Design, Development, and Testing at Sea of the Mission Control System for the MARIUS Autonomous Underwater Vehicle *

P. Oliveira, A. Pascoal, V. Silva, C. Silvestre

Institute for Systems and Robotics, Instituto Superior Técnico
Av. Rovisco Pais, 1096 Lisboa Codex, Portugal
E-mail address: antonio@isr.isr.ist.utl.pt

Abstract-This paper describes the design and development of a Mission Control System for the MARIUS AUV, and presents the results of sea tests for system design validation carried out in Sines, Portugal.

I INTRODUCTION

Among the challenges that face the designers of underwater vehicle systems, the following is of the utmost importance: design a computer based *Mission Control System* that will *i)* enable an operator to define a vehicle mission in a high level language, and translate it into a mission plan, *ii)*provide adequate tools to convert a mission plan into a Mission Program that can be formally verified and executed in real-time, and *iii)* endow an operator with the capability to follow the state of progress of the Mission Program as it is executed, and modify it if required.

Meeting those objectives poses a formidable task to underwater system designers, who strive to develop vehicles that can be programmed and operated by end-users that are not necessarily familiarized with the most intricate details of underwater system technology. Identical problems face the designers of complex robotic systems in a number of areas that include advanced manipulators, industrial work cells, and autonomous air and land vehicles. The widespread interest of the scientific community in the design of Mission Control Systems for advanced robots is by now patent in a sizeable body of literature that covers a wide spectrum of research topics focusing on the interplay between event driven and time-driven dynamical systems. The former are within the realm of Discrete Event System Theory [4], whereas the latter can be tackled using well established theoretical tools from the field of Continuous and Discrete-Time Dynamical Systems [5].

Early references in this vast area include the pioneering work of K.S. Fu [6], Saridis [9] and Albus [1], which set the ground for the study of learning control systems, intelligent machine organization, and general architectures for autonomous undersea vehicles, respectively. For an overview of recent theoretical and applied work in the field, the reader is referred to [2, 10], which contain a number of papers on the design of advanced control systems for unmanned underwater vehicles, combined underwater vehicle and manipulator systems, intervention robots, and air vehicles.

As part of the international effort to develop advanced systems for underwater vehicle mission control, IST has designed a first version of a Mission Control System for the MARIUS AUV [3], shown in Fig. 1. This paper provides a brief summary of the framework for design, analysis and implementation of the Mission Control System proposed, and reports the results of a series of sea tests for system validation conducted in Sines, Portugal. The work reported here has been influenced by the solid body of research carried out by INRIA/IFREMER in France, with applications to the VORTEX vehicle, and at NPS in the U.S. with applications to the PHOENIX vehicle, see [7] and the references therein.

The organization of the paper is as follows: Sec-

*This work was supported by the Commission of the European Communities under contract MAS2-CT92-0021 of the MAST-II programme, and by the Portuguese PRAXIS programme under contract 3/3.1/TPR/23/94.

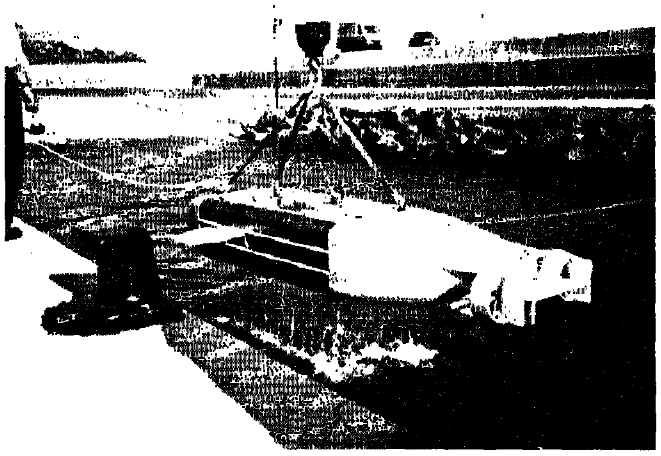


Fig. 1: The MARIUS AUV.

tion describes the basic framework adopted for Mission Control System design and implementation using the software programming environments CORAL and ATOL. Section illustrates the basic steps involved in the design of a Mission Program for a simple mission example, describes the set-up for mission execution and mission follow-up from a shore station, and reports the results of running the mission described at sea.

