UUV TASK DEFINITION USING METAPROGRAMMING

Milton Yukio Godoy Saito, saito.milton@gmail.com
Fabio Kawaoka Takase, fktakase@usp.br
Newton Maruyama, maruyama@usp.br
Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos
Escola Politécnica da Universidade de São Paulo
Av. Prof. Mello Moraes, 2231, 005508-900, São Paulo, Brasil

Abstract. *In this work, a metaprogramming language is proposed for task specification of an unmanned underwater vehicle (UUV) named VSOR. The VSOR is an open frame type UUV with pressure vessels for embedded electronics and batteries, and a set of eight undersea thrusters. The vehicle has been devised to provide inspection and intervention capabilities in deep water oil field missions. The software consists of two parts, the first one interacts with the user through a programming language and the second one implements a command line interpreter that can send commands to low-level layers of the embedded system software. The use of metaprogramming is specially usefull when users must program the VSOR to perform complex tasks without knowing the low level programming details of the embedded system. The metaprogramming system has been initially used to specify vehicle maneuvering in test tank trials. The tasks specify different optimal maneuvers, that must be accomplished by the vehicle, in order to estimate its hidrodynamic coefficients (both drag coefficients and added mass inertia). The metaprogramming system offers a flexible environment which allows easy modification of instructions and parameters during experimental trials. These features allow much faster deployment of task specifications when compared to customised programming solutions that need to access low-level layers of the embedded system software.*

Keywords*: metaprogramming, scripts, programming, task definition, unmanned underwater vehicle.*

1. INTRODUCTION

Nowadays, the brazilian oil production in offshore fields represents a significant portion of the total oil production. Although costly, deepwater exploration - water depths of more than 400m - is already successful. Nevertheless, there is a great technological demand arising from the need to reduce deepwater production costs and to enable oil production in new offshore fields which have been discovered in ultra-deepwaters (3000m). In this context, a semi-autonomous unmanned underwater vehicle (UUV), named VSOR, is being developed at the Mechatronics Engineering Department at the University of Sao Paulo. The name VSOR is an acronym for *Veículo Submarino Operado Remotamente*, (*Remotely Operated Underwater Vehicle*). The classification of the VSOR, an open frame UUV type, as semi-autonomous is due to the combination of a remotely operated mode together with some degree of autonomous behaviour. The autonomous behaviour is provided by a real-time embedded software that performs sensor data acquisition and can generate trajectory planning and position control when close to the target. Due to limited space in the pressure vessels for embedded electronics and the severe energy consumption constraints, only one single board PC-104 computer with several I/O interfaces is utilised. Embedded software with hard constraints has been historically designed as custom solutions, using techniques such as the *executive while* (Burns, 2001). In the *executive while* the process scheduling is a priori estimated. Each process is given a set of time slots in a way that all processes are able to respect their deadlines. In the case of a complex embedded system as the VSOR, this hard coded solution is not adequate as the inclusion of new sensors and actuators would imply in recalculating the a priori scheduling. In order to accommodate new hardware and software configurations, the embedded system software is developed using a Real Time Operating System, the VxWorks from WindRiver Inc. The set of concurrent processes that composes the embedded system software are developed independently, and with priorities assigned, the VxWorks can handle the process scheduling. Although much more flexible, this solution still demands a great attention on process scheduling, process priorities, process interactions and resource allocation in order to meet process deadlines (Ramamritham and Stankovic, 1994). Usually, the control system designer does not have a fine knowledge about the embedded system low-level programming details. In this work, a new metaprogramming language for task specification of the VSOR is introduced. The metaprogramming system has been initially used to specify vehicle maneuvering in test tank trials. The tasks specify different optimal maneuvers that must be accomplished by the vehicle, in order to estimate its hidrodynamic coefficients (both drag coefficients and added mass inertia). The metaprogramming system offers a flexible environment which allows easy modification of instructions and parameters during experimental trials. These features allow much faster deployment of task specifications when compared to customised programming solutions that need to access low-level layers of the embedded system software.

