From data-flow task to multitasking: applying the synchronous approach to active vision in robotics

Eric Marchand, E. Rutten, F. Chaumette

From Data-Flow Task to Multi Tasking : Applying the Synchronous Approach to Active Vision in Robotics

ric Marchand, ric Rutten, and François Chaumette

Abstract— **In this paper, we apply the synchronous approach to real time active visual 3D reconstruction using a camera mounted on a robot end-effector. It illustrates the adequateness of SIGNAL, a synchronous data flow programming language and environment, for the specification of a system dealing with various domains in control theory and computer vision. More precisely, our application consists in the 3D structure estimation of a set of geometrical primitives using an active vision paradigm. At the level of camera motion control, the visual servoing approach is specified and implemented in SIGNAL as a function from sensor inputs to control outputs. Furthermore, the 3D structure estimation method is based on the "structure from controlled motion" approach (constraining camera motion for optimal estimation). Its specification is made in parallel to visual servoing, and involves the delay mechanism of SIGNAL for the specification of filters. This reconstruction involves to be focused on each object; we thus present a perception strategy for sequencing several estimations, using task preemption and time intervals in SIGNAL. It allows to consider in an unified framework the various aspects of the application: from data-flow task specification to multi-tasking and hierarchical task preemption. The integration of these techniques is validated experimentally by their implementation on a robotic cell. Merits and drawbacks of the proposed framework with respect to more usual asynchronous approaches are finally discussed.**

Keywords— **Synchronous language, SIGNAL, real-time, data-flow tasks, preemption structures, visual servoing, structure from controlled motion, perception strategies.**

I. INTRODUCTION

IN this paper, we apply the synchronous approach to the integration of control systems and particularly to real time active vision. We present the integration of different new techniques for the structure estimation of a robot environment by means of an active vision scheme. Recovering 3D structure from images is one of the main issues in computer vision. The approach that we have chosen to get an accurate three-dimensional geometric description of a scene consists in controlling the motion of a moving camera in order to improve the quality of the perceptual results. Such constraints are ensured using the *visual servoing approach* to define the sensor-based control of the robot. However, the integration issue of such system is rarely clear in the literature; most of the time, it results from the integration of various programs running under a real time operating system. This kind of techniques cannot provide a safe implementation on which correctness properties can be checked (the implementation is far from original specification, the translation in an executable program is error-prone, modification in specification yields to complex changes in the program,...) Thus, to improve the engineering process, it is necessary to provide users tools which automatizes the analysis of the different levels of specifications, from the definition of low level processes to their integration into complex systems and their compilation into automatically generated executable code.

In order to address these points, the technique involved in this paper is the *synchronous approach to reactive real time systems* [8], and particularly the language SIGNAL [32]. Reactive systems [8][25] are characterized by the fact that their pace is determined by their environment. An interpretation of the synchrony hypothesis is that all the relevant values involved in a computation (input, output and internal) are present simultaneously within the single instant of logical time when the system reacts to its inputs. In other words, it is valid if the system can be proved to react rapidly enough to perceive all relevant external events. It is an abstraction of the commonly used infinite loop of automatic controllers (input acquisition, computation, output return). This form of synchrony is however a realistic abstraction, since it is actually present in numerous domain such as digital hardware (zero-delay), and control theory (design of control laws). It facilitates the semantical manipulations on programs. It has the advantage that it guarantees deterministic behaviors, which is not the case of real-time operating systems (dependent on unknown or uncontrolled parameters) or general purpose languages like ADA (inherently non-deterministic, especially in the composition of sub-processes). The synchronous semantics provides support for a whole set of tools assisting the design of real-time applications. The analysis at the different levels of abstraction, from requirements down to code generation, and possibly implementation on specific hardware through co-design, performance evaluation, and optimization are performed on sound formal bases. The analysis and verification techniques handle the logical time aspects of discrete event systems. Having this guarantee on behaviors, it is then possible to make very accurate estimations of timing properties of applications, according to the specific hardware architecture which can also be multi-processor; this quantitative analysis is the purpose of

ric Marchand is with IRISA/Universit de Rennes 1, TEMIS Project, Campus de Beaulieu, 35042 Rennes-cedex, France; e-mail: marchand@irisa.fr.
ric Rutten is with IRISA/INRIA Rennes, EP-ATR Project, Campus de Beaulieu, 35042 Rennes-cedex, France; e-mail: rutten@irisa.fr.
Franois Chaumette is with IRISA/INRIA Rennes, TEMIS Project, Campus de Beaulieu, 35042 Rennes-cedex, France; e-mail: chaumett@irisa.fr.

a system like SYNDEX [47] which can be used in combination with SIGNAL.

A family of languages [24] is based on this synchrony hypothesis, featuring among other ESTEREL [12], LUSTRE [23], SIGNAL [32] and also STATECHARTS [26]. They all feature complete environments, with sets of tools based upon their formal semantics, and support specification, formal verification, optimization and generation of executable code. Their aim is to support the design of safe critical applications, especially those involving signal processing and process control. The synchronous technology and its languages are available commercially, and applied in industrial contexts [9]. Among synchronous languages, SIGNAL is a real-time synchronized data-flow language [32]. Its model of time is based on instants, and its actions are performed within the instants. SIGNAL*GTi* [42] is an extension that introduces intervals of time, and provides constructs for the specification of hierarchical preemptive tasks executed on these intervals.

This paper presents a real-size experiment of SIGNAL, on a robotics system, using this new language extension [37]. It concerns the reconstruction of complex scenes from 2D images, acquired by a camera mounted on a robot end-effector. It is illustrative of the synchronous methodology and its adequateness for that class of systems. Indeed, such an application allows us to show benefits of using SIGNAL in the following domains involved in robotics and computer vision: robot control, estimation algorithms, and task level programming. More precisely, the active visual reconstruction problem presented in this paper is handled at three levels:

- The lowest level concerns the *control of the camera motion*. A new approach to vision-based control was introduced a few years ago [20]. The basic idea consists in considering a vision system as a specific sensor dedicated to a task and included in a control servo loop. At this level, a robot task is seen as a *data flow function* computing the flow of control values for the actuator from the flow of sensor input data. In that case, the synchrony hypothesis clearly applies to the equations defining a sensor-based control law, and benefits to the implementation.
- The second level concerns the *structure estimation aspect*. Embedded in the same formalism, the "*structure from controlled motion*" paradigm allows us to obtain an optimal estimation of the parameters of a 3D geometrical primitive [15]. Its specification involves parallelism with the camera motion control task, as well as a dynamical aspect, in that it is defined in function of past measured values.
- The high level deals with *perception strategies*. Since the approach for structure estimation involves to gaze on the considered primitive, we present a method for connecting up many estimations in order to recover the complete spatial structure of scenes composed of several objects (we only consider in this paper segments and cylinders). Developing perception strategies able to perform a succession of robust estimations without any assumption on the number and on the localization of the different objects is thus necessary. There, the task-level programming consists in specifying different robot tasks and sequencing them by associating them with modes on which they are enabled [36]. For the specification of such hierarchical and parallel transition systems, we use SIGNAL*GTi* the extension of SIGNAL to the notions of *task* and *time interval* [42].

The remainder of this paper is organized as follows: Section II is devoted to image-based control loop description and its specification. In Section III, structure from motion aspects based on an active vision paradigm are considered. Section IV is devoted to perception strategies and their specification in terms of a hierarchy of tasks. A critical discussion is then proposed in Section V where merits and drawbacks of the proposed framework with respect to more usual asynchronous approach are discussed.

II. EQUATIONAL SPECIFICATION OF VISUAL SERVOING

Two main approaches are currently used in robot control based on visual data [50]: the *position-based control* which is achieved by computing, from the visual data, the 3D position and orientation of the camera with respect to its environment, and the *image-based visual servoing*, which consists in specifying a task as the regulation in the image of a set of visual features [2][20][22][28][30][13]. In the remainder of this paper, we will only refer to this last approach since it is able to provide robust and stable closed-loop control laws. This section recalls the application of the task function approach to visual servoing and the expression of the resulting control law, before the presentation of its specification in SIGNAL.

A. Visual Sensing - the Interaction Matrix

We first examine what data can be extracted from an image and incorporated in a vision-based control scheme. In fact, it has been shown [20] that such an ability relies on the explicit knowledge of the spatio-temporal evolution of a visual feature with respect to camera motion (in the following, we represent this evolution by *the interaction matrix* related to the considered feature).

Let us model a camera by a perspective projection. Without loss of generality, the camera focal length is assumed to be equal to 1, so that any point with coordinates $\underline{x} = (x,\, y,\, z)^T$ is projected on the image plane as a point with coordinates $\underline{X} = (X,\, Y,\, 1)^T$ with:

$$\underline{X} = \frac{1}{z}\,\underline{x} \tag{1}$$

Let us consider a geometrical primitive $\mathcal{P}_s$ of the scene; its configuration is specified by an equation of the type:

$$h(\underline{x}, \underline{p}) = 0\ , \forall \underline{x} \in \mathcal{P}_s \tag{2}$$

where h defines the kind of the primitive and the value of parameter vector $\underline{p}$ stands for its corresponding configuration.

Using the perspective projection equation (1), we can define from (2) the two following functions [20]:

$$\left\{ \begin{array}{l} g(\underline{X}, \underline{P}) = 0\ , \forall \underline{X} \in \mathcal{P}_i \\ 1/z = \mu(\underline{X}, \underline{p}_0) \end{array} \right. \tag{3}$$

where:

- $\mathcal{P}_i$ denotes the projection in the image plane of $\mathcal{P}_s$
- g defines the kind of the image primitive and the value of parameter vector $\underline{P}$ its configuration.
- function μ gives, for any point of $\mathcal{P}_i$ with coordinates $\underline{X}$, the depth of the point of $\mathcal{P}_s$ the projection of which results in point $\underline{X}$.
- parameters $\underline{p}_0$ describe the configuration of μ and are function of parameters $\underline{p}$.

More precisely, for planar primitives (a circle for example), the function μ represents the plane in which the primitive lies. For volumetric primitives (sphere, cylinder, torus,...), function g represents the projection in the image of the primitive limbs and function μ defines the 3D surface in which the limbs lie (see Figure 1). Function μ is therefore called limb surface.

