Simulation and Scheduling of Real-Time Computer Vision Algorithms

F.Torres[1], F.A.Candelas[1], S.T.Puente[1], L.M.Jiménez[2], C.Fernández[2], and R.J.Agulló[2]

[1] Physics, Systems Engineering and Signal Theory Department. University of Alicante, Spain

[2] Science and Technology Dept. System Engineering and Automation Div. University Miguel Hernández, Spain

Abstract. A fully integrated development tool for computer vision systems has been built in the framework of this paper.
There are many applications that help the user in the design of such systems, using graphical interfaces and function libraries. Even in some cases, the final source code can be generated by these applications.
This paper goes a step beyond; it allows the development of computer vision systems from a distributed environment. Besides, and as a distinctive characteristic with regard to other similar utilities, the system is able to automatically optimize task scheduling and assignment, depending on the available hardware.

1 Introduction

The proposed tool establishes a mechanism for the development of general computer vision applications from the posing stage until codification, scheduling and the necessary hardware selection to its execution. This process is complex, and usually causes time charges in the scheduling problem resolution and in the debugging of a specific codification for each equipment.

The exposed system integrates all the tasks and automates great part of the work that must be carried out by the programmer, who in this way can think in the problem analysis and the solution exposition. The debugging and scheduling tasks are analyzed depending on the accumulated experience in the applications design for specific hardware environments, being implemented in well-defined and robust modules.

The application structure establishes two main modules that are the visual workspace and the processing task scheduler. A database works as a link between both modules collecting the necessary information about the nature of the tasks designed by the programmer.

Standard systems as *Khoros* (Khoral Research) have been considered prior to this development. *Khoros* with his visual environment *Cantata* is highly modular and extensible with distributed processing capabilities. Other systems as *Gargoyle* (Univ. Chicago) are focused on active vision applications. Our system adds capability of distributed development (not jus distributed execution) and interface with scheduling algorithms.

1.1 Scheme of the Application

In the Fig. 1 the basic system architecture is schematically shown. The two main modules can be observed: the visual workspace (together with its associated database) and the scheduling module. Also, the interrelations between the different elements and the information flow are exposed.

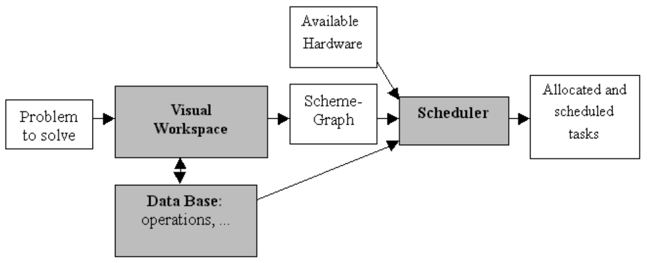


Fig. 1. Scheme of the application

In short, the objectives of the principal components of the application can be deduced from its input and output data (Table 1, Table 2).

Table 1. Data flow of Visual Workspace

Input data	Ouput data	Database contents
Ploblem to be solved	Scheme	Blocks description

Table 2. Data flow of Scheduler

Input data	Ouput data
Task graph (from a sheme)	Allocated and scheduled task
Blocks description	
Available hardware	

The first section of the article explains what the visual workspace consists of and which are its main functionalities. It makes a special point of the different characteristics in relation to other similar environments.

Next, the interface module is described in detail. All the required operations to obtain a task graph (required input in the scheduler) from a scheme (visual workspace output) are specified.

Finally, the third section focuses on the scheduler subsystem, describing all the offered possibilities and the different output information that can be obtained.

An example is added in order to offer a general view of the system, and to show the integration level obtained. A reconstruction process from images acquired by a pair of stereoscopic cameras has been chosen for this purpose.

2 The Visual Workspace of EVA[1]

2.1 What Does the Visual Workspace Consist of?

The visual workspace allows to specify the algorithm or application to be developed as a diagram formed by a sequence of graphic objects (which will be called IPOs – Image Processing Objects), which represent the different operations to be executed. The data (or images) interchange and the execution flow are indicated through the present connections between different IPOs. The visual workspace provides tools to create a scheme, that is to say, to draw IPOs that represent the operations and to interconnect them, as well as the tools to be able to execute the program.

The environment by itself does not execute a scheme, but it works as an interface or front-end application between the user and the processing software modules[2]. The way to operate is to convert the graphic representation into a scheme of commands for the software modules interpreter to refresh the state of the scheme in the screen.

In addition to a communication to the software modules interpreter, the visual workspace can reach remote applications like image servers or viewers. Actually it is not the application which connects to the viewer, but an special kind of IPO, viewer IPO, which is able to ask the interpreter for an image and to send its data to a server.

The EVA application is designed in a distributed way. It consists of different applications: an interpreter of the software modules server, a visual workspace for computer vision systems, and a viewer. Whichever of these applications can be executed several times and in several machines, provided that a TCP-IP connection between them is established.

