# A Versatile Parallel Architecture for Vision Based Control Applications

* P. Rives, * J.J. Borrelly,

(*) INRIA - Centre de Sophia-Antipolis
2004 route des Lucioles,
BP 109, 06561 Valbonne Cédex

** J. Gallice, ** P. Martinet

(**) Laboratoire d'Electronique,
Université Blaise Pascal de Clermont Ferrand,
U.R.A 830 du CNRS,
Les Cézeaux, 63177 Aubière Cédex

## Abstract

*In this paper, we discuss an hardware and software architecture for implementing vision based control approaches applied to robotics. In such approaches, the sampling rate of the robot's closed loop control is directly given by the image processing time. That implies strong time constraints to the vision system. In a previous paper [8], we have presented the design phase of a new machine vision based on a parallel architecture concept and first simulation runs concerning its expected performances have been also shown. The complete vision machine is now available and we detail in this paper both its hardware and software aspects. We also present the first results obtained for two real vision based control applications.*

## 1 Introduction

During the ten last years, many improvements was done in the field of sensor based control applied to robotics. A large framework is already existing [11] and allows us to design closed loop control schemes based on exteroceptive sensors like proximity, range finder or force sensors. Applying such an approach to more sophisticated sensors like vision is more recent [9], [3]. This is mainly due to the large time consuming of the image processing algorithms which are, until recently, incompatible with the sampling rate of the closed loop control schemes. These time constraints involve the use of specific hardware and software architectures having powerfull real time capabilities and modularity. Presently, only a few available industrial product may satisfy these requirements with relatively low performancies with regard to their high cost. Concerning research products, a new generation of image processing systems appears using Transputer nets [2], DSP processors, or dedicated processor like Sympati [12]. Unfortunately, these systems are often designed for general trends in image processing and do not take into account specificities of the vision based control approach. Due to this fact, we developped a new architecture (hardware and software) specially dedicated to this kind of applications. In a previous paper [8], we have presented the design phase of this new machine vision based on a parallel architecture concept and first simulation runs concerning its expected performances have been also shown. The complete vision machine is now available and we detail in this paper both hardware and software aspects with a special emphasis on the use of VxWork real time operating system at the system level. The paper is organized as follows. In a first part, after having briefly recalled the general problem of vision based control and emphasized the consequencies on the architecture of the vision system, we present the details of the hardware composed by three basic modules (corresponding to three different VME boards). In the second part, we discuss programming aspects of such a system. In the last part, we present the performances obtained on two real time vision based control experiments : a deep underwater bore-hole reentry task and high speed vehicle guidance task.

## 2 Hardware aspects:

Implementing efficiently a vision based control approach needs to solve several problems at the image processing level. The major difficulties are induced by real time constraints and the processing of large dataflow from the sequence of images. These problems can be summarized as follows :

- **Timing aspect :** In a vision based control scheme, the sampling rate of the closed loop control is directly given by the image processing time. Thus, to ensure the performances of the control, this sampling rate has to be sufficiently high. For the most current applications in visual servoing, a rate of 10 images per second can be considered as the lower limit.
- **Image processing aspect :** In a *scene analysis approach*, the main goal is to get high level description of a scene by means of sophisticated time consuming algorithms. By opposite, in a vision based control approach, we only want to extract from the image the necessary and sufficient informations able to constitute a feeback vector for the closed loop control scheme. Consequently, we only request low or intermediate levels processings providing contour based or region based



0-8186-5420-1/93 $03.00 © 1993 IEEE

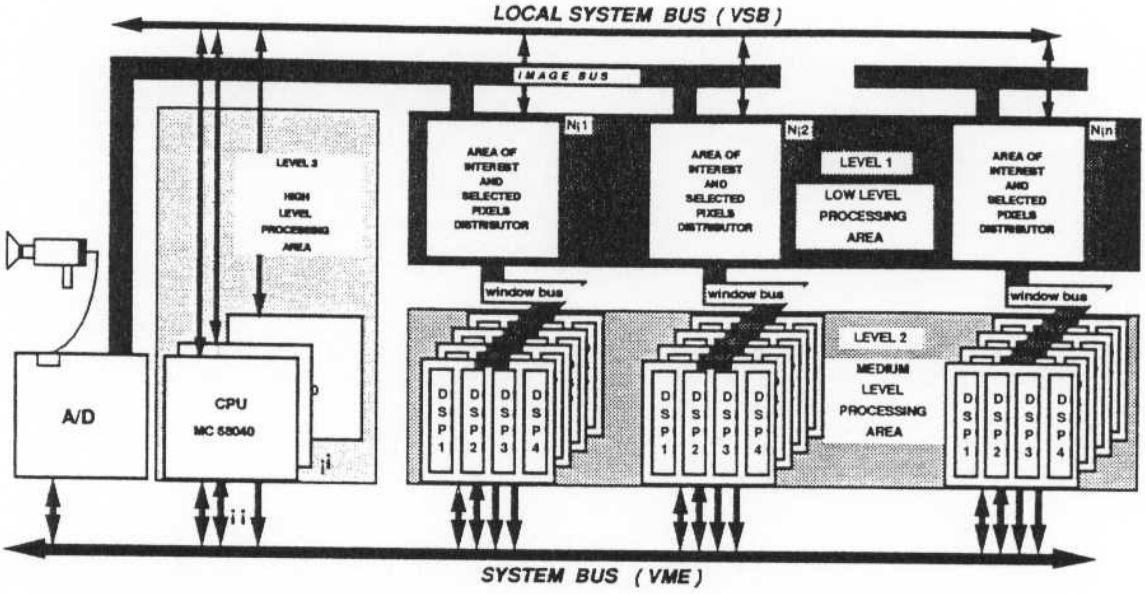


Figure 1: Overview of the general system

features like points, lines, area, center of gravity and so on. Moreover, in the most cases, these features are located in some regions of interest and then, the processing of the whole image is not necessary.

