Appeared in IROS '95, Pittesburgh Pa, August 1995.

Integration and Control of Active Visual Processes

James L. Crowley
LIFIA - IMAG
46 Ave Félix Viallet
38031 Grenoble, France
and
Henrik Christensen
Laboratory of Image Analysis, Aalborg University
Fr. Bajers Vej 7 D
DK-9220 Aalborg, Denmark

ABSTRACT

This paper presents a software skeleton system for experiments in integration and control of real time active vision systems. This skeleton has been constructed by a consortium of six laboratories as part of a long term basic research investigation of issues of integration and control of real time vision.

The first chapter describes the problems of integration and control and the context for their investigation The second chapter presents the SAVA III skeleton system which has been developed to support our experiments in integration and control. An overview of the skeleton system is presented, and then the communication mechanisms are described. The use of a rule based interpreter based on CLIPS 5.1 permits each module to send messages which can define rules and functions within the other modules, essentially allowing modules to program each other. The rule interpreter is used to construct the control component for the processing cycle for each module. The interpreter is also used for the control part of pre-attentive demons which detect perceptual events within the modules.

The SAVA system is illustrated with a description of a recent demonstration of a visual navigation system constructed in the SAVA skeleton. The components of the visual navigation system are presented and details are given concerning the use of stereo for obstacle detection, and the use of landmarks for updating the estimated position.

1 Introduction

Active vision is the use of controlled camera motion and control of processing to simplify and accelerate visual perception. This sub-discipline has grown steadily from ideas expressed in the mid 80’s by Bajcsy [Bajcsy 88], Aloimonos [Aloimonos et al. 88], Ballard [Ballard 91] and others, to what now appears to be a “paradigm shift”. Part of this shift has been the recognition that a vision system can not be designed in isolation of the task. Vision must serve a purpose, and in particular should be designed in the context of the perceptual needs of the task and the 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. 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. This paper reports on the development of this system. A more complete description of the results of the project may be found in the book [Crowley –Christensen 94].

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.
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.

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

[1]The Partners in project ESPRIT BR 3038 are Aalborg University (DK), University of Surrey (UK), KTH, The Royal Institute of Technology (S), University of Linkoping (S), LTIRF-INPG (F) and LIFIA-INPG (F). The project has been continued under BR 7108 with the addition of University of Genoa (I).

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.

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 experimented with 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 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.

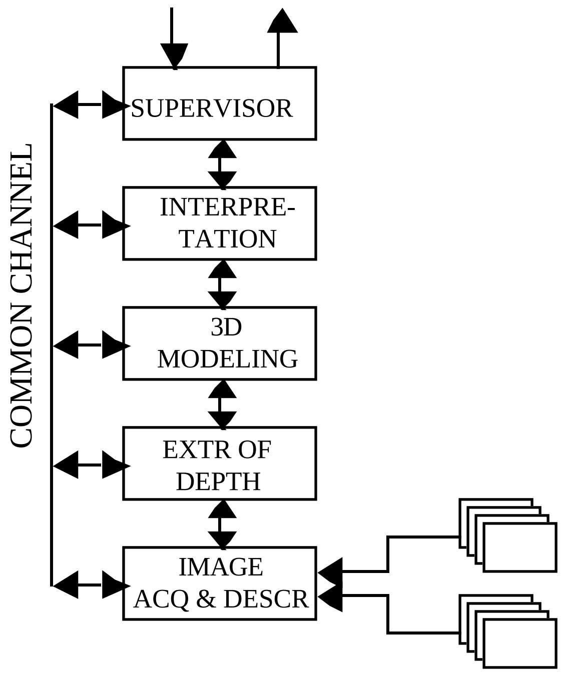

Figure 1. The VAP system architecture

The architecture adopted by the consortium is shown in figure 1. The system has a data flow part, which 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 different approaches to integration. 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. As the project advanced, it was realised that a more flexible processing model was needed to make computations both efficient and robust.

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 2.

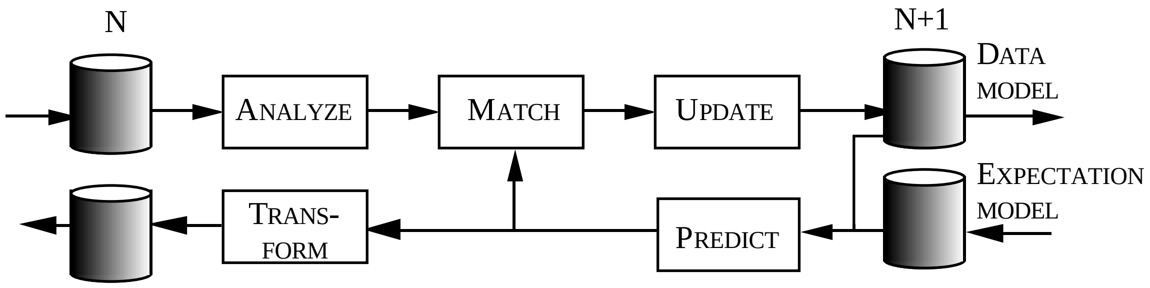

Figure 2. Basic Predict-Match-Update cycle for the module architecture.

