



---

# **Project SUMARE**

---

Deliverable D6.5 : ROV configuration with  
adaptive sensing and guidance

**Coordinated by: Stefan Rolfes**

**Date: September 2003**

**Participant partners: I3S** (*S. Rolfes, M.-J. Rendas*)

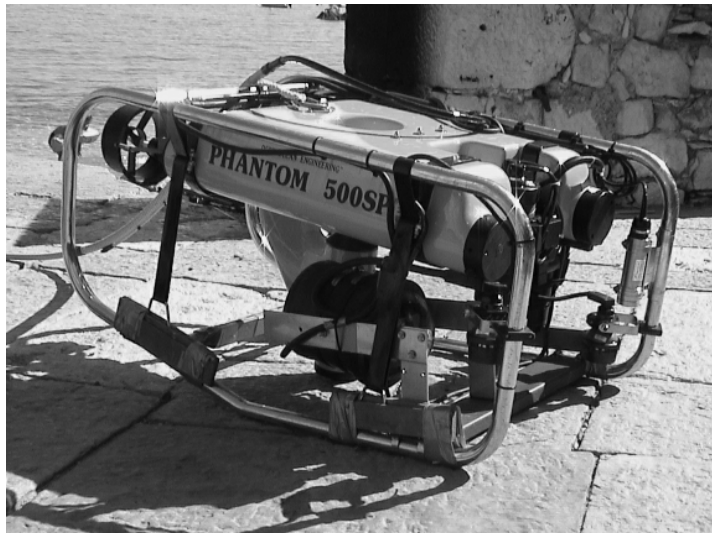
# Table of contents

- 1 Introduction ..... 3
- 2 Software configuration..... 4
  - 2.1 Multiple Client / Server application ..... 5
  - 2.2 Low-level controller ..... 5
  - 2.3 Navigation behaviours..... 6
- 3 Mission-controller ..... 8
- 4 SyncCharts ..... 9

# 1 Introduction

This deliverable reports on the configuration of the software that is actually running on the ROV Phantom (see Figure 1). The software has been modified after the Orkney campaign in 2000 (see Deliverable D8.1) in order to improve the data acquisition and the adaptive sensing and guidance system, and was shown to perform well during the second Orkney campaign in september 2002 (see Deliverable D8.2). The document is organized in 3 sections:

- *Software architecture*: This section presents the overall software architecture and presents a detailed description of the adaptive sensing and guidance sub-system.
- *Mission-controller*: This section describes the mission-controller that ensures safe sequencing of the platform operations for complex missions.
- *SyncCharts*: This section presents the kernel of the mission-controller.



**Figure 1 : The Phantom 500SP.**

## 2 Software architecture

The software architecture of Phantom is a distributed system that integrates a pool of independent applications, each providing a set of methodologies that are exported to other applications. The advantages of this architecture are that: (i) new applications can immediately use the already existing methodologies and (ii) the software can be distributed on several PCs, increasing in this way the computational power.

An overview of the actual software architecture is shown in Figure 2. The individual applications can be divided into three levels. The first level includes the data-acquisition modules which communicate directly with the available sensors. The second level includes all available Navigation systems. The last level is constituted by the low-level controllers that generate the commands to the vertical and horizontal thrusters, as well as to the tilt-platform on which the sonar is mounted.