Figure 2 represents a situation with four machines. Two of them, 3 and 4, work as servers. The other two, 1 and 2, have the visual workspace and the viewers, and are specifically assigned to users.

2.2 Visual Workspace Appearance

Hereafter, the components of the visual workspace are detailed. Figure 3 shows a scheme the way it is represented in the environment.

[1] EVA (Computer Vision Environment). This application belongs to a CICYT project coordinated by different Spanish Universities.

[2] In this paper the development of the software modules is not presented.

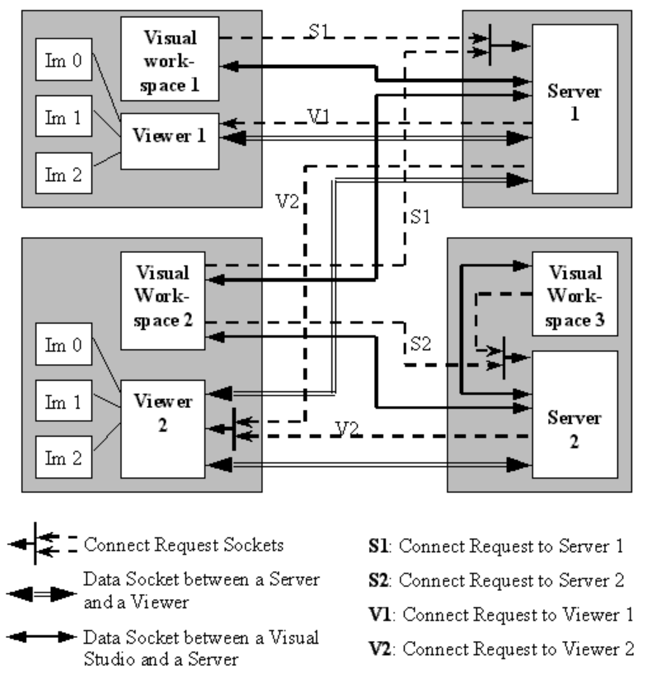
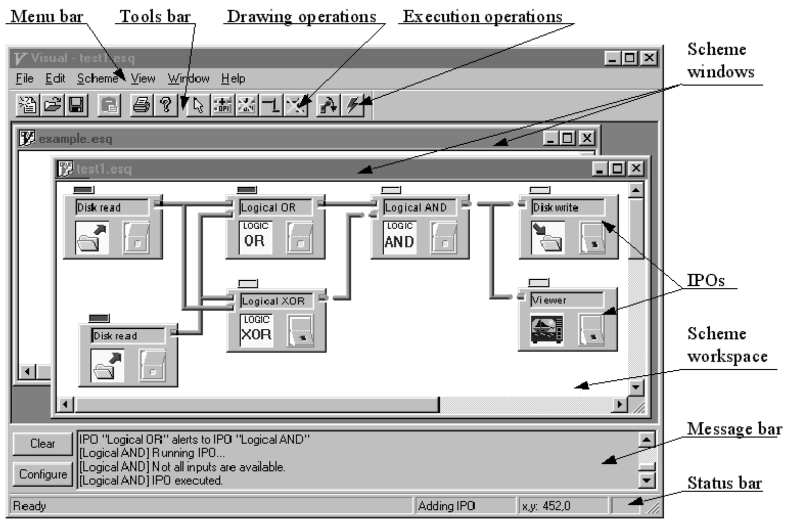

Visual work-space 1
Im 0
Im 1
Im 2
Viewer 1
Server 1
S1
V1
V2
S1
Visual Work-space 2
Im 0
Im 1
Im 2
Viewer 2
Visual Work-space 3
Server 2
S2
V2
Connect Request Sockets
Data Socket between a Server and a Viewer
Data Socket between a Visual Studio and a Server
S1: Connect Request to Server 1
S2: Connect Request to Server 2
V1: Connect Request to Viewer 1
V2: Connect Request to Viewer 2

Fig. 2. EVA Architecture



IPO "Logical OR" alerts to IPO "Logical AND"
[Logical AND] Running IPO...
[Logical AND] Not all inputs are available.
[Logical AND] IPO executed.

Fig. 3. Layout of the visual workspace

The visual workspace is a MDI application. In each window a scheme can be drawn and executed with IPOs and interconnections between them.[1].

3 The Image Processing Objects (IPOs)

In the visual workspace the algorithms are expressed as graphic schemes, which can be designed easily by the user with drawings composed of some basic graphic objects. These objects are the IPOs (Image Processing Objects) and the connections between them.

3.1 What is an IPO?

An IPO is a graphic element that represents an operation to be executed in a scheme. It has several inputs, each one receiving data from other IPOs (mainly they work with data representing images). In the same way, the IPOs have several outputs through which they give back the results data after the execution of the operation they represent.