The *analysis* block, in figure 2, 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.

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.

The following chapter describes the implementation of the VAP architecture.

2 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

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

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, 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 rule interpreter for the control component and the demon interpreter within each module. CLIPS 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 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.

2.1 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 3. The standard module is composed of a number of procedures (shown as rectangles) that are called in sequence by a scheduling process.

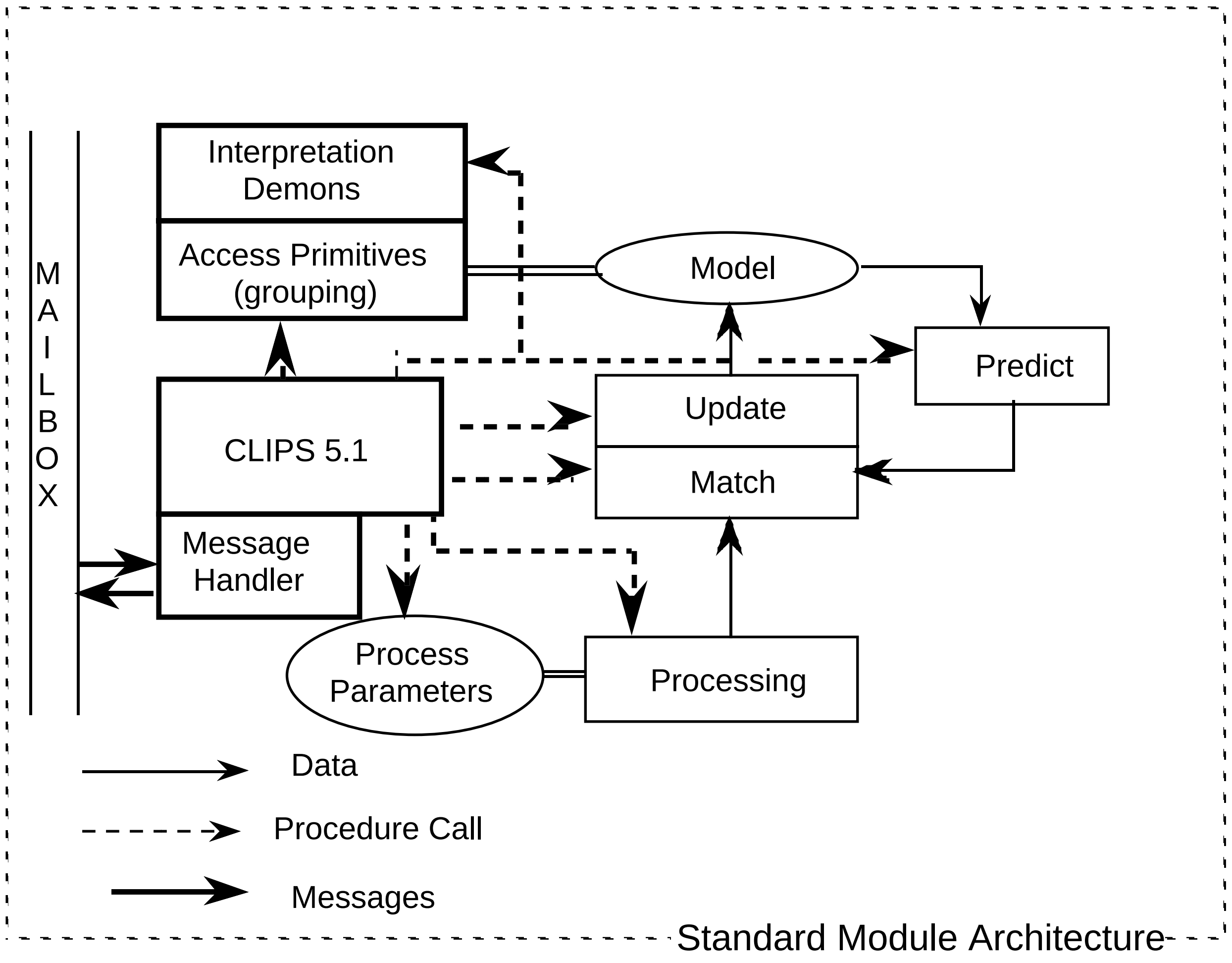

Figure 3 Architecture of a Standard Module in SAVA III

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.