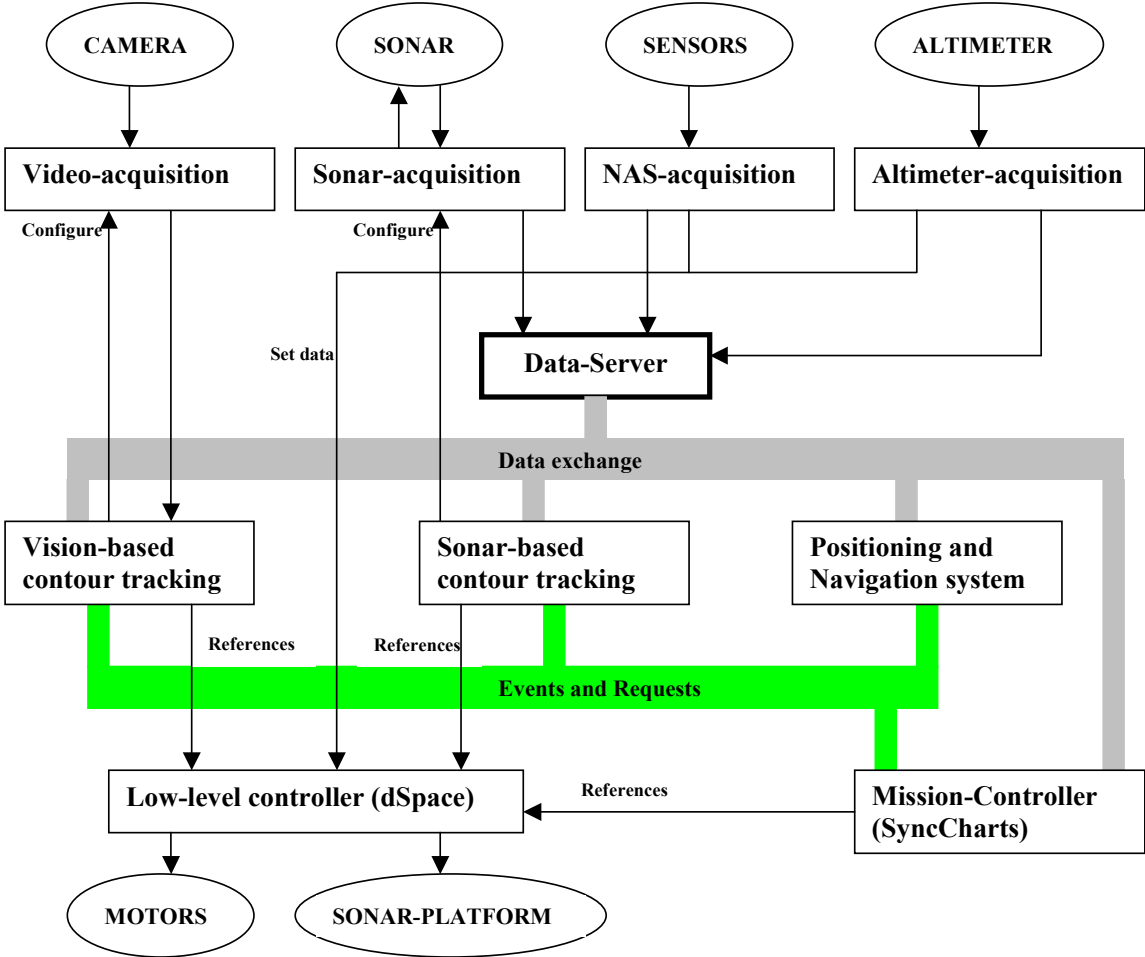
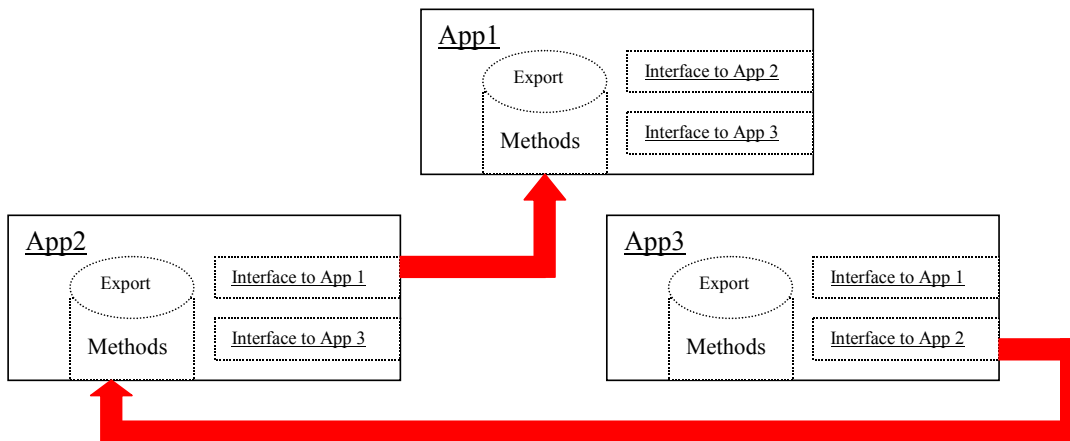


Figure 2 : Software-configuration of the PHANTOM.