Each IPO have some characteristics or own properties, which can be divided in two groups. In one hand the general properties, which represent values that have all the IPOs, such as the number of inputs or outputs, etc.; and on the other hand the particular properties, which depend on the kind of operations the IPO performs. An example of particular property can be the selection of a certain operation or function that the IPO must execute between all the possible operations of a type.

In the Fig. 4 an arithmetic operation IPO, which computes the add function of two inputs, is shown:

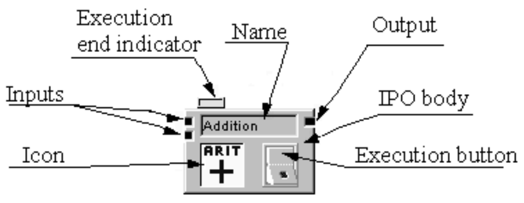


Fig. 4. IPO parts

In the figure the different parts of an IPO can be observed:

- IPO name. This name depends on the operation the IPO represents, although it can be modified by the user.
- IPO icon. It's a small drawing that helps to easily identify the operation the IPO represents.

- Execution button. The user can switch on or off the button. When the button is on, the IPO starts its execution, and the operation will be carried out provided all its inputs are available. When the button is off, the IPO resets.
- Execution indicator. It is the small indicator that is on top of the IPO, and points out whether its associated operation has been executed.
- Input and output connections. They are the square points that are on the sides of the IPO. On the left side there are the input connections, and on the right side there are the output ones. The number of inputs or outputs connections can be chosen depending on the IPO kind. There are IPOs that, because of its nature, have no input or output, IPOs that necessarily have a certain number of inputs or outputs, IPOs that have maximum and minimum limits in the inputs or outputs, etc.

3.2 Considerations of IPO Execution

The execution of each IPO of a scheme can be controlled manually through the execution button each IPO has. To execute an IPO and to operate input data, the execution button must be activated. This operation can be performed manually by the user, switching on the IPO button with the mouse, or automatically when the automatic execution mode is activated. Besides, before executing an IPO, the previous IPOs should have been correctly executed, so that their outputs are available.

When the execution of an IPO is achieved, this activates its execution indicator, operates input data according to specific properties, and, if the execution is correct, generates output data, that is sent to other IPOs which are connected. There are also IPOs as the viewer one or the disk writer (from the kind of IPO disk access) which don't produce any output data to other IPOs, and that only represent graphically or save the results. This kind of IPOs can be considered as outputs schemes, where the data flow ends.

The IPOs pass its data through some objects called connections.

3.3 IPOs connections

The connections are the way in which the dependence between input data of an IPO and output data generated by other IPOs is represented. A connection is represented in a scheme as a pipe that starts in an output IPO and stops, through possible ramifications, in the inputs of another IPOs.

When a IPO has not yet been executed (either it has been added recently to the scheme, or it has been reseted), the connections that start from this one are drawn as broken pipes to indicate that new data have not yet reached its destination. On the contrary, when an IPO is correctly executed and sends the data to its destination IPOs, the output connections appear as continuous pipes, without any cut, so representing circulation of the data to destination IPOs. This can be observed in the scheme of Fig. 5, where the XOR IPO has not yet been executed, and that's why the pipe to the AND IPO is broken. The same happens with the pipe between AND and the Disk Writer IPO.

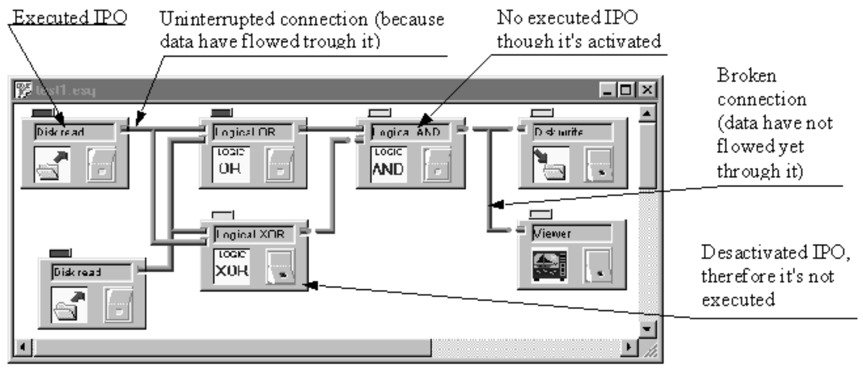

Executed IPO
Uninterrupted connection (because data have flowed trough it)
No executed IPO though it's activated
Broken connection (data have not flowed yet through it)
Desactivated IPO, therefore it's not executed
test1.esq
Disk read
Logical OR
LOGIC
OR
Logical AND
LOGIC
AND
Disk write
Logical XOR
LOGIC
XOR
Viewer
Disk read

Fig. 5. A scheme example

4 Results Viewer: Viewer IPO

4.1 Viewer IPO Performance

A viewer IPO is not able to create by itself a window where to show the image received by the input connection. Instead, if it knows how to communicate to a server.