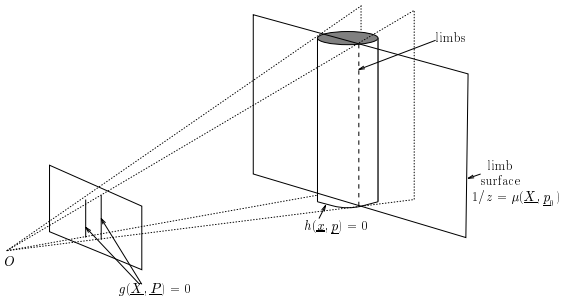


Fig. 1. Projection of the primitive in the image (g) and limb surface (μ)

Let $T_c = (V, \Omega)$ be the camera kinematic screw where $V = (V_x, V_y, V_z)$ and $\Omega = (\Omega_x, \Omega_y, \Omega_z)$ represent its translational and rotational components. The time variation of $\underline{P}$, which links the motion of the primitive in the image to the camera motion T_c, can be explicitly derived [20] and we get:

$$\dot{\underline{P}} = L_{\underline{P}}^T(\underline{P}, \underline{p}_0)\, T_c \tag{4}$$

where $L_{\underline{P}}^T(\underline{P}, \underline{p}_0)$, called the interaction matrix related to $\underline{P}$, fully characterizes the interaction between the camera and the considered primitive.

We may thus choose as visual features in a visual servoing framework the parameters $\underline{P}$ which describe the configuration of one or several primitives observed in the image (such as the coordinates of a point, the orientation and distance to origin of a line, the inertial moments of an ellipse, etc).

The design of a vision-based task now consists in selecting the visual features $\underline{P}$, able to realize the specified task, and their desired value $\underline{P}_d$ to be reached in the image. As shown in the next section, the control law able to perform such a task is essentially based on the interaction matrix related to $\underline{P}$ (we will see in Section III that the interaction matrix is also involved in our 3D structure estimation method).

B. Expression of the Task Function and Control

Embedding visual servoing in the task function approach [43] allows us to take advantage of general results helpful for analysis and synthesis of efficient closed loop control schemes. We only recall the obtained results, all the developments being fully described in [43] and, in the particular case of vision-based control, in [20]. We define a vision-based task, $\underline{e}_1$:

$$\underline{e}_1 = C\,(\underline{P} - \underline{P}_d) \tag{5}$$

where:
- $\underline{P}_d$ is the desired value of the selected visual features;
- $\underline{P}$ is their current value, measured from the image at each iteration of the control law;
- C is called combination matrix and can be defined as:
  - $C = W L_{\underline{P}}^{T^+}(\underline{P}, \widehat{\underline{p}_0})$ if the 3D parameters $\underline{p}_0$, involved in the interaction matrix, can be estimated on-line (using for example the 3D structure estimation method that we present in Section III). In that case, W is defined as a full rank matrix such that $\mathrm{Ker}\, W = \mathrm{Ker}\, L_{\underline{P}}^T$ and C has to be computed at each iteration of the control loop.
  - $C = W L_{\underline{P}}^{T^+}(\underline{P}_d, \underline{p}_{0_d})$ if the value of the interaction matrix can not be updated at each iteration of the control law. Assumptions on the shape and on the geometry of the considered primitives in the scene have thus generally to be done in order to compute the desired values $\underline{p}_{0_d}$. Such a choice allows us to avoid the on-line estimation of parameters $\underline{p}_0$. In that case, we set W as a full rank matrix such that $\mathrm{Ker}\, W = \mathrm{Ker}\, L_{\underline{P}}^T(\underline{P}_d, \underline{p}_{0_d})$ and C, which is now constant, is computed only once at the beginning of the control loop.

F When the vision-based task does not constrain all the six camera degrees of freedom, a secondary task, such as a trajectory tracking, can be combined with $\underline{e}_1$. It can be expressed as the minimization of a cost function h_s, with gradient function $\underline{g}_s$. The task function $\underline{e}$, minimizing h_s under the constraint $\underline{e}_1 = 0$, takes the form:

$$\underline{e} = W^+ \underline{e}_1 + \left(\mathbf{I}_6 - W^+ W\right) \underline{g}_s^T \tag{6}$$

where W^+ and $\mathbf{I}_6 - W^+W$ are two projection operators which guarantee that the camera motion due to the secondary task is compatible with the regulation of $\underline{P}$ to $\underline{P}_d$.

A general control scheme aimed at minimizing the task function $\underline{e}$ is described in [43]. We here only present the simplified control scheme that we have used to perform the experimentations described further. Similar control approaches can be found in [27] and [40].

For making $\underline{e}$ exponentially decreases and then behaves like a first order decoupled system, we have [20]:

$$T_c = -\lambda \underline{e} - \widehat{\frac{\partial \underline{e}}{\partial t}} \tag{7}$$

where:
- T_c is the desired camera velocity given as input to the robot controller;
- λ is the proportional coefficient involved in the exponential convergence of $\underline{e}$;
- $\widehat{\frac{\partial \underline{e}}{\partial t}}$ can be written under the form:

$$\widehat{\frac{\partial \underline{e}}{\partial t}} = W^+ \widehat{\frac{\partial \underline{e}_1}{\partial t}} + (\mathbf{I}_6 - W^+ W) \frac{\partial \underline{g}_s^T}{\partial t} \tag{8}$$

The choice of the secondary cost function generally allows us to know $\frac{\partial g}{\partial t}^T$. On the other hand, vector $\widehat{\frac{\partial e_1}{\partial t}}$ represents an estimation of a possible autonomous target motion. If the target moves, this estimation has to be introduced in the control law in order to suppress tracking errors. It can be obtained using classical filtering techniques such as Kalman filter [30] [14]. In our case, since we are interested in the 3D reconstruction of static scenes, we will assume that $\frac{\partial e_1}{\partial t} = 0$.

C. Towards implementation

From the point of view of programming, these algorithms have two specific features. First, they have an equational nature: they express relations between various flows of data, in a declarative way. In particular, the iterative aspect in the control loop (at each instant) is completely implicit. Second, they are synchronous: the equations involve values of the different quantities within the same instant. Classical programming methods are not well adapted to specify and program such algorithms. Asynchronous imperative languages require the explicit management of low level aspects of the implementation (like the sequencing of computations imposed by data dependencies). Hence, we use the synchronous data flow language SIGNAL, providing the adequate high-level of abstraction for specification, as well as a coherent model of time. A more complete discussion on the merits of SIGNAL is developped in Section V. In the following, we first briefly present the synchronous language SIGNAL, then we show how the control law presented before can be specified in this language.

D. Data Flow Equations in SIGNAL

SIGNAL [32] is a synchronous real-time language, data flow oriented (*i.e.,* declarative) and built around a minimal kernel of operators. This language manipulates signals, which are unbounded series of typed values, with an associated clock determining the set of instants when values are present. For instance, a signal `X` denotes the sequence $(\mathtt{x}_t)_{t\in T}$ of data indexed by time t in a time domain T. Signals of a special kind called `event` are characterized only by their clock *i.e.,* their presence. Given a signal `X`, its clock is noted `event X`, meaning that the event is present simultaneously with `X`. The constructs of the language can be used in an equational style to specify relations between signals *i.e.,* between their values and between their clocks. Systems of equations on signals are built using a composition construct. Data flow applications are activities executed over a set of instants in time: at each instant, input data is acquired from the execution environment. Output values are produced according to the system of equations considered as a network of operations.

The kernel of the SIGNAL language is based on four operations, defining elementary processes, and a composition operation to build more elaborate ones.

- *Functions* are instantaneous transformations on the data. For example, signal Y_t, defined by the instantaneous function f in: $\forall t,\ Y_t = f(X_{1_t}, X_{2_t}, \ldots, X_{n_t})$ is encoded in SIGNAL by: `Y := f{ X1, X2,..., Xn}`. The signals `Y`, `X1`,..., `Xn` are required to have the same clock.
- *Selection* of a signal `X` according to a boolean condition `C` is: `Y := X when C`. The operands and the result do not generally have identical clock. Signal `Y` is present if and only if `X` and `C` are present at the same time and `C` has the value `true`; when `Y` is present, its value is that of `X`.
- *Deterministic merge*: `Z := X default Y` defines the union of two signals of the same type. The clock of `Z` is the union of that of `X` and that of `Y`. The value of `Z` is the value of `X` when it is present, or otherwise that of `Y` if it is present and `X` is not.
- *Delay Operator*, a "dynamic" process giving access to past values of a signal, will be presented in Section III-C.

Composition of processes is the associative and commutative operator "`|`" denoting the union of the underlying systems of equations. In SIGNAL, for processes P_1 and P_2, it is written: `(|`P_1 `|`P_2`|)`.

Hierarchy, modularity and re-use of processes are supported by the possibility of defining process models, and invoking instances.

The SIGNAL compiler performs the analysis of the consistency of the system of equations, and determines whether the synchronization constraints between the clocks of signals are verified or not. This is based on an internal representation featuring a graph of data dependencies between operations, augmented with temporal information coming from the clock calculus. If the program is constrained so as to compute a deterministic solution, then executable code can be automatically produced (in C or FORTRAN). The complete programming environment also contains a graphical, block-diagram oriented user interface where processes are boxes linked by wires representing signals, as illustrated in Figure 2.b.

E. Application to Visual Servoing

A robot control law, at the relatively lowest level, consists in the regulation of a task function, which is an equation $c = f(s)$ giving the value of the control c to be applied to the actuator, in terms of the values s acquired by the sensors. The control of the actuator is a continuous function f, more or less complex. Such a task can be composed of several sub-tasks, with a priority order. The implementation of such a control law is made by sampling sensor data s into a flow of values s_t, which are used to compute the flow of commands c_t: $\forall t, c_t = f(s_t)$. This kind of numerical, data flow computation is the traditional application domain of data flow languages in general, and of SIGNAL in particular. Furthermore, as indicated by the time index t in this schematical equation, the simultaneous presence of the values involved is adequately handled by the synchrony hypothesis.

A modular description of the visual servoing process (in the case where C and W are chosen constant) is given in Figure 2, also representing a block-diagram of the corresponding SIGNAL program. At a high level, the visual servoing process is composed of three different sub-modules:

- a `CAMERA_OUTPUT` module which provides a flow of image information at video rate: `P`.

