Control laws, Tasks and Procedures with ORCCAD: Application to the Control of an Underwater Arm

to appear in *International Journal of Systems Science*

D. Simon, K. Kapellos*
INRIA Sophia-Antipolis, B.P. 93
06902 SOPHIA-ANTIPOLIS Cedex, FRANCE

B. Espiau
INRIA Rhone-Alpes, 655 avenue de l'Europe
38330 MONTBONNOT ST MARTIN, FRANCE

Abstract

Software reliability is a major issue in the design of control architecture for robots operating in hostile or poorly known environments.

The ORCCAD control architecture gathers control laws in continuous time at the low levels and discrete time logical aspects at higher levels. While some performances can be checked using simulations, crucial properties such as dead-lock avoidance, safety and liveness can be formally verified at both levels, using in particular some advantages of synchronous programming and associated tools. In the framework of the Union project, the underlying programming style using mission composition from basic actions and formal verification is illustrated by the design of an underwater structure inspection mission simulation using a ROV fitted with a manipulator. Since the physical system is not yet ready only realistic simulation results are provided, but it is expected that they will be rather easy to transfer on the real plant.

1 Introduction

A mobile robot aimed to operate in an hazardous environment, like a long range AUV or a planetary rover, is a typical example of critical system. We mean here that, for such a system, like for a satellite, any repairing or recovery operation, even a mission reconfiguration, which would involve the intervention of a human operator is always costly, often difficult and sometimes impossible. This is why such systems should be at least provided with capacities of on-line adaptation, like self replanning or sensor-based control. However, this is not sufficient and we have to be sure, as far as possible, that the system will behave correctly, *before launching*. More precisely, once a mission has been defined, we would like to verify that:

- its specifications are correct, i.e. that they correspond to the desired goals,
- its programming conforms to specifications,
- the constraints induced by real-time and implementation issues do not disturb its behavior.

Therefore, and assuming that the hardware structure (mechanics, sensors, computer architecture) is given, this points out the necessity of validating all the algorithmic and software issues, from the point of view of their implementation as well as from that of their functionalities. This mainly concerns two aspects, which should not be considered independently: the control issues, modeled

*This work is funded by the Esprit III Basic Research Action UNION

through discrete- or continuous-time differential equations, and the logical ones, represented by discrete events. If we now adopt the user's point of view, it appears that his main interest is in the specification of complex missions or applications in an easy and safe way. For that purpose it is necessary to define properly what are the activities he should handle and how to *compose* them in order to meet his requirements.

In classical robot controllers, activities are mainly defined as motion control, i.e. following a predefined trajectory either in joint or operational space. Using exteroceptive sensors, when allowed, is difficult and usually limited to the definition of guarded commands. Besides, modern control theories like the Task-function approach ([Samson91]) allows to specify and design more efficient actions in different spaces. In particular, it promotes the use of exteroceptive sensors to design sensor-based control laws like visual servoing ([Espiau92]) or force-feedback control ([Espiau90]). Hybrid tasks main be defined, where some directions of space are controlled in position or velocity while others are controlled using force-feedback, e.g. to perform automatic contour following of an unknown object. Finally, redundant tasks, i.e. when the dimension of the primary goal is smaller than the dimension of the physical system, can be designed in the same framework by adding a secondary cost function: such a task is detailed in section 5.1. Such different control laws are used in the design of basic actions of the robot and are then scheduled and synchronized, according to the context, to perform a more complete mission. For example, to perform an assembly operation, we may use first trajectory tracking until discovering the socket with a camera, switch to visual-servoing for fine approach and finally switch to force-feedback control to perform an assembly operation after contact detection.

From the verification side, the compositionality principles must preserve the coherence between underlying mathematical models, in order to be able to perform formal computations at any level. As an example, the use of a single synchronous reactive language as a target for automatic translation is a way of preserving a logical structure whatever the complexity of the application. A consequence of this point of view is that the basic entities have to be carefully studied and also that composition operators should have a proper semantics.

The ORCCAD architecture is aimed to provide users with a set of coherent structures and tools to develop robotic applications in this framework. Continuing a previous work ([Simon94]), we show in this paper how concepts and validation tools are currently tested in the framework of the Union project [Rigaud95], where canonical missions have been defined for a hybrid underwater platform, i.e. the Ifremer's Vortex vehicle fitted with a PA10 arm.

This paper is organized as follows: in the next section we present the main concepts we designed inside the ORCCAD architecture. In section 3 we briefly review some available formal verification methods. Section 4 describes a mission scenario for the Vortex/Pa10 system. Some of the structures we use to design this mission are detailed in sections 5 and 6 where simulations results are provided. Guidelines for the future are given in the conclusion of the paper.

2 ORCCAD: an Hybrid Systems-Oriented Architecture

ORCCAD ([Simon93]) is a development environment for specification, validation by formal methods and by simulation, and implementation of robotic applications.

The formal definition of a robotic action is a key point in the ORCCAD framework. It is based on the following basic principles:

• in general, physical tasks to be achieved by robots can be stated as automatic control problems which can be efficiently solved in real-time by using adequate feedback control loops. In this framework, let us mention that the *Task-Function* approach ([Samson91]) was specifically developed for robotic systems;

• the characterization of the physical action is not sufficient for fully defining a robotic action: starting and stopping times must be considered, as well as reactions to significant events observed during the task execution;

• since the overall performance of the system relies on the existence of efficient real-time mechanisms at the execution level, particular attention must be paid to their specification and their verification.

A robotic application should therefore handle all these aspects coherently. Its specification must be modular, structured and accessible to users with different expertise. The *end-user* concerned with a particular application should be provided with high level formalisms allowing to focus on specification and verification issues; the *control systems engineer* needs an environment with efficient design, programming and simulation tools to express the control laws which then are encapsulated for the end-user.

In ORCCAD, two entities are defined in order to fulfill these requirements. The *Robot-Task* (RT) models basic robotic actions where control aspects are predominants. The *Robot-Procedure* (RP) are used to logically and hierarchically compose RTs and RPs in structures of increasing complexity from basic actions upto a full mission specification.

The RT characterizes in a structured way continuous time closed loop control laws, along with their temporal features related to implementation and the management of associated events. More formally, a RT is the entire parameterized specification of:

• an elementary servo-control task, i.e. the activation of a control scheme structurally invariant along the task duration;

• a logical behavior associated with a set of signals (events) which are expected before or during the task execution.

These events may be pre-conditions, post-conditions and exceptions which are themselves classified in type 1 (weak), type 2 (strong) and type 3 (fatal) exceptions.

This characterization of the interface of a RT with its environment in a clear way, using typed input/output events, allows to compose them in an easy way in order to construct more complex actions, the RPs: briefly speaking, they specify in a structured way a logical and temporal arrangement of RTs in order to achieve an objective in a context dependent and reliable way, providing predefined corrective actions in the case of unsuccessful execution of RTs. More formally a RP is the complete specification of

• a main programme, characterizing the nominal execution of the action, composed by RTs, RPs and conditions,

• a set of triplets (exception event, processing, assertion), which specifies the processing to apply to handle the exception and the information to transmit to the planification level (if provided), and

• a local behavior defining the logical co-ordination of the previously considered items.

Figure 1 summarizes the exceptions processing organization in ORCCAD. Type 1 events are locally processed in the RT, e.g. by parameters or gains modification. Type 2 exceptions are treated in the RP leading to switch to a different RT in nominal conditions and to a new RP in case of failure. Type 3 exceptions lead to a mission abortion through a safety behavior which may be context dependent ([Kapellos94]).

These well defined structures allows to systematize and thus automatize formal verification on the expected controller behavior. This is also a key to design automatic code generators.

Using robust control-laws and tuning the gains and parameters with a simulation tool like SIMPARC ([Astraudo92]) ensures the stability of the physical system during RTs execution with specified performance. On the other hand the logical behavior of the RT is verified to be correct to ensure critical properties of liveness and safety.