- **Tracking aspect :** The features extracted from the image have to be tracked along the sequence. We have also to take into account appearing and disappearing features due to the motions of the camera and objects in the scene.

From these requirements, we have developped an original architecture characterized by its modularity and its real time capabilities (figure 1). This architecture implements the concept of active window. An active window is attached to a particular region of interest in the image and has in charge to extract a desired feature in this region of interest. At each active window is associated a temporal filter able to perform the tracking of the feature along the sequence. Several active windows can be defined, at the same time, in the image and different processings can be done in each window. The functionnalities of such an architecture are presented at the figure 2.

As above mentionned, we deal with image processing algorithms which only request low or intermediate levels processings. This aspect is clearly taken into account at the hardware level by using a mixed architecture :

- **at the low level:** We have to deal with repetitive processings through the window like convolution or filtering. *Pipeline Architecture* has been chosen to perform efficiently such processings. In practice, that is done by means of VLSI convoluors which admit until 5X5 mask size. This step can be performed in real time with a delay directly function of the size of the mask. The input of the algorithm is the active window and the values of the mask parameters corresponding to the desired processing. The output will be constituted by a list of pixels which potentialy belong to the geometric features present in the window.

- **at the intermediate level:** We find merging-like algorithms wich take in input the list of pixels of interest provided by the low level stage and merge them to compute a more structured representation of the geometric features present in the window (for exemple, slope and distance to the origin for a line or coordinates of center and radius for a circle). In our hardware, this part is done by using a **MIMD Architecture** and it is practically implemented by means of multi DSP's boards.

The details of the hardware has been already described in a previous paper [8], and we just recall the principles. Three basic modules have been defined corresponding, in practice to three different VME boards). Dependant of the complexity of the application, the system will be built around one or more of these basic modules in such a way that we will be able to attempt video rate performances.

The three basic modules are the following :

- **WINDIS Window Dispatcher Subsystem (figure 3):** This module has in charge the win-