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 current definition of theses phases used in most modules 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 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.

2.2 Communications Between Modules.

Modules communicate control, data requests, reply and synchronisation information using message passing based on Unix Sockets. The SavaSend() function converts a message into an ascii string and sends it to the mailbox of the destination 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.

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. Build replies TRUE if successful and FALSE if unsuccessful. In either case, the response is sent back to the sender by the function reply.

```
(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. If ?command corresponds to a previously defined function, then ?body is executed using the CLIPS “eval” command, as shown by the rule “interpret-function-messages”. If the message evaluates to a “NIL”, then reply does not send a message.

```
(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)
)
```

2.3 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.

The control part of a demon is encoded as rules while the processing part is generally in C. 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.

Having a rule interpreter provides makes it possible to have 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.

3 The Visual Navigation Demonstrator

This section illustrates the use of SAVA III by presenting an overview of the a visual navigation system constructed for the milestone 1 demonstration of VAP-II presented in June 1993. We first provide an overview of the demonstration and the configuration of the SAVA III system which was prepared for the demonstration. We then describe technical details of how vision is used to detect and avoid obstacles and to locate and use landmarks for position estimation. In the final section we describe how rule based control can be used to operate competing visual tasks and react to events.

3.1 Demonstration Overview

Perception serves at least two purposes for the navigation system of a mobile robot [Crowley 85]:

1) Correction of the estimated position as maintained by odometry.

2) Detection and location of obstacles.

Our demonstration uses an interactive graphics tool developed for the MITHRA project to construct a floor-plan "map" of the environment [Causse-Crowley 93]. This map contains walls, furniture, doorways, and visual landmarks (beacons). The tool allows us to specify a number of attributes for each object. In the case of a visual landmark, we can include a symbolic name for its shape, and a three dimension position for a reference point. The interactive specification software also permits us to overlay a network of places and routes on the environment. Using this network we can specify a navigation mission consisting of tasks of the form "go-to <place>" [Crowley 87].

The visuals tasks of obstacle detection and beacon location are contradictory. In order to detect obstacles, the head must look in front of the robot. In order to detect landmarks, the head must look at a wall. Further more, when an obstacle is detected, the system must react to its presence and use vision to steer the robot to a free path. We use a rule based control system to balance these competing activities. This rule base implements the finite state graph shown in figure 4.

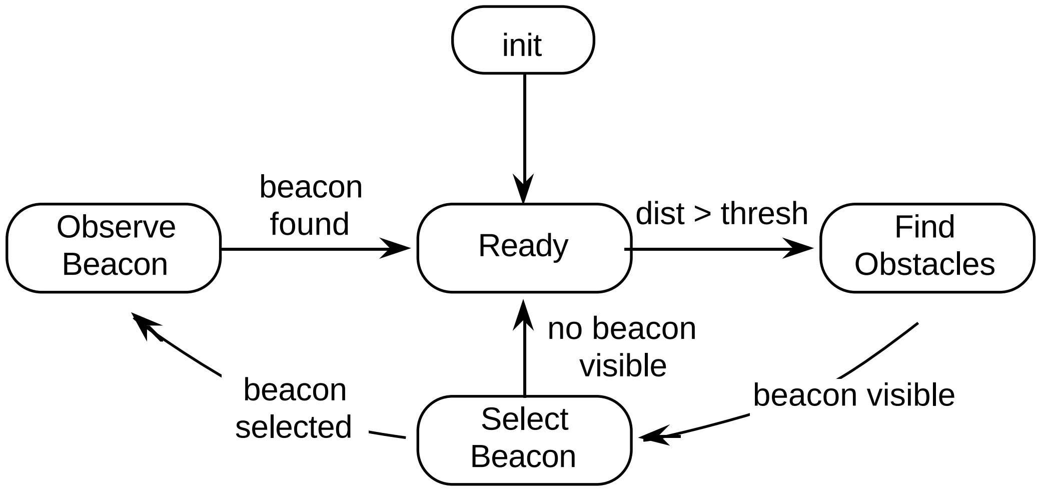

Figure 4 State Transitions Graph for perceptual control of navigation demo

A typical mission involves navigating from a corner of our laboratory to a passage between computer terminals, around a corner and out the door of the laboratory. Once out the door, the vehicle turns around and returns to its starting position.