From the discrete event systems point of view, the RP ensures a robust control of the physical system seen as a collection of RTs. Here, verification mainly concerns the correctness of the logical

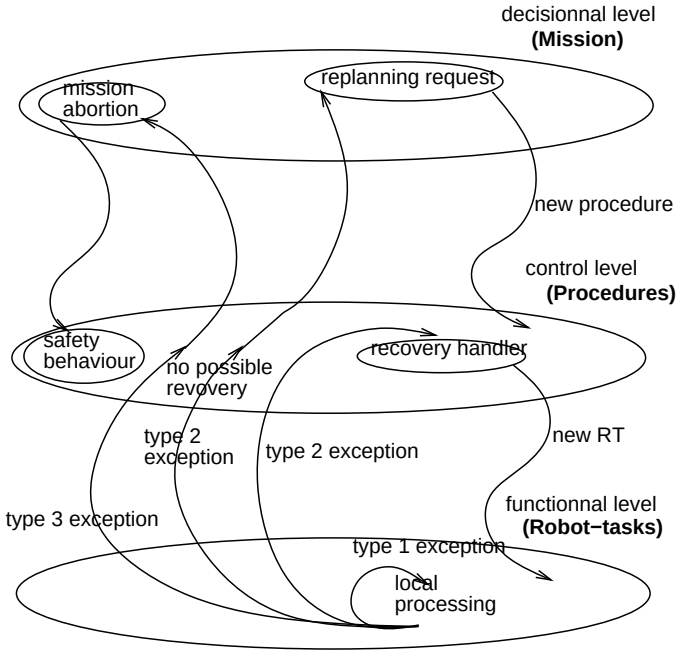


Figure 1: Reactivity and exception handling

and temporal behavior, by proving critical properties and conformity with application's requirements. These aspects will be illustrated by examples in the last sections of the paper.

3 Formal Verification: an Overview

Programme verification is a research topic which is almost as old as computer science itself. Among the existing possibilities, the model checking approach is widely used when problems can be represented under the form of certain classes of transition systems. In the case of reactive systems, which is the most relevant for robotics applications, an idea consists in expressing programmes in a language with a well-defined operational semantics, like in the *synchronous* approach. Then, the comparison to specifications can be performed, either by expressing global properties in temporal logics, or by expressing the specification as a transition system and verifying its equivalence with the whole system. There exists tools for this approach, like EMC ([Clarke83]) or AUTO ([de Simone89]), both of them are friendly interfaced with the ESTEREL ([Berry92]) synchronous language. Generally, the user can either build abstract views and observe the resulting behavior on a reduced automaton, which is easy when this last is small, or express the property to be checked under the form of a regular criterion and verify its satisfaction by bisimulation, a technique which may prove that the programme augmented with the regular criterion exhibits the same behaviour than the original specification ([Sangiorgi95]).

It is also possible to extend this approach from boolean automata to timed ones. The idea consists in introducing timed states, that is to say states which include implicit counters. Among the existing tools, let us mention that a time extension to ARGOS ([Maraninchi92]) allows the use of the temporal prover KRONOS ([Henziger92]). Finally hybrid systems have been a topic of growing interest in the recent years. Their study comes from the fact that physical systems are never purely continuous-time ones nor only finite state machines, but really combine intimately discrete issues and continuous components. Nevertheless current methods only concerns linear hybrid automata where variables change at constant rate and thus available tools are not suited to analyse non-linear systems like robots. In ORCCAD the verification methods we use are summarized in figure 2. See for example [Espiau95] for an extended review of possible approaches of formal verification for robotics.

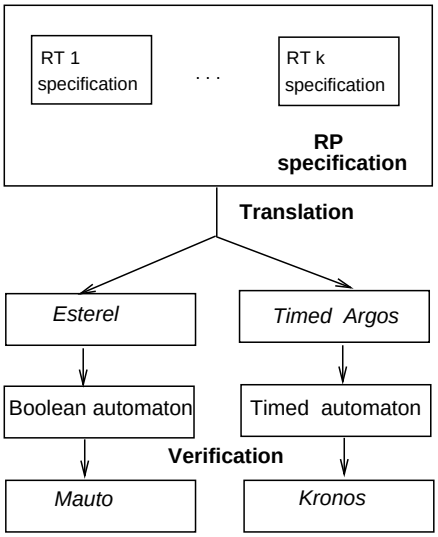


Figure 2: Specification-verification process in ORCCAD

An important field where formal verification is required is the one of underwater vehicles or robots. However, only a few works, like [Coste-Manière95] or [Simon95b], verify explicitly some properties. The interested reader may also consult [Kalavanis e.a.95], where several approaches of control architecture are described, some of them allowing potentially to perform different types of verification.

4 Experimental Plant and Mission Scenario

Let us now focus to the case study which will be considered in the following. In the framework of the Union project [Rigaud95] we plan to assess the design and verification methods of Orccad with an underwater testbed application. The mission will take place in a pool, using the Vortex vehicle fitted with a manipulator.

Vortex is a R.O.V. designed by Ifremer as a testbed for control laws and control architectures. Up to now it is equipped with six screw propellers, four in the horizontal plane and two in the vertical one. Two vertical thrusters will be added so that the vehicle will be actively controllable along and around all its three motion axis.

The vehicle is fitted with a traditional sensing set such as compass, inclinometers, depth-meter and gyro-meters allowing to measure most of its internal state. A video camera is used for target tracking tasks and a belt with eight ultrasonic sounders allows to perform positioning and wall following tasks. Control algorithms are computed on an external VME backplane running Pirat, a C++ library of objects used for control design. At a higher level, i.e. control laws and mission management, ESTEREL is used to design Robot-tasks and Procedures, consistently with the ORCCAD approach.

Vortex is currently being equipped with an electric powered Mitsubischi Pa10 arm with 7 degrees of freedom, fixed under the vehicle on its z axis. Control algorithms for the arm are run on a second, modified, Pirat controller. Up to now the two controllers only have a low bandwidth communication capability, therefore control laws for Vortex and the Pa10 arm will run independently, and *only short synchronization messages will be exchanged* on the communication link. We use this system, depicted in figure 3 to test the programming style and verification process induced by the RT and RP concepts given the following mission scenario.

The mission simulates the inspection of an underwater structure i.e. a vertical pipe (figure 4).

Starting from the initial position, Vortex is set in station keeping mode while the arm is folded

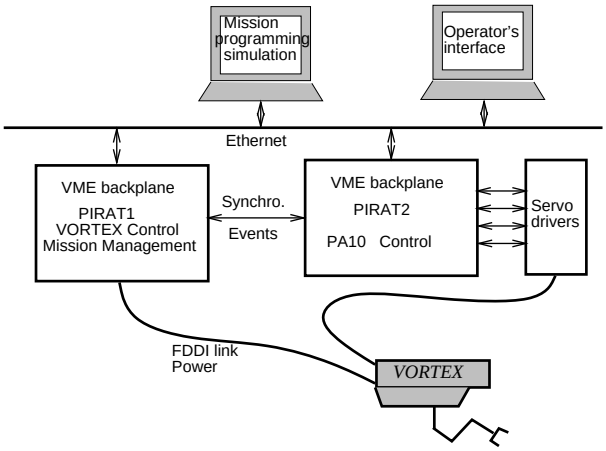


Figure 3: Experimental plant

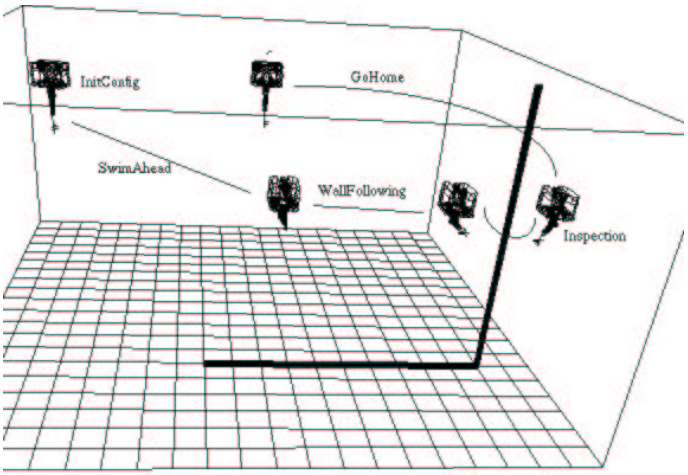


Figure 4: Mission description

from its initial extended position to its parking position. The arm is then locked using the brakes, while Vortex swims ahead at a defined heading angle up to detection of a wall at a defined distance. The control law is then switched to the wall following mode up to reaching a corner, where the vehicle is locked in station keeping mode.

A second phase is then started to work with the arm while the vehicle is stabilized. The arm is first driven in teleoperated mode to point to some locations in the workspace, then it is operated in robot mode to track again the stored locations. During this phase, Vortex can be stabilized in two ways, using target tracking with the camera or using the sounders belt. Once the arm operation is finished an operator call sets it again in parking position while the vehicle comes back to its initial position under the crank.

The system operator also defined several general exceptions and recovery behaviours. At any time, the detection of a water leak or of an hardware failure must lead to abort the mission and to leave the system in a situation as safe as possible, i.e. setting an alarm and emergency surfacing with the arm locked in folded position. The vehicle must never cross a depth limit so that the arm cannot hit the bottom of the pool. It is worth noting that these recovery behaviours are mission- and context-dependent, i.e. even if a water leak should be a fatal exception for any underwater vehicle the defined recovery procedure here assumes that the surface is free and could not be used e.g. during under-ice exploration. Other exceptions more specific to a given system or control algorithms are also defined inside the subtasks involved in the mission.