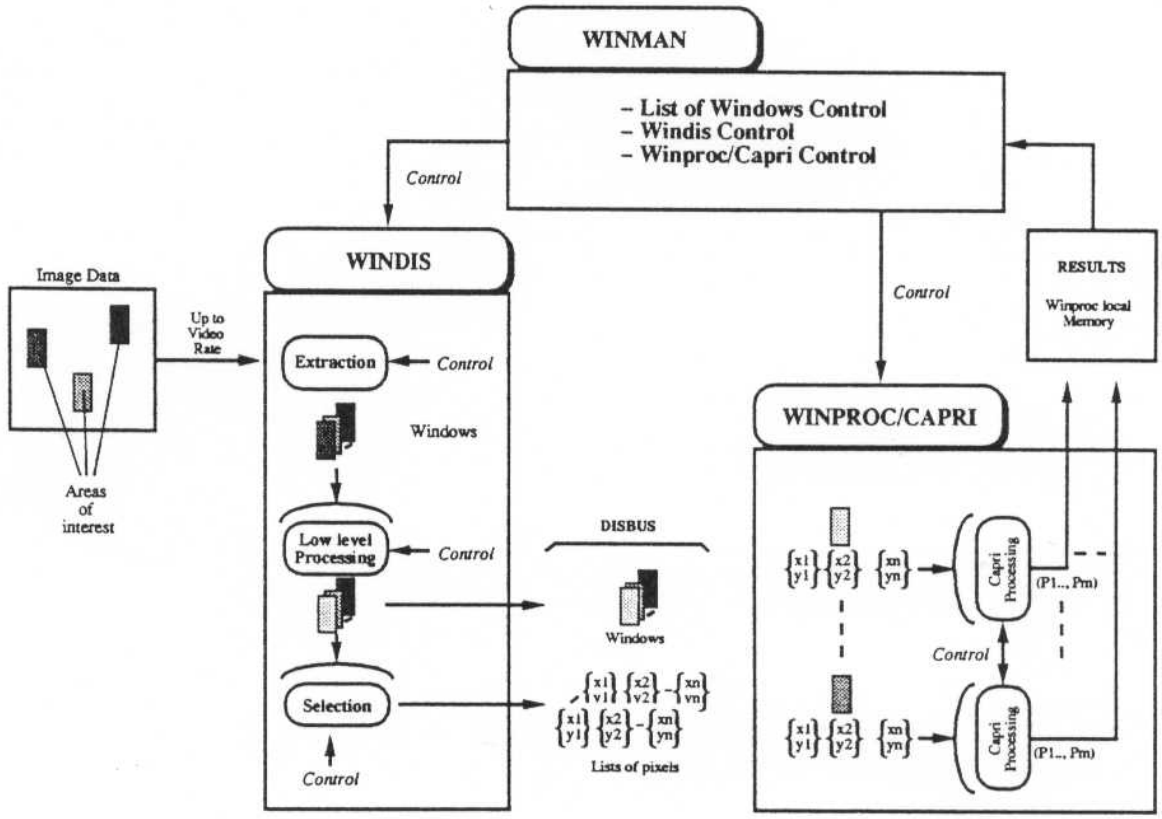


Figure 2: Data flow through Windis/Winproc subsystem

dow extraction, the execution of low level processings and the distribution of active windows toward the Window Processing Subsystem. After low level processing and thresholding, the list of selected pixels and grey levels corresponding to the different active windows are dispatched to the intermediate level processing through a 20MHz pixel bus.

- **WINPROC Window Processing Subsystem (figure 4):** We have associated one to sixteen DSP 96002's modules (CAPRI modules) with one distributor module. DSP modules are plugged on VME mother boards and execute intermediate level processings on the dataflow from the window bus. Window processing modules provide a geometric description of the searched primitive in each window. At each VME mother board can be associated four DSP modules.

- **WINMAN Window Manager Subsystem :** The window manager controls distributor and DSP modules, and executes high level processing of applications tasks. Moreover, it has in charge the tracking of the active windows along the sequence. A 68040 based cpu board implements this module.

For each level, we have introduced parallelism allowing us to reach video rate for most of applications tasks. All the modules satisfy to VME requirements and accept in input MAXBUS video bus from Datacube. The management of such a system is a tricky job and requests the use of a real time operating system. For facility reasons, we have chosen to develop all the system level under VxWorks O. S. We present in the next section the different layers of the software from the application level until the implementation aspects using the real time tasks scheduled by VxWorks.

## 3 Programming aspects:

### 3.1 System Overview

Figure 5 shows the general software organization as well as its relationships with the underlying hadrware.

The hardware is comprised of an image frame grabber (*DIG*), the WINDIS and WINPROC subsystems. Each of the hardware components have the capability to generate interrupts to the HOST (WINMAN) in the following conditions:

- *DIG :* Every 40 ms synchronized with video.

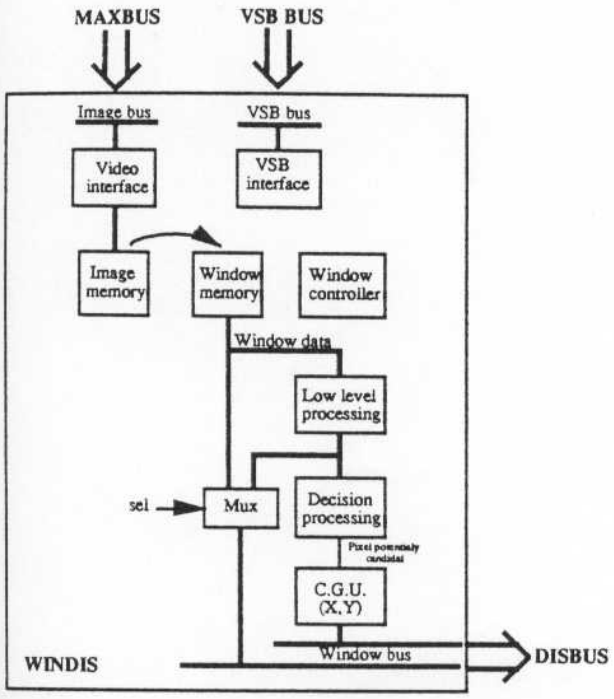


Figure 3: WINDIS subsystem

- *WINDIS :* either after completion of the input image swapping (video) or at the end of a pending distribution on DISBUS.
- *WINPROC :* The interrupts are generated by any of the supported CAPRI module, either at the end of processing of a window or at the end of processing a list of windows or both.

In this context a real-time application can be defined as a set of real time communicating tasks managed by a real-time Kernel (VxWorks). Part of them are 'system' tasks, while the others are 'user' tasks.

### 3.1.1 System software

To each of the hardware device correspond a driver (*digDrv, wdDrv, wpDrv/cpDrv*) which catches the device interrupt and manages the device registers. In the case of the WINPROC device, the driver has been separated into two parts:

- *wpDrv* which manages the shared memory device and board registers.
- *cpDrv* which manages the CAPRI modules (load DSP program,start/stop DSP activity, send interrupt to DSP ..)