II Mission Control System Design

This section describes a framework for the design of Mission Control Systems for AUVs. Due to space limitations, only the key concepts will be presented here. For complete details the reader is referred to [8]. The framework proposed builds on the concepts of *System Task*, *Vehicle Primitive*, *Mission Procedure*, and *Mission Program*, that will be explained in the sequel.

System Task The concept of System Task arises naturally out of the need to organize into distinct, easily identifiable classes, the algorithms and procedures that are the fundamental building blocks of a complex Underwater Robotic System. For example, in the case of an AUV, it is convenient to group the set of all navigation algorithms to process motion sensor data into a Navigation Task that will be responsible for determining the attitude and position of the vehicle in space. A different task will be responsible for implementing the procedures for multi-rate motion sensor data acquisition. In practice, the number and type of classes adopted is dictated by the characteristics of the Robotic System under development, and by the organization of its basic functionalities, as judged appropriate by the Robotic System designer. These considerations lead naturally to the following definition:

A Vehicle System Task (abbv. *System Task - ST*) *is a parametrized specification of a class of algorithms or procedures that implement a basic functionality in an Underwater Robotic System.*

The implementation of a System Task requires the interplay of two modules: *i)* a *Functional module* that contains selected algorithms and procedures and exchanges data with other System Taks and physical devices, and *ii)* a logical *Command module*, embodied in a finite state automaton, that receives external commands, produces output messages, and controls the selection of algorithms, procedures, and data paths to and from the Functional module.

The design of the Functional module is carried out using well known tools from such diverse fields as navigation, guidance and control, instrumentation and measurements, communication theory, and computer science. The design of the Command module amounts to specifying a finite state automaton [4] that deals with the logical aspects of the System Task.

Vehicle Primitive The concept of Vehicle Primitive plays a central role in the general framework for Mission Control System design described in this paper. A Vehicle Primitive corresponds to an atomic, clearly identifiable action performed by an Underwater Robotic System, and constitutes the basic building block for the organization of complex robot missions. The following definition is offered:

A Vehicle Primitive (VP) is a parameterized specification of an elementary operation mode of an Underwater Robotic System. A Vehicle Primitive corresponds to the logical activation and synchronization of a number of System Tasks that lead to a structurally and logically invariant behavior of an underwater robot.

Associated with each Vehicle Primitive, there are sets of *pre-conditions* and *resource allocation* requirements that must be met in order for the Primitive to be activated, as well as a set of Vehicle Primitive *errors*. During operation, a Vehicle Primitive will generate messages that will trigger the execution of a number of System Tasks. The conditions that determine the occurrence of those events are dictated by the logical structure of the Vehicle Primitive itself, and by the types of message received from the underlying Vehicle System Tasks. The normal or abnormal termination of a Vehicle Primitive will generate a well defined set of *post-conditions* that are input to other Vehicle Primitives, and will release the resources that were appropriated during its execution.

By exploring the use Petri nets for the modeling of discrete event systems [4] it is possible to show that a Vehicle Primitive can be embodied in a Petri net structure defined by the five-tuple $(P_{VP}, T_{VP}, A_{VP}, w_{VP}, \mathbf{x}_{VP_0})$, where P_{VP}, T_{VP}, and A_{VP} denote sets of places, transitions, and arcs respectively, w_{VP} is a weight function, and $\mathbf{x}_{VP_0}$ is the initial Petri net marking. The set of places P_{VP} can further be decomposed as $P_{VP} = P_{pre} \cup P_{res} \cup P_{err} \cup P_{loc} \cup P_{pos}$, where $P_{pre}, P_{res}, P_{err}, P_{pos}$, and P_{loc} denote the subset of places that hold information related to the pre-conditions, resource allocation, errors, post-conditions, and the remaining state of the Petri net, respectively.