Following the ORCCAD approach we designed the necessary RTs and RPs given by figure 5. Although this mission still remains rather simple compared with real operation at sea, the overall complexity and interleaving of exceptions, recovery behaviours and parallel activities justify that properties important for the system safety must be checked during mission design and before execution. The global resulting automaton is impossible to be understood by human beings and even too big to be displayed. In the next section we focus on the way we designed and validate some of the control laws and RTs needed to implement the mission scenario.

Procedures	Sub−Proc. and Tasks	Main Events	Comments
InitCruiseConfig	Vortex: StationKeeping Pa10: GoToPark	Parked	Using absolute position Arm folded and locked
ReachWorkingArea	Vortex: SwimAhead WallFollowing Pa10: GoToPark	WallDetected CornerDetected	Navigation Using US sensors
DoInspection	Vortex: KeepStableBase Pa10: Pa10_Working Pa10BrakesOff	StopKeepStable Operator's calls	Stabilization in workspace Inspection with arm tip Brakes are released
GoHome	Vortex: GoToPoint Pa10: GoToPark	WayPointReached	Come back to initial position
Emergency	Vortex: GoUp Pa10: GotoPark	Hardware failure Water leak Operator's interrupt	Emergency recovery behaviour
KeepStableBase	Vortex:KeepStableCamera KeepStableUS	TargetFound TargetLost	Visual servoing Distance towalls servoing
Pa10_Working	Pa10: Pa10_TT_JS Pa10_TT_SE3 Pa10_Teleop	JointLimits Operator's calls	Traj. tracking in joint space Traj. track. in operationnal space Teleoperation mode
GoToPark	Pa10: Pa10_TT_JS Pa10BrakesOn	ParkPosReached ArmLocked	Arm folded Arm locked by brakes

Figure 5: Summary of tasks and procedures

5 Design and Analysis of Robot-tasks

5.1 Trajectory Tracking of the PA10

5.1.1 Control of Redundant Tasks

The control law which is used for the PA10 manipulator belongs to the class of decoupling/feedback linearization in a dedicated task space. Its general expression is given in (3). The goal assigned to the manipulator is defined as the regulation to zero of an $n-$dimensional output function $e(q,t)$, where q is a vector of generalized coordinates, aimed to represent in a clever way the user's objective. In the present case, the output function includes the tracking by the arm tip of a trajectory in the 6-dimensional space of frames (SE(3)). Since the robot has 7 joints, one degree of freedom is available for achieving simultaneously a secondary task, which can be expressed as the minimization of a scalar cost $h_s(q)$. Classical secondary goals of that kind are the avoidance of kinematic singularities or of joint limits, the minimization of the velocity norm, etc.... The two tasks are finally gathered into a single one through the expression:

$$e = J_1^{\dagger} e_1 + \alpha (I_6 - J_1^{\dagger} J_1) \frac{\partial h_s}{\partial q} \tag{1}$$

where e_1 expresses the trajectory tracking task:

$$e_1 = \begin{pmatrix} P(q) - P^*(t) \\ A(q,t) \end{pmatrix} \tag{2}$$

A being a parameterization of the attitude error, P the position of the tip and α a positive weighting factor. $J_1 = \frac{\partial e_1}{\partial q}$ is the jacobian matrix of e_1.

The final control law is given below.

$$\Gamma = -k\hat{M}\left(\widehat{\frac{\partial e}{\partial q}}\right)^{-1} G\left(\mu De + \widehat{\frac{\partial e}{\partial q}}\dot{q} + \widehat{\frac{\partial e}{\partial t}}\right) + \hat{N} - \hat{M}\left(\widehat{\frac{\partial e}{\partial q}}\right)^{-1}\hat{f} \tag{3}$$

Here, Γ is the array of control actuator torques, M and N gather the Lagrangian dynamics, and k, μ, G, D are tuning parameters. The "hats" indicate that more or less complex models of the concerned terms can be used. In fact, it should be emphasized that the RT designer may select easily the adequate models and tuning parameters in ORCCAD, since they belong to some predefined classes in an object-oriented description of the control.

5.1.2 Design of a Robot-task

Action Description The reference trajectory is defined in the SE3 space with respect to the vehicle coordinates system. It will be used to drive the end effector to inspection positions and, with slight modifications, to control the tip position in teleoperated mode using a master-arm with a different kinematics. The Modified-Denavit-Hartenberg (MDH) parameters, working range and main dynamic parameters of the PA10 are listed in table 1.

Prior to any motion, correctness of the initial conditions and software initializations must be checked. The action must be aborted if a joint limit is reached. The servoing task duration is the one of the reference trajectory.

Link	σ	α	d	θ	r	*Limits(rad)*
1	0	0	0	q_1	0.315	-2.35 2.35
2	0	$\pi/2$	0	q_2	0	-1.64 1.64
3	0	$-\pi/2$	0	q_3	0.450	-2.35 2.35
4	0	$\pi/2$	0	q_4	0.0	-2.35 2.35
5	0	$-\pi/2$	0	q_5	0.50	-2.35 2.35
6	0	$\pi/2$	0	q_6	0.0	-3.14 3.14
7	0	$-\pi/2$	0	q_7	0.080	*no limits*

Link	*Masses*	*Inertia*	*Moments*
1	9.78	0.3646, 0.3738, 0.3673	-1.632
2	8.41	0.0851, 0.894, 0.0851	.532
3	3.51	0.0502, 0.0588, 0.0529	$-.314$
4	4.31	0.0291, 0.0321, 0.0291	.199
5	3.45	0.1173, 0.1206, 0.1221	$-.568$
6	1.46	0.0016, 0.0078, 0.0016	$-.044$
7	0.24	0.002, 0.0002, 0.0002	$-.007$

Table 1: Denavit-Hartenberg and dynamic parameters of the PA10 arm (in air)

Specification in Continuous Time On the basis of these informations and requirements, a set of objects defined in the RT modeling [Kapellos94]) can now be instantiated by setting adequate numerical values. The resulting specification defines the action from a continuous time point of view, i.e independently of time discretization and other implementation related aspects which are considered in a next design step. A graphical representation of the RT designed using the RT editor

of the ORCCAD system is given figure 6. Some of these objects are now described and the reader is referred to [Kapellos95b] for a complete description of a similar RT.

TGPA10 computes on line the reference trajectory $R_d(t) = [P_d(t), A_d(t)]$ in the SE(3) space. The initial position $R_{init} = [P_{init}, A_{init}]$ is considered to be the current one, while the final position $R_{fin} = [P_{fin}, A_{fin}]$, as well as the trajectory duration T are given by the parameters of the Robot-task. Velocity and acceleration profiles of the trajectory are fixed by choosing the following r(t) function:

$$\begin{array}{l} P_d(t) = P_{init} + r(t)[P_{fin} - P_{init}] \\ A_d(t) = A_{init} A(u, r(t)\Theta), \qquad t \in [0, T] \end{array} \tag{4}$$

where u is the vector of the axis of the rotation and Θ is the value of the rotation angle.

For simulation purposes, the PA10 object models the dynamics of the arm using the state representation:

$$\dot{X} = f(X, U, t), \quad Y = g(X), \quad X = [q_i, \dot{q}_i, i = 1, \ldots, 7]$$

where the dynamics (including hydrodynamics and hydrostatic forces) f is computed using Dynamechs ([McMillan95]) and the direct kinematics g is computed using the Symoro software ([Dombre88]).

KINPA10 computes the attitude of the frame linked to the end effector of the arm from the values of the MDH parameters given in table 1. This attitude is expressed in the frame linked to the binding point of the arm on the vehicle.

TFRT computes the task function $e(q, t)$ given by equations 1 and 2. The choice of the attitude error parameterization, of the secondary objective to minimize and the value of the weighting between the primary and secondary task functions are left to the designer. In this case the axis/angle parameterization is used for rotations while the avoidance of joint limits using Greville's algorithm is selected as a secondary objective.

CORED computes the control vector Γ to be sent to the actuators. Further simulations and experiments allow to tune the values of the gains k, μ, G and D.

The discrete event behaviour of the RT is specified by the instantiation of three objects. More precisely,

-OBSPREC tests the conditions to start the action (initializations, ...) and sends the "*start*" event to the discrete event controller.

-BUTPA10 detects a joint limit hit and reports it to the discrete event controller by emitting the event "*joint_lim*". "Software" limits are computed using the values of table 1 together with a user's defined threshold.

ATR computes the RT desired behaviour: it is driven by the observed input events and it emits output ones. The designer has to specify the processing type for the declared exceptions. In our case, accordingly to the action description, the T2 type is associated to "*joint_lim*" event. Thanks to the strong typing of events, the corresponding ESTEREL code is automatically generated through the Graphical User Interface (GUI) and is proven to be correct in any case ([Kapellos94]).