This paper is organized as follows. Section 2 summarises the mechanical design of the VSOR and its thruster system. Sections 3 and 4 present the control architecture and the sensor suite. Section 5 follows with the presentation of the metaprogramming language. Section 6 describes the basic procedures for programming the VSOR using the metaprogramming language. Section 7 presents some experimental results. And finally, some conclusions about the feasibility of the method are drawn in Section 8.

2. THE VSOR DESIGN: Mechanical Design and Thruster System

The vehicle is constructed with aluminum tubular structure with the following dimensions, $l = 1.4m$ x $w = 1.2m$ x $h = 0.9m$, equipped with three pressure vessels with the same dimensions, $l = 1.0m$ x $d = 0.167m$. Its weight in air is about 200Kg and the weight buoyancy force is 2Kgf positive.

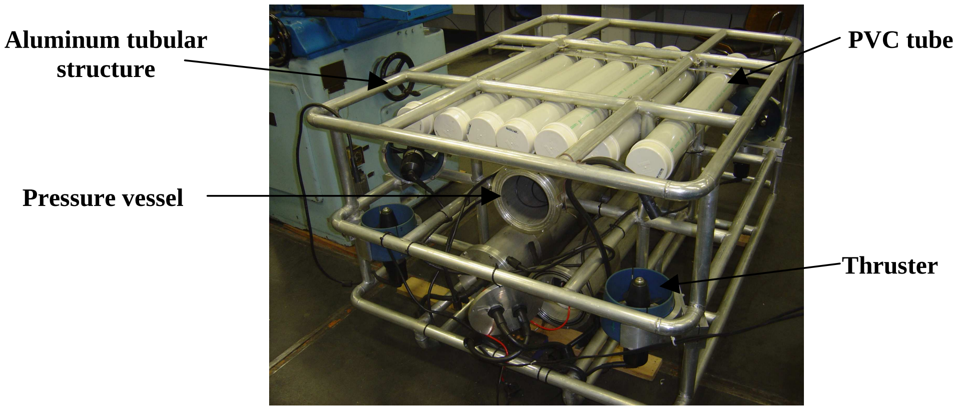


Figure 1. Physical layout of the VSOR.

On the top of the VSOR chassis there is a layer of PVC tubes for buoyancy properties, a pressure vessel for the electronics and sensors, and the four horizontal thrusters located in the corners (see Fig. 1). The bottom part of the vehicle chassis consists in two pressure vessels that contain batteries and four vertical thrusters arranged in the corners. A small vessel $l = 0.15m$ x $d = 0.12m$ is localised approximately in the vehicle mass center carries an Inertial Measurement Unit. Modular structural components allow that the vehicle can be easily reconfigured. The overall structure of the vehicle is symmetric with respect to both the *xz* and *yz* planes. This particular thruster configuration enables full controllability of the vehicle motion.

3. CONTROL ARCHITECTURE

The VSOR control architecture, Fig. 2, consists of two main modules: the surface computer system and the sub-sea computer system. The surface system (host) is a standard IBM-PC computer running Windows XP OS. This computer is responsible for the task-level control commands and it is connected to the sub-sea computer through a Ethernet cabling (4-26 AWG cable) using a TCP/IP stack. The sub-sea computer system is responsible for the sensor system processing (sensor raw data acquisition and position and velocity estimation) and the low-level control system. It consists of an embedded computer system with a set of PC-104 type boards that runs the VxWorks Real-Time operating system (VxOs). Apart from the CPU board with a low power NS Geode running on 300MHz, the set of PC-104 boards includes an A/D board, a D/A board and a multi-serial board. This computing platform provides a stable and familiar programming environment (Altshuler, 2003).

The programs and the VxOS kernel are built in the surface computer using the WindRiver Tornado IDE and downloaded to the sub-sea computer system. Historically, software written for embedded systems has been designed using a simple control loop under the *while executive* concept. The loop calls subroutines, each of which manages part of the hardware or the software. This hardware oriented custom software construction is at the base of proposals on the hardware & software co-design (Coumeri & Thomas). In the case of a complex system such as the VSOR, this hard coded solution is not adequate as the need of inclusion of new sensor and actuators are always likely. The VSOR requires a preemptive multitasking software architecture, i.e., tasks must switch based on timing and priorities. In VxOS, each task is assigned a priority, and the scheduler ensures that the CPU is allocated to the highest-priority task that is on ready state (WindRiver, 2002). The scheduling algorithm tries to accomplish with all deadline constraints, responding for particular events (interrupt and task switching, for example) with minimal latency. The complexities of managing multiple tasks running seemingly concurrently are solved therefore by the Real-Time Operating System Scheduler.