In this way, to be able to see images that arrive at the viewer IPOs, the viewer can reside in the same machine than the visual workspace, or in a machine which can be accessed through a network from the machine where the visual workspace is being executed.

4.2 Communication with IP Sockets

The connection between the visual workspace application and the server can be achieved through sockets, some objects to communicate between applications of a same machine or machines that coexist in a TCP-IP based network. A server will have assigned an IP address (the machine address where it is being executed in a network) and a port number that identifies it.

5 Visual Workspace / Scheduler Interface

It constitutes one of the main contributions of the present project. Its mission is to serve as an union link between the two applications.

The visual workspace that is described in the previous sections offers as a result a scheme of operations to be executed. As it has already been explained, these schemes are composed of elementary blocks (the so-called IPOs) interconnected between them.

Therefore, the elements that will be served as inputs for the interface are:

- Used IPOs.
- Connections between IPOs.
- Environment database.

The database contains detailed information about each one of the IPOs that can take part in the scheme. This information will be essential to convert the input data (scheme of operations) to a task graph (set of tasks, precedence relations and computation times required by the scheduling subsystem).

More in detail, the output interface must contain the following elements:

- List of elementary subtasks that take part in the process (generally, an IPO will be composed by several tasks of lower level).
- Data relative to the computation time for the subtasks.
- Precedence and exclusion relations (they will be described later).
- Information about preemptive and non preemptive tasks.
- Communication requirements and their associated costs.

5.1 Information about IPOs

We will study in depth which are the contents of the IPOs database required by the interface.

Subtasks Each IPO in the environment constitutes an artificial vision operation. It's possible, therefore, to split it in elementary blocks at many different levels. A high level division would represent a small number of operations and a high complexity degree; while a low level division would require more operations each one showing less complexity.

A multilevel representation has been used for the environment, so the user can work with elementary operations or, if he wish, he can use higher level operators. This way, he does not need to know the internal performance of each one of the algorithms he uses. For example, he can add in the system a block that carries out a certain morphological filter, instead of adding blocks for each one of the elementary operations of erosion and dilation. It's always possible, as it has been mentioned before, to use elementary blocks for very specific algorithms.

On the contrary, the task scheduler requires a representation at the lowest possible level, to be able to get an efficient implementation. Whichever algorithm is capable to be executed faster if we use the maximum parallelism of the system; if we use at the same time different processors to execute each subtask. On the other hand, the fact of executing in a certain order each one of the steps in an algorithm can also produce a speed improvement. But to achieve that point, it's necessary to know in detail the process. And the first data to be known are the subtasks in which a task can be divided. Besides, the more detailed this subdivision, the better the results.

Processor First of all, each one of the subtasks mentioned before must be characterized by the kind of function or kind of processors that can execute it.

First, in an artificial vision system there will be mainly two kinds of processor: CPUs and image processing boards (IAPBs). Most tasks will be carried out using whichever of these two elements. But there will be operations (for example image acquisition) that will be only executed by the processing board. In the same way, there will be complex algorithms that will be only implementable using CPUs.

This way, it's necessary to specify the kind or kinds of processors able to execute each subtask. This information will be used as a restriction in order to get a feasible scheduling.

Execution Time This value must also be specified for each subtask. A standard measure is used: the number of operations required. This way it will be possible to consider the different processor speeds in task assignment and scheduling.

Depending on the processors, there will be fixed or variable execution times. In the latter case, times will be represented by a formula according to the input image size and depth, or in general, according to whichever variable related to the block.

Possible Parameters Finally, the blocks also admits the possibility to be characterized according to:

- Different possibilities to perform the same operation.
- Image kind and size.
- Etc.

Most information is used at present by the scheduling system. The remaining data has been added anticipating possible new scheduler features.

5.2 Interface process

As it has been mentioned before, the output that must produce this interface is a task group. The information contained in this graph is organized in two lists:

- **Task and associated data list**
  Each task is described in terms of:
  - Execution time.
  - Preemption.
  - Communication requirements.
- **Precedence relations list**
  Considers task pairs in a fixed order:
  - Source task.
  - Destination task.

To obtain these lists a top-down procedure, which starts from a rough general description to refine it step by step, has been designed.

The necessary stages in this process are specified:

- **Obtaining the initial graph**
  A first graph is created assigning one task to each IPO. In this way, a tasks list and precedence list is generated. The precedence list is generated according to the present connections in the scheme.
- **Exploding the initial graph**
  Sequentially, each task (so far representing only one IPO) is exploded in its elemental components. This operation requires the following modifications in the lists:
  - Elimination of the task corresponding to the IPO and creation of as many new tasks as components it has.
  - Creation of new elements in the precedence relation list, binding each one of the IPO components.