5.1.3 Time-constrained Specification

The passage from the above continuous time specification to a description taking into account implementation aspects is done by specifying temporal properties, i.e. sampling periods, duration of computations, communications and synchronizations between the processes. It is also finally necessary to enter the localization of data processing codes.

At this stage of specification the basic entities are Module-tasks, i.e. real-time tasks, usually periodic, which implement an elemental functional module of the RT. Since MTs may, possibly, be distributed over a multi-processor target architecture they communicate using typed ports and specific synchronization mechanisms. Each MT owns one input (output) port for each data it consumes (resp. produces).

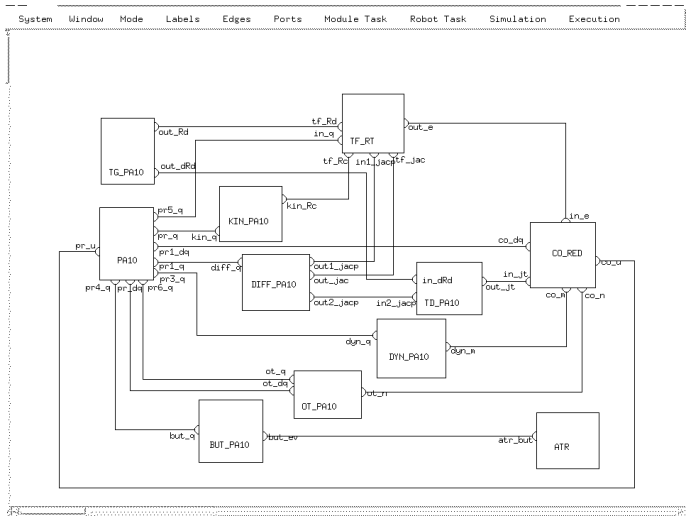


Figure 6: Graphical Representation of the Robot-task Specifying a Trajectory Tracking Action in the Frame Space

Let us describe explicitly some aspects concerning the particular RT design. The TGPA10 module owns two output ports, *out_ Rd*, *out_ dRd*, which transmits at every period R_d and its differential *dRd* to TFRT (fig. 6). Since no data is needed by the TGPA10 module, no input port is specified.

Three different types of synchronization are associated to establish communications. CORED and PA10 communicate in an *Asynchronous/Synchronous* way so that the control outputs sent to the actuators are the last computed. The *Event* type is used for the communications between observers and the ATR module, which is the one ensuring the safe and ordered transmission of the events, and *Asynchronous/Asynchronous* type is adopted for other communications.

It is finally necessary to associate sampling periods with the modules. For the TGPA10, it is set to 15ms and its code execution duration is estimated at 90μs, while for the CORED module the period is fixed at 10ms. The names of the files where are located the initialization and computation codes are indicated as well.

5.1.4 Analysis of a Robot-task

Prior to the real-time code generation we analyse the Robot-task specification in order to ensure safety properties of the network of modules like dead-lock avoidance and temporal consistency using Petri net modelling and analysis ([Simon97]). Besides, validation of the control performance can be checked through realistic simulations, taking into account both the plant model and the controller's timed behaviour. As an illustration of the last point we present a few simulation results. Figure 7 plots the evolution of the x and z coordinates of the origin of the end-effector frame in SE(3) (left part) together with the values of the joint variables during a 3ss motion. The Simparc simulation programme was automatically generated from the GUI and allows to tune control gains and temporal attributes upto matching the end user's requirements e.g. expressed as a maximum allowable value for the tracking error.

5.2 Vortex control: Wall following and stabilization

There are two main tasks involving vehicle control during the phases of workspace reaching and inspection. The first one is the wall following and corner stabilization using the U.S. sensors belt and the second is the stabilization in front of a known target, here a vertical pipe, with the camera. This last one is designed from slight modifications of a pipe following task described in [Rives96].

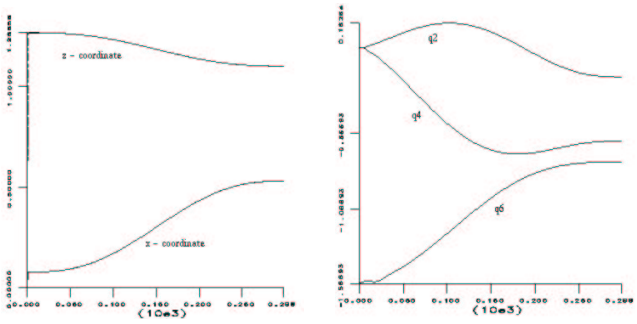


Figure 7: Simulation Results of the PA10TTSE3 task

Several kinds of wall following and stabilization tasks can be performed according to the sensors configuration and value of set points. In [Simon94] sensors were located in the corners of the vehicle, thus allowing a "spiral" task in the pool with automatic crossing of the corners and obstacle avoidance. For locking in a pool corner, a "square" configuration like the one depicted in figure 8 is more adequate.

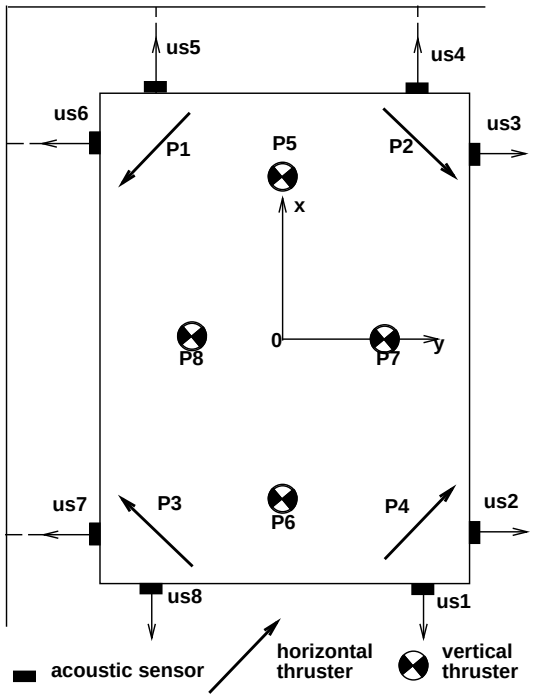


Figure 8: Vortex configuration (top view)

Although 3 sensors would be enough to control motions along the x and y axis and around the yaw axis, we can use four of them in order to improve reliability and decoupling. Sensors number 4,5,6 and 7 are here used to perform left wall following and docking. According to the task-function approach ([Samson91]), we can design the following first sensor-based task-function to be servoed to zero in order to control the x, y and yaw motions (assuming that the pool angle is a square angle):

$$e_1 = \begin{bmatrix} (us4 + us5) - 2 * x_d \\ (us6 + us7) - 2 * y_d \\ (us5 - us4) + (us7 - us6) \end{bmatrix} \triangleq \begin{matrix} x\ dist. \\ 0\ y\ dist. \\ yaw \end{matrix} \quad (5)$$

A second task-function uses the internal sensors to control the depth of the vehicle and its roll and

pitch angles:

$$e_2 = \begin{bmatrix} (z - z_d) \\ (\phi - \phi_d) \\ (\theta - \theta_d) \end{bmatrix} \triangleq 0 \begin{matrix} depth \\ roll \\ pitch \end{matrix} \quad (6)$$

From this 6 dimensional hybrid task-function we are now able to compute the elementary forces and torques along and around the axes of the vehicle using simple PID controllers with fixed gains. These elementary forces and torques are finally gathered to compute the total desired force screw applied to the vehicle using its set of thrusters through a mapping and anti wind-up function.

Control laws for the vehicle and the arm were tested and tuned using the realistic simulations provided by Simparc ([Astraudo92]). Some classes of the Dynamechs library ([McMillan95]) were integrated in the former model of Vortex to compute the arm dynamics and the interaction between the two subsystems. Simparc allows to set many temporal features of the system and of the controllers. Figure 9 shows the front sensors signals during the last seconds of the docking phase. The used numerical values are the ones we expect to use in real experiments, i.e. sampling rates for the PA10 and Vortex controllers of 10 ms and 100 ms respectively. The US sensors synthesized by ray tracing are triggered every 360 ms with a 90 ms shift between two successive sounders. The logical behavior of this Robot-task is not detailed here but follows the general structure described in the previous section.

Once we have designed and validate all the necessary RTs we may logically en temporally compose them to design and verify more increasingly complex actions upto reaching the mission level as shown in the next section.