At the beginning of the mission, the robot verifies that the floor-plan in front of him is free of obstacles. It then locates the nearest beacon and measures the angle. The angle to the beacon provides a one dimensional measurement for updating the estimated position and orientation using an extended Kalman filter. The robot then verifies again that its path is free. Both of these activities occur while the robot is moving.

The structure of the demonstration system is shown in figure 5. 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.

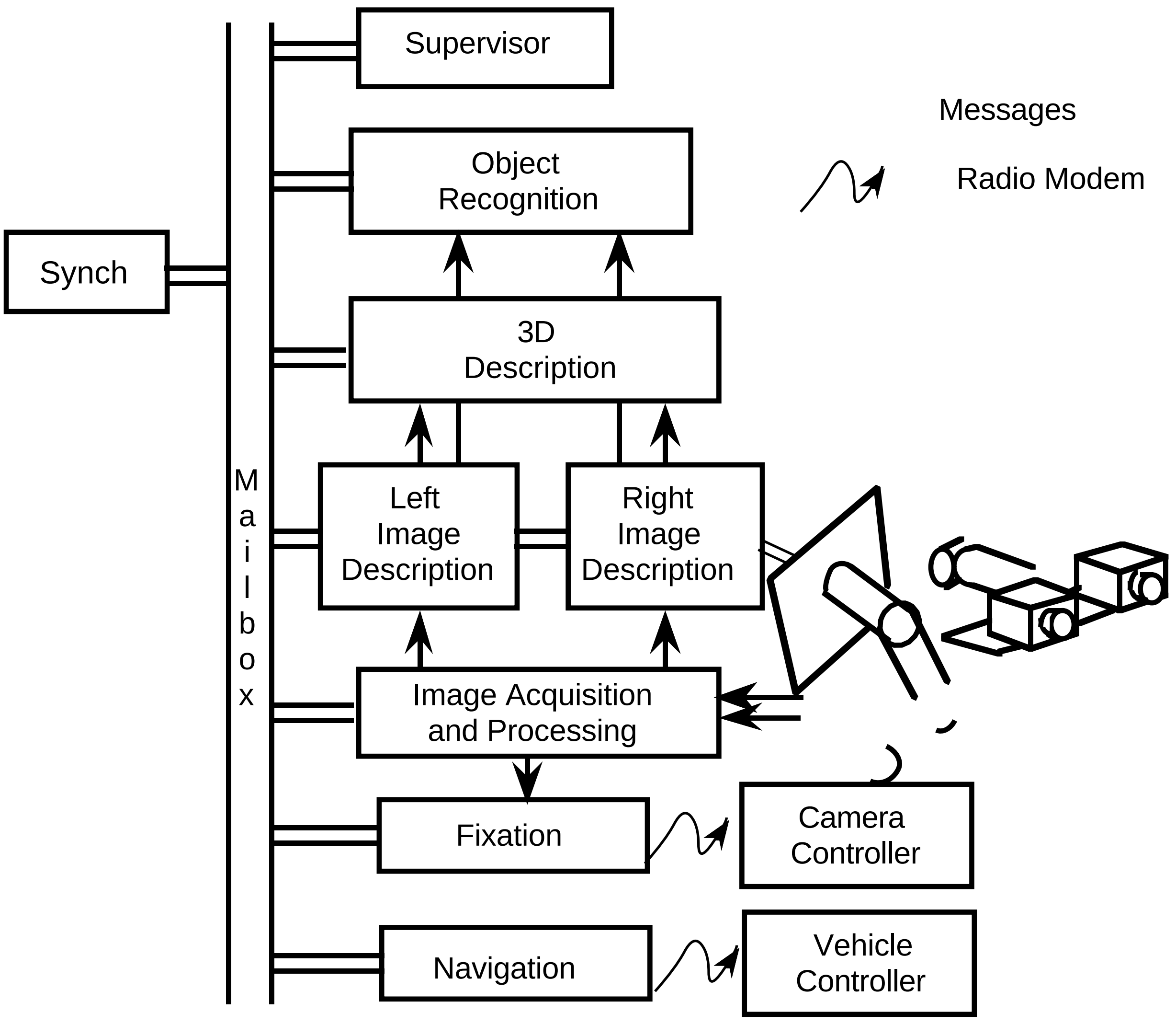

Figure 5 A Distributed Multi-process Vision System.

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.

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.

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. The four classes of model access procedures are:

Class 1 Procedures for extracting parametric primitives that are close to an ideal geometric entity such as a line or a point. Such primitives are easily implemented and have a complexity that is proportional to the number of primitives in the model.

Class 2 Procedures for extracting parametric primitives that are similar to an ideal "prototype". The prototype is specified an estimated value and a standard deviation for each of the primitive parameters. Similarity is measured by a Mahalanobis Distance. Parameters may be labelled as a "wildcard" by specifying a large (or infinite) standard deviation.

Class 3 Procedures for extracting pairs of primitives that satisfy some geometric relation, such as a junction, an alignment, or a parallelism. The geometric relation is defined as a set of parameters and specified as an estimate and a standard deviation.

Class 4 Procedures for extracting grouping of groupings. If performed by brute force, complex forms such as lines and contours composed of sets of line segments can require an exponential number of computations. By structuring the interrogation as a grouping, we can limit the theoretical complexity to $O(N^3)$ and maintain an average case cost that is nearly O(N).

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.

3.2 Visual Obstacle Detection

The process for obstacle detection is a good example of the implementation of a task-specific vision process within the SAVA system. Our experience with vision for obstacle detection indicates that none of the current techniques for image description are sufficiently robust to work in all cases. Accordingly we have designed vision processes for detecting specific pre-defined classes of obstacles. Our vehicle is equipped with 24 ultrasonic range sensors which function reasonably well for flat walls (with a textured surface) and large objects such as cabinets, but which are very unreliable for the detection of table legs and chair legs. Because table and chair legs are a particular problem, we have defined a vision process specifically for this case.

The 2D description module in SAVA III uses edge segments and grouping of edge segments to describe the contents of the image. The problem with edge segments is that they are often broken at unpredictable points along a contour. Thus we have defined an obstacle detection procedure which detects edges which are not on the floor plane based on the difference of orientation of edges in the right and left image, under a homographic transformation.

Given a pair of stereo cameras, it is very easy to write a 3 x 3 homogeneous coordinate matrix which represents a projective transformation from one image to the other for points within the ground plane. Let ${}^{G}P$, represent a point on the ground plane, and let ${}^{l}P$ and ${}^{r}P$ represent the image of this point in the left and right camera. The 3 x 3 transformation matrix ${}^{l}_{r}\mathbf{D}$ will transform the point ${}^{r}P$ into a point ${}^{l}P'$ in the left image. For the image of all points on the ground plane,