  The final result of this process is a completely defined graph, to the detail specified in the visual workspace database.

6 Task Scheduler

It constitutes the last stage in the resolution of vision problems. Depending on the available hardware, an optimized space-time scheduling is calculated.

The system is generic and realistic: a static scheduler has been designed that, on one hand, considers subtasks interruption and, besides, takes into account precedence relations. In this way, we can assure that the offered solution is a feasible solution and therefore can be implemented in practice.

6.1 Characteristics of a static scheduling

In static scheduling the processor assignment, as well as start execution times for each subtask, are calculated before its real execution.

One of the main objectives of this scheduler is to reduce the final execution time of a subtasks group, as well as to obtain information about the possibility of executing it within certain time limits. [2] [3] [4] [5]. The algorithms that perform static scheduling can be classified in two groups: first of all, those that give an heuristic solution. This solution is usually based according to a cost weighted by the subtasks to be executed or another important information, so an exhaustive search of solutions is performed, and it tries to calculate the better possible one. On the other hand we have the algorithms that search an approximated solution, and they will stop in the moment that the achieved solution is acceptable. These algorithms are usually based on a solution and refine it in following iterations.

The major advantage of a static scheduling is that the required calculation time to perform the scheduling is spent in the subtasks compilation, what results to be more efficient at the execution time than the dynamic scheduling.

We must also have in mind that these systems do not always give an optimal solution because its calculation, practically in all cases, is a NP-hard problem [6]. An NP-hard ploblem is a seemingly intractable decision ploblem for which the only known solutions are exponential funcions of the ploblem size. Furthermore, there is not tolerance to treat with extern events that could appear during the system execution.

For the application, we have adopted static scheduling methods because they perfectly fit in the system nature: either the tasks to be executed or the available hardware are known a priori, what means that unexpected events will never appear. Moreover, the user needs to know in a reliable way the execution time associated to a certain scheme. Systems using dynamic scheduling are not able to offer such data.

6.2 Relations between Subtasks

Two kinds of relationship between subtasks can been established, one according to their creation instant and the other according to the capacity of being or not being able to be blocked between them [7][8].

Def. *Precedence relations:* a subtask i precedes another, j if and only if j can not start execution until the i subtask has been finished. The precedence relations can be expressed with a guided graph where the nodes are the subtasks to be executed and the edges represent the actual precedence relations. Independent subtasks can be executed concurrently.

Def. *Exclusion relations:* a subtask i excludes another j, when the i subtask is being executed the j subtask can not be executed. This condition serves to guarantee the access to shared resources, as input/output ports or printers.

6.3 Kinds of Subtasks

To perform the system scheduling we need to know the subtasks graph in which this system is divided. The interface module generates this information from the scheme coming from the artificial vision system editor.

The subtasks of this graph can be divided in three groups:

CPU This kind of subtask is to be executed only in a processor of CPU kind, and having in mind the characteristics of it, they will be able to be preemptive. The main purpose in vision systems will be to execute high level operations not available in hardware; as well as synchronizing execution with the rest of processors when necessary.

IAPB Low level subtasks that are only executed in a IAPB processor, as for example the image acquisition, filtering, etc. They will be able to be preemptive. To execute such a task in a certain IAPB, a CPU processor should have sent previously the proper order. These orders can be sent in packets, so making a better use of the available broadband. Therefore, before executing a IAPB subtask, a CPU task and a communication one called CPU/IAPB will be required.

CPU/IAPB They permit to send data between CPU and IAPB processors. This data can be, from single instructions to execute a certain process, to the results achieved in a certain processing. These subtasks can not be stopped and require the two implied processors to be available during the transmission; it's to say, we can consider that they are being executed in the two processors at the same time.

6.4 Scheduling and Allocation

The exposed development allows multiple possibilities of task scheduling/assignment, the user can choose among them according to the results he wishes to obtain:

1. Temporal scheduling using only one processor of each kind.
2. Temporal scheduling provided a previous spatial assignment has been performed manually.
3. Spatial assignment based on a temporal scheduling previously calculated for only one processor.
4. Simultaneous assignment and scheduling. These methods give the better results because they consider the possible interrelations between spatial and temporal distributions.
5. Information about the maximum number of processors of each kind necessary for a minimum time execution.

For the algorithms of kind 2, 3 and 4, it's necessary to know how many processors of each kind are available. This is part of the information required by the scheduler: the available hardware.

The scheduler gives, whichever the chosen algorithm, the following information about the scheduling results:

1. Spatial and temporal distribution graph: shows graphically which subtasks are executed in each processor and their start and finish times. Besides, a color code allows to distinguish between subtasks types.
2. Processor usage: indicates how busy each processor is (percentage of time the processor is not idle).
3. System usage: average time for all the available processors.
4. System Execution time: time necessary to fully execute the scheme.
5. Subtasks graph: graph used to obtain the scheduling.
6. Information about the subtasks: it allows to view the information relative to each subtask, displaying only those groups selected by the user.