Based on the framework introduced, a Vehicle Primitive programming environment named CORAL has been developed. The left side of Figure 2 depicts the organization of the CORAL software tools that are available to *edit* and *generate* a *Library of Vehicle Primitives* which implement the complete set of atomic actions required for a specific Underwater Robotic System. Each Vehicle Primitive, embodied in its equivalent Petri net, can be input either graphically via a CORAL graphic input interface, or via a textual description using the declarative, LR1 synchronous language CORAL. A CORAL compiler/linker is in charge of accepting the vehicle primitive textual descriptions, and producing a Vehicle Primitive Library that is an archive containing the syntax and semantic descriptions of all Vehicle Primitives, as well as the data sets required for their execution.

In order to run the Vehicle Primitives described before, a CORAL Engine has been developed that accepts Vehicle Primitive descriptions and executes them in real-time.

Mission Procedure/Mission Program Given a mission to be performed by an Underwater Robotic System, the generation of the corresponding mission plan requires the availability of a set of entities aimed at specifying robot *Actions* at a number of abstraction levels. Those entities - henceforth referred to as *Mission Procedures* - allow for modular Mission Program generation, and simplify the task of defining new mission plans by modifying/expanding existing ones. The above introduction motivates the following definition:

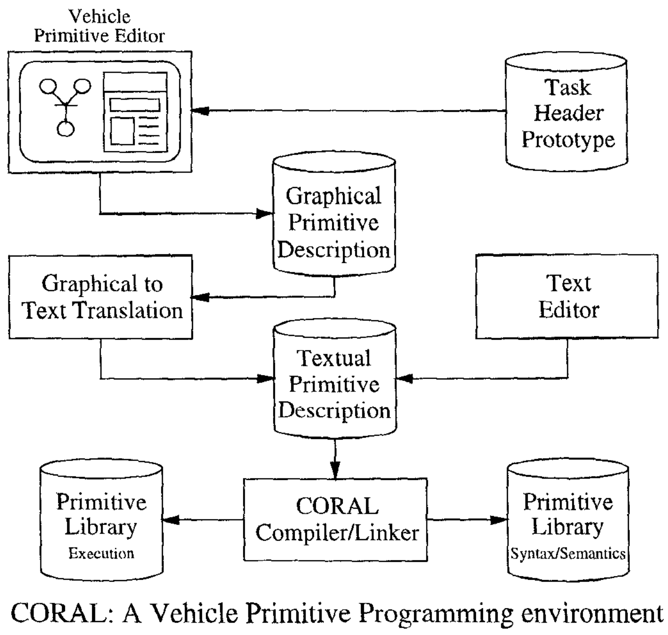
A Mission Procedure is a parameterized specification of an Action of an Underwater Robotic System. A Mission Procedure corresponds to the logical and temporal chaining of Vehicle Primitives - and possibly other Mission Procedures - that concur the execution of the specified Action.

According to the definition, the execution of a robot mission entails the execution of a number of well defined Actions specified by Mission Procedures, which in turn synchronize the operation of Vehicle Primitives. In practice, the activation of Mission Procedures and Vehicle Primitives will be triggered by conditions imposed by the mission plan structure, and by messages received from the underlying Vehicle Primitives during the course of the mission.