The VSOR embedded system software modules are designed to run on a VxOS microkernel. It consists of a set of sensors and actuators software interfaces and a control process that is responsible for:

- Scheduling each sensor and actuator concurrent processes;
- Managing communication with the remote control station;
- Storing sensors and actuators data into an embedded hard disk;
- Calculating position and velocity feedback loops;
- Coordinating the global test.

When the tests take place, all data are stored into an embedded hard disk for offline analysis.

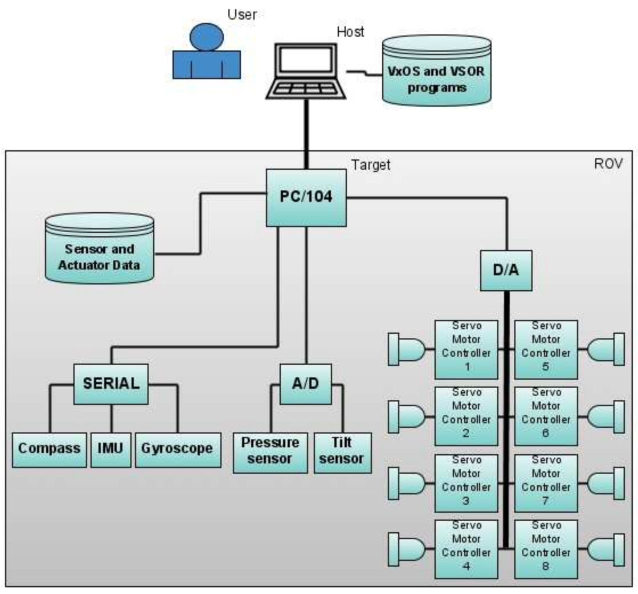


Figure 2. Control architecture schematic diagram.

4. SENSOR SUITE

The VSOR sensor suite is composed by five different sensors that are summarised in Table 1.

Table 1 – Sensor system.

Variable	Sensor(Manufacturer)	Precision	Update Rate	Output	Inteface
Heading	Compass TCM2 (PNI)	≈1º	13Hz	Digital	RS-232
Roll and Pitch	Tilt Series 757 (Applied Geomechanics)	≈2º	20Hz	Analog	A/D card
Depth	Pressure Sensor, MPX5100DP (Motorola)	3.5cm	20Hz	Analog	A/D card
Yaw Rate	Fiber Optic Gyro, E-Core 2000 (KVH)	Bias < 2º/h	10Hz	Digital	RS-232
Linear Acceleration	Inertial Measurement Unit, VG700A (Crossbow)	Bias < 12mg	100Hz	Digital	RS-232
Angular Velocity	Inertial Measurement Unit, VG700A (Crossbow)	Bias < 20º/h	100Hz	Digital	RS-232

For each sensor an interface software module is implemented allocating the required resources, such as physical memory used as circular buffers and a set of I/O and timer interrupts.

5. METAPROGRAMMING

One of the main goals of the use of metaprogramming is to let users work at a higher level of abstraction than if they decide to use hard coded solution. Essentially, metaprogramming is formally defined as writing computer programs that write or manipulate other programs. The use of a metalanguage, in which the user writes verbose statements to describe complex tasks in a human-readable fashion, simplifies the programming task, and broadens the range of actual users of the system. The VSOR embedded system software is therefore implemented to run programs written in a metalanguage organized in scripts.

Scripting programming languages, also called scripting languages or script languages, are computer programming languages that do not require compilation, i.e., they are typically interpreted. Thus, scripts are often distinguished from programs, because programs are converted permanently into binary executable files before they can run. Scripts remain in their original form and are interpreted command-by-command each time they run. So, script reduces the traditional edit-compile-link-run process, as a result, it automates tests when there is a list of variables to be declared and updated frequently.