Finally, we must emphasize that once the initial simulation has been performed, it's possible to obtain another simulation with the same data using a different algorithm or modifying the available hardware.

7 Example of the System Performance

To evaluate the functionality of the presented system, an example is outlined below showing a stereo correspondence algorithm. The example algorithm calculates the stereo correspondence between edge features obtained as the zero-crossings of the convolution of each image with the $\nabla^2 G$ (Laplacian of Gaussian) operator [9]. The algorithm implements the cross-channel coherence between filters tuned to different spatial frequencies [10].

Without deepening in the algorithm, whose details can be found in the previously indicated references, the images obtained from a stereo pair of parallel axis [10] are filtered at three different spatial frequency channels. The correspondence at each channel establishes a set of candidates based on the similarity constrains of sign and direction, within a maximum disparity interval. The last module establishes a coherence checkup between channels to accomplish a classification of matching candidates.

The following figures show step by step how the system behaves. Figure 6 represents the first step where the algorithm is represented as a visual workspace scheme. Figure 7 shows the task graph obtained from the scheme. Task procedures computation times and other data are available at this level. Figure 8 outlines the automatic allocation and scheduling results provided there are 3 CPUs and 2 IAPBs in the system. Figure 9 shows an alternative schedule using manual assignment. Finally, Fig. 10 represents the automatic allocation and scheduling provided there are no hardware restrictions.

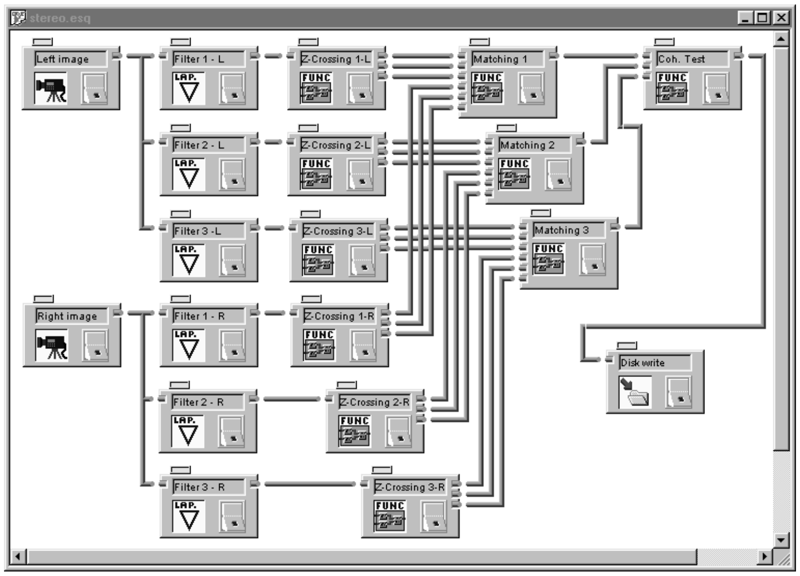


Fig. 6. Stereo algorithm scheme (Visual workspace view)

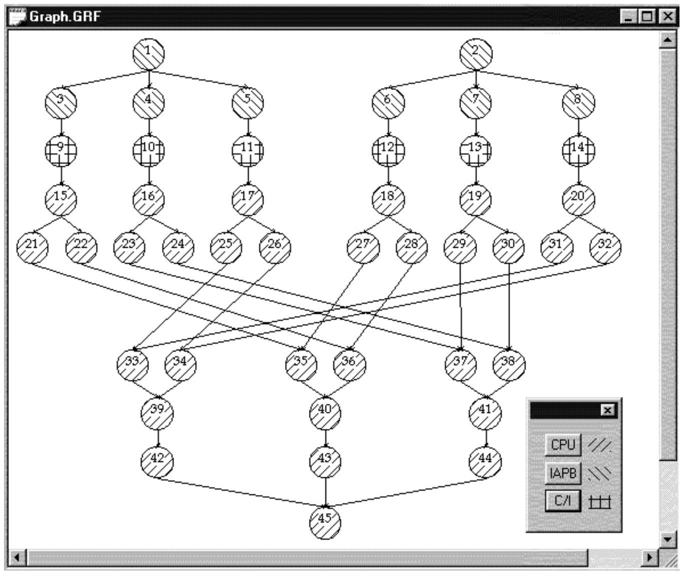


Fig. 7. Stereo algorithm task graph (Scheduler view)