On top of these drivers and especially for the window processing applications we find the window management layer :

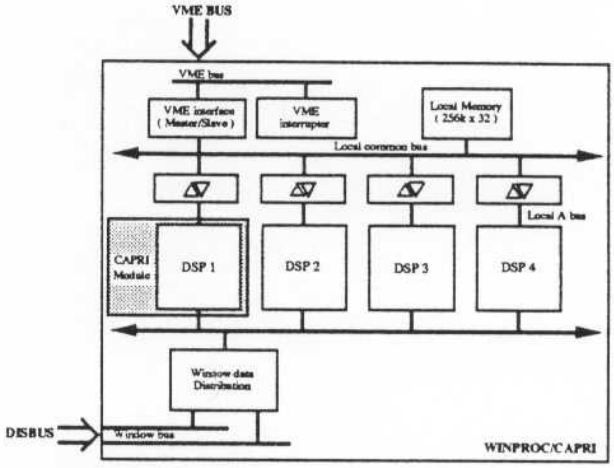


Figure 4: WINPROC/CAPRI subsystem

- *winDrv* is the HOST part of the window driver which manages individual windows by programming the hardware through the appropriate drivers.
- *Windows* is the DSP part of the window driver and is in charge to run the user processing function for the requested window.
- *winMgr* deals with the temporal aspects of the window processing.

### 3.1.2 Application software

The application software is separated into five parts:

- *Processing* is the set of user defined processing functions to be run on the DSP modules.
- *hardConfig* defines the involved hardware (initialize drivers)
- *winDefine* setup the processing characteristics of all the windows of the application.
- *timeSetup* gives the temporal aspects of the window processing.
- *Results* is the set of the overall result processing functions to be applied on the results of the intermediate level processing.

## 3.2 Building an application

To build an application the user must first define the DSP processing functions and assign them an unique identifier as well as the data structures for the function parameters and results. These functions will be called by the *Windows* driver and must respect the following prototype:

```
int     process_func(x,y,a,b,p,r)
int     x,y,a,b; /* window geometry */
param*  p;       /* pointer to parameters */
result* r;       /*pointer to results */
```

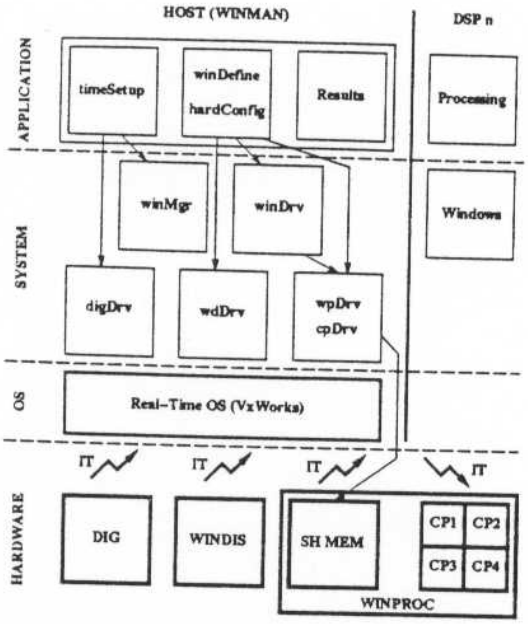


Figure 5: General Software Organization

The DSP programs can then be compiled and linked with the *Windows* driver. The processing functions need not exist on all the CAPRI modules provided they have a unique identifier.

The parameters and results of each DSP function will be located in the WINPROC shared memory, and will be used by the DSP and the application (*Results*) through pointers.
The *winDrv* driver provides functions to allocate double banks of parameters or results for a window and the associated DSP.
Then,one must fill the window descriptor of each window, including :

- *Geometry* : The window geometry is defined by the window position and size in the source image, windows are rectangular areas.
- *Low level processing* : The low level processing concerns the WINDIS attribute such as the filter size and coefficients and the selection of the pixels of interest.
- *Intermediate level processing* : The intermediate level processing takes its input data from the low level and is done by up to 4 CAPRI modules. Each of the DSP module will run the appropriate user defined function with standard parameters and results.
- *Global processing* : This level is in charge to get the DSP results for each window, update the window geometry and compute the control vector to send to the robot.

An application supports three types of processing :

- *multiple-dsp* is used when the *same* window is sent to several CAPRI modules after the low level processing. A maximum of four modules is allowed provided they lie on the same WINPROC device.
- *multiple-window* is used when a list of windows is send to a CAPRI module before starting the processing. The multiple-window mode can be used together with the multiple-dsp mode provided that all the windows of the list are sent to the same set of CAPRI modules.
- *multiple-rate* is used when some windows are activated less frequently than other. The window rate is derived from the 40 ms video rate. All windows running at a selected rate must be sent to the same set of CAPRI modules.

The user also has to define the window distribution policy by building the lists of windows and configure the *digDrv* to generate appropriate events from the video rate. One or more list of windows will be attached to each generated event.

### 3.3 Running an Application

Figure 6 shows the data and control flow during execution.

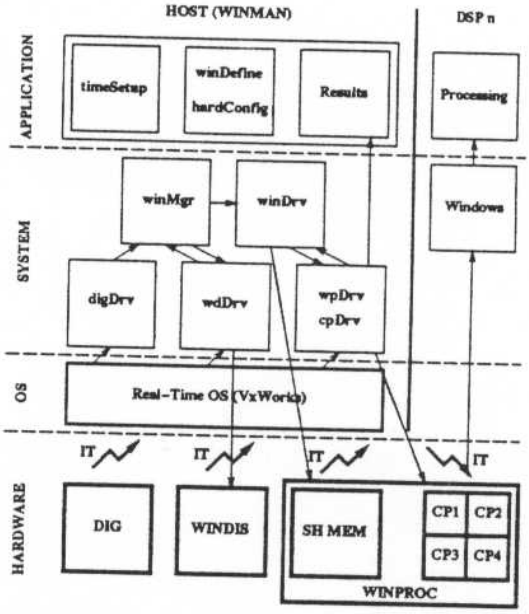


Figure 6: Data Control flow

An application is comprised of two processes :