In this work, metaprogramming is understood as writing of a set of verbose commands into scripts that specify tasks that the VSOR must perform.

6. BASIC PROCEDURES FOR PROGRAMMING

It might be worthwhile to express instances of the problem as sentences in a simple but powerfull language, which means allowing task specification into higher abstraction levels. In order to do this some basic metalanguage instructions are created:

- Input thrusters signals: cosine, step, cosine-step combinations and respective parameters;
- Tests control signals: timer, start, abort and stop execution test;
- Basic services*:* sample signal visualization and process control;
- VSOR motion: linear and rotational motion, and control tests using PID controllers;
- The execution flow control and conditional statements were also included in the VSOR test instruction set.

Basically, the grammar defines regular expressions of the VSOR metalanguage instructions, defined as the following (Gamma, 2001):

<expression> <parameter_1> <parameter_2> ... <parameter_N>

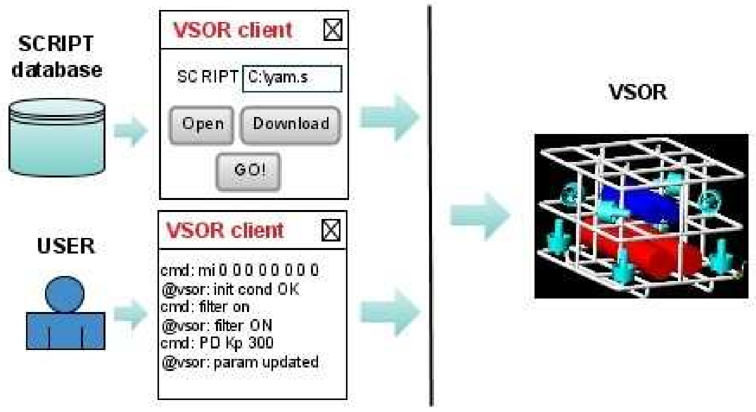
The symbol *expression* is the start symbol, that defines which instruction will be executed, than *parameter_1*, *parameter_2,…, parameter_N* are the instruction related parameters.

For example:

ms <sign> <amplitude> <period> <shift>

The **ms** expression defines a thruster input signal instruction. The *sign* symbol selects which signal type (i.e., sine, step or both combined). Depending of the *sign* selected, it is necessary to provide a list of parameters. For example: ***ms sin 10 8 2*** (a sine signal with amplitude: 10volts, period: 8seconds and shift: 2volts).

The system offers flexible commands to control the VSOR. There are two methods for test execution: send instructions directly to the VSOR embedded system and through instruction scripts. The last method is used to automate complex maneuvering tests. Sending command-by-command is useful to customise tests, debug the system and make interventions during execution. Fig. 3 shows the ways to interact with the VSOR during tests. In Fig. 3, the top VSOR client window refers to an example of script use. And the bottom VSOR client window is an example of a direct interaction with VSOR, You must note that the embedded system send messages to notify the user about the request status.



```
cmd: mi 0 0 0 0 0 0 0 0
@vsor: init cond OK
cmd: filter on
@vsor: filter ON
cmd: PD Kp 300
@vsor: param updated
```

Figure 3. Different methods to interact with the VSOR during experimental tests.