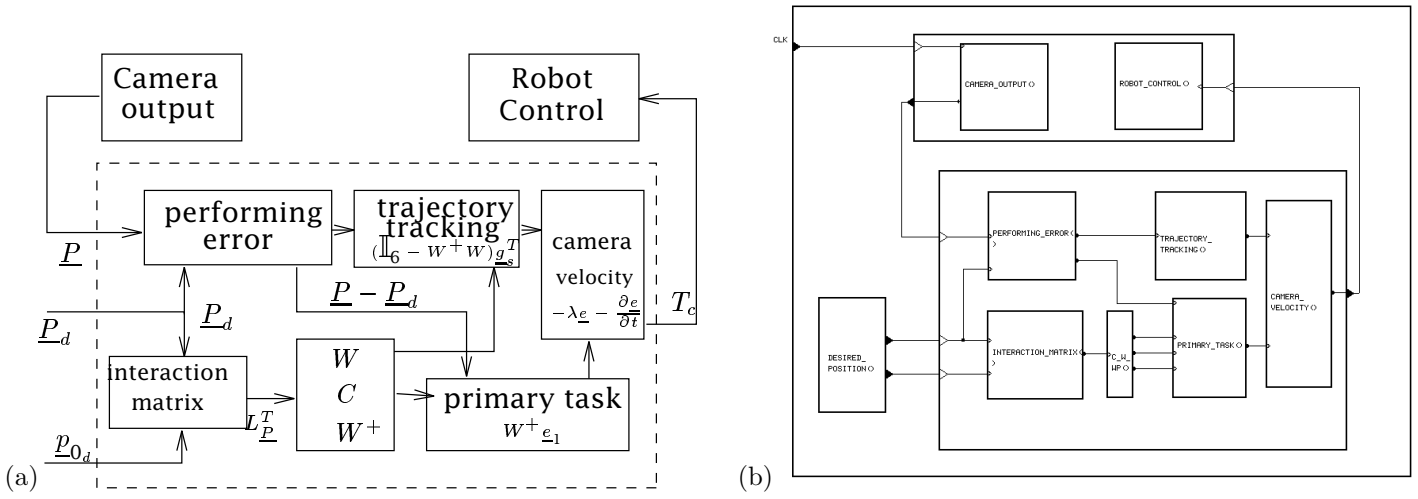


Fig. 2. (a) Modular description of a general visual servoing process, (b) SIGNAL specification.

• this information is received by the control module as input. This process computes the corresponding camera velocity `TC` using the task function approach.
• this camera velocity is transmitted to the `ROBOT_CONTROL` module.

The control module itself is hierarchically decomposed into sub-modules : the `PERFORMING_ERROR` process computes the error $\underline{P} - \underline{P}_d$; the `INTERACTION_MATRIX` process computes $L_{\underline{P}}^T(\underline{P}_d, \underline{p}_{0_d})$; from the output of the `INTERACTION_MATRIX` module, a process computes the matrixes W, W^+ and C which are used with the `PERFORMING_ERROR` module to determine the camera velocity for the `PRIMARY_TASK` ($W^+\underline{e}_1$); a module performs a secondary task (here `TRAJECTORY_TRACKING`). This trajectory tracking is performed only when the error $\underline{P} - \underline{P}_d$ is less that some threshold ε, otherwise it is null. The final `CAMERA_VELOCITY` is then computed using the two flows of data coming from the `PRIMARY_TASK` and the secondary `TRAJECTORY_TRACKING` task.

In conclusion, a SIGNAL program of the control process can be written as in Figure 3. Figure 2.b shows the same program presented with the graphic interface of SIGNAL.

```
(| P := CAMERA_OUTPUT{}
 |(| L := INTERACTION_MATRIX{pd,Pd}
   | error := PERFORMING_ERROR{P,Pd}
   | tau := PRIMARY_TASK{L,error}
   | traj := TRAJECTORY_TRACKING{pd} when error < epsilon
          default NULL_MOTION{}
   | Tc := CAMERA_VELOCITY{tau,traj}   |)
 | SEND_VELOCITY{Tc} |)
```

Fig. 3. The equation of the whole control process.

F. Experimental Results: visual servoing

The whole application presented in this paper has been implemented with SIGNAL on an experimental testbed composed of a CCD camera mounted on the end effector of a six degrees of freedom cartesian robot (see Figure 4). The image processing part is implemented in C and performed on a commercial image processing board (EDIXIA IA 1000). The implementation of the control law, as well as the 3D structure estimation and the automata which will be described further, are implemented using the SIGNAL language and run on a SPARC Station 10. Details are given in [37] and [36].

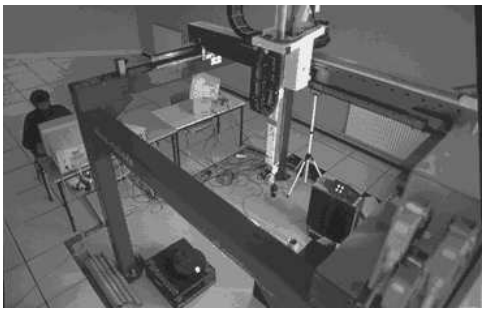


Fig. 4. Experimental cell (camera mounted on a 6 dof robot)

We here present the results of the realization of the positioning task with respect to a cylinder. Let us note that we want the cylinder to appear centered and vertical in the image. After the convergence of the vision-based task, successive trajectory trackings (around the cylinder then along its axis) are performed. More precisely the secondary task consists in moving the camera with a velocity $V_x = 5cm/s$, $V_y = 10cm/s$, $-V_x$ and $-V_y$ successively.

Figure 5.a represents the initial image acquired by the camera and the selected cylinder (note the superimposed white lines). Figure 5.b contains the image acquired by the camera after the convergence of the vision-based task. In Figure 5.c are plotted the behavior of the four components of $\underline{P} - \underline{P}_d$. Let us point out the exponential decay of these evolutions during the convergence phase (iteration 0 to 170). The graphics shown in Figure 5.d (respectively Figure 5.e) represent the evolution of the translational (resp. rotational) components of the camera velocity T_c. Let us note that a rotational motion compensates for the translational motion around the cylinder,

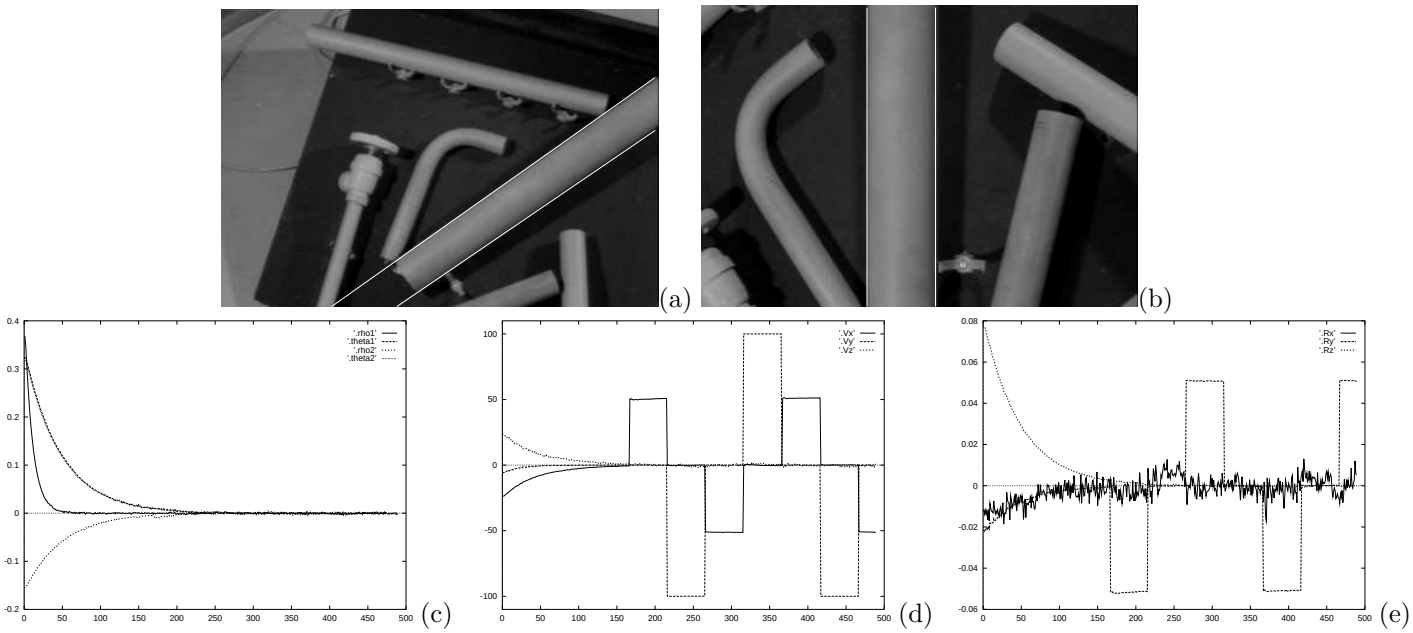


Fig. 5. Positioning with respect to a cylinder and trajectory tracking, plots versus number of frames.

and makes it be static in the image plane during the trajectory tracking.

III. Data Flow Processes for Active 3D Reconstruction

The determination of the 3D description of a scene from 2D images is one of the main issues in computer vision. The work presented in this section is concerned with the processing of a sequence of images acquired by a moving camera to get an exact and complete description of geometrical primitives [15]. The camera motion will be performed using the visual servoing approach presented above. The estimation of the considered primitive will be achieved in parallel to the computation of the control law. Furthermore, we will see that performing the control law needs the use of previous values of the estimated parameters of the 3D primitive. This introduces the need for the other features of Signal, in order to specify delays and parallelism.

A. 3D Structure Estimation Using Dynamic Vision

The observability of the camera motion which is necessary for the 3D structure estimation characterizes a domain of research called dynamic vision. Approaches for 3D structure recovery may be divided into two main classes : the discrete approach, where images are acquired at distant time instants [16][21], and the continuous approach, where images are considered at video rate [1][19][49]. The method presented here is a continuous approach which stems from the interaction matrix related to the considered primitive. Hence, this reconstruction method is embedded into the framework presented in Section II.

As previously stated, a geometrical primitive is defined by an equation $h(\underline{x}, \underline{p}) = 0$. Using the relation between the time variation of $\underline{P}$ in the image sequence and the camera velocity T_c, we are able to compute the value of the parameters $\underline{p}$ of the considered primitive [15].

First, from the resolution of a linear system derived from relation (4), we obtain the parameters $\underline{p}_0$ which represent the position of the limb surface:

$$\underline{p}_0 = \underline{p}_0(T_c, \underline{P}, \dot{\underline{P}}) \tag{9}$$

Then, knowing the position of the primitive in the image described by (3) and using geometrical constraints related to the considered primitive, we can estimate the parameters $\underline{p}$ which fully define its 3D configuration:

$$\underline{p} = \underline{p}(\underline{P}, \underline{p}_0) \tag{10}$$

From a geometric point of view, this approach leads to determine the intersection between the limb surface and a generalized cone, defined by its vertex located at the optical center and by the image of the primitive (see Figure 6).