$${}^{l}P = {}^{l}_{r}\mathbf{D}\ {}^{r}P.$$

Points above the ground plane will be projected to the left of the corresponding point in the left image, as shown in figure 6. Points below the ground plane (reflections) will be projected to the right. We can use this mapping to detect vertical edge segments for which one of the end points is near the ground plane than the other. Such features include table legs, chairs and door frames, all of which are crucial features for our robot.

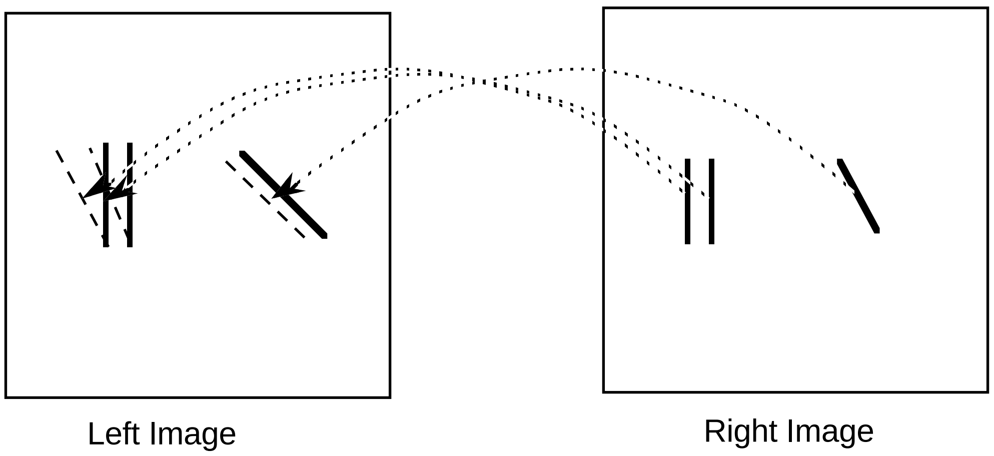

Figure 6 Obstacle detection by mapping edge segments using the homographic transformation defined by the floor plane. Horizontal segments in the floor plane map to segments in the left image with the same orientation. Vertical segments map to a different orientation.

Obstacle detection operates for a given head position and camera vergence angles. With our current 25 mm lenses, at a height of 1.5 meter, at an angle of 45 degrees, the field of view of the floor is a rhomboidal section whose size is very nearly that of a rectanlge composed of four sheets of A4 paper. (or a single sheet of A2 paper). This rhomboidal region constitutes the region of interest in which which visual features constitute potential obstacles. We have defined four arm-head configurations with overlapping fields of view such that the composition is a region of a distance from 1 to 2 meters in front of the robot and roughly 1 meter wide. For each position, a 3x3 homographic transformation is calibrated by imaging a piece of A2 paper decorated with dark squares. Calibration of this matrix is performed by a simple least squares technique [Crowley et al 93].

The visual features which are available are edge segments, expressed both as end-points and as well as in the midpoint, direction and length representation [Crowley et al 92]. Since very short segments have unreliable orientations, we select segments from the right image whose length is longer than a minimum value, typically 6 pixels. The end-points for such segments are projected to the left image to create a list of edge segments from the right image for which we will seek a match.

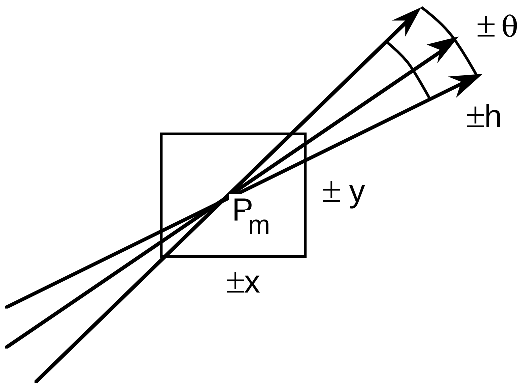

Figure 7 Similarity with respect to a prototype segment.

The SAVA 2D description module contains a vocabulary of procedures for associative access. One such procedure is called Get-Prototype-Segment, illustrated in figure 7. The arguments for this procedures are a vector of parameters (x_p, y_p, h_p, θ_p), and a set of standard deviations for matching

(σ_x, σ_y, σ_h, σ_θ). The procedure Get-Prototype-Segment computes the similarity in orientation, position and length between the prototype segment and each segment in the model. All segments which have a difference less than one standard deviation in orientation, position and length are placed in a list of possible matches. This list is sorted based on the combined Mahalanobis distance, defined by:

$$\text{Dist} = \left(\frac{x_p - x}{\sigma_x}\right)^2 + \left(\frac{y_p - y}{\sigma_y}\right)^2 + \left(\frac{h_p - h}{\sigma_h}\right)^2 + \left(\frac{\theta_p - \theta}{\sigma_\theta}\right)^2$$

In order to use “Get-Prototype-Segment” to detect matching segments, we must furnish estimated uncertainties for the difference in orientation, position and length. The position tolerance is a sum of factors due to the uncertainty of the vergence angle of the head, and the precision of detection of segments. A value of 10 pixels is typical. The length uncertainty is set to the half length of the mapped (prototype) segment. The orientation angle must be large because it is precisely the difference in orientation that signals an obstacle.