#### 3.3.1 Distribution

The distribution process is a single real-time task (part of *winMgr*) which reads the set of lists of windows from a message queue and calls the *wdDrv*, *cpDrv* functions. This message queue is filled by the *digDrv* depending

of the distribution policy previously setup. At the beginning of a distribution, the WINDIS input image is swapped in order to get the proper image. The *winDrv* uses double banks of lists of windows to be processed. At the end of the distribution of a list of windows, each involved CAPRI module will receive an interrupt to start processing, provided the previous list was finished. The CAPRI modules signal the end of processing of a list by generating an interrupt, allowing the distribution process to continue for that module.
Figure 7 shows the distribution of four lists of windows for a total of nine windows and three CAPRI modules. The lists $l1$ to $l3$ are sent every video sample while list $l4$ is sent every two samples. In this example, the distribution of $l3$ will require the processing of list $l1$ to be finished on $cp1$ and $cp2$.

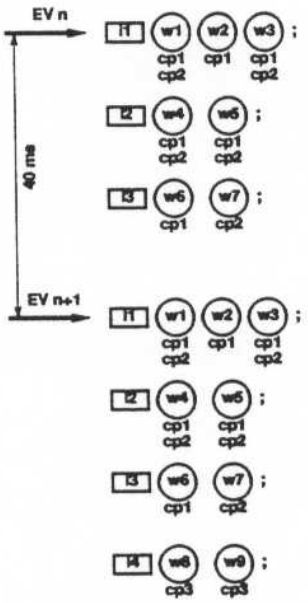


Figure 7: Distribution timings

### 3.3.2 Result processing

The result processing is done by a set of user tasks which read messages from user defined message queues. These message queues must be assigned to the windows during the configuration phase. The messages contain the window and CAPRI module identifiers, so that the user tasks can call the appropriate result processing function. The user tasks are also in charge of the swapping of the double bank of results. At the end of a window processing, the CAPRI modules will generate an interrupt to *wpDrv* with the window and module identifiers. The *wpDrv* then write this information to the window message queue, waking up the user task. It is not recommended to use the same message queue for windows running at different rates.

## 4 Performance analysis:

To validate the machine vision above presented, we have implemented two real time vision based control applications. The first one concerns an underwater robotics task and the second addresses the problem of automatic road following.

### 4.1 Underwater robotics task:

In the near future, underwater robotics will certainly be one of the most exciting application fields for autonomous vehicles. Nowadays, underwater operations are performed directly by skin-divers or teleoperated from an underwater manned vehicle. In all cases, they are very expensive. Among the different types of underwater missions which will be useful to automatize, docking missions play a major role. For these reasons, IFREMER (the French agency in charge of oceanographic research) has started a program aimed at designing an underwater autonomous vehicle, NADIA, which will be able to perform automatic docking missions. In this program, we are in charge of validating sensor based control approaches for this purpose. More precisely, The experiment presented here focuses on the problem of positioning a mechanical structure with respect to a deep underwater bore-hole by using a visual servoing approach. In opposite to classical robotics approaches, the visual servoing approach [13],[5], does not require an explicit 3D reconstruction of the scene. The basic idea is to assume that a task can be fully specified in terms of features in an image. The task is achieved when a desired image of the scene is reached. In terms of control, this can be formulated as a problem of regulation to zero of a certain output function directly defined in the image frame. When implementing a visual servoing application several problems need to be solved . In [3], [10], the readers will find more details on the theoretical framework around this kind of approach, we focus here on the critical problem of real time image processing. For this particular experiment, we have to compute the values of the parameters of two ellipses corresponding to the projection onto the image of the two circles bounding the bore-hole in the scene. From the image processing point of view, we have to perform the fitting and the tracking of the two ellipses at video rate along the sequence. To reach these real-time performances, one WINMAN board, one WINDIS board and one WINPROC with two DSP modules are necessary. Two active windows have been used, one for each ellipse. The pixels belonging to contours of the ellipses are extracted ,in the WINDIS board, by an edge operator using a 3X3 mask and flushed, via the DISBUS to the WINPROC board. The WINPROC board is configurated such that one DSP module estimates the current values of the parameters of the ellipses by using a moment representation while the second DSP module has only in charge to display results in the image framebuffer. Due to the relative motion between the camera and the bore-hole, the size and the location of the two ellipses have to be updated at video rate. The tracking of the two ellipses along the sequence of images, is done by a very simple temporal filter applied on the size and position of the two active

windows. This filter is implemented on the WINMAN board. Figure 8 presents the points of interests which have been selected by the low level processing module. Figure 9 shows the two active windows and the two ellipses which have been computed by the DSP modules from the points of interest. The Table 1 gives the computation times versus the number of points of interest selected in the active window by the low level and can be interpreted as follows :

- **Distribution:** The distribution time on the DISBUS rated at 20 Mhz, including low-level processing and WINDIS/WINPROC register setup.
- **Parameters:** The computation time required to determine the ellipse parameters on a single DSP.
- **Equation:** The computation time required to determine the contour of the ellipse for display.
- **Display:** The computation time required to actually display the ellipse and the window geometry through the VME bus towards the image frame buffer.

