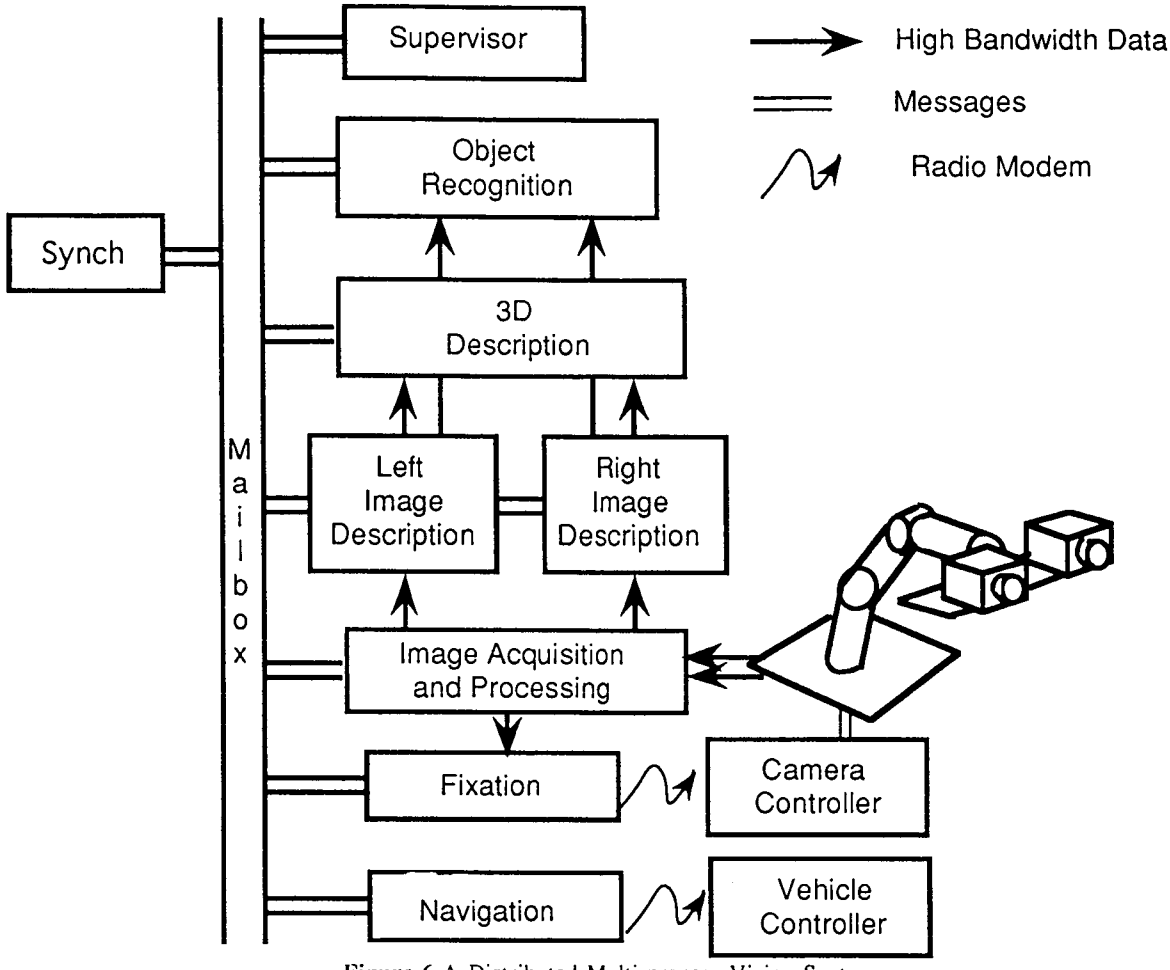

Figure 6 A Distributed Multi-process Vision System.

The edge extraction process begins by calculating the horizontal and vertical derivatives within the region of interest. These derivatives are then combined by table look-up to compute the gradient magnitude and orientation. Points which are extrema in magnitude are marked as potential edge points and compared against two thresholds. Hysteresis thresholding is applied so that only regions of edge points containing at least one point above the threshold are considered. Adjacent edge points with a similar orientation are grouped to form line segments. Edge segments are represented by a vector of parameters that includes the mid-point, orientation and half-length.

Three classes of image processing procedures are available in the image processing module

1) Edge Segment Extraction. On command, the module will transfer the pixels within a region of interest to an edge extraction card produced by the consortium. This card computes the gradient magnitude and orientation and detects pixels which are extrema in gradient magnitude. Detected pixels are grouped in single raster scan to construct edge segments. Gradient magnitudes are compared to two thresholds to provide a hysteresis based thresholding.