The worst case difference in orientation for mapped segments will occur for segments which are perpendicular to the ground plane. By observing such segments, we note that for our camera configuration, such segments have a difference in orientation of about 15 degrees. To add robustness to noise, we set the orientation standard deviation to 20°.

Any segment in the right image for which no candidate match is found in the left image is rejected. For each projected segment for which a match is found, the difference in orientation between the projected segment and the corresponding segment is computed. If this difference in orientation exceeds 10°, then the segment is considered as an obstacle.

When an obstacle segment is detected, its position in the left image is sufficient to compute the necessary turning angle. A bounding rectangle is computed for all obstacle segments. If the center of this rectangle is to the left of the image center, then a right turning angle is computed, otherwise a left turning angle. The turning angle is obtained from the pixel position of right (or left) edge of the rectangle. This pixel value is transformed to degrees using a calibrated ratio of pixel to degrees. The robot is then turned by this amount, and the obstacle test is repeated.

3.3 Visual Localisation

The position state of our standard vehicle controller is the vector $[x, y, \theta]^T$ which gives the vehicles current position and orientation in a global coordinate space. Our standard vehicle controller maintains an estimated of the position vector and its covariance matrix using a Kalman filter fed by odometry. The covariance matrix for this position estimation grows as the vehicle moves. The principle cause of position uncertainty is uncertainty in the orientation, which acts as a lever arm. Uncertainty orientation arises both from from drift while moving and from errors in the estimated center of rotation when turning. The vehicle must use observations of the external environment to keep the position state vector aligned the the real world.

The vehicle controller provides a command for correcting estimated position using an extended Kalman filter. In previous work we have developed and demonstrated a position correction technique based on matching walls observed with a set of 24 ultrasonic range sensors [Crowley 89]. In this demonstration we adapt the vehicle controller to update the position state from the observation of known land-marks in the visual environment. This technique is an adaptation of the technique developed in the thesis of Chenavier [Chenavier-Crowley 92].

A world model is provided to the navigation controller. This model contains a description of walls, obstacles and landmarks. Landmarks have a “type” which symbolically names their shape, a 3D position, and a visibility cone. The navigation controller maintains a list of landmark in the current room, and periodically updates the Cartesian distance to each landmark. Each time the localisation state becomes active, the navigation controller selects a visible landmark. The landmark location is transformed to robot centered coordinates using the estimated position of the vehicle. Fixation is directed to the 3D position of this landmark, using a command to the fixation controller. Perceptual grouping procedures in the 2D description module are used to detect the position of the landmark in the image. Visual servoing is used to bring the landmark reference point to the center of the image. The observed angle to the landmark is then determined from the head controller. The difference between the observed and predicted angle provides a correction vector for an extended Kalman filter.