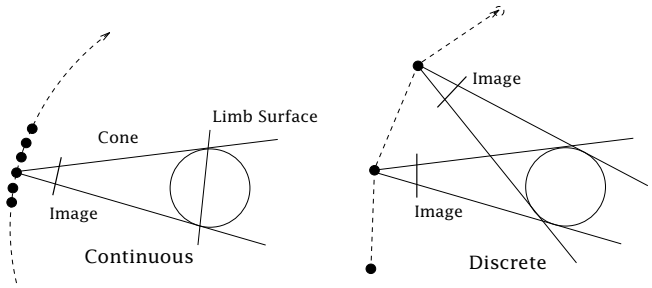


Fig. 6. Difference between continuous (on the left) and discrete (on the right) approaches for 3D structure estimation.

This approach has been applied to the most representative primitives (*i.e,* point, straight line, circle, sphere and cylin-

der) [15]. Let us note that in the case of a cylinder, this method can be applied using the projection of only one limb in the image (let us however note that more precise results are obtained using the projection of the two limbs in the image).

B. 3D Structure Estimation Using Active Vision

Important errors on the 3D structure estimation can be observed when no particular strategy concerning camera motion is defined. This is due to the fact that the quality of the estimation is very sensitive to the nature of the successive motions of the camera [19]. Active vision is thus necessary to improve the accuracy of the estimation results by generating adequate camera motions. Active vision is defined in [6] as an intelligent data acquisition process. Since the major shortcomings which limit the performance of vision systems are their sensitivity to noise and their low accuracy, the aim of active vision is generally to elaborate control strategies for adaptively setting camera parameters (position, velocity,...) in order to improve the knowledge of the environment. A theorical analysis of active vision proposed in [3] shows that ill-posed, non-linear and unstable problems for a passive observer are transformed into well-posed, linear, and stable ones for an active observer.

In our case, constraining camera motion can greatly improve the quality of the structure estimation. Indeed, it has been shown [15] that two vision-based tasks have to be realized in order to obtain a better robustness and accuracy in the results. First, a **fixating task** is required to obtain a non biased estimation of the 3D parameters. More precisely, the primitive must constantly appear at the same position in the image while the camera is moving. Futhermore, some positions of the primitive in the image do minimize the influence of the measurements errors. So, in order to obtain an *optimal estimation*, a **gaze control task** constrains the camera motion so that the object remains fixed at its specified position in the image. For example, a cylinder must appear vertical (or horizontal) and centered in the image [15].

A control law in closed-loop with respect to visual data is perfectly suitable to generate such motions. In the visual servoing framework presented in Section II, these tasks can be expressed as the regulation to zero of a primary task defined by $\underline{e}_1 = C(\underline{P} - \underline{P}_d)$ where $\underline{P}_d$ is the specified position of the primitive in the image, and where C is chosen as $C = L^T_{\underline{P}}(\underline{P}, \widehat{\underline{p}_0})$ (let us note that the interaction matrix is now updated at each iteration of the control loop using the measured value of $\underline{P}$ and the estimated value $\widehat{\underline{p}_0}$ of the parameters describing the limb surface). A trajectory tracking has also to be performed simultaneously in order to move the camera around the object of interest.

C. Parallel Dynamical Processes in SIGNAL

The described estimation scheme involves computations on the past values of signals, performed in parallel with the camera motion control. This introduces the need for constructs in the language enabling the expression of dynamical behaviors, as well as parallel ones. As mentioned in Section II-D, SIGNAL comprises constructs enabling this:

- *delay* on the values of a signal gives access to the past value of a signal. For example, equation $ZX_t = X_{t-1}$, with initial value V_0, defines a dynamic process which is encoded in SIGNAL by: `ZX := X$1` with initialization `ZX init V0`. Signals `X` and `ZX` have the same clock. Derived operators include delays on N instants (`$N`), and a `window M` operation giving access to a whole window in the past values (from times $t-M$ to t), as well as combinations of both operators.
For example, a filter defined by equation $y_t = (x_t + x_{t-1} + x_{t-2})/3$ is written in SIGNAL:
`(| Y := (X + ZX + ZZX)/3 | ZX := X$1 | ZZX := X$2 |).`
- *parallelism* between processes is obtained simply with the composition operator "`|`", which can be interpreted as parallelism with signals carrying instantaneous communication between processes.

D. Application to 3D Structure Estimation

Access to past values. As shown on equation (9), the 3D structure estimation method is based on the measure of $\dot{\underline{P}}$ which is computed using the current and the past values of the position of the primitive in the image (*i.e* $\underline{P}_t$ and $\underline{P}_{t-1}$). The velocity of the camera T_c between these two instants t and $t-1$ is also necessary for the estimation.

The past values of $\underline{P}$ and T_c can be easily expressed using the delay operator. If `P` is a signal carrying the position of the primitive in the image and `Tc` the velocity of the camera, the estimation `p` of the 3D primitive parameters $\underline{p}$ is expressed as in listing of the Figure 7 where `ZP := P$1` is the position of the primitive in the image at time $t-1$ and `ZTc := TC$1` is the camera velocity at time $t-1$. Thus, the language structures meet the data flow nature of the estimation algorithm which uses at each time t the value of parameters $\underline{P}$ and T_c at time t and $t-1$.

```
(| p := ESTIMATION{P,ZP,ZTc}
 | ZP := P$1
 | ZTc := Tc$1
 |)
```

Fig. 7. Using the DELAY operator for the estimation.

Besides, we smooth the output of this process, by computing the mean value of the current estimation and of the two previous ones. As already stated, such a filter is simply expressed in SIGNAL with the delay operator.

Parallelism. The estimation process is added to the control process of Section II in such a way that it is executed in parallel with the control law, as shown in Figure. 8. The interaction matrix is now computed for each new value provided by the measurement and the estimation processes. In the previous section (visual servoing without estimation), assumptions on the shape of the primitive had to be made; here, the geometrical structure of the primitive is estimated on-line. According to the synchrony hypothesis, the value at instant t of the interaction matrix is updated using the estimated parameters

and the current position of the primitive in the image at the same logical instant t.

D.0.a Results: Estimation of the Structure of a Cylinder. We here give the results obtained using the proposed 3D reconstruction method in the case of a cylinder. More details about this derivation can be found in [15]. As already stated, the cylinder must always appear centered and horizontal or vertical in the image sequence during the camera motion in order to obtain a non-biased and robust estimation. As in the previous section, the secondary task consists in a translational motion along the horizontal axis of the image plane ($V_x = 5cm/s$).

Figure 9 reports the error between the real value of the radius of the cylinder displayed in Figure 5.b and its estimated value. Let us note that the cylinder radius is determined with an accuracy less than 0.5 mm whereas the camera is one meter away from the cylinder (and even less than 0.1 mm with good lighting conditions).

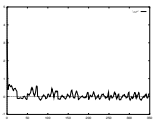


Fig. 9. Successive errors between the estimated and real values ($40mm$) of the cylinder radius displayed in Figure 5

IV. Hierarchies of Preemptive Tasks for Scene Reconstruction

We are now interested in investigating the problem of recovering a precise description of a 3D scene containing several objects. As already stated, the visual reconstruction scheme presented above involves to gaze on the considered primitive in the scene. This can be done on only one primitive at a time. Hence, reconstructions have to be performed in sequence. We present in this section the specification of such a sequencing which is stated in terms of a hierarchical parallel automaton [37].

The previous sections gave a framework for the specification and implementation in Signal of control tasks, and its application to vision-based tasks. Once a library of control tasks is available, the specification of higher-level and more complex behaviors requires the possibility to combine these tasks in various ways. Especially, one wants to combine them in sequence or in parallel, starting and interrupting them on the occurrence of events, that can be either external (coming from logical sensors) or internal. This level of robot programming necessitates preemption structures for concurrent tasks. The purpose of Signal*GTi* is precisely to augment Signal with objects and operations for the construction of such preemptive hierarchies of data flow tasks. Thus, preemption and sequencing of data flow tasks is handled in an extension to Signal using the notion of time interval. This enables the specification of hierarchical interruption structures, associating data flow computations to execution intervals, and making transitions from the one to the other in reaction to events.

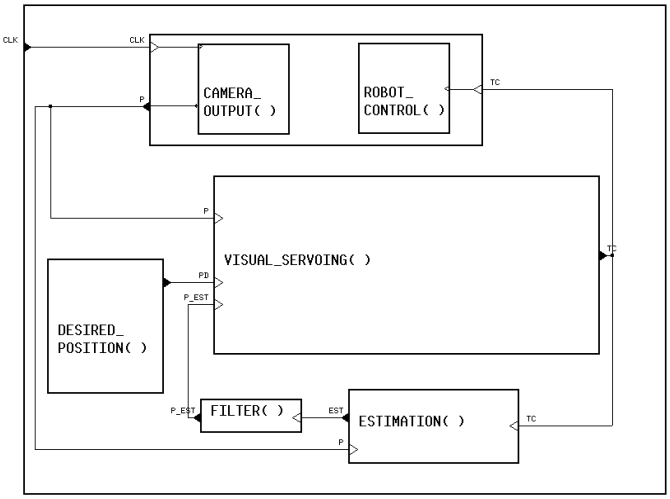
A. Vision Tasks for Complex Scene Reconstruction

In order to successively perform the estimation of the 3D structure of each primitive of the scene (assumed to be only composed of polyhedral objects and cylinders), a collection of 2D visual data (in fact a set of segments) is first created. With this collection of segments, a process selects a primitive, and after a recognition process which estimates the nature of the considered primitive (segment or cylinder), an optimal estimation of its 3D structure is performed. After the reconstruction of the selected primitive, the set of segments is updated, then, a new selection is done. The scene reconstruction process ends when the collection of segments is empty. We now detail the different steps involved in this strategy.

The first step in the whole scene reconstruction process is to build the set of segments (denoted Ω) visible in the image for the initial camera position. A weight is given to each element of Ω. This weight is function of the length and the position of the corresponding segment in the image in accordance with a given strategy. The segment with the highest weight is extracted from Ω, then, an optimal estimation based on this segment is performed.

Each segment is assumed to correspond to the projection in the image of either a limb of a cylinder, either a 3D segment. Since the structure estimation method is specific to each kind of primitives, a preliminary recognition process is required. To determine the nature of the observed primitive, we first assume that it is a cylinder, and a one limb-based estimation is performed. When this estimation is done, two competing hypotheses can be acting: the observed primitive is a straight line or the observed primitive is a cylinder. A maximum likelihood ratio test is used to determine which one of these two hypotheses is the right one [36].

If the primitive has been recognized as a cylinder, a two-limbs based estimation is used to provide a more robust and precise estimation. However, it is impossible, without *a priori*




```
(| TC := CONTROL{P,Pd,p}
 | ( | p := ESTIMATION{P,ZP,ZTC}
     | ZP := P$1
     | ZTc := Tc$1 |)
 |)
```

Fig. 8. Control and estimation in parallel.

knowledge on the structure of a cylinder, to determine the position in the image of its second limb. To solve this problem, we use the results obtained with the one limb-based estimation involved in the previous recognition process. These results are good enough to predict the position of the second limb in the image by projecting the 3D estimated cylinder in the image. Then, after a simple matching step, a robust estimation based on the two limbs can be performed.