2) Edge Chain Extraction: In place of edge segments, another module may request edge chains. Edge points are are computed by the same algorithm as for edge segments. A one pass raster-chaining algorithm is used to construct a list of edge chains within the region of interest. The edge chaining code is computed on a co-processor card based on the Intel 680

3) Measures for Ocular Reflexes. In order to avoid communicating images, the measurements on which ocular-motor reflexes are based have been placed in this module. Measures include coarse to fine computation of phase for convergence, and gradient based measurements for aperture and focus.

Image Tracking and Description

An image description is maintained by a tracking process which uses a first order Kalman filter to track edge segments. This tracking process improves the stability of image primitives, permits the system to maintain correspondence of image features over time, and provides an estimate of the position and velocity of image primitives as well as the uncertainty of these

estimates. It also permits information about the movement of the head or vehicle to be used to compensate for movements by the robot. A vocabulary of model access and grouping procedures give associative access to the 2D description modules. These procedures are used by a library of "demon" procedures which can be enabled in order to provide data driven interpretation of the image description.

Separate image description modules exist for the right and left cameras. The 2D image descriptions are maintained by a tracking process that uses a first order Kalman filter to track image description primitives. This tracking process improves the stability of image primitives, permits the system to maintain correspondence of image features over time, and provides an estimate of the position and velocity of image primitives as well as the uncertainty of these estimates.

Model access primitives use matching and grouping to interrogate the contents of the token model. A set of demons may be invoked by other modules to interrogate the description after each update using the model access primitives. Access to the 2D model is provided by a large vocabulary of model access and grouping procedures. It is also possible to compose sequences of these grouping procedures, extracting, for example, all the junctions near an ideal line. These procedures may be called by other modules within the system, or they may be invoked by a set of interpretation demons. These demons are placed on an agenda by messages from other modules. After each update cycle the demon agenda is executed.

3D Geometric Scene Description Module

In addition to a description of images, the skeleton system maintains a geometric description of the scene. This geometric description expresses the structure within a region of interest of the scene in terms of 3D parametric primitives. This module assumes that the phase based convergence reflex maintains the cameras converged on an object. Convergence maintains edge segments from a region of interest in the scene in the similar positions in the image. The image description access primitive "FindPrototypeSegment" is used to construct a list of possible matching segments in the left and right image. This list is sorted based on similarity of length, orientation and position. The most likely matches are selected for 3D reconstruction.

Reconstruction requires camera calibration. A novel procedure for dynamic auto-calibration of cameras has been developed. This procedure permits a reference frame for a pair of stereo cameras to be constructed for any scene objects. The projective transformation matrices from object centered coordinates can be obtained by direct observation (no matrix inversion) and can be maintained by a very simply operation. These matrices make it possible to reconstruct the 3D form of objects in an object centered reference frame. As with the image tracking and description module, the geometric description is maintained by a tracking process in order to provide stability and to maintain correspondence over time.

Symbolic Scene Interpretation

The symbolic scene interpretation maintains a symbolic description of the scene in terms of known object categories (or classes) and qualitative relation. This description is built up and maintained by interrogating the contents of the image and scene description modules. The SAVA III symbolic description process was implemented using the CLIPS rule interpreter system. Rules implement a hypothesize and test process which is triggered by demons. Working memory of the production system serves as a blackboard into which recognition procedures can poste their results.

Process Supervisor

The process supervisor maintained a list of places and routes which the system is to travel, as well as a data base of "landmarks" which the system is to find during mission execution. The supervisor plans a navigation which it then executes by sending commands to the other modules. An interesting aspect of the supervisors operation was coordinating between the competing tasks of watching in front of the robot for obstacles and searching for landmarks for position correction. Obstacles must be searched at least once every 50 cm, while landmark detection is required whenever the uncertainty of the estimated position passes a certain threshold. Both operations require command of the camera head. This balancing act was performed by a finite state automata programmed as a set of rules.

Navigation Control

The navigation module controls vehicle actions by sending commands to an on-board vehicle control program. The on-board program, known as the "standard vehicle controller", provides asynchronous independent control of forward translation and rotation. The on-board controller acts like auto-pilot, stabilizing the vehicle and estimating its position. The controller accepts both velocity and displacement commands, and can support a diverse variety of navigation techniques. The controller is capable of responding to commands at any time using a simple serial line protocol. New commands for displacement immediately replace previous commands. This permits visual servoing to be used to pilot the vehicle.

Position and orientation are modelled in the vehicle controlled using Kalman filter to maintain an estimated position and covariance. The control protocol includes a command to correct the estimated position and orientation and their uncertainty from external perception using Kalman filter update. This command has been used to update the estimated position by observing the angle to known objects. The LIFIA standard vehicle controller is described in greater detail in [Crowley-Reignier 93]. The navigation module contains procedures to detect and avoid obstacles, and to locate and use landmarks for updating the vehicle's estimated position.