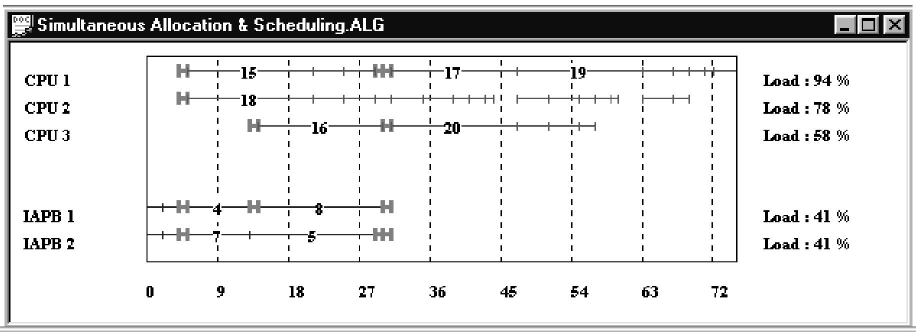
8 Conclusions

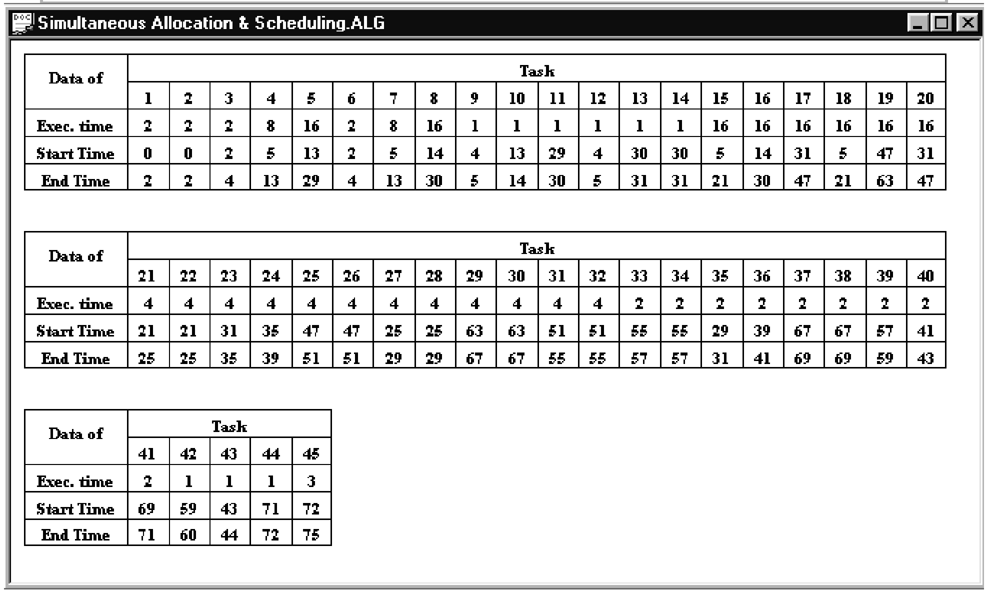
The visual workspace becomes a powerful tool in the development of applications in computer vision, with the added possibility of hierarchical design in complex processes.

The application offers a huge amount of information to the user, as compared to other similar tools. Particularly, hardware selection becomes an easy task once the time requirements are fixed. And it is also possible to check for the feasibility of a certain real time processing algorithm by just simulating the system with no hardware restrictions.

The general algorithm (simultaneous assignment and scheduling) does not always find the optimal solution. In fact, on complex systems like those we are dealing with, it can not be assured that the optimal solution has been found without checking all the possible schedules, which in most cases is an unaffordable work (references on this point can be found on [6] and [11]). What the system gives is a pseudo-optimal schedule based on a heuristic algorithm. The performance evaluation of our algorithm in comparison with other scheduling techniques is not easy as most of these techniques can not be applied to scenarios as complex as those our algorithm deals with (preemption, etc.).

On the other side, the 'no hardware restrictions algorithm' (hardware requirements for optimal performance: figure 10) does give an optimal solution in terms of execution time, as it uses as many processors as needed.




Simultaneous Allocation & Scheduling.ALG

Data of	Task																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Exec. time	2	2	2	8	16	2	8	16	1	1	1	1	1	1	16	16	16	16	16	16
Start Time	0	0	2	5	13	2	5	14	4	13	29	4	30	30	5	14	31	5	47	31
End Time	2	2	4	13	29	4	13	30	5	14	30	5	31	31	21	30	47	21	63	47

Data of	Task																			
	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
Exec. time	4	4	4	4	4	4	4	4	4	4	4	4	2	2	2	2	2	2	2	2
Start Time	21	21	31	35	47	47	25	25	63	63	51	51	55	55	29	39	67	67	57	41
End Time	25	25	35	39	51	51	29	29	67	67	55	55	57	57	31	41	69	69	59	43

Data of	Task				
	41	42	43	44	45
Exec. time	2	1	1	1	3
Start Time	69	59	43	71	72
End Time	71	60	44	72	75

Fig. 8. Automatic task allocation and scheduling (Hardware: 3 CPU's, 2 IAPB's)