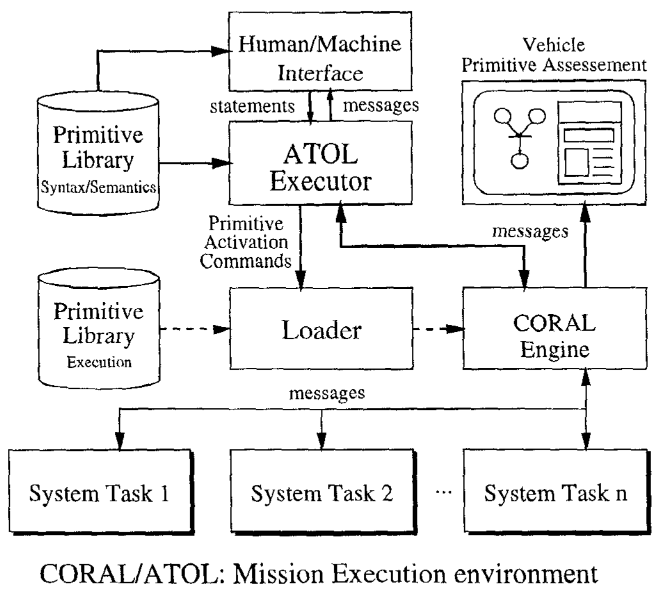
In principle, simple Mission Programs could be embodied into - higher level - Petri nets that would implement the necessary Mission Procedure structures. However the analysis of even a simple mission plan programmed using that methodology will convince the reader that the complexity of the resulting Petri net structure can become unwieldy. See Section for a detailed example. Furthermore, the approach described does not lend itself to capturing situations where the mission plan includes logical, as well as procedural statements (e.g., do loops for the repeated execution of Mission Procedures and Vehicle Primitives, etc.). These considerations motivated the need to define a specific environment for Mission Program/Mission Procedure design and implementation, named ATOL, which is currently being developed.

The framework for Mission Control System design and implementation proposed in this paper leads to the general structure of Figure 2 (right side), which captures the interaction among System Tasks, Vehicle Primitives, and Mission Program/Mission Procedures, at both programming and run-time. In the figure, the Human/Machine Interface provides the user with a text editor, and an on-line checking mechanism for the syntax and semantics of ATOL statements.

From an execution point of view, the ATOL Executor - running an ATOL Mission Program - is-



Vehicle Primitive Editor
Task Header Prototype
Graphical Primitive Description
Graphical to Text Translation
Text Editor
Textual Primitive Description
Primitive Library Execution
CORAL Compiler/Linker
Primitive Library Syntax/Semantics
CORAL: A Vehicle Primitive Programming environment



Human/Machine Interface
statements
messages
Primitive Library Syntax/Semantics
ATOL Executor
Vehicle Primitive Assessement
Primitive Activation Commands
messages
Primitive Library Execution
Loader
CORAL Engine
messages
System Task 1
System Task 2
...
System Task n
CORAL/ATOL: Mission Execution environment

Fig. 2: Mission Control System Organization.

sues commands to the CORAL Loader, which transfers selected Vehicle Primitive descriptions from the Vehicle Primitive Library to the CORAL Engine. The Engine runs the Primitives selected by interacting with the System Tasks, and issues messages that condition the execution of the ATOL Mission Program. During mission execution, the status of any Vehicle Primitive can be displayed on a *Vehicle Primitive Assessment module* that allows visualizing the flow of markings on the corresponding Petri nets.