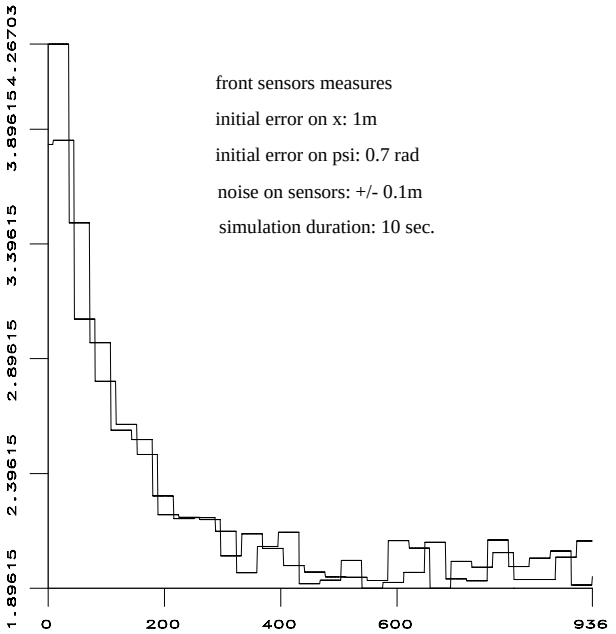


Figure 9: Outputs of Front Sensors During Docking

6 Design and Analysis of Procedures

The user-oriented description of the mission given section 4 naturally guided us to design the four principal procedures which are described below. They will be used afterwards to compose the application.

- InitCruiseConfig consists in preparing the vehicle for cruising. It is composed of the StationKeeping and Pa10ttjs tasks which are launched concurrently to stabilize the platform while the arm is driven to its parking configuration. The *ArmLocked* event is a post-condition of the procedure and stops it.

- ReachWorkingArea is used to navigate in the pool until the vehicle reaches the inspection place as described in section 4. The Pa10BrakesOn task is active all along the procedure duration to keep the arm motionless and folded. The SwimAhead task drives the vehicle in the pool up to a wall detection (*WallDetected* event) and is followed by the WallFollowing task active up to reaching a corner of the pool. According to our previous knowledge of the environment the corresponding (*CornerDetected*) event can be built using US sensors and absolute positioning information to be sure that we are locked in the right corner (the one where there is a pipe).

- DoInspection is used to coordinate actions of the platform and of the arm in order to accomplish the main task which is inspecting the pipe. Points of the target may be learn by teleoperation or recovered using predefined locations. This procedure is extensively detailed in the following paragraph.

- GoHome is in charge of driving the vehicle to its homing position and preparing it to be pulled out. It uses GoToPoint and StationKeeping tasks which concerns the platform activity and Pa10BrakesOn and Pa10ttjs tasks which concerns the one of the arm.

We now detail the specific procedure DoInspection. It is complex enough to enlighten the specification and analysis process we propose in the framework of the Orccad approach.

6.1 The DoInspection Procedure: Specification and Analysis

DoInspection procedure aims at performing manipulation operations with the arm while ensuring the stability of the base. Since the base and the arm are controlled separately despite of mutual disturbances, an important requirement is that the arm can only move when the base is stable enough and stops its motion otherwise. Motions of the arm can be performed again after recovering platform stability. It is clear that the need to handle concurrency, preemption and synchronization is very strong in this procedure as well as the stability of the overall physical system; these reasons justify our choice to present it in detail.

More precisely, the procedure can begin when the vehicle has reached the working area, i.e. it is positioned in front of the cylinder to be inspected within a predefined threshold (see fig. 13). During arm operations the vehicle is controlled using the KeepStableCamera RP. The presence of the camera signal is monitored all along the procedure. Loosing this signal, e.g. due to an uncontrolled motion of the vehicles, raises a *TargetLost* signal switching to the KeepStableUs task up to stability and video signal recovery (*TargetFound*).

Once the vehicle is stable in front of its target the actions concerning the arm can be run concurrently with the platform stabilization on the second controller. The Pa10Teleop may begin under control of predefined signals coming from the operator's interface and keyboard to touch some locations on the target and store their coordinates. Upon an operator's command, the arm controller is set in the mode of automatic trajectory tracking in operational space Pa10ttse3 to point again the visited locations. As explained in section 5.1 several secondary task-functions, like joint-limits and singularities avoidance, can be switched according to the operation context. In both modes, a platform unstability temporary blocks the arm motions (Pa10BrakesOn task) up to recovering vehicle control.

Let us now see how we progressively specify and analyse this procedure by hierarchically composing actions of increasing complexity. In particular we detail in the following paragraphs two (sub)procedures: the first one concerns the stabilization of the base while the second deals with 'safe' motions of the arm. At each step of the specification validation by formal methods and simulations are proposed.

6.1.1 The KeepStableBase Procedure

Since the camera sampling rate is about ten times faster than the U.S. sensors, it is expected that the first one will be more robust with respect to the disturbances coming from the arm motions. However, if the camera looses the target, we can switch to sounders stabilization until being able to recover the target visual tracking mode, centered facing the pipe. Thus, several different actions may be concurrently used in a context dependent way to achieve a common goal. This redundancy is useful to increase the safety and efficiency of the system and such a situation where several RTs are exceptions of each other is a typical structure in our procedures.

Thus this procedure is composed by the KeepStableCamera and KeepStableUs RTs. The first one is the main programme while the second one is a recovery programme associated with the *TargetLost* type 2 exception issue from the KeepStableCamera task. In the case of the recovery programme execution, the end of the procedure is achieved either by reception of the *TargetFound* event giving the possibility to come back to the main programme, or by the *StopKeepStable* event awaited all along the evolution of the procedure. Thus, using knowledge about the geometry of the pool, this action is able to achieve stability of the mobile base even in the case of video tracking failure. This specification is depicted in figure 10.

```
Name : KeepStableBase
Pre-cond:OkInit[30ms], SignalIsValid[30ms]
Main programme:
      KeepStableCamera
T2 exceptions: TargetLost
      do
        KeepStableUs
      until targetFound
T3 exceptions: WaterLeak, ...
Post-cond : StopKeepStable [10ss]
```

Figure 10: KeepStableBase RP specification

Assuming that the individual Robot-tasks were previously validated, the analysis process now mainly concerns the verification of the logical behaviour of the discrete event controller given by the translation of the RP specification into the Esterel language. Nevertheless, the logical correctness of the programme does not guarantee that all possible sequences of tasks are compatible, correctly parameterized and finally that switching is smooth enough to avoid undesired transient behaviours. As pointed in section 3 the state of the art in verification of hybrid systems does not provides usable methods to prove assertions related to continuous time varying variables of the system's state. Thus simulation remains a complementary method to be used for a complete verification of our procedure.

Logical Behaviour Verification Firstly, the satisfaction of *crucial properties* is checked. Concerning the *safety property* (any fatal exception must always be correctly handled) we proceed as follows. Knowing the user's specification defining the fatal exceptions and the associated processing, we build a criterion defined by abstract actions like:

"*Error = /Water_Leak? and not /Accent!*".
The abstraction of the global procedure automaton with respect to this criterion is then computed. The absence of the "Error" action in the resulting automaton proves that the safety property is verified.

The *liveness property* (the RP always reaches its goal in a nominal execution) is proved in a similar way. The "Success" signal is emitted at the end of all successful achievements of a RP. The abstraction of the procedure automaton crossed with an adequate criterion built with this signal must be equivalent by bisimulation to a one state automaton with a single action, the "Success" one.

Conflicts detection : We are interesting here to check that during the RP evolution it does not exist instants where two different RTs are competing for using a same resource of the system. We consider the physical resources controlled by the RTs (the arm and the vehicle) as well as the software resources used by the controllers (real-time tasks). For example, we want to verify that the RTs KEEPSTABLECAMERA and KEEPSTABLEUS never compete to apply different desired force screws to Vortex thrusters during all the RP evolution. We reduce the global automaton (fig. 11) to the only interesting signals STARTSERVOINGVORTEX and STOPPEDSERVOINGVORTEX. Thus we can check that these two signals alternate during the RP life insuring that a control law can be started only after confirmation that the previous one is stopped. This transition happens in an atomic reaction of the automaton thus minimizing the delay during which the vehicle is not controlled (typically one sampling period according to the real implementation).

The *conformity of the RP behaviour with respect to the requirements* must be also verified. For example, we proved that the loss of the visual signal is always handled by switching to the KEEPSTABLEUS task by observing the global automaton with respect to the *TargetLost* event (specified as a type 2 exception) and the *StartServoingUs* signal (fig. 11).

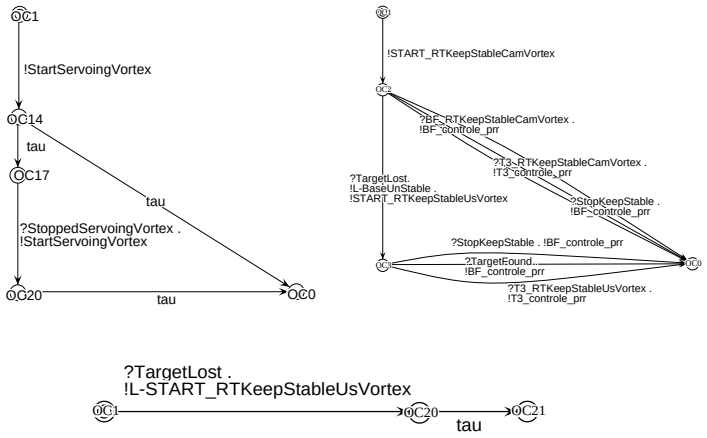
Finally, in order to help the user during the phase of RPs specification we use *abstract views* by switching the body of a non-atomic behavioural activity into a single one at a relevant level of abstraction. Actually, every mismatch in the behaviour's specification is reflected in the resulting automaton. Figure 11 illustrates the nominal execution of the overall mission at the level of abstraction of an RT; only starting and final instants of RTs are considered making abstraction of their internal evolution.