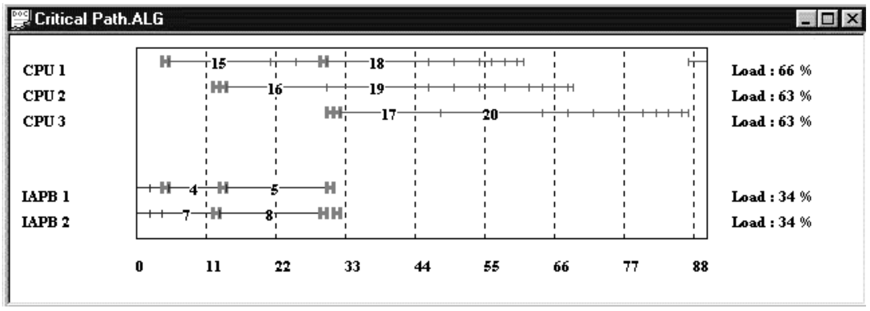


Fig. 9. Automatic scheduling (Hardware: 3CPU's, 2 IAPB's)

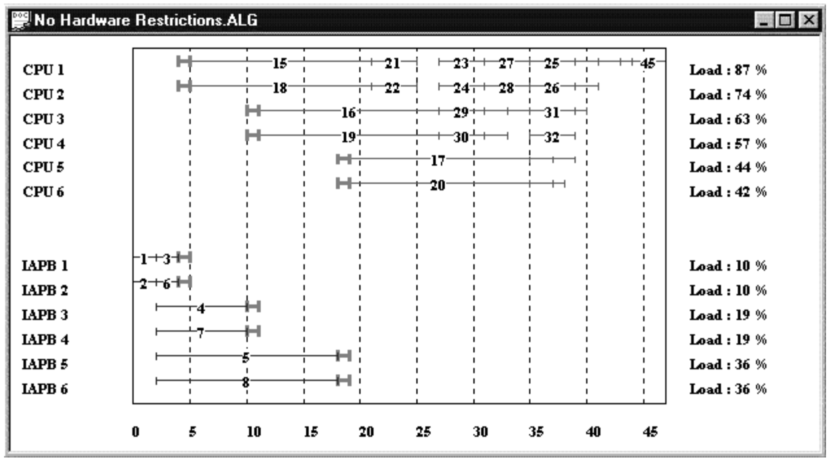


Fig. 10. Hardware requirements for optimal performance

Future improvements include the consideration of communication costs in order to make the simulations even more realistic. New scheduling techniques [12][13]are being investigated at present with promising results.

9 Acknowledgments

This work was supported by the Spanish Government through CICYT in the framework of project TAP-96-0629-C04-01.

References

1. Ben Shneiderman, Designing the User Interface, Addison Wesley, 1998.
2. Lo, V.M., Heuristic Algorithms for Task Assignment in Distributed systems. IEEE Trans. Computers, Vol C-37, N 11, Nov, 1988, pages 1384-1397.
3. Sarkar, V., and J. Hennesy, Compile-Time Partitioning and Scheduling of Parallel Programs. Symp. Compiler Constructrion, ACM Press, New York, N.Y., 1986, pages 17-26.
4. Shirazi, B., M. Wang and G. Pathak, Analysis and Evaluation of Heuristic Methods for Static Task Scheduling. In Parallel and Distributed Computing, Vol,. 10, 1990, pages 222-232.
5. Stone, H. S. Multiprocessor Scheduling with the Aid of Network Flow Algorithms. IEEE Trans. Software Eng, Vol. SE-3 N 1, Jan 1977, pages 85-93
6. Phillip A. Laplante, Real-Time Systems Design and Analysis. IEEE Press, New York (1997).
7. F. Torres Medina, Arquitectura paralela para el procesado de imgenes de alta resolucin. Aplicacin a la inspeccin de impresiones en tiempo real. PhD. ETSII, Polytechnic University of Madrid, 1995.
8. C.M. Krishna, Kang G. Shin, Real-time systems. McGraw-Hill, New York (1997).

9. Marr, D. Vision, Ed. Freeman, 1982
10. Mayhew, J.E.W. and Frisby, J.P. Psychophysical and Computation Studies Towars a Theory of Human Stereopsis, Artificial Intelligence, 17, pp.349-386, 1981.
11. Nimal Nissanke, Realtime Systems. Prentice Hall, London (1997).
12. Lee, B., A. R. Hurson, and T.-Y Feng, A Vertically Layered Allocation Scheme for Data Flow Systems. In J. Parallel and Distributed Computing. Vol. 11, N 3, 1991, pages 175-187.
13. Wang, M., et al,. Accurate Communication Cost Estimation in Static Task Scheduling. In Proc 24th Ann. Hawaii Int'l Conf. System Sciences. Vol I, IEEE CS Press, Los Alamitos