III Mission Program Design Using Coral. Tests at Sea

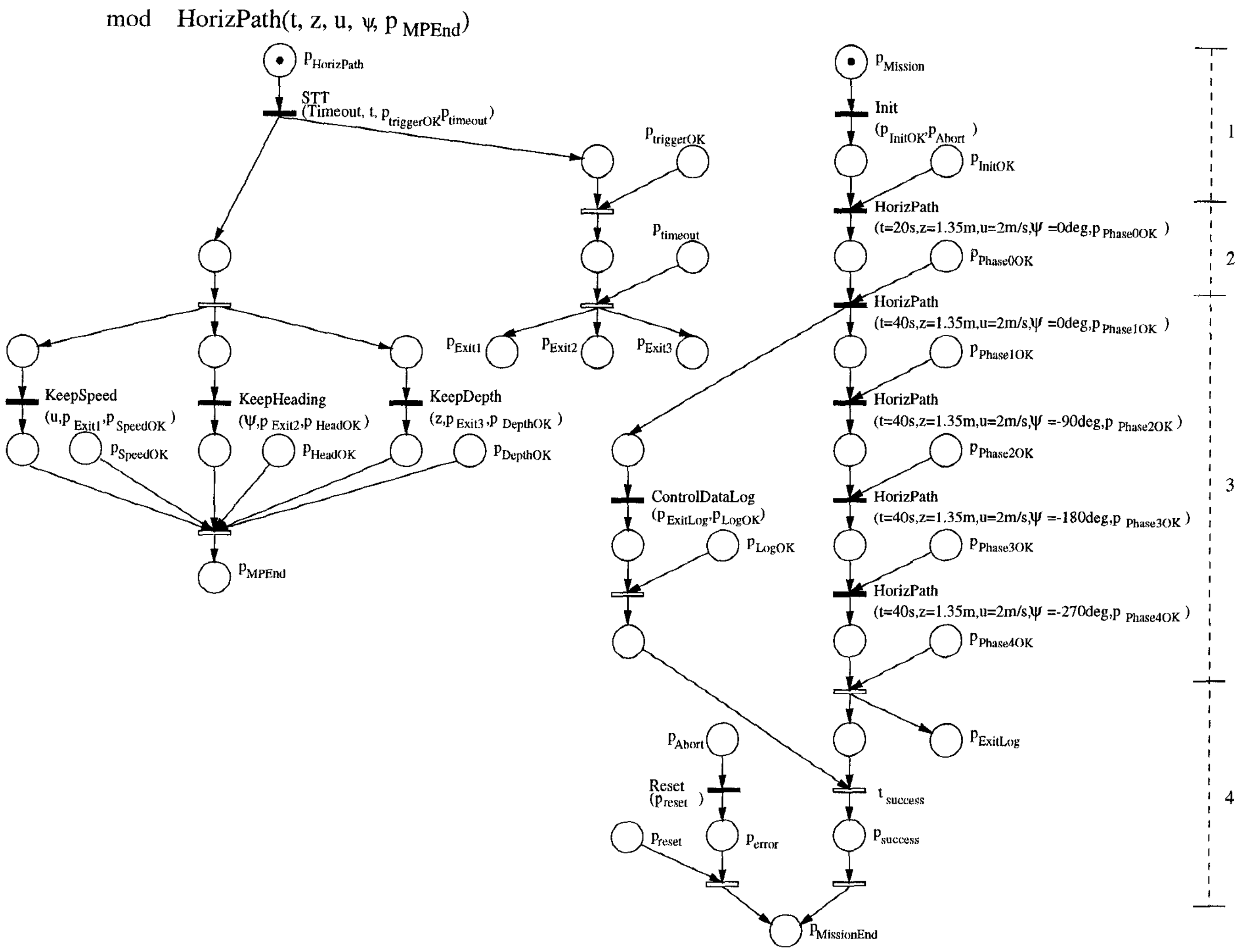
This section outlines the programming of a simple mission using CORAL, and presents the results of its execution using the MARIUS AUV [3]. The prototype vehicle is 4.5 m long, 1.1 m wide and 0.6 m high. It is equipped with two main back thrusters for cruising, four tunnel thrusters for station keeping maneuvers, and rudders, elevator and ailerons for vehicle steering in the vertical and horizontal planes. The vehicle has a dry weight of 1060 kg, a payload capacity of 50 kg, and a maximum operating depth of 600 m. Its maximum rated speed with respect to the water is 2.5 m/s. At the speed of 1.26 m/s, its expected mission duration and mission range are 18 h and 83 km, respectively.

The mission example requires that the AUV trace a square shaped trajectory, at constant depth and speed of 1.35m and 2.0m/s, respectively. The square maneuver is obtained by requesting the vehicle to change its heading by -90 deg every 40 seconds. The initial heading is 0 deg.

The design of the corresponding Mission Program involves a Mission Procedure named *HorizPath*, whose implementation using the CORAL programming environment is shown in Figure 3. This Mission Procedure parametrizes the action of keeping constant heading ψ, depth z, and speed u of the vehicle, for a period of time t.

The *HorizPath* Mission Procedure starts by setting a timer to generate a timeout after the required execution time has elapsed. This is done by issuing an timeout command with the required Mission Procedure duration time t. To perform the maneuver, three Vehicle Primitives are called in parallel: *KeepSpeed* with a velocity set-point u, *KeepDepth* with a depth set-point of z, and *KeepHeading* with a heading set-point of ψ. The generation of a timeout terminates the execution of *HorizPath* by exiting the three Vehicle Primitives.