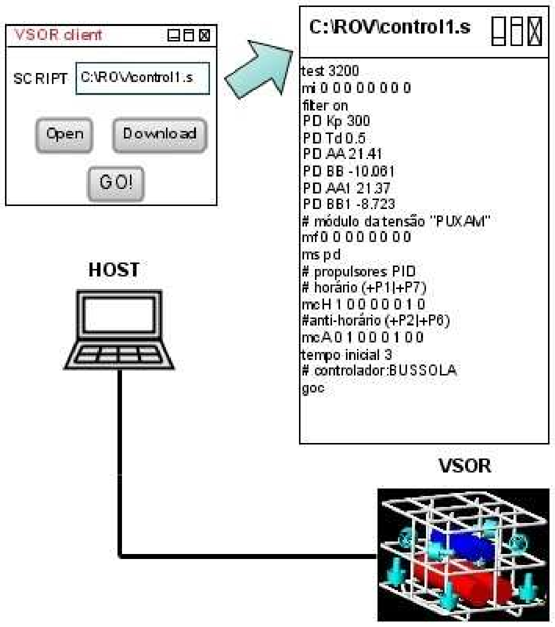
7. EXPERIMENTAL RESULTS

Two examples extracted from the trials that have been conducted in the Naval and Oceanic Engineering Department test tank are shown in the next two subsections. The main purpose of those tests was the estimation of hydrodynamic coefficients of the UUV and this work will focus on the use of metaprogramming and scripts.

7.1. Proportional-Derivative control test

This experimental trial consisted of a Proportional-Derivative (PD) control test. The main goal in this experiment is to maintain the VSOR geo-orientation. The initial geo-orientation reference, called by *set point*, is set manually sending an instruction directly to the VSOR before running the test.

The PD test is programmed like in the script illustrated in Fig. 4. The script contains all parameters and conditions required by a usual test, such as name of the log file and start conditions. In addition, coefficients values are defined and all thrusters that can be used by the feedback controllers are setup. Basically, the PD controller calculates the control effort output based on the compass heading signal. The calculated control action is converted into an equivalent analog signal and then send to the specified thrusters.



```
test 3200
mi 0 0 0 0 0 0 0 0
filter on
PD Kp 300
PD Td 0.5
PD AA 21.41
PD BB -10.061
PD AA1 21.37
PD BB1 -8.723
# módulo da tensão "PUXAM"
mf 0 0 0 0 0 0 0 0
ms pd
# propulsores PID
# horário (+P1|+P7)
mcH 1 0 0 0 0 0 1 0
#anti-horário (+P2|+P6)
mcA 0 1 0 0 0 1 0 0
tempo inicial 3
# controlador:BUSSOLA
goc
```

Figure 4. General view of a PD test.

Figure 5 illustrate results achieved in the PD test. The plotted curve indicates that, in the beginning of the test, the VSOR angular orientation is not the same as the desired setpoint (314º). So, the test started with the controller being saturated. As a result, the vehicle heading orientation has a fast change. The plot illustrates that the VSOR heading orientation converges to 314º.

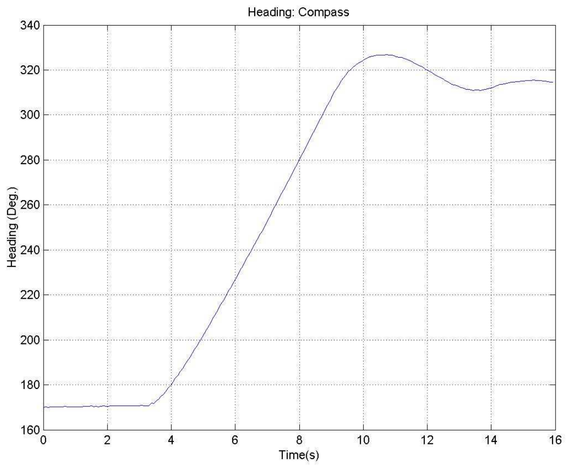


Figure 5 - Experimental heading data from compass

7.2. Sinusoidal control experiment

In order to obtain hydrodynamic coefficients, it is necessary to apply specific torque profiles, such as sinusoidal torque. This torque is created by applying a sinusoidal signal into a pre-specified set of thrusters. With the correct torque profile the VSOR can spin 360 degrees around itself and can turn back to the original position. In Fig. 6, the torque that results from a sinusoidal input signal are illustrated. The torque is estimated using the yaw rate signal provided by the gyroscope.