Landmarks are located using the perceptual grouping procedures in the 2D description module. A demon has been placed in the 2D description module for each class of landmark. Initial experiments have been performed using dark triangles on paper posted on the wall. We permitted the use of artificial land-marks to concentrate on debugging of the basic navigation control processes. We have recently begun using computer screens, which provide easily detected rectangles. There are about 15 within the robot's experimental environment. Triangles and computer screens are detected using the procedure for groupings of groupings. We plan to write landmark detection demons for vertical objects such as pillars, window frames and doors in the near future. Such vertical objects will be detected as a pair of anti-parallel segments with a certain separation.

Updating Position from Beacon Observation

The angle to a known landmark provides a 1-D constraint on position that can be used with an extended Kalman filter to update the vehicle's estimated position and covariance. The key to such an estimate is the model of the observation process, ${}^{Y}_{X}\mathbf{H}$. The vehicle's state vector is the estimation position and orientation.

$$\hat{X}(t) \equiv [\hat{x}_r(t),\ \hat{y}_r(t), \hat{\theta}_r(t)]^T$$

This state vector is accompanied by a covariance matrix, $\hat{\mathbf{C}}_x(t)$, estimated from an odometric error model. The observation is the relative angle to the landmark. This observation angle is predicted from the position of vehicle and the position of the landmark $\hat{X}_b = (x_b, y_b)$. The predicted vector to the landmark is the difference in position between the robot and the landmark, $(\Delta x(t), \Delta y(t))$

$$\hat{\psi}(t) = {}^{Y}_{X}\mathbf{H} = F(\hat{x}_r(t),\ \hat{y}_r(t), \hat{\theta}_r(t), x_b, y_b) = \mathrm{Tan}^{-1}\left(\frac{\Delta y(t)}{\Delta x(t)}\right)$$

As this observation function is non-linear, we must estimate it with a first derivative taken near the expected value. Thus we replace the observation model with a linear approximation given by the Jacobian of the function $F(\hat{X}(t), \hat{X}_b)$.

$${}^{Y}_{X}\mathbf{H} \approx {}^{Y}_{X}\mathbf{J} = \frac{\partial F(\hat{X}(t))}{\partial \hat{X}(t)}$$

The Jacobian is a row vector which contains the derivative of $F(\hat{X}(t))$ with respect to each term in the state vector. Having written the Jacobian approximation to ${}^{Y}_{X}\mathbf{H}$, the rest is classical extended Kalman filter. The Jacobian permits us to estimate the uncertainty of the prediction as a variance based on the vehicle's estimated position.

$$\sigma^2_{\psi} = {}^{Y}_{X}\mathbf{J}\ \hat{\mathbf{C}}_x(t)\ {}^{Y}_{X}\mathbf{J}^T$$

The observation of a landmark gives an observation angle $\psi(t)$. The difference between the predicted angle and the observed angle gives a correction term $\Delta\hat{\psi}$. The Kalman gain matrix tell how much this correction term contributes to updating each component of the vehicle's state. The vector is computed as a relative waiting of the uncertainty of the position prediction and the observation. If we assume that the observation is perfectly precise, the equation simplifies to :

$$\mathbf{K}(t) := \mathbf{C}^*_X(t) \; {}^X_Y\mathbf{J} \; [\sigma^2_\psi]^{-1}$$

Where ${}^X_Y\mathbf{J}$ is the derivative of each term in the state vector as a function of the observed direction, $\psi(t)$. A transpose of ${}^Y_X\mathbf{J}$ is commonly used for this term. $\mathbf{K}(t)$ is a three one vector which tells the contribution to each term of the state vector $\hat{X}(t) \equiv [\hat{x}_r(t), \hat{y}_r(t), \hat{\theta}_r(t)]^T$ given by the observation $\Delta\psi(t)$. With this gain matrix we can update the state vector and its uncertainty by :

$$\hat{X}(t) := X^*(t) + \mathbf{K}(t)\, [\Delta\psi(t)]$$

and

$$\hat{\mathbf{C}}(t) := \mathbf{C}^*(t) - \mathbf{K}(t) \; {}^X_Y\mathbf{J} \; \mathbf{C}^*(t)$$

Experiments with an earlier implementation of this process appear in [Chenavier-Crowley 92].

4 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 have then used to the embedded components of the skeleton system to construct visual "competences" which may serve the navigation task. These competences are used as needed by the navigation system. 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.