The Mission Program can be explained with the help of Figure 3, which shows four distinct phases:



mod HorizPath(t, z, u, ψ, p MPEnd)
p HorizPath
STT (Timeout, t, p triggerOK p timeout)
p triggerOK
p timeout
p Exit1
p Exit2
p Exit3
KeepSpeed (u, p Exit1, p SpeedOK)
p SpeedOK
KeepHeading (ψ, p Exit2, p HeadOK)
p HeadOK
KeepDepth (z, p Exit3, p DepthOK)
p DepthOK
p MPEnd
p Mission
Init (p InitOK, p Abort)
p InitOK
HorizPath (t=20s,z=1.35m,u=2m/s,ψ =0deg,p Phase0OK)
p Phase0OK
HorizPath (t=40s,z=1.35m,u=2m/s,ψ =0deg,p Phase1OK)
p Phase1OK
HorizPath (t=40s,z=1.35m,u=2m/s,ψ =-90deg,p Phase2OK)
p Phase2OK
HorizPath (t=40s,z=1.35m,u=2m/s,ψ =-180deg,p Phase3OK)
p Phase3OK
HorizPath (t=40s,z=1.35m,u=2m/s,ψ =-270deg,p Phase4OK)
p Phase4OK
ControlDataLog (p ExitLog, p LogOK)
p LogOK
p ExitLog
p Abort
Reset (p reset)
p reset
p error
t success
p success
p MissionEnd
1
2
3
4

Fig. 3: Mission Procedure and Mission Program in CORAL.

in phase 1, all vehicle System Tasks are initialized by calling the *Init* Vehicle Primitive; in phase 2, the *HorizPath* Mission Procedure is called for a period $t = 20$ s, with a velocity set-point of $u = 2$ m/s, a depth set-point of $z = 1.35$ m, and an heading set-point of $\psi = 0$ deg. At the end of this phase, the vehicle is headed north, and ready to start the required square maneuver; phase 3 calls the *HorizPath* Mission Procedure repeatedly, with heading set-points of 0deg, -90deg, -180deg, and -270deg, while maintaining the remaining input set-points equal to those in phase 2. The required duration of each Mission Procedure call is $t = 40$ s. In parallel, the Vehicle Primitive *ControlDataLog* is called to start logging control loop data for later off-line analysis.

In order to assess the performance of the Mission Control System of MARIUS, a series of tests were conducted at sea in Sines, Portugal, in January 1996. The tests included programming and running the mission described above. Throughout the mission, the vehicle pulled a buoy with an antenna, thus enabling radio communications between the vehicle and a shore station. The software for Mission Control was run on the computer network installed on-board the AUV. Figures 4 through 7 display some of the data acquired in the course of the mission, which was executed to perfection. Figures 4 and 5 show the commanded and measured heading, and the rudder activity, respectively. Figures 6 and 7 show the slight variations in heading and depth due to the wave action in shallow water.

References

[1] J. Albus, "System Description and Design Architecture for Multiple Autonomous Undersea Vehicles," National Institute of Standards and Technology, Technical Note 1251, September 1988.

[2] P. Antsaklis, K. Passino, *An Introduction to Intelligent and Autonomous Control*, Kluwer Academic Publishers, 1993.

[3] G. Ayela, A. Bjerrum, S. Bruun, A. Pascoal, F-L. Pereira, C. Petzelt, J-P. Pignon, " Development of a Self-Organizing Underwater Vehicle - SOUV, *Proceedings of the MAST-Days and Euromar Conference*, Sorrento, Italy, November 1995.

[4] C. Cassandras, *Discrete Event Systems. Modeling and Performance Analysis*,Aksen Associates Incorporated Publishers, 1993.

[5] G. Franklin, J. Powell, M. Workman, *Digital Control of Dynamic Systems*, Addison-Wesley, 1990.