We have validated the whole experiment including the control loop on our testbed constituted by a 6 degrees of freedom arm with a camera mounted on its end effector. The performances reached at the image processing level allow to ensure a 25Hz sampling rate for the complete visual servoing loop.

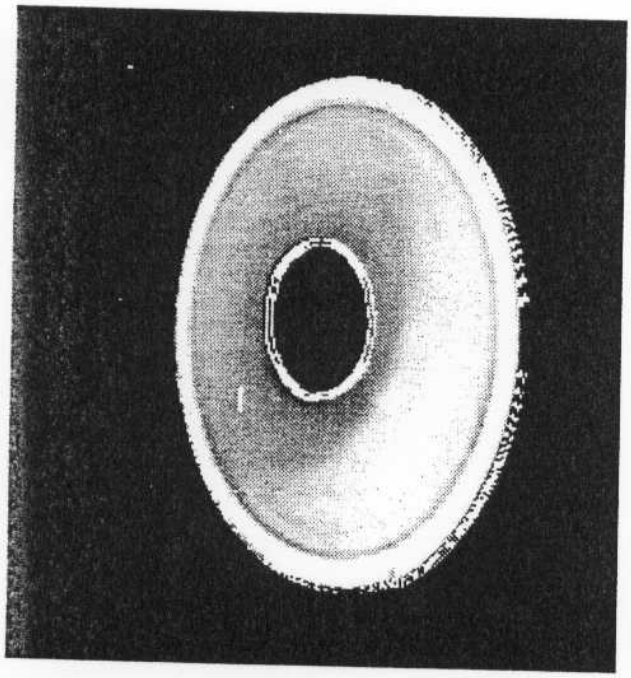

Figure 8: Low level processing of the ellipses

## 4.2 Automatic road following:

The second experiment deals with automatic vehicle guidance on highway. Le Laboratoire d'Electronique de Clermont-Ferrand has developped an algorithm using data provided by an on board camera which allows to estimate the current position of the vehicle (lateral offset from the straight line, side slipping angle ) with respect to a normalized road [1],[4]. This estimation constitutes the vector measurement of a state feedback controller driving the steering system. Assuming the road model is such that be shown in the figure ??, we want to extract the parameters $(\rho, \theta)_j, j = 1, 2, 3$ characterizing the three white lines of the highway. Moreover, we want to track the lines from one image to the next. In terms of image processing, we have to :

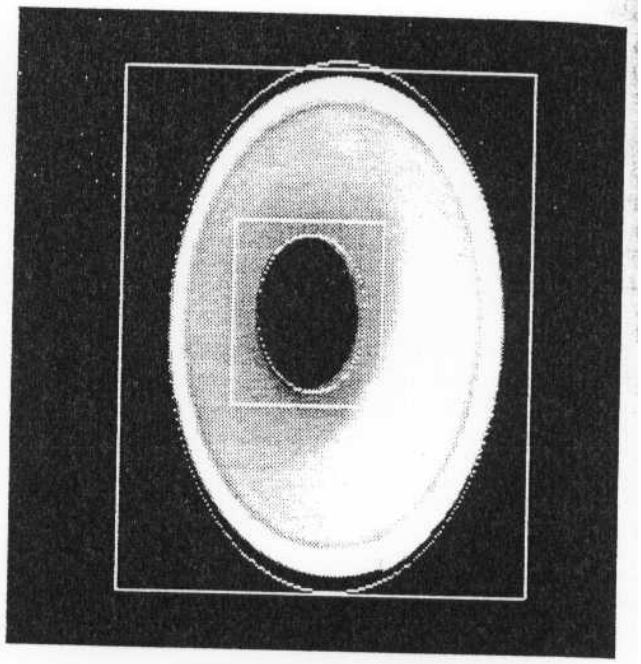

Figure 9: Intermediate level processing of the ellipses

- Compute the parameters $(\rho, \theta)_j$ for each white line.
- Compute from image $(k)$ to image $(k+1)$ a prediction of the parameters $(\rho, \theta)_j$ at time $(k+1)$.

For ensuring good performances at the control level, all these computations must be done with a sampling period the closest than possible to video rate. The computation of the parameters $(\rho, \theta)_j$ is performed by attaching a list of active windows at each white line $D_j$. The low level processing extracts in each window the points of interest corresponding to the left edge of the white line. That is done by a 3X3 Prewitt's convolution mask following by an adaptative tresholding. The selected points of interest are flushed to the DSP's modules in charge of the computation of the parameters $(\rho, \theta)_j$. We use for this computation a Hough transform working on a reduced research space centered around the predicted values of the parameters $(\rho, \theta)_j$ computed at time $(k-1)$. Results are shown

| Window | 1 | 2 |
|---|---|---|
| Size | 176x224 | 64x80 |
| Pix of interest | 1533 | 502 |
| Distribution | 2.40 ms | 0.70 ms |
| Parameters | 10.40 ms | 3.43 ms |
| Equation | 4.64 ms | 3.32 ms |
| Display | 28 ms | 11.58 ms |

Table 1:

on the figures 11 and 12. In the first one presents the points of interest detected at the low level by the Prewitt operator. The second one shows the segments of lines estimated at the intermediate level by the Hough transform. The temporal tracking of the lines along the sequence is done by a Kalman filtering. The prediction step is used for updating the locations and sizes of the active windows at time $(k+1)$.