Each optimal estimation ends when all the primitive parameters have been accurately computed with a sufficient precision. In parallel with an optimal estimation, we can also realize a coarse estimation of other primitives selected in Ω. It is coarse since the camera motion is not adequate for these primitives. Therefore, this leads to inaccurate results. The interest of this estimation is that it provides 3D information about the scene which can be used if the application does not necessitate a precise estimation for all the primitives present in the scene (such an application is for example the grasping of an object, whose 3D location has to be accurately determined, while avoiding obstacles, whose 3D exact location is generally less critical). Such coarse estimations end when the corresponding segments get out from the image or when the optimal estimation ends.

Furthermore, the optimal estimation described in Section III considers that the primitives have an infinite length. In order to determine their length, the vertices of the primitives have to be observed in the image, which generally implies a complementary camera motion. For accuracy issues, this motion is performed in the direction of the primitive axis, at a constant range, and until one of the two endpoints of the primitive appears at the image center. Once the camera has reached its desired position, the 3D position of the corresponding end point is computed as the intersection between the primitive axis and the camera optical axis. A motion in the opposite direction is then generated to determine the position of the other end point. Such a camera motion, based on visual data, is again performed using the visual servoing approach described in Section II.

A Hierarchical Parallel Automaton as Controller. This kind of complex robotics strategy involves the use of several subsystems (such as the different tasks described in the previous section). Achieving the complete operation requires a dynamic scheduling of these elementary subsystems. Furthermore, we want to specify combinations of tasks. For example, we want to combine the effects of several tasks executed in parallel (*e.g.,* a primary vision-based task combined with a trajectory tracking. Another example used here is the coarse estimation of some primitives performed in parallel with the optimal estimation of an other primitive).

We thus have developed a method for connecting up several estimations based on the definition of a hierarchical parallel automaton. This automaton is able to connect up the different stages of the reconstruction process: selection, visual servoing, optimal estimation of the selected primitive and concurrently, coarse estimations. Each state of our automaton is associated with a certain task such as the creation or the update of Ω, the structure estimation process, the camera motion control using visual servoing, etc (see Figure 10). The transitions between the states are discrete events and are function of the image data, the value of the estimated parameters of the primitives, and the state of Ω.

A framework to schedule such tasks and to program such hierarchical automata is now presented.

B. Preemption and Sequencing of Data Flow Tasks in SIGNAL

This section introduces recent extensions to SIGNAL, handling tasks execution over time intervals and their sequencing [42]. A data flow application is executed from an initial state of its memory at an initial instant α, which is before the first event of the reactive execution. A data flow process

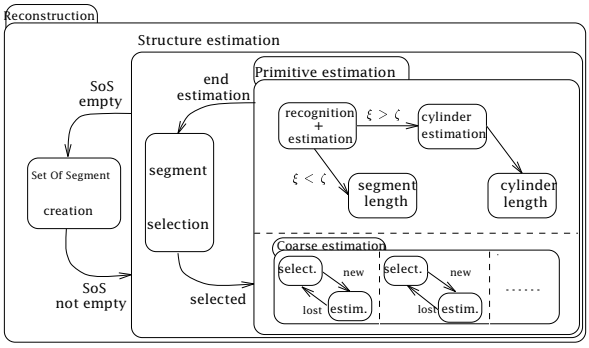


Fig. 10. Hierarchical parallel automaton for the application.

has no termination specified in itself; therefore its end at an instant ω can only be decided in reaction to external events or the reaching of given values. Hence, ω is part of the execution, and the time interval on which the application executes is the left-open, right-closed interval $]\alpha,\omega]$.

Time intervals. They are introduced in order to enable the structured decomposition of the interval $]\alpha,\omega]$ into sub-intervals as illustrated in Figure 11, and their association with processes [42]. A sub-interval `I` is delimited by occurrences of bounding events at the beginning `B` and end `E`: `I := ]B, E]`. It has the value `inside` between the next occurrence of `B` and the next occurrence of `E`, and `outside` otherwise.

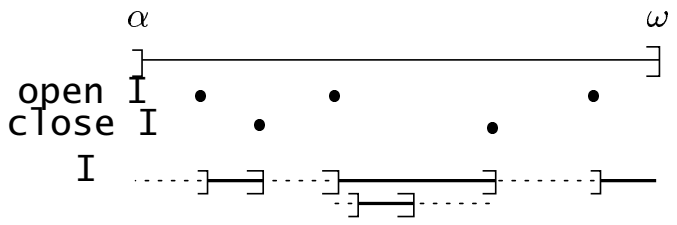


Fig. 11. Time intervals sub-dividing $]\alpha,\omega]$.

Like $]\alpha,\omega]$, sub-intervals are left-open and right-closed. This choice is coherent with the behavior expected from reactive automata: a transition is made according to a received event occurrence and a current state, which results in a new state. Hence, the instant where the event occurs belongs to the time interval of the current state, not to that of the new state.

The operator `compl I` defines the complement of an interval `I`, which is `inside` when `I` is `outside` and conversely. Operators `open I` and `close I` respectively give the opening and closing occurrences of the bounding events. Occurrences of a signal `X` inside interval `I` can be selected by `X in I`, and reciprocally outside by `X out I`. In this framework, `open I` is `B out I`, and `close I` is `E in I`.

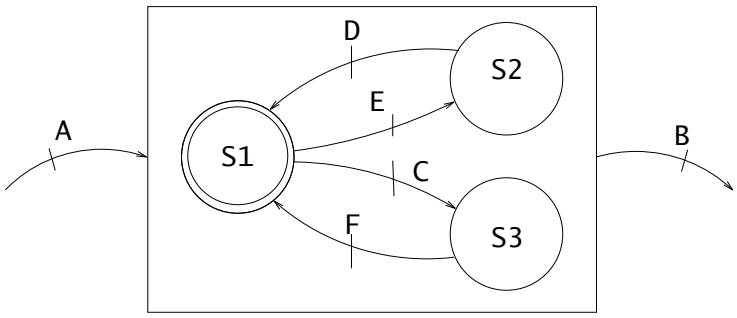
Tasks. They consist in associating a given (sub)process of the application with a given (sub)interval of $]\alpha,\omega]$ on which it is executed. Tasks which are active on $]\alpha,\omega]$ represent the default case: they are *remanent* throughout the whole application. Inside the task interval, the task process is active *i.e.*, present and executing normally. Outside the interval, the process is inexistent *i.e.*, absent and the values it keeps in its internal state are unavailable. In some sense, it is out of time, its clock being cut.

Tasks are defined by the process `P` to be executed, the execution interval `I`, and the starting state (current, or initial) when (re-)entering the interval. More precisely, the latter means that, when re-entering the task interval, the process can be re-started at its current state at the instant where the task was *suspended* (*i.e.,* in a temporary fashion); this is written in SIGNAL `P on I`. Alternatively, it can be re-started at its initial state, as defined by the declaration of all its state variables, if the task was *interrupted* (*i.e.*, aborted in a definitive fashion); this is written `P each I`. The processes associated with intervals can themselves be decomposed into sub-tasks. Hence, the specification of *hierarchies* of complex behaviors is possible.

Task control. Task control is achieved as a result of constraining intervals and their bounding events, and associating activities to them. *Parallelism* between several tasks is obtained naturally when tasks share the same interval, or overlapping intervals. *Sequencing tasks* then amounts to constraining the intervals of the tasks. Using `on` and `each`, as defined above, enables to control activities, and more elaborate behaviors can be specified. Hence, it is possible to specify hierarchical parallel automata or place/transition systems.

Each time interval holds some state information, and events cause transitions between these states. In the simple behavior illustrated in Figure 12.a, a transition leads from state `S1` to state `S2` on the occurrence of an event `E`, except if the event `C` occurs before, leading in place `S3`. If `E` and `C` append synchronously or are constrained to be equal, then both places `S2` and `S3` are entered. This is a sub-behavior attached to a place entered upon event `A` and left upon event `B`. This can be coded by intervals such that the closing of the one is the opening of the other, as in Figure 12.b.

An encoding of intervals and tasks into the SIGNAL kernel is implemented as a pre-processor to the SIGNAL compiler, called SIGNAL*GTi*. Data flow and sequencing aspects are both in the *same language* framework, thus relying on the *same model* for their execution and the verification of correctness of programs. From the point of view of synchronous programming languages, an approach related to our integration of data-flow and sequencing is ARGOLUS [35]. It integrates ARGOS (hierarchical parallel automata) with LUSTRE (data flow). However ARGOLUS focuses on semantics issues and has not been applied to control systems. In our case, we try to specify sequencing in a more declarative style. A general study of preemption and concurrency, lead in combination with the imperative synchronous language ESTEREL [12], resulted in the possibility to control the starting, suspension, resuming and termination of external tasks [11]. However, the fact that these tasks can not be defined within the same language framework limits the control on interactions between different levels.



(a) Hierarhical place/transition system

```
(| S1 := ](D in S2) default (F in S3),
            E default C] init inside
 | S2 := ]E in S1, D] init outside
 | S3 := ]C in S1, F] init outside
 |) each ]A, B] init outside
```

(b) specification in SIGNAL*GTi*

Fig. 12. Sequencing, concurrency and nested preemption in SIGNAL*GTi*.

C. Application to the Reconstruction Strategy

The visual reconstruction process based on the hierarchical parallel automaton proposed in Section IV-A has been specified using the general notion of tasks and time intervals defined above. This automaton is able to sequence the different vision tasks (selection, gaze control, optimal estimation of the selected primitive and concurrently, coarse estimation of the other ones) and to provide a robust estimation of the spatial organization of the scene [36].

Once the automaton specification is completed, programming such an automaton is quite easy using the presented tasks sequencing framework. The source code in SIGNAL of the application is very close to the specification because programming is performed via the specification of constraints or relations between all the involved signals. At this step of the description, we show in Figure 13.a a part of the SIGNAL code to illustrate the encoding of such a hierarchical parallel automaton with our approach based on time interval description.