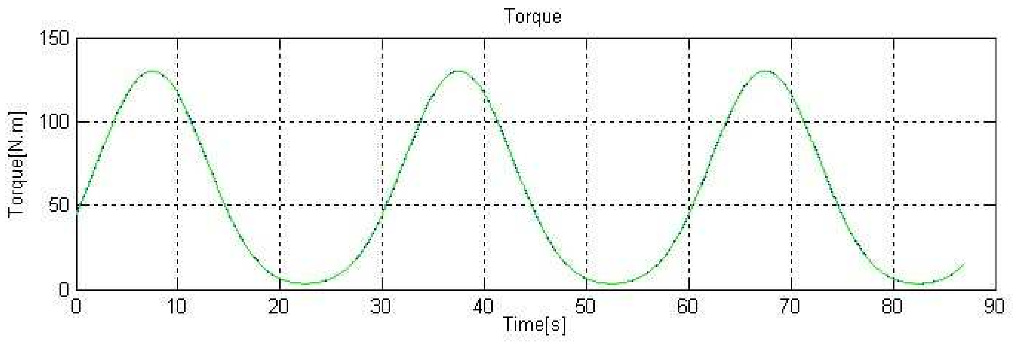


Figure 6. VSOR torque during a rotation experiment.

8. CONCLUSIONS

In this work, an UUV embedded control system has been presented. Users without concurrent programming skills have a mean to define complex tasks for the VSOR. This has been achieved successfully using metaprogramming and scripts. Experimental trials performed in the test tank have been successful and allowed the acquisition of a great amount of experimental data in a short time, proving to be an efficient tool for task definition and automation.

The use of metaprogramming to define tasks hid from the task programmer the complexities of the computational implementation of the concurrent tasks of the embedded software, such as:

- The policies to write to the log file the data sampled from sensors. This task is I/O expensive and have to be well managed in order not to consume too much CPU time.
- The policies to write and read from/to the circular buffers configured for each sensor. The configuration of the circular buffer size and its access to write and read must be well balanced as the writing access must be performed using hardware interruptions and reading access must be synchronized with the log file access.
- Complex algorithms such as the prediction algorithm used to overcome the limited update rate of the compass caused by the embedded Earth electromagnetic field effect compensation of the compass. Without this prediction algorithm the sinusoidal control experiment could not be performed.
- Careful implementation of reference signals to avoid drift problems using absolute time references.

As result from this computational implementation details hiding the metaprogramming proved to be a very useful mean to define tasks to the VSOR in the experimental tests performed in the test tank. The following steps in this research and development field are the implementation of a planner task and a monitor task that interacts deliberatively with the VSOR through metaprogramming.

9. ACKNOWLEDGEMENTS

The authors would like to thank the CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) for the financial support (process no. 550934/2005-7), the CENPES-PETROBRAS research center for the logistic support, the FDTE (*Fundação para o Desenvolvimento Tecnológico de Engenharia*) for the awarded scholarship and the WindRiver Inc. for the VxWorks 5.5.1 and the Tornado 2.2.1 academic licenses.

10. REFERENCES

Altshuler, R.C., Apgar, J.F., Ashford, A.C., Broxton, M.J., Edelson, J.S., Khan, C.J., Khripin, A., Knaian, A.N., Kraft, A.D., Lovell, S.D., Mcletchie K.W., Mazzone, L.A., Newburg, S.O., Rorschach, K.L., Stark, J.C., Horng, D.E., Uechi, K.A., 2003. "ORCA-VI: An Autonomous Underwater Vehicle". Massachusetts Institute of Technology.

Burns, A., Wellings, A., 2001. "Real Time Systems and Programming Languages: Ada 95, Real-Time Java and Real-Time C/POSIX" , Addison-Wesley, 3rd Edition,.

Coumeri, S.L. e Thomas, D.E, 1995. "A Simulation Environment for Hardware-Software Codesign". Proc. of the 1995 International Conference on Computer Design: VLSI in Computers and Processors, pp. 58-53, Oct. 1995.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 2001. "Design Patterns". Ed. Addison-Wesley.

Ramamritham, K., Stankovic, J.A, 1994. Scheduling algorithms and operating systems support for real-time systems. Proc. IEEE, vol.82, no.1, pp.55-67, Jan. 1994.

WindRiver, 2002. "User's guide".

11. RESPONSIBILITY NOTICE

The authors are the only responsible for the printed material included in this paper.