5. Conclusions

According to the principle of "purposiveness" a vision system operates in order to furnish an observation function for some task. In order to enrich our task domain, we have adapted the VAP Skeleton system to serve as the visual component for a mobile robot navigating in an indoor environment. We stress that the visual navigation is not, in itself, the goal of the project. Visual navigation is a task which is sufficiently rich in events to explore the problems of integration and control of an active perception system.

During the last four years, the VAP consortium has constructed a number of demonstrations of continuously operating vision systems. In each of these systems explicit control of sensor motion and processing has permitted the system to operate in real time, with increasingly degrees of robustness. The consortium experience has verified the VAP hypothesis that control of continuously operating process is basic to the design of a general purpose real time vision system.

Bibliography

[Aloimonos et. al. 88] Aloimonos, J. Y., I. Weiss, and A. Bandyopadhyay, "Active Vision", International Journal of Computer Vision, Vol. 1, No. 4, Jan. 1988.

[Bajcsy 88] R. Bajcsy, "Active Perception", IEEE Proceedings, Vol 76, No 8, pp. 996-1006, August 1988.

[Ballard 91] D. Ballard, "Animate Vision", Artificial Intelligence, Vol 48, No. 1, pp. 1-27, February 1991.

[Bolles-Horaud 84] Bolles, R. C. and Horaud, P., "Configuration Understanding in Range Data", Second ISRR, August, 1984.

[Brooks 81] R.A. Brooks, "Symbolic Reasoning among 3-D models and 2-D images", Artificial Intelligence, Vol. 17, pp. 285-348, 1981.

[Brown 90] C. Brown, Prediction and Co-operation in Gaze Control, Biological Cybernetics 63, 1990.

[Crowley 87] Crowley, J. L., "Coordination of Action and Perception in a Surveillance Robot", IEEE Expert, Vol 2(4), pp 32-43 Winter 1987. (Also in the 10th I.J.C.A.I., Milan 1987).

[Crowley et. al. 88] J. L. Crowley, P. Stelmaszyk and C. Discours, "Measuring Image Flow by Tracking Edge Lines", Second I.C.C.V., Tarbon Springs, Fla, Dec. 1988.

[Crowley 91] Crowley, J. L. "Towards Continuously Operating Integrated Vision Systems for Robotics Applications", SCIA-91, Seventh Scandinavian Conference on Image Analysis, Aalborg, August 91.

[Crowley-Reignier 93] J. L. Crowley et Patrick Reignier, "Asynchronous Control of Rotation and Translation for a Robot Vehicle", Robotics and Autonmous Systems, Vol 10, No. 1, January 1993.

[Crowley-Christensen 93] J. L. Crowley and H. I. Christensen, Vision as Process, Springer-Verlag Basic Research Series, To appear, 1993.

[Draper et al. 89] B. A. Draper, R. T. Collins, J.Brolio, A. R. Hansen, and E. M. Riseman, "The Schema System", International Journal of Computer Vision, Kluwer, 2(3), Jan 1989.

[Eklundh 92] Eklundh, J. O. and K.Pahlavan, Head, "Eye and Head-Eye System", SPIE Applications of AI X: Machine Vision and Robotics, Orlando, April 92.

[Granum-Christensen 1990] E. Granum & H.I. Christensen, Dynamic Robot Vision, In: Traditional and Non-traditional Robotics Sensors, T. Henderson (Ed.), NATO ASI Series in Computer Science, Springer Verlag, 1990.

[Hanson-Riseman 1978] Hanson, A.R. & Riseman, E.M., VISIONS: A Computer Vision System for Interpreting Scenes, in Computer Vision Systems, A.R. Hanson & E.M. Riseman, Academic Press, New York, N.Y., pp. 303-334, 1978.

[Huang 83] Huang T. H., Image Sequence Processing and Dynamic Scene Analysis, Springer Verlag, Berlin, 1983.

[Kalman 60] Kalman, R. E. "A New Approach to Linear Filtering and Prediction Problems", Transactions of the ASME, Series D. J. Basic Eng., Vol 82, 1960.

[Krotkov 90] Krotkow, E., Henriksen, K. and Kories, R., "Stereo Ranging from Verging Cameras", *IEEE Trans on PAMI*, Vol 12, No. 12, pp. 1200-1205, December 1990.

[Marr 82] Marr, D., Vision, W. H. Freeman, San Francisco, 1982.

[Tsotsos 87] Tsotsos, J. "Image Understanding", The Encyclopedia of Artificial Intelligence, S. C. Shapiro (Ed), Wiley, New York, 1987.

[Tsotsos 88] Tsotsos, J. "A Complexity Level Analysis of Computer Vision", International Journal of Computer Vision, 1(4), Jun 1988.

[Weems 89] C. Weems, "The Image Understanding Architecture", International Journal of Computer Vision, Kluwer, 2(3), Jan 1989.