Figure 13.b illustrates the specification of the behavior of the system using a possible execution trace. An `SoS_creation` phase creates Ω; it is active during the time interval `IE`. In alternance with this (*i.e.,* when not in `IE`), the `Structure_estimation` process is active on `IREC`. It is itself an alternance between a *primitive selection* process, on interval `IC`, and the *primitive reconstruction* on `IR`. The selection chooses a primitive to be estimated in the set of segments Ω. If Ω is empty (*i.e.*, `SoSempty=true`), it causes the *structure estimation* (`IREC`) to exit. When Ω is not empty, the primitive reconstruction process for the chosen primitive is itself decomposed into sub-activities. It begins on `Inat` with a *recognition process* which estimates the *nature* of the considered primitive (segment or cylinder) and ends with the boolean event `Cylinder`. It continues with an optimal estimation of the different parameters of its 3D structure, according to the value of `Cylinder`: in the case of a segment (`Cylinder=false`), only its length on interval `Isl` is determined; in the case of a cylinder (`Cylinder=true`), a two-limbs based estimation is performed on `Icv`, and then the length of the cylinder is estimated on `Icl`. In parallel with the optimal estimation, a coarse estimation of some primitives can be performed on intervals $\mathtt{Ir}_i$. After each estimation of a primitive, Ω is updated and a new selection is performed on `IC`.

The two following advantages of using SIGNAL for our application can be emphasized:

Termination. A data-flow process defines, like our vision tasks, a behavior, but not a termination : this aspect must be defined separately. One way of deciding on termination of a task is to apply criteria for reaching a goal: when a certain value $\underline{P}_d$ is acquired by the sensor, the task is considered to have reached its goal, hence it ends. Let us consider the case of the length computation. We have a trajectory tracking task, and the goal is defined by a certain position of the endpoint of the primitive in the image (it must appear in the center of the image). In fact, we have to minimize the error $(\underline{P} - \underline{P}_d)$ between the current position of the endpoint and its desired position. The goal is reached with a precision ε when condition $||\underline{P} - \underline{P}_d|| \leq \varepsilon$ is satisfied. The evaluation of this condition must be performed at all instants: hence, this evaluation is another data flow treatment. The instant when the condition is satisfied can be marked by a discrete event, which, causing termination of the task, can also cause a transition to another task at a higher level of the reactive sequencing. In this sense, this event can be used to specify the end of the execution interval of the task. Evaluation of such conditions can be made following a dynamic evolution: a sequence of modes of evaluation of $\underline{P}$, or of the criterion, can be defined, becoming finer (and possibly more costly) when close to interesting or important values.

Parallelism. Parallelism between two tasks is transparent to the programmer using the composition operator. This is the case, for example, of the coarse estimation processes and the optimal estimation process. To perform these estimations, they both use the same information (*i.e.*, the measure of camera velocity, the measures performed in the image at current and previous instants), in such a way, according to the synchronous hypothesis, that they can use it at the same logic instant. In fact, we have here a parallelism of specification, and the compiler monitors all the synchronization and com-

Application:
(| IE :=] when SoSempty, when not SoSempty] init inside
| IREC := comp IE
| SoS_creation each IE
| Structure_estimation each IREC
|)

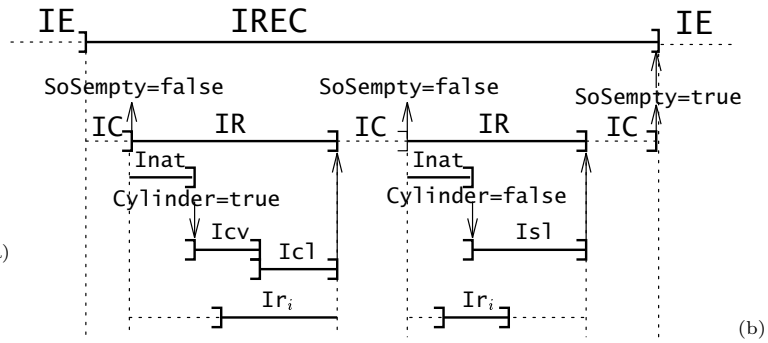
Structure_estimation:
(| IC :=] close Icl default close Isl, SoSempty] init inside
| IR := comp IC
| Choice each IC
| Primitive_estimation each IR
|)

Primitive_estimation:
(| Optimal_estimation
|(| Coarse_estimation$_1$ | ... | Coarse_estimation$_n$ |)
|)

Coarse_estimation$_i$:
(| Ir$_i$:=] New_Segment, Segment_Lost] init outside
| Coarse_est each Ir$_i$ |)

Optimal estimation:
(| Inat :=] close Isl default close Icl, Cyl] init inside
| Nature each Inat
| Icv :=] when Cyl, when (| $prec$ |< ε_{cv})] init outside
| Cylinder_vertex each Icv
| Icl :=] close Icv, when (| $prec$ |< ε_{cl})] init outside
| Cylinder_length each Icl
| Isl :=] when not Cyl, when (| $prec$ |< ε_{sl})] init outside
| Segment_length each Isl |)

(a)



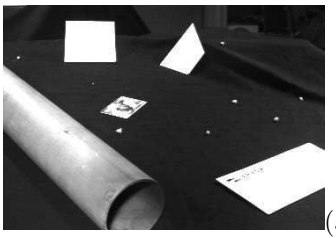
(b)

Fig. 13. Specification of the hierarchy of tasks in SIGNAL*GTi* (a) and Specification of the sequencing in terms of activity intervals: a possible trace (b).

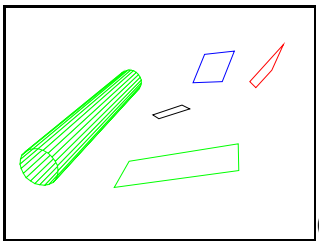
munication problems.

D. Complex Scene Reconstruction

The example reported here (see Figure 14) deals with a scene composed of a cylinder and five polygons which lie in different planes. In Figure 14.a is displayed the initial image acquired by the camera. Figure 14.b represents a view of the 3D reconstructed scene.



(a)



(b)

Fig. 14. View of the scene (a) and 3D model of the reconstructed scene (b)

Figure 15 shows a graphical view of the reconstruction environment built with OSF/Motif. The management of this graphical interface is programmed with SIGNAL; only X11 functions are defined and called as external functions. From the bottom to the top of the environment are represented: the current state of the different time intervals (i.e., activity of tasks), the evolution of the parameters describing the selected primitive, the error between the current and the desired position of the primitive in the image, a symbolic representation of the automaton, the image with the set of segments Ω superimposed on it and the current representation of the reconstructed scene.

V. Related Work and Discussion

A. Design of Reactive Systems: Synchronous and Asynchronous Approach

A review of the techniques classically used for real-time programming is given in [10]. The most common time model for concurrent programming is asynchrony . Two main approaches raised: one is based on formal methods (such as finite state automata or Petri Nets), the other is based on tools able to express concurrency (real time operating systems, or concurrent programming languages [7][29][39]). Let us examine first the transition based systems. The *finite state automata* are well known tools, deterministic, efficient and they allow verification capabilities. However, the composition of little automata can yield to a very big one often impossible to understand. Furthermore a little change in the specification provokes a complete transformation of the automaton. Finally, let us point out that parallelism and preemption of tasks are not supported by this object. The *Petri Nets* are often used for small applications and, if they support concurrency, they do not support hierarchical design, and their determinism is not clear. The second approach is based on the expression of the concurrency. *Concurrent Programming Languages* such as OCCAM (CSP [29]) or ADA [7] have numerous advantages. They are well structured and allow a good modularity. But they are asynchronous and thus non deterministic. The synchronization between processes is performed during the execution and is unpredictable. Thus, they can hardly be used for reactive systems implementation. The most classical way for real time systems integration is the connection of classical programs using *real-time Operating System (OS)* primitives. Here, the main problem is that there is a set of programs to study and connect. Diagnostic and maintenance is hard,

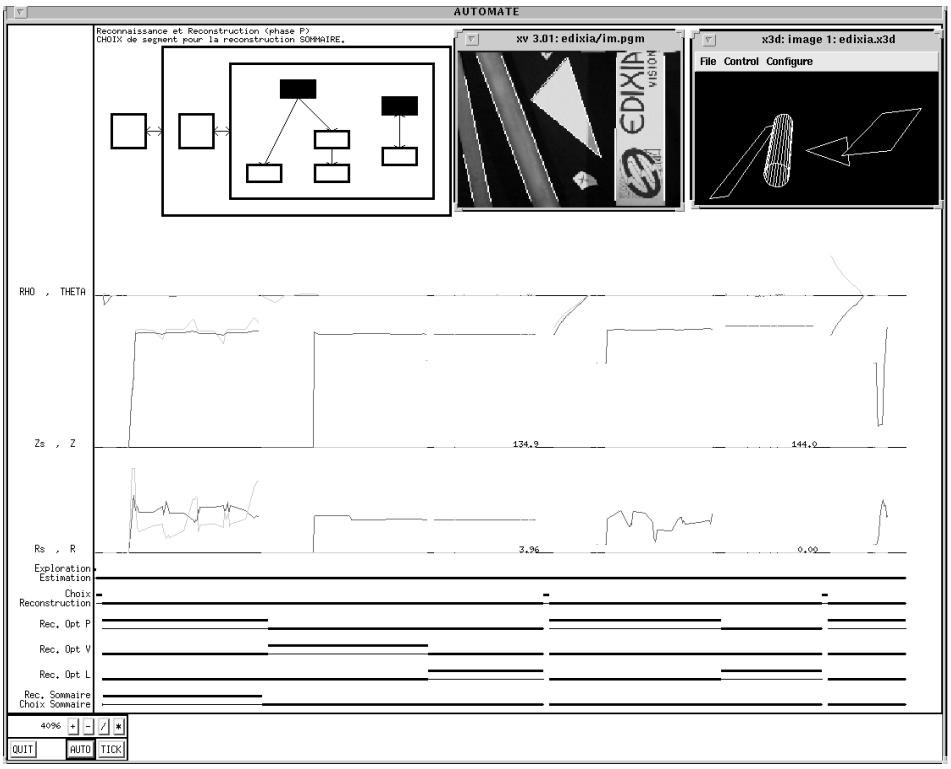


Fig. 15. The synchronous environment for 3D scene reconstruction under the OSF / Motif window management system

temporal constraints are not expressed in the programs description, but are satisfied using the OS primitives for process synchronization/communication. This leads to systems which are generally non deterministic [39], and on which no safety properties can be formally guaranteed.

On the other side, we found the family of synchronous languages[8][10][24]. In [34], a number of formal specifications, implementations and verification of a reactive system are proposed. This book proposes a comparative survey of various languages (among which synchronous languages such as SIGNAL [5] but also LUSTRE, ESTEREL, STATECHARTS, but also asynchronous languages such as ADA,...) used to specify, verify and implement a controller for a robotic production cell. The main drawbacks raised by a classical asynchronous implementation of reactive systems can be avoided using this class of languages. Resulting from the synchrony hypothesis these languages are deterministic, they allow concurrency and hierarchical specification. They provide safety, logical correctness (respect of the input/output specification), and temporal correctness. Furthermore, they are based on a mathematically well defined semantics, thus they support verification tools. For example, the compilation of SIGNAL code provides a graph on which static correctness proofs can be derived. It can also produce an equivalent dynamical equation system, on which dynamical properties can be proved.