Validation of Smooth Switching After the verification of the logical behaviour of the procedure we focus now on a particular phase of the system's evolution: the transition between the two control tasks. Logical analysis provides no information about the transient behaviour of the physical system during RT switching. Up to now the ORCCAD environment is able to simulate individual Robot-tasks. In the context of this work we developed structures to simulate RP as sequences of RTs. What is characteristic is that the discrete event controller is the same as the one which will be used for execution in real world([Pissard95],[Kapellos95]).

Figure 11 illustrates results of a 10 sec simulation. The transition from KEEPSTABLECAMERA task to the KEEPSTABLEUS, triggered by the *TargetLost* exception, is done smoothly as it is indicated by the evolution of the x and y coordinates of Vortex. We must point out that the generated RTA automatically handles the management of the switching mechanism at the level of the Real-time Operating System (RTOS) i.e. creation, suspension, resumption and deletion of real-time processes together with checking for preconditions completion.

6.1.2 PA10SAFEMOVESE3 Procedure

We now briefly present the SAFEMOVESE3 procedure related to motion control of the arm in operational space. It uses the PA10TTSE3 task for trajectory tracking. Upon reception of a type 2 exception stating that the vehicle is unstable the arm is stopped using PA10BRAKESON to help re-



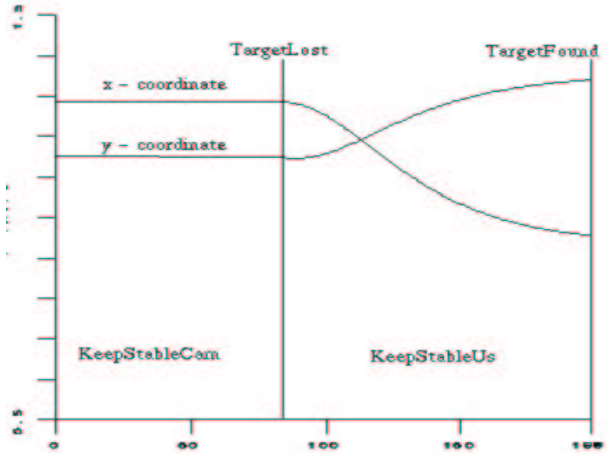


Figure 11: KEEPSTABLEBASE procedure: abstract view and simulation

covering stability. Once it is done Pa10TtSe3 is reactivated up to the end of the desired trajectory receiving the *PosAtteined* post-condition. As this specification is slightly more complex that the one of the keepStableBase procedure we only show its abstract view and simulation results where we can see a motion interrupt while the base tries to stabilize again (see figure 12 and the lowest plot figure 13).

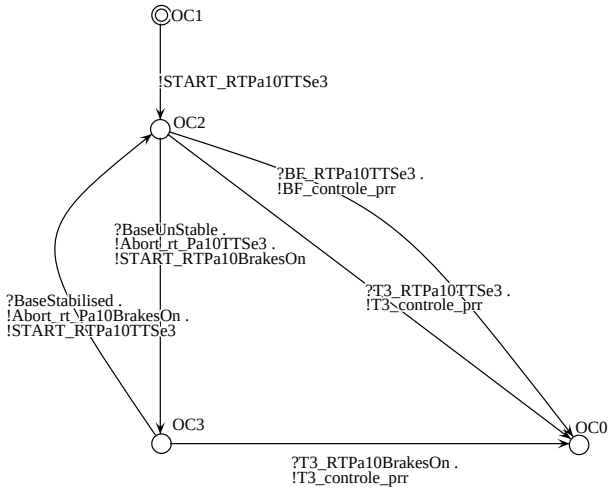


Figure 12: Abstract view of the Pa10SafeMoveSe3 procedure

6.1.3 DoInspection Specification

At this stage we can ensure that, for the KeepStableBase and Pa10SafeMoveSe3 procedures, the logical behaviour of the controller is correct and that the simulated sequencing of tasks works correctly. We are ready now to compose these actions with the other ones to obtain the DoInspection procedure.

Its main programme consists of two parallel branches concerning the arm and the base. Respectively, they are specified as follows:

```
SEQ(do
        Pa10BrakesOn
    until  BaseStabilized;
    repeat 2 times
        Pa10SafeMoveJS;
        Pa10SafeMoveSe3;
    end repeat
    ;
    do
        Pa10SafeMoveJs
    until  ParkingPositionReached
    emit  InpectionOK
    )
```

This specification states that the Pa10 actuators are blocked until the base is stable. A soon as this condition is satisfied the arm is driven to inspect two locations in two steps: a motion in the joint space drives the arm in the region of interest and sequently a new motion in the tip frame space moves the end-effector in front of the inspection point. We remark that the meaning of 'safe' motion is that the base and the arm do not move concurrently. Finally, the arm reaches the parking position and signals that the inspection is finished.

The second parallel branch of the main programme concerns the base of the vehicle and it is specified as follows:

```
SEQ(do
      loop
           BaseSearchTarget;
             emitBaseStabilized
           KeepStableBase;
      end loop
      until  (InpectionOK)
    )
```

It states that the base is situated in front of the target such that the axis of the cylinder is exactly in the center of the camera's image with the right appearance. Sequently, it signals that the base is in a motionless position and keeps it using KEEPSTABLEBASE. We remind that this procedure handles the loss of the video signal and that the 'loop' instruction aims precisely to come back to the vision based control. *InspectionOK* signal, emitted by the branch of the parallel instruction handling the Pa10 activity interrupts the stabilization of the base and the procedure terminates.

6.1.4 DOINSPECTION Analysis

Verification of the Logical Behaviour The crucial properties of safety, liveness and absence of conflicts are verified, like for all the previous procedures. Interesting properties to be proved here concerns the conformity of the specification with respect to the mission requirements and in particular the right synchronizations between tasks. Let us now present a few results.

We want to certify that the arm is motionless when the platform is recovering stabilization using the ultra-sonic sensors. The actions involved in this property are KEEPSTABLEUS and PA10TTSE3 and therefore the signals in respect of which we observe and reduce the global automaton are *START-KeepStableUs*, *BFKeepStableUs*, *STARTPa10ttse3*, *BFPa10ttse3* and *AbortPa10ttse3*. The result is given by the top plot of figure 13. We remark that the two tasks always run in sequence: from the state 'OC3' either KEEPSTABLEUS is executed (state 'OC4') or PA10TTSE3 (state 'OC5'). In particular, if KEEPSTABLEUS run from state 'OC5' the PA10TTSE3 task is aborted and only re-activated after the end of the KEEPSTABLEUS task. The automaton given by the top plot of figure 13 is a more detailed abstract view of the procedure at the RT level and was used to check the right ordering of actions during tasks switching.

Smooth Switching Validation We must recall that the base and the arm are separately controlled so that the stability of the whole system cannot be formally insured. The procedures previously presented only concerns every sub-system.

Besides, in this procedure, coordinated activities are specified. Of course, from the logical point of view we proved that an image loss, probably due to the arm's motion, is correctly handled but the only way we have to test that their mutual disturbances permits the realization of the mission remains simulation.

Figure 13 presents a 15 sec simulation of the DOINSPECTION procedure. The two plots at the top of the figure illustrate the evolution of the x and y coordinates of the base, while the lowest plot represents the evolution of the x coordinate of the origin of the end-effector frame of the arm. Vertical lines indicate the instants of detection of the mentioned events which imply a Robot-task switching in the procedure. The results appear to be satisfactory : the motion of the vehicle during tasks switching is smooth enough to allow for the video signal recovery and recentering of the camera in front of the pipe. Moreover, the motions of the arm are correctly synchronized with the motions of the vehicle, i.e. the arm is always motionless except when the pipe's image is stabilized and centered in the vision's system window. It must be point out that the simulated discrete event controller, managing the arm and vehicle motion tasks, is the same that theone to be downloaded on the actual controller thus improving the confidence we may have in the final system's behaviour.