## 2.1 Multiple Client / Server application

Each individual application **exports** its methods to the other applications of the system, and simultaneously **imports** required methods (see Figure 3). The entire system is thus a multiple server/client application. The methodology that has been used is **DCOM** (Distributed Component Object Model). In essence the services that are provided are not different from those provided by Sockets or pipes. The difference is that DCOM is transparent to the developer. Remote methods can be accessed by simple function calls, such that the programmer does not need to spend time on coding and decoding of the data that are transmitted between the server and the client.



**Figure 3 : Exportation and Importation of methodologies.**

## 2.2 Low-level controller

The low-level controllers run on a dSpace-board (implementation in *Simulink*) and generate commands for the vertical and horizontal thrusters according to the selected vertical and horizontal control modes and actual reference values. The interface is implemented in C++ (based on DCOM) and ensures only the communication to the acquisition and navigation systems. In the actual configuration two vertical and two horizontal controllers are implemented.

### Vertical controllers:

- *Auto-depth* (maintains the ROV at constant depth from the sea surface)
- *Auto-altitude* (maintains the ROV at constant altitude from the sea bottom)

### Horizontal controllers:

- *Auto-heading* (maintains the ROV at a desired orientation)
- *Rate-controller* (imposes the horizontal rotation rate of the ROV)

These controllers constitute the basic building blocks of all navigation systems (see Figure 2) which select the appropriate controllers and impose the required references. In addition to these control modes it is also possible to actuate directly on the thrusters, enabling the direct implementation of alternative control modes of the motors on the navigation behaviours.

### 2.3 Navigation behaviours

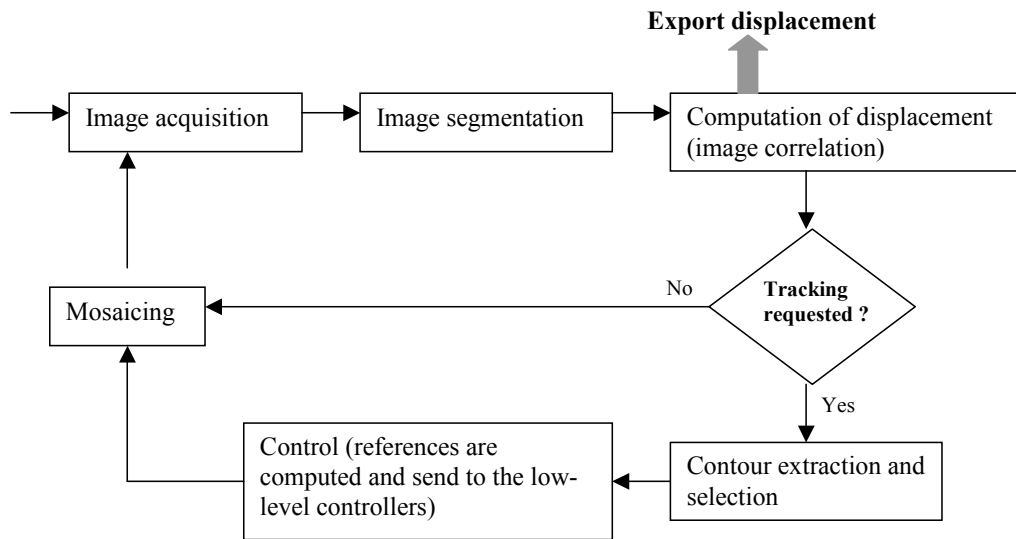
The actual software configuration of the ROV provides three different navigation behaviours:

- *Visual-based contour tracking*
- *Sonar-based contour tracking*
- *Open-loop navigation*: list of references (heading, duration, depth/altitude, speed)