B. Control of Visual processes

Concerning the specification and implementation of vision systems, most of them result from the integration of various programs running on different boards connected by a real time OS (such as VX WORKS). Sequencing is usually performed using finite state automata or Petri Nets. Therefore the integration of such systems is rarely described (nevertheless, a rapid survey of these methods, is given in [48] in the case of real time visual inspection). More formal approaches, such as the Discrete Event Systems (DES) formalism [41], appear recently in the literature concerning computer vision [4][46]. Modeling of visual behaviors of agents engaged in navigational tasks has been proposed in [31]. The use of DES formalism allows to synthesize complex behaviors. However, only the supervisor design is done using this formalism and the design of control task is not taken into account. Dealing with the task level (image processing oriented), an interesting approach has been proposed in [44], dealing on a functional data-flow implementation of vision algorithms.

More recently, the approach of applying synchronous languages has been adopted in the ORCCAD environment [45][17]. This approach aims at a complete design environment for robot programming (the application domain of ORCCAD is not limited to robot vision), and has more general goals than ours. Each action is modeled with a functional aspect and a behavior. The central object of this approach, called *Robot-Task*, is a set of control loops (*Module tasks*) associated with a local behavior. Around the *Robot task*, the ap-

plication level consists in scheduling several actions involved in the subsystems. ESTEREL [12] has been used to program the application level and local behavior of each *Robot task*. This allows to derive a single automaton for the whole application which can be used for correctness checking. However, if task-level programming is done using ESTEREL, the data-flow specification of control loops is done using another formalism at a different level, and controlled as an external task. We think that these *Module Tasks* could be easily implemented with SIGNAL (most of the tasks presented in [45][17] use the task function approach [43]), and can take advantage in using a data-flow language. In that case, interactions between the two languages might be supported by a set of exchange formats, currently being designed.

C. Discussion

First of all, we must recognize that a data-flow language such as SIGNAL is not adapted to all kinds of computation. For example, image processing or linear algebra cannot generally be performed with SIGNAL (or with difficulty, see [33] for image processing). Arrays are available in SIGNAL, but the algorithms involved for this kind of computation (for example the inverse of a matrix) are not data-flow. Thus we use external functions written with other languages (C and Fortran), which are called from the SIGNAL program. The same method is used in our application for the management of the set of segments which is obviously not the application domain of synchronous languages. However, the use of such functions is not performed asynchronously: they are considered as any function defined in SIGNAL, thus we do not leave the synchronous framework. Furthermore, the management of asynchronous inputs or interruptions is not supported. However, this is not necessary in this kind of application where the inputs are provided regularly and periodically (here at video rate). Finally, dynamical management of time at the execution is not treated here, but this is not necessary due to the regular aspect of the loops.

Let us now emphasize the merits of synchronous languages, and more particularly SIGNAL, for this kind of applications. Dealing with implementation issues, advantages can be found at both control and task level. The data flow framework is particularly appropriate for the specification of visual servoing because of the equational and data flow nature of the closed-loop control laws, which can be implemented as control functions between sensor data and control outputs. The possibility of implicitly specifying parallel behaviors has been proved useful for the 3D structure estimation using active vision. The synchrony hypothesis corresponds well to the model of time in the equations defining the control laws. The second point concerns tasks sequencing and preempting. The language-level integration of the data flow and sequencing frameworks have been achieved as an extension of SIGNAL: SIGNAL*GTi*. It enables the design of time intervals, their association with data flow processes and provides constructs for the specification of *hierarchical preemptive tasks*. This way, it offers a multi-paradigm language combining the data flow and multi-tasking paradigms, for hybrid applications blending (sampled) continuous and discrete transition aspects. Using SIGNAL*GTi*, we can design a *hierarchy* of parallel automata, thus we have the advantages of both the automata (determinism, tasks sequencing) and concurrent programming languages (parallelism between tasks) without their drawbacks. Note that SIGNAL*GTi* has not been developed only for the application presented in this paper, but for the specification of other complex applications (such as behavioral animation in computer graphics [18] or the design of a transformer power station [38]).

The semantics of SIGNAL is also defined via a mathematical model of multiple clocked flows of data and events. SIGNAL programs describe relations on such objects, thus programming is done via constraints. The compiler calculates the solutions of the system and may thus be used as a proof system. Thus, its programming environment, which is not limited to the compiler, features tools for the automated analysis of formal properties. The compilation of SIGNAL code provides a dependencies graph on which static correctness proofs can be derived: SIGNAL checks *automatically* the network of dependencies between datas and detects causal cycles, temporal inconsistencies from the point of view of time indexes. SIGNAL synthesizes *automatically* the scheduling of the operations involved inside a control-loop (note that this work is an error-prone task when done by hand in classical C-like languages), and this scheduling is proved to be *correct* from the point of view of data dependencies. Furthermore, the SIGNAL-code is thus easy to modify since the re-synthesis is automatic. Finally, the compiler synthesizes *automatically* a global optimization of the dependencies graph.

The SIGNAL environment provides other tools (note that they have not been used directly in our application): SIGNAL can also produce an equivalent dynamical equation system, on which dynamical properties can be proved. The absence of deadlocks (liveness), reachability of states (or on the contrary non-reachability of a "bad" state), or properties specific to the application can thus be checked. These properties (both static and dynamic) checking tools are important at two levels: for development purposes it is important to verify that the systems really has the expected or required behavior ; and for the certification of the safety of the systems, which is meaningful regarding safety-critical application.

Work is going on concerning the distribution of SIGNAL programs on parallel machines, with automatic generation of separate code modules and of their communications. Finally, the compilation of SIGNAL into VHDL opens the ways towards hardware/software co-design. We claim that this tools are very important in the context of software engineering and that other classical asynchronous languages do not provide them (other synchronous languages like ESTEREL or LUSTRE provide this kind of tools).

As a conclusion, the contribution of the synchronous approach, and of SIGNAL in particular is that it has a programming style closer to control engineer's specification and that it

provides him with a set of tools releaving him from error-prone works. Even if some other languages are sometimes provided with interesting complementary functionalities (management of the duration of tasks, dynamic scheduling, ...), they do not offer such tools based on formal model.

VI. Conclusion

The goal of this paper was to show that synchronous languages are suitable to specify and to implement control applications and that they are better suited than classical languages. We have chosen the example of an active vision system which has to manage vision tasks at different levels: camera motion control (vision-based closed-loop control), estimation algorithms (3D structure estimation from controlled motion), and perception strategies (vision tasks sequencing). The whole application can be specified in SIGNAL, from the discrete event driven sequencing down to the servoing loop. Experiments have been carried out on a robotic cell, with real data acquired by a moving camera.

We have proved that it is feasible to program active vision applications with the synchronous language SIGNAL. It is, of course, also possible with other more classical languages and SIGNAL cannot perform (or with difficulty) every kind of computation. However, what is done using SIGNAL (specification of computations, parallel composition of processes, multi-tasking, and executable code generation) is done automatically and correctly. In conclusion, we argue that a synchronous data-flow approach is advantageous for three main reasons:

- based on the formal model of the languages, processing and analysis of the programs are automated and implemented in a set of tools ;
- the programming style we have proposed for this kind of application is independent from the sequential aspect of the computer architecture and thus very close to the original specification of the control theorist;
- it allows to consider in an **unified framework** the various aspects of the application: from data-flow task specification to multi-tasking and hierarchical task preemption (this latter point had not been previously demonstrated by a real size application).

It would be also interesting to consider a generalization of the structure of data-flow tasks proposed in this paper, towards a programming environment dedicated to the design of sensor-based control tasks following the task-function approach, as presented in the perspectives of [45].

Finally, the application presented here deals with structure estimation. However, the methodology presented here can be applied to other kind of application. For example, target tracking by visual servoing is possible. The implementation of a Kalman filter used to estimate the target motion as proposed in [14] is something easy to do with SIGNAL. Schematically, the integration of a filter process in the visual servoing program is very similar to the integration of the structure estimation process. The estimation of the target motion can be performed in parallel with the main visual servoing process and access to past values in the filter will be performed using the delay instruction.

Acknowledgment

This work was partly supported by the MESR (French Ministry of the University and Research) within project VIA (*Vision Intentionnelle et Action*) and under contribution to a student grant. The authors thank Samuel Ketels and Florent Martinez who have implemented in part some of the experimental work reported here.

References

[1] G. Adiv, "Inherent ambiguities in recovering 3D motion and structure from a noisy flow field," *IEEE Trans. on on Pattern Analysis and Machine Intelligence*, vol. 11, no. 5, pp. 477–489, May 1989.

[2] P.K. Allen, A. Timcenko, B. Yoshimi, and P. Michelman, "Automated tracking and grasping of a moving object with a robotic hand-eye system," *IEEE Trans. on Robotics and Automation*, vol. 9, no. 2, pp. 152–165, April 1993.

[3] Y. Aloimonos, I. Weiss, and A. Bandopadhay, "Active vision," *Int. Journal of Computer Vision*, vol. 1, no. 4, pp. 333–356, January 1987.

[4] Y. Aloimonos, E. Rivlin, and L. Huang, *Designing Visual Systems: Purposive Navigation,* In Active Perception*, Aloimonos Y. Editor*, pp. 47–102, Lawrence Erlbaum Assoc., publishers, Hillsdale, NJ, 1993.

[5] T. Amagbegnon, P. Le Guernic, H. Marchand, and E. Rutten, "SIGNAL – the specification of a generic, verified production cell controller," in *Formal Development of Reactive Systems - Case Study Production Cell*, C. Lewerentz and T. Lindner, Eds. Lecture Notes in Computer Science, 891, Springer-Verlag, 1995.

[6] R. Bajcsy, "Active perception," *Proceedings of the IEEE*, vol. 76, no. 8, pp. 996–1005, August 1988.

[7] T.P. Baker and O. Pazy, "Real time features for ADA 9x," in *IEEE Real-Time Systems Symposium*, December 1991, pp. 172–180.

[8] A. Benveniste and G. Berry, "Real-time systems designs and programming," *Proceedings of the IEEE*, vol. 79, no. 9, pp. 1270–1282, September 1991.

[9] A. Benveniste, "Synchronous languages provide safety in reactive systems design," *Control Engineering*, pp. 87–89, September 1994.

[10] G. Berry, "Real time programming: Special purpose languages or general purpose languages," in *11th IFIP World Congress*, San Francisco, California, August 1989, pp. 11–17.