We have validated both low level and intermediate level processing on a benchmark using one up to ten windows ($Roi_0 \ldots Roi_9$) whose the characteristics are given at Table 2.

| Region of interest | Size [X,Y] | P1 [min, max] | P2 [min, max] | N |
|---|---|---|---|---|
| Roi0 & Roi5 | [80,80] | [0.4,0.5] | [300,400] | 78 |
| Roi1 & Roi6 | [50,40] | [0.2,0.3] | [300,400] | 20 |
| Roi2 & Roi7 | [50,40] | [0.2,0.3] | [300,400] | 22 |
| Roi3 & Roi8 | [46,84] | [0.7,1] | [200,300] | 59 |
| Roi4 & Roi9 | [120,109] | [-0.9,-0.8] | [50,200] | 100 |

Table 2:

For each active windows, we define :

- *[size_X,size_Y]* the size of the window.
- *[P1min, P1max]* the research space for the parameter $\theta$ expressed in radians.
- *[P2min, P2max]* the research space for the parameter $\rho$ expressed in pixels.
- $N$ the number of points of interest selected by the low level.

The size of the accumulator used in the Hough transform has been fixed to 10X100 for all the experiments.

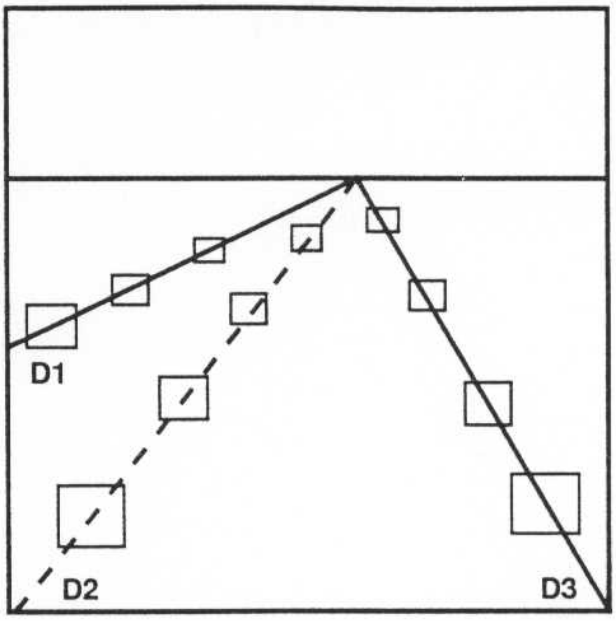


Figure 10: Model of the road

The Table 3 shows the execution time versus the number of windows for both the low and intermediate level. This result corresponds to the case where only one DSP is used. For pointing out the multiprocessing aspect, we have done a second experiment with two DSP. The first one is dedicated to the extraction of the left edges of the white lines while the second extracts right edges. The same benchmark that previously is used. The Table 4 gives the number of points of interest selected by the low level for each side of the line in a window. Table 5 summarizes the variation of the execution time versus the number of windows. At the present time, we can process until three windows at video rate. These results have been obtained with non optimized code and we expect, in a next future, to be able to reach 4 or 5 windows by DSP at video

| Regions of interest | Roi0 | Roi0 - Roi1 | Roi0 - Roi2 | Roi0 - Roi3 | Roi0 - Roi4 |
|---|---|---|---|---|---|
| Execution Time (ms) | 40 | 40 | 40 | 80 | 80 |

| Regions of interest | Roi0 - Roi5 | Roi0 - Roi6 | Roi0 - Roi7 | Roi0 - Roi8 | Roi0 - Roi9 |
|---|---|---|---|---|---|
| Execution Time (ms) | 120 | 120 | 120 | 160 | 160 |

Table 3:

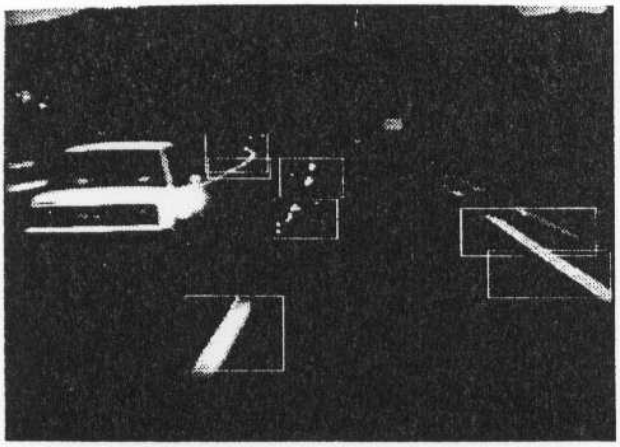

Figure 11: Points of interest detected at the low level

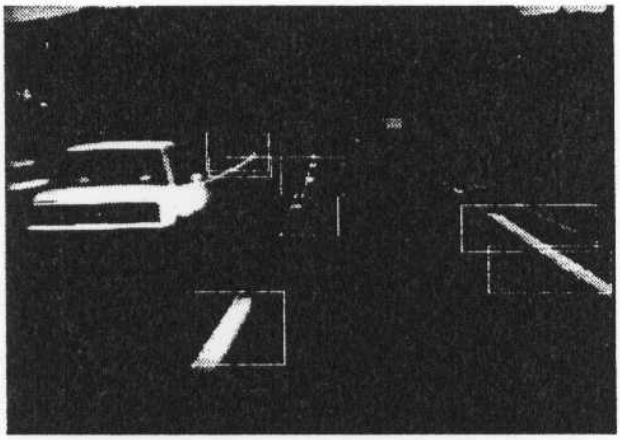

Figure 12: Segments of lines estimated by the Hough transform

rate. We can also note as an important result, that multiplying by two the number of DSP increases the computing power by a factor two. That means that, for a small number of DSP, the communication time is neglected with regard to the computation time.

| Regions of Interest | Roi0 | Roi1 | Roi2 | Roi3 | Roi4 |
|---|---|---|---|---|---|
| N (left edge) | 78 | 20 | 22 | 59 | 100 |
| N (right edge) | 78 | 21 | 14 | 37 | 102 |

Table 4:

| Regions of Interest | Roi0 | Roi0 - Roi1 | Roi0 - Roi2 | Roi0 - Roi3 | Roi0 - Roi4 |
|---|---|---|---|---|---|
| Execution Time (ms) | 40 | 40 | 40 | 80 | 80 |

Table 5:

## 5 Conclusion

We have presented, in this paper, a new machine vision system dedicated to vision based control applications in robotics. This kind of application is characterized by strong real time constraints and large dataflow in input from the sequence of images. Our approach consists to split the problem in two levels. The first one is devoted to the tracking of features along the sequence by means of active windows associated to the regions of interest. The second one is to attach to these windows some dedicated image processings (hard and soft) which perform the extraction of the desired feature into each window. To implement these two levels, we use a mixed architecture :

- **Pipeline Architecture** based on VLSI chips, is associated to the low level allowing to efficiently perform processings like convolution which are repetitive through the window.
- **MIMD Architecture** based on multi DSP's processors, has in charge the intermediate level.

We have validated this architecture on two vision based control experiments. The first one was based on an ellipse fitting algorithm while the second uses Hough transform technique. The results obtained are very promising and should be allowed us to implement complex vision based control applications at video rate.

## References


[1] R. Chapuis: *Suivi de primitives images, application à la conduite automatique sur route*, Ph-d Thesis, University of Clermont-Ferrand, France, Janvier 1991.

[2] J.P Derutin, B. Besserer, T. Tixier, A. Klickel: *A Parallel Vision Machine: TRANSVISION*, CAMP91, Computer and Machine Perception, Paris, December 1991.

[3] Espiau B., Chaumette F., Rives P. : *A New Approach to Visual Servoing in Robotics*, IEEE Transaction on Robotics and Automation, Vol. 8, No. 3, June 1992

[4] F. Jury, R. Chapuis, J. Gallice: *Mobile position estimation in a dynamic environment*, RFIA-AFCET Congress, Lyon, Novembre 1991.

[5] M. Kabuka, E. McVey, P. Shironoshita : *An Adaptive Approach to Video Tracking*, IEEE Journal of Robotics and Automation, 4(2):228-236, April 1988.

[6] H. Kubota, K. Fukui, M. Ishikawa, H. Mizogushi, Y. Kuno: *Advanced Vis ion Processor with an Overall Image Processing Unit and Multiple Local Processing Modules*, MVA'90 IAPR Work-shop on Machine Vision Applications, Tokyo, Japan, 1990.

[7] H. Kubota, Y. Okamoto, H. Mizogushi, Y. Kuno: *Vision Processor for Moving Object Analysis*, CAMP91, Computer and Machine Perception, Paris, De cember 1991.

[8] Martinet P.,Rives P.,Fickinger P.,Borrelly J.J : *Parallel Architecture for Visual Servoing Applications*, Workshop on Computer Architecture for Machine Perception, Paris, Dec. 1991

[9] P. Rives, F. Chaumette, B. Espiau: *Visual Servoing Based on a Task Function Approach*, First International Symposium on Experimental Robotics, Montréal, Canada, June 1989.

[10] P. Rives, H.Michel: *Visual Servoing Based on Ellipse Features*, SPIE: Intelligent Robots and Computer Vision XI: Algorithms and Techniques, Boston, MA, 7-10 Sept. 1993

[11] C. Samson, B. Espiau, M. Le Borgne: *Robot Control: the Task Function Approach*, Oxford University Press, 1991.

[12] *SYMPATI 2: SYstème Multiprocesseur Adapté au Traitement d'Images* Document Technique, SIR/CEN de Saclay, 1991

[13] L. E. Weiss, A. C. Sanderson: *Dynamic Sensor-Based Control of Robots with Visual Feedback*, IEEE Journal of Robotics and Automation, Vol. RA-3, n. 5, Oct. 1987.