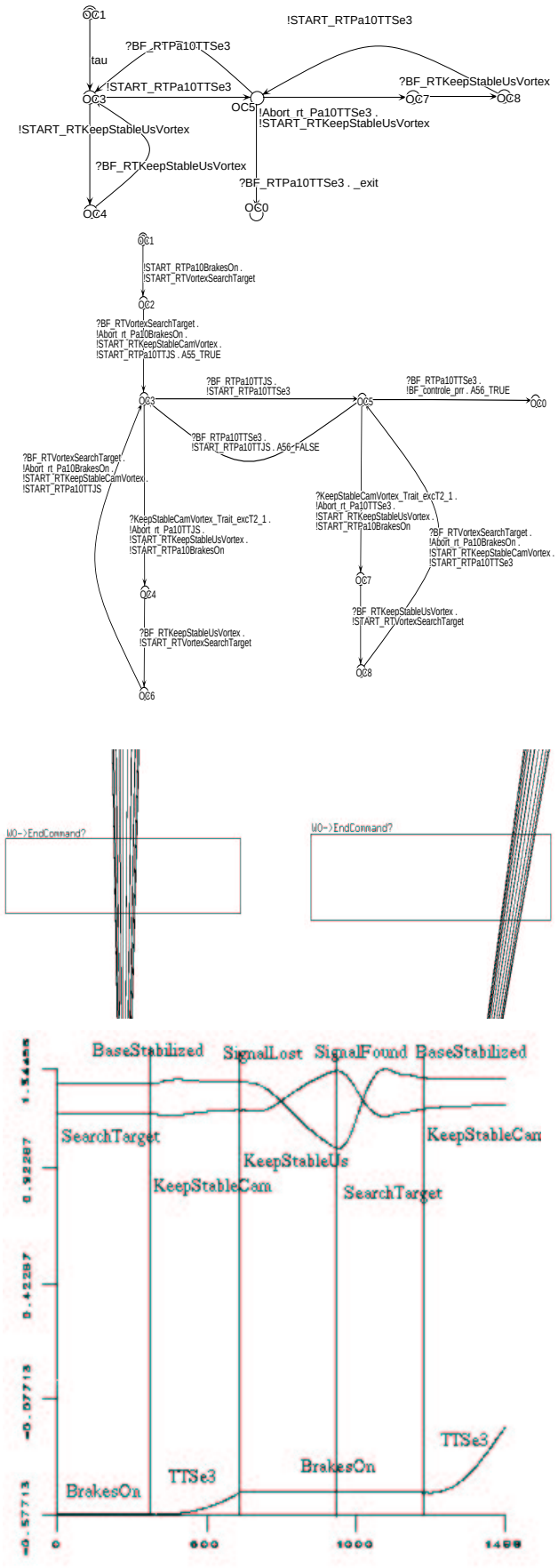


Figure 13: DoInspection procedure: a) abstract view at RT level b) simulation of an evolution

Finally, following the same programming methodology all the RPs, RTs and recovery handlers necessary to the execution of the mission are gathered in the embedding INSPECTION procedure as outlined in figure 14. At the mission level most internal details are hidden to the end-user as shown in figure 15 depicting the foreseen run-time operator's interface.

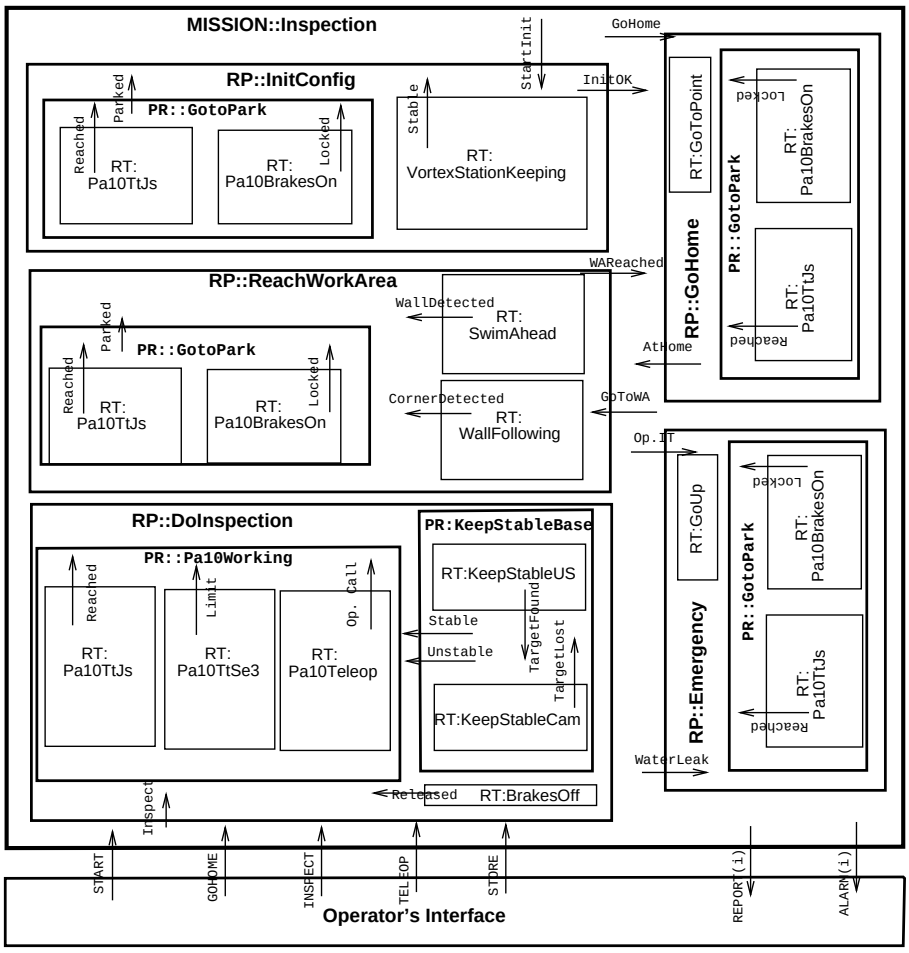


Figure 14: Outline of the MISSION procedure

7 Concluding remarks

We have described in this paper a methodology for mission programming using actions composition, and for its formal verification. It is currently applied to the simulation of an underwater inspection operation in the framework of the UNION project. It has been proved that the logical behaviour of a class of elementary tasks is correct in a generic way, i.e whatever the size of the problem ([Kapellos94]). The absence of deadlocks and the right working of the recovery procedures were checked. Several views of reduced behaviours were also produced. Finally, the verification approach was used to perform some optimization in the handling of real-time tasks.

Since tools for the verification of temporal features and hybrid systems are not mature enough, we only verify the logical behaviours of robotic actions involved in the application. Thus, realistic simulations remains useful to check numerical properties, i.e. tasks synchronization delays or control law performance. Previous experiences shown that this kind of simulation is helpful to prepare experiments and to save time on the real site. We hope that this statement will be confirmed by the experiments planned by the end of the UNION project.

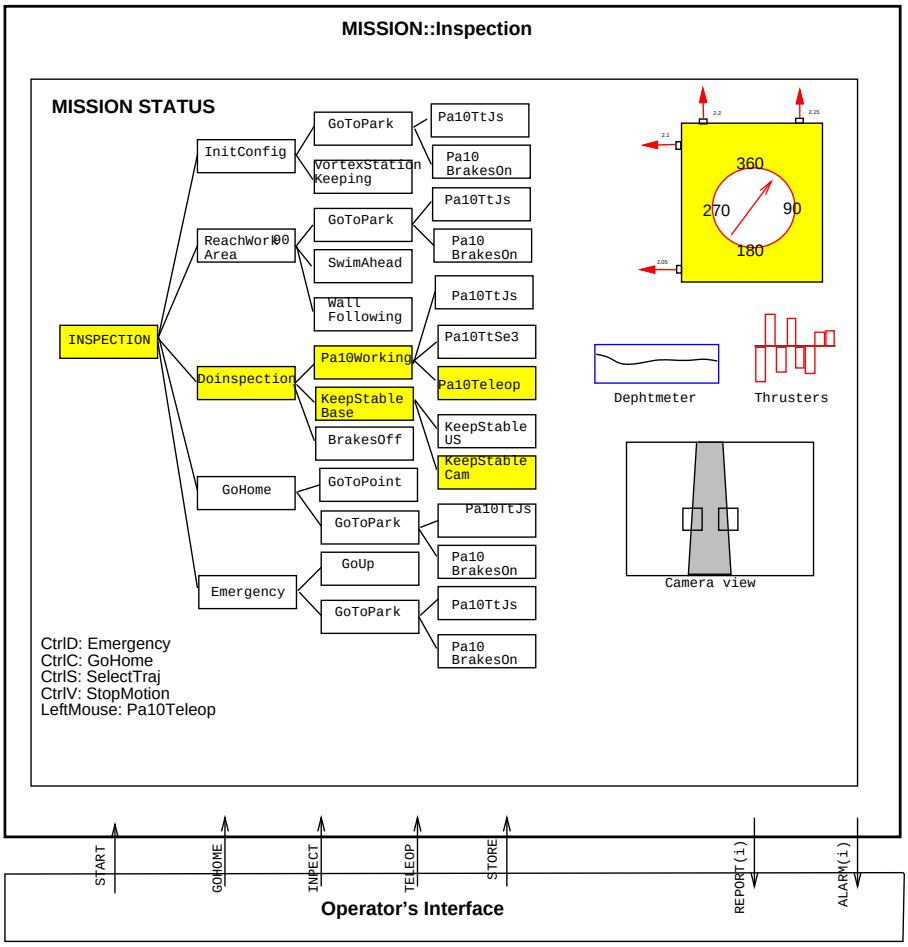


Figure 15: Operator's interface

Although the mission we describe in this paper looks simple compared with a real operation at sea, the interleaving of events and activities of the full mission automaton justifies our hierarchical design and verification process, where we design complex actions from already validated ones lying at a lower level. As an illustration of the complexity, let us mention that the overall automaton of the application has 715 states and 8,597 transitions, which clearly excludes to build it by hand or to verify it visually. This size is a consequence of the existence of a true parallelism between arm and vehicle controls all along the operation.