[11] G. Berry, "Preemption in concurrent systems," in *13th Conf. on Foundations of Software Technology and Theoretical Computer Science*. LNCS 761, Springer Verlag, Bombay, India, December 1993.

[12] F. Boussinot and R. De Simone, "The ESTEREL language," *Proceedings of the IEEE*, vol. 79, no. 9, pp. 1293–1304, September 1991.

[13] A. Castaño and S. Hutchinson, "Visual compliance: Task-directed visual servo control," *IEEE Trans. on Robotics and Automation*, vol. 10, no. 3, pp. 334–342, June 1994.

[14] F. Chaumette and A. Santos, "Tracking a moving object by visual servoing," in *Proc. of 12th IFAC World Congress*, Sidney, Autralia, July 1993, pp. 643–648.

[15] F. Chaumette, S. Boukir, P. Bouthemy, and D. Juvin, "Structure from controlled motion," *IEEE Trans. on on Pattern Analysis and Machine Intelligence*, vol. 18, no. 5, pp. 492–504, May 1996.

[16] C. Chien and J.K. Aggarwal, "Model construction and shape recognition from occluding contour," *IEEE Trans. on on Pattern Analysis and Machine Intelligence*, vol. 11, no. 4, pp. 372–389, February 1989.

[17] E. Coste-Manière, B. Espiau, and D. Simon, "Reactive objects in a task level open controller," in *IEEE Int. Conf. on Robotics and Automation*, Nice, France, May 1992, vol. 3, pp. 2732–2737.

[18] S. Donikian and E. Rutten, "Reactivity, concurrency, data-flow and hierarchical preemption for behavioral animation," in *Eurographics Workshop on Programming Paradigms in Graphics*, Maastricht, The Netherlands, September 1995.

[19] B. Espiau and P. Rives, "Closed-loop recursive estimation of 3D features for a mobile vision system," in *IEEE Int. Conf. on*

Function	`Y := f{ X1, X2,..., Xn}`	instantaneous transformations on the data
Selection	`Y := X when B`	selection according to a boolean condition
Deterministic merge	`Z := X default Y`	defines the union of two signals
Delay	`ZX := X$1`	defines a dynamic process ($zx_t = x_{t-1}$)
Composition	`(| P1 | P2 |)`	union of the underlying systems of equations can be interpreted as parallelism between processes
Time Interval	`I = ]A,B]`	time interval I delimited by occurence of events `A` and `B`
Complement	`compl I`	complement of an interval `I`
Opening	`open I`	opening occurence of the bouding event
Closing	`close I`	closing occurence of the bouding event
Selection	`X in I`	occurence of a signal `X` inside interval `I`
	`X out I`	occurence of a signal `X` outside interval `I`
Task Control	`P each I`	`P` is executed while `I` (re-started at its current state)
	`P on I`	`P` is executed while `I` (re-started at its initial state)

TABLE I

QUICK SIGNAL AND SIGNAL*GTi* REFERENCE GUIDE

Robotics and Automation, Raleigh, North Carolina, April 1987, vol. 3, pp. 1436–1443.

[20] B. Espiau, F. Chaumette, and P. Rives, "A new approach to visual servoing in robotics," *IEEE Trans. on Robotics and Automation*, vol. 8, no. 3, pp. 313–326, June 1992.

[21] O. Faugeras, *Three-dimensionnal computer vision: a geometric viewpoint*, MIT press, 1993.

[22] J.T Feddema, C.S.G. Lee, and O.R. Mitchell, "Weighted selection of image features for resolved rate visual feedback control," *IEEE Trans. on Robotics and Automation*, vol. 7, no. 1, pp. 31–47, February 1991.

[23] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, "The synchronous data flow programming language LUSTRE," *Proceedings of the IEEE*, vol. 79, no. 9, pp. 1305–1320, September 1991.

[24] N. Halbwachs, *Synchronous programming of reactive systems*, Kluwer, 1993.

[25] D. Harel and A. Pnueli, "On the development of reactive systems," in *Logics and Models of Concurrent Systems*, K.R. Apt, Ed., New York, 1985, vol. 13 of *NATO ASI Series, Springer Verlag*, pp. 477–498.

[26] D. Harel, "STATECHARTS: a visual formalism for complex systems," *Science of Computer Programming*, vol. 8, no. 3, pp. 231–274, June 1987.

[27] K. Hashimoto, T. Kimoto, T. Ebine, and H. Kimura, "Manipulator control with image-based visual servo," in *IEEE Int. Conf. on Robotics and Automation*, Sacramento, California, April 1991, vol. 3, pp. 2267–2271.

[28] K. Hashimoto Editor, *Visual Servoing : Real Time Control of Robot manipulators based on visual sensory feedback*, World Scientific Series in Robotics and Automated Systems, Vol 7, World Scientific Press, Singapore, 1993.

[29] C.A.R. Hoare, *Communicating Sequential Process*, Prentise-Hall Int., 1985.

[30] A.E. Hunt and A.C. Sanderson, "Vision-based predictive robotic tracking of a moving object," Tech. Rep. CMU-RI-TR-82-15, Carnegie-Mellon University, January 1982.

[31] J. Košeká, H. Christensen, and R. Bajcsy, "Discrete event modeling of visually guided behaviors," *Int. Journal of Computer Vision*, vol. 14, no. 2, pp. 179–191, March 1995.

[32] P. Le Guernic, M. Le Borgne, T. Gautier, and C. Le Maire, "Programming real time application with SIGNAL," *Proceedings of the IEEE*, vol. 79, no. 9, pp. 1321–1336, September 1991.

[33] P. Le Guernic, "The SIGNAL programming environment," *Algorithms and Parallel VLSI Architecture II,* Elsevier Science Publishers, pp. 347–358, September 1992.

[34] C. Lewerentz and T. Lindner, *Formal Development of reactive systems*, vol. 891 of *Lecture Notes in Computer Science*, Springer Verlag, 1995.

[35] M. Jourdan, F. Lagnier, F. Maraninchi, and F. Raymond, "A multiparadigm language for reactive systems," in *IEEE Int. Conf. on Computer Languages*, Toulouse, France, May 1994.

[36] E. Marchand and F. Chaumette, "Active visual 3D perception," in *IEEE Int. Workshop on Vision for Robots*, Pittsburgh, USA, August 1995, pp. 10–17.

[37] E. Marchand, F. Chaumette, and E. Rutten, "Real time active visual reconstruction using the synchronous paradigm," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS'95*, Pittsburgh, USA, August 1995, vol. 1, pp. 96–102.

[38] H. Marchand, E. Rutten, and M. Samaan, "Synchronous design of a transformer station controller in SIGNAL," in *Proc of 4th IEEE Conf. on Control Applications*, Albany, New York, USA, September 1995.

[39] R. Milner, *A Calculus of Communicating Systems*, Number 92 in Lecture Notes in Computer Science. Springer Verlag, 1980.

[40] N. Papanikolopoulos, B. Nelson, and P.K. Khosla, "Six degree-of-freedom hand/eye visual tracking with uncertain parameters," *IEEE Trans. on Robotics and Automation*, vol. 11, no. 5, pp. 725–732, October 1995.

[41] P.J. Ramadge and W.M. Wonham, "The control of discrete events systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–97, January 1989.

[42] E. Rutten and P. Le Guernic, "Sequencing of data flow tasks in SIGNAL," in *ACM SIGPLAN Workshop on Language, Compiler and Tool Support for Real-Time Systems*, Orlando, Florida, June 1994.

[43] C. Samson, B. Espiau, and M. Le Borgne, *Robot Control: the Task Function Approach*, Clarendon Press, Oxford, United Kingdom, 1991.

[44] J. Sérot, G. Quénot, and B. Zavidovique, "Functionnal programming on a dataflow architecture applications in real-time image processing," *Machine Vision and Application*, vol. 7, no. 1, pp. 44–56, Winter 1993.

[45] D. Simon, B. Espiau, E. Castillo, and K. Kapellos, "Computer-aided design of a generic robot controller handling reactivity and real-time controller issues," *IEEE Trans. on Control Systems Technology*, vol. 1, no. 4, pp. 213–229, December 1993.

[46] M.T. Sobh and R. Bajcsy, "Visual observation under uncertainty as a discrete event processs," in *IAPR Int. Conf. on Pattern Recognition*, The Hague, The Netherlands, August 1992, pp. 429–432.

[47] Y. Sorel, "Massively parallel computing systems with real-time constraints: the algorithm-architecture adequation methodology," in *Massively Parallel Computing systems Conference*, 1994.

[48] A.D.H. Thomas, M.G. Rodd, J.D. Holt, and C.J. Neill, "Real time industrial visual inspection: A review," *Real Time Imaging*, vol. 1, no. 2, pp. 139–158, June 1995.

[49] A.M. Waxman, B.K. Parsi, and M. Subbarao, "Closed-form solutions to image flow equations for 3D structure and motion," *Int. Journal of Computer Vision*, vol. 1, no. 3, pp. 239–258, October 1987.

[50] L.E. Weiss, A.C. Sanderson, and C.P. Neuman, "Dynamic sensor-based control of robots with visual feedback," *IEEE Journal of Robotics and Automation*, vol. 3, no. 5, pp. 404–417, October 1987.

ric Marchand was born in Rennes, France in 1970. He graduated in Computer Science from IFSIC (Universit de Rennes 1) in 1993. He received a Ph.D in Computer Science from the University of Rennes 1 in June 1996. He is currently in Postdoc at Yale University. His research interests include robotics, perception strategies and especially the cooperation between perception and action. The considered applications are 3D reconstruction and multi-sensors cooperation. He is also interested in high-level languages for reactive and real-time systems programming.

ric Rutten graduated in Computer Science in 1987 from Institut National des Sciences Appliques and University of Rennes 1, France. He received a Ph.D in Computer Science from the University of Rennes 1 in 1990. The area of his doctoral research was task-level languages for robot programming. Since 1992, he holds an INRIA research position at IRISA in Rennes, in the Real-Time Programming team, where the SIGNAL language is being designed. His research interests extend obliquely from high-level languages for robot programming to models for the design of programming environments. They are currently in the synchronous approach to the design of discrete event controllers and real-time systems. The applications considered are in reactive and safety-critical systems, e.g. robotic systems.

Franois Chaumette was born in Nantes, France, in 1963. He graduated from Ecole Nationale Supérieure de Mécanique, Nantes, France in 1987 and received the Ph.D degree in Computer Science from the University of Rennes, France, in 1990. In 1991, he received the AFCET / CNRS Prize for the best french thesis in Automatic Control. Since 1990, he has been with INRIA, at IRISA in Rennes, France, where his current title is “Chargé de Recherche”. His research interests include robotics, computer vision, and especially the coupling of these two research domains (vision-based control, active vision and purposive vision).