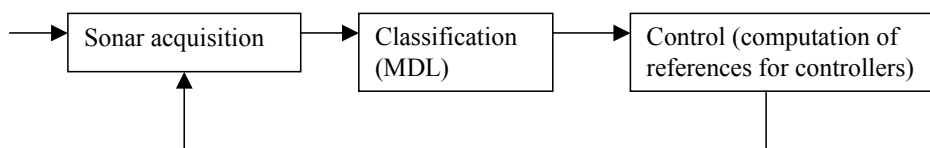
The first two navigation behaviours involve active environment perception, using either visual information or sonar data. The flow diagram of the **visual contour tracker** is shown in Figure 4. The two principal methods that are exported are *image-segmentation* and *tracking*. Both can be requested by a human operator or by the mission controller. After acquisition of the image, it is segmented (see Deliverable 3.3a) and the displacement with respect to the previous image is found by optimizing the correlation between the actual and previously segmented image. The estimated displacement is (i) exported to the positioning system that fuses the visual information and the non-acoustic sensor data in order to obtain an updated position estimate and (ii) to track a contour previously selected (best correspondance). If the tracking mode has been requested, it generates references that are issued to the low-level controllers (see Deliverable D4.1).

Figure 5 shows the flow diagram of the **sonar-based contour tracker**. A first learning step (not indicated in the figure) learns the characteristics of the sonar returns from the two materials on each side of the contour. The tracking step, triggered by the human operator, acquires the sonar data from the data-server, classifies it, and generates the references for the low-level controllers (see Deliverable D4.2). Both perception-based trackers use the rate controller .

The **open-loop** mode has been extensively used during the Orkney 2002 campaign to obtain video footage along transects or predefined patterns (see Deliverable D8.2). The execution of a user-defined mission (sequence of references) is controlled by the mission-controller and does not rely on perception based navigation systems. In this mode only the image segmentation method of the visual contour tracker has been requested in order to provide visual feedback to the operator.



**Figure 4 : Flow-diagram of the visual-based contour tracker.**



**Figure 5 : Flow-diagram of the sonar-based contour tracker.**

### 3 Mission-controller

The flow-diagram of the mission controller is shown in Figure 6. A user interfaces allows the definition of complex missions. Dependent on the mission types it imports the required methods (e.g. image segmentation, tracking, positioning,...) and exports the events that are needed to execute the mission. The kernel of the mission controller is implemented by the graphical real-time systems language SyncCharts, which ensures an appropriate reaction of the system (via requests of available methods) to incoming events that are generated either internally by the SyncCharts, or externally by the requested methods.