The construction of these demonstration systems has led us to construct a distributed system composed of a number of cyclic processes. By placing an interpreter at the heart of each of these modules we have obtained a tool which permits experiments in control of perception with a minimum of effort. The fact that the control logic is declarative has greatly enhanced the clarity and simplicity of our experiments. SAVA III has become a tool which has permitted us to demonstrate that a continuously operating active vision system requires control of perception.

Bibliography

[Aloimonos et. al. 1988] 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 1984] 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] Brown C. , “Prediction and Cooperation in Gaze Control”, Biological Cybernetics 63, 1990.

[Causse-Crowley 93] O. Causse and J. L. Crowley, “A Man Machine Interface for a Mobile Robot”, IROS ‘93, Tokyo, July 1993.

[Chehikian 91] Chehikian A., “A One Pass Edge Exctraction Agorithm” 7th Scandanavian Conference on Image Analysis, Aalborg, 1991.

[Crowley 85] Crowley, J. L., “Navigation for an Intelligent Mobile Robot”, IEEE Journal on Robotics and Automation, 1 (1), March 1985.

[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] Crowley, J. L., P. Stelmaszyk and C. Discours, “Measuring Image Flow by Tracking Edge-Lines”, Second ICCV, Tarpan Springs, Fla. 1988.

[Crowley 89] Crowley, J. L., “World Modeling and Position Estimation for a Mobile Robot Using Ultrasonic Ranging”, 1989 IEEE Conference on Robotics and Automation, Scottsdale AZ, May 1989.

[Crowley et. al. 90] Crowley, J. L., P. Bobet and K. Sarachik, “Dynamic World Modeling using Vertical Line Stereo”, First European Conference on Computer Vision, (ECCV-1) Antibes, France, 1990.

[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 et. al. 92] J. L. Crowley, P. Stelmaszyk, T. Skordas and P. Puget, “Measurement and Integration of 3-D Structures By Tracking Edge Lines”, International Journal of Computer Vision, Vol 8, No. 2, July 1992.

[Crowley-Chenavier] F. Chenavier and J. L. Crowley, “Position Estimation for a Mobile Robot Using Vision and Odometry”, 1992 IEEE Conference on Robotics and Automation, Nice, May , 1992.

[Crowley et al. 93] J. L. Crowley, P. Bobet et C. Schmid , “Auto-Calibration of Cameras by Direct Observation of Objects”, Image and Vision Computing, Vol 11, no. 2, March 1993.

[Crowley-Christensen 93] Vision as Process, Results from Project ESPRIT BR 3038, Springer Verlag Basic Research Series, to appear 1993.

[Crowley-Demazeau 93] J. L. Crowley et Y. Demazeau, “Principles and Techniques for Sensor Data Fusion“, Signal Processing, March 1993.

[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.

[Draper et al. 89] Draper B. A., 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, Fla. 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.

[Hanson-Riseman 1987] A.R. Hanson & E. Riseman, “A Knowledge Based Approach to Vision. In: Vision, Brain and Co-operative Computation”, (Eds.) M.A. Arbib & A.R. Hanson, MIT Press, Cambridge, Mass, 1987.

[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.

[Kalman 61] Kalman, R. E. and R. S. Bucy, “New Results in LInear Filtering and Prediction Theory”, Transaction of the ASME, Series D. J. Basic Eng., Vol 83, 1961.

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

[Rimey-Brown 1991] R.D. Rimey & C. Brown, “Controlling Eye Movements with Hidden Markov Models”, Intl Journal on Computer Vision, April 1991.

[Tsotsos 87] J.K. Tsotsos, “Representational Axes and Temporal Co-operative Processes, In: Vision”, Brain and Co-operative Computation, (Eds.) M.A. Arbib & A.R. Hanson, MIT Press, Cambridge, Mass, pp. 361-418, 1987.