[6] K.S. Fu,"Learning Control Systems-Review and Outlook," *IEEE Transactions on Automatic Control*, Vol.AC-15, No.2,1970

[7] M. Lee, R. McGhee, Editors, Proceedings of the IARP 2nd Workshop on Mobile Robots for Subsea Environments, Monterey, California, May 1994.

[8] P. Oliveira, A. Pascoal, V. Silva, C. Silvestre, "Design, Development, and Testing of a Mission Control System for the MARIUS AUV", *Proceedings of the 3rd Workshop on Mobile Robots for Subsea Environments*, Toulon, France, March 1996.

[9] G. Saridis, "Analytical Formulation of the Principle of Increasing Precision with Decreasing Intelligence for Intelligent Machines,"*Automatica*, Vol. 25, pp. 461–467.

[10] K. Valavanis, G. Saridis, A. Pascoal, P. Lima, F-L. Pereira, editors. *Proc. of the Joint U.S./Portugal Workshop on Undersea Robotics and Intelligent Control*, Lisbon, Portugal, March 1995.

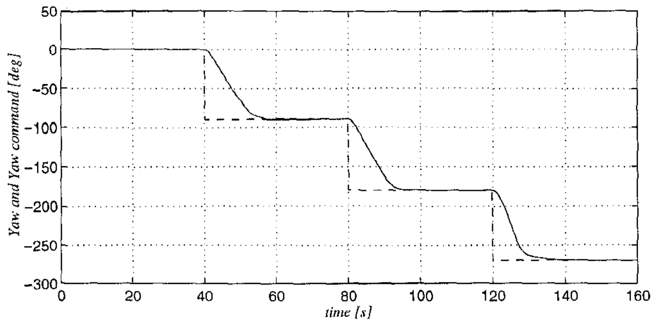


Fig. 4: Commanded and measured heading.

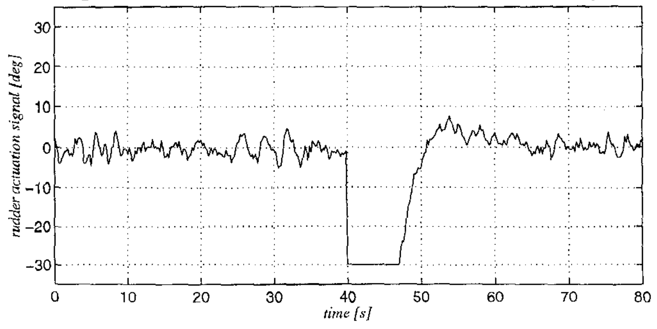


Fig. 5: Rudder deflection.

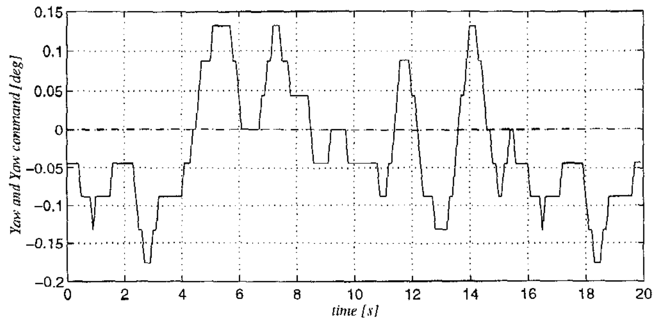


Fig. 6: Measured heading (zoom in).

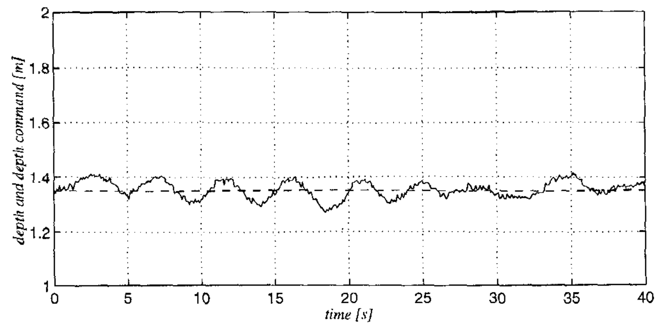


Fig. 7: Commanded and measured depth.