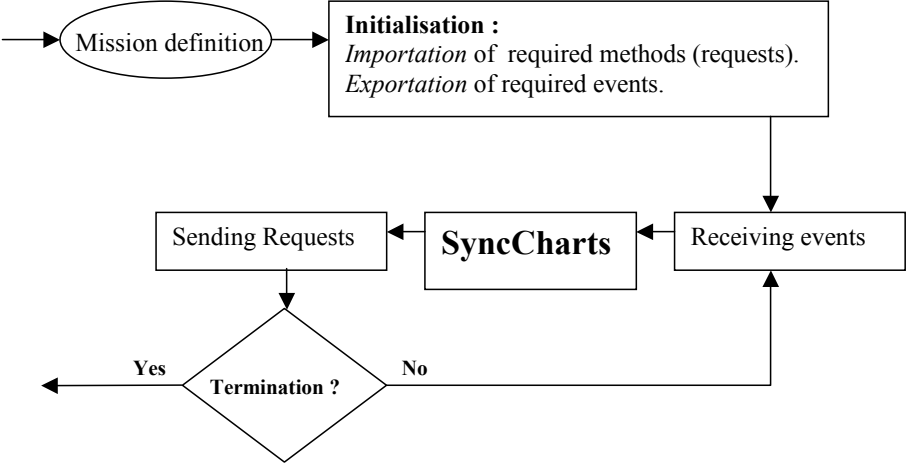


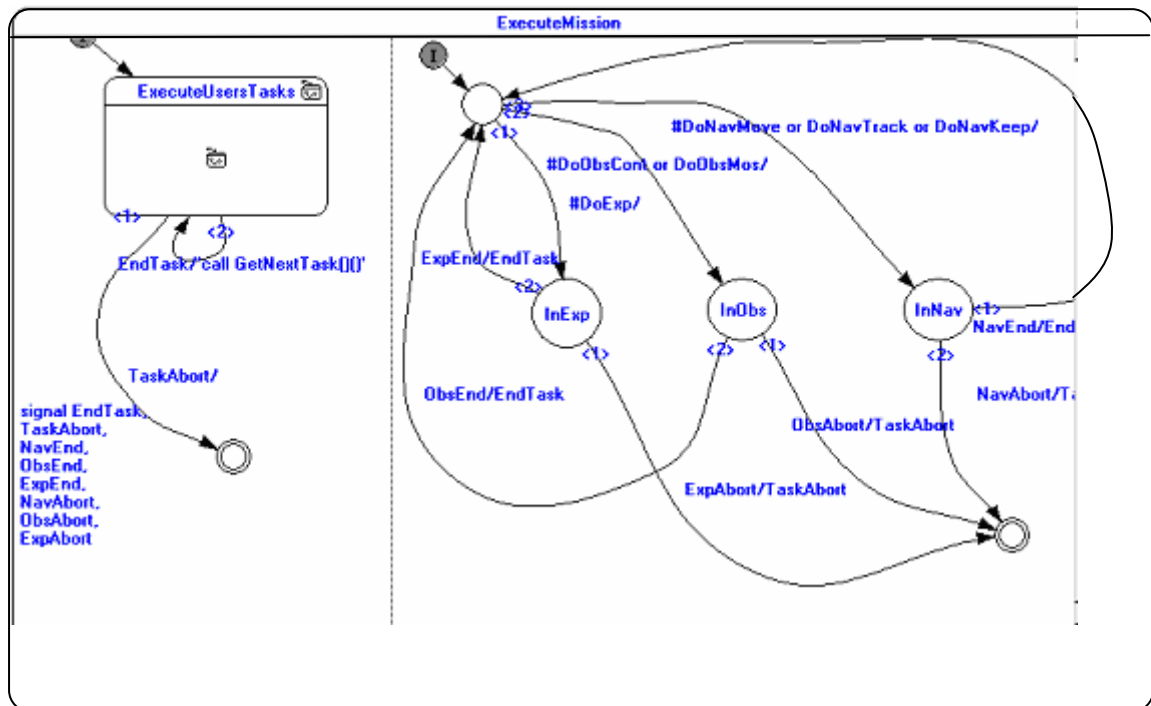
Figure 6 : Flow-diagram of the mission controller.

## 4 SyncCharts

SyncCharts are a state based visual synchronous model that is dedicated to reactive system modeling (parallel state machine). Though using a simple graphical syntax, SyncCharts may exhibit complex instantaneous behaviors, mixing concurrent evolutions. Available compilers produce a C-code that can easily be integrated to an application.

The state machine reacts to external events that are generated by the external methods. At each state the Synchroncharts generate a set of functions-calls (that are translated to requests to external methods) and ensure an appropriate reaction of the system on the available information

A graphical representation of the Synchroncharts, shown in the sequel, provides a detailed description of the reactivity of the system.



**Diagram 1 : the mission controller.**

The diagram above presents the higher hierarchical level (top macro-state) of the mission controller. It consists of two state machines that run in parallel, each one being defined on each side of the dashed vertical line inside the definition of this macro-state.

The state machine on the left shows that upon start (dark circle on the top left) the machine enters the composed macro-state *ExecuteUsersTask* (that is defined in another diagram below). Emission of the event *EndTask* makes this state machine to leave the state *ExecuteUsersTask* and re-enter that state again, while executing the method *GetNextTask* in the transition (and this guarantees that some internal variables will be correctly updated). Finally, the event *TaskAbort* (that is emitted by the state machine on the right) takes this machine to its halting state (indicated by the double circle).

This diagram also defines a number of *signals*, see the list on the left) that are visible inside this state and all other internally defined macro-states.

The state machine on the right maintains information on what kind of behavior the platform is implementing: a simple navigation behavior (like a “go to” task), an observation behavior (track a given line) or an exploration behavior (corresponding for instance to completely mapping a given area).

This component is still under development, and all interfaces with other software modules are not done yet. For this reason, we do not include here detailed description of the macro-states in the previous Figure.