Despite its success, we consider that this work still leaves several questions open. Among them, we can distinguish three main classes of problems which should be solved for ensuring the widest possible applicability of the proposed approach.

1. *Specification.* Clearly, the coding of the elementary behaviours "by hand" remains rather messy and error prone. As we did for the robot-tasks, the automatic translation of a specification stated in a "natural" end-user's language into ESTEREL programmes is a first necessity ([Espiau96]). Another requirement is that the connection to a possible planning level should be ensured properly. In that ultimate case, the specification under the form of synchronized procedures would be produced automatically.

2. *Verification and Diagnosis.* The available tools for logical aspects seem to have reached a quality level which is high enough for our applications. Even the size seems not to be a problem thanks to new verification techniques using Binary Decision Diagrams, a compact form of storage state space encoded with booleans ([Bryant92]). However, in the critical applications we address,

logical aspects are obviously not sufficient for understanding the behaviour of the application. At least, considering *timed* aspects is necessary. Some tools, based on timed transition systems, begin now to be available in the academic world. However they still suffer of various problems: mainly, expressing the properties to check is difficult, and the diagnosis provided by a temporal prover is generally understandable by an expert only. A great effort should therefore be done for making these techniques really utilizable.

3. *Handling of Symbolic Information.* Again, in practice, logics is necessary but not sufficient. Clearly, an end-user has to specify some issues under the form of symbolic (or linguistic) variables, and to understand messages from the system with the same semantics. This points out the necessity for a system like ORCCAD to handle symbolic aspects (inputs, outputs and computations) with the same rigor as it does for events or continuous variables. Is it not a challenge?

References

[Astraudo92] C. Astraudo, J.J. Borrelly, "Simulation of Multiprocessor Robot Controllers", *Proc. IEEE Int. Conf. on Robotics and Automation*, Nice, May 1992.

[Berry92] G. Berry, G. Gonthier: " The Synchronous Programming Language ESTEREL: Design, Semantics, Implementation", *Science Of Computer Programming*, Vol 19 no 2, pp 87-152, 1992.

[Bryant92] R.E. Bryant: "Symbolic boolean manipulation with ordered binary-decision diagrams", *ACM Computing Surveys*, 293-318,September 1992.

[Clarke83] E.M. Clarke, A. Emerson, A.P. Sistla: "Automatic Verification of Finite State Concurrent Systems using Temporal Logic Specifications: a practical approach", *Proc. 10th ACM Symp. on Principles of Programming Languages*, pp 117-126, 1983.

[Coste-Manière95] E. Coste-Manière, M. Perrier, A. Peuch "Mission Programming: Application to Underwater Robots" *4th Int. Symp. on Experimental Robotics*, Stanford, USA, June 30- July 2, 1995.

[de Simone89] R. de Simone, D. Vergamini: *Aboard* AUTO, INRIA Technical Report no 111, 1989.

[Dombre88] E. Dombre, W. Khalil: *Modélisation et commande des robots*, Edition Hermès, Paris 1988.

[Espiau90] B. Espiau, J.P. Merlet and C. Samson: "Force-feedback control and non-contact sensing: a unified approach", in Proc. *8th Symposium RoManSy*, Krakow, Poland, July 1990.

[Espiau92] B. Espiau, F. Chaumette and P. Rives: "A new approach to visual servoing in robotics", *IEEE Trans. in Robotics and Automation*, vol. 8, pp 313-326, June 1992.

[Espiau95] B. Espiau, K. Kapellos, M. Jourdan, and D. Simon "On the Validation of Robotics Control Systems. Part I: High level Specification and Formal verification", Inria Research Report no 2719, November 1995

[Espiau96] B. Espiau, K. Kapellos, E. Coste-Manière, and N. Turro. Formal mission specification in an open architecture. in Proc. *Intl Symposium on Robotics and Manufacturing ISRAM96*, Montpellier, France, May 1996.

[Henziger92] T. Henzinger, X. Nicollin, J. Sifakis and S. Yovine, " Symbolic Model-Checking for Real-Time Systems",*LICS 92*, IEEE Computer Society Press, June 1992.

[Kalavanis e.a.95] K.P. Kalavanis and al., editors, "International Program Development in Undersea Robotics and Intelligent Control", *Proc. of the Joint US Portugal Workshop*, Lisboa, Portugal, march 1995.

[Kapellos94] K . Kapellos: *Environnement de programmation des applications robotiques réactives*, PhD dissertation, Ecole des Mines de Paris, Sophia Antipolis, France, November 1994.

[Kapellos95] K. Kapellos, S. Abdou, M. Jourdan, B. Espiau "Specification, Formal Verification and Implementation of Tasks ans Missions for an Autonomous Vehicle" *4th Int. Symp. on Experimental Robotics*, Stanford, USA, June 30- July 2, 1995.

[Kapellos95b] K. Kapellos and B. Espiau, "Implementation with Orccad of a Method for Smooth Singularity Crossing in a 6-DOF Manipulator", Inria Research Report no 2654, September 1995

[McMillan95] S. McMillan, D. Orin and B. McGhee, "Efficient Dynamic Simulation of an Underwater Vehicle with a Robotic Manipulator", *IEEE Trans. on Systems, Man and Cybernetics*, vol. 25, no 8, August 1995.

[Maraninchi92] F. Maraninchi, " Operational and Compositional Semantics of Synchronous Automaton Compositions", *CONCUR, LNCS 630*, Springer Verlag, 1992.

[Pissard95] R. Pissard-Gibollet, K. Kapellos, P. Rives, J.J. Borrelly, "Real-Time Programming of Mobile Robot Actions Using Advanced Control Techniques" *4th Int. Symp. on Experimental Robotics*, Stanford, USA, June 30- July 2, 1995.

[Rigaud95] V. Rigaud, "UNION Esprit Basic Research Action: Main objectives of the project", Workshop on Undersea Robotics and Intelligent Control, March2-3 1995, Lisboa, Portugal

[Rives96] P. Rives and R Pissard-Gibollet "Sensor-based tasks : from the specification to the control aspects", in Proc. *Intl Symposium on Robotics and Manufacturing ISRAM96*, Montpellier, France, May 1996.

[Samson91] C. Samson, M. Le Borgne, B. Espiau: *Robot Control: the Task-Function Approach*, Clarendon Press, Oxford Science Publications, U.K., 1991.

[Sangiorgi95] D. Sangiorgi, "On proof method for bisimulation", in: *Proc. MFCS'95*, J. Wiedermann, P. Háiek (ed.), LNCS, 969, Springer Verlag, p. 479–488, 1995.

[Simon93] D. Simon, B. Espiau, E. Castillo, K. Kapellos: " Computer-aided Design of a Generic Robot Controller Handling Reactivity and Real-time Control Issues", *IEEE Trans. on Control Systems Technology*, vol 1, no 4, December 1993.

[Simon94] D. Simon, E. Coste-Manière, R. Pissard, V. Rigaud, M. Perrier, A. Peuch: "A Reactive approach to underwater Vehicle control: the mixed Orccad/Pirrat programming of the Vortex vehicle", 2^{nd} *Workshop on Mobile Robots for Subsea Environment*, International Advanced Robotics Programme, Monterey, Mai 1994.

[Simon95b] D. Simon, K. Kapellos, B. Espiau "Formal Verification of Missions and Tasks: Application to Underwater Robotics" *Int. Conf. on Advanced Robotics, ICAR'95*, Barcelona, Spain, sept. 1995.

[Simon97] D. Simon, E. Castillo and P. Freedman, "Design and Analysis of Synchronization for Real-time Closed-loop Control in Robotics", to appear in *IEEE Trans. on Control Systems Technology*