






Article

# Development of a Modular Software Architecture for Underwater Vehicles Using Systems Engineering

Carlos A. Zuluaga <sup>1</sup>, Luis M. Aristizábal <sup>1</sup>, Santiago Rúa <sup>2</sup>, Diego A. Franco <sup>1</sup>, Dorie A. Osorio <sup>1</sup>  
and Rafael E. Vásquez <sup>1,\*</sup>

<sup>1</sup> School of Engineering, Universidad Pontificia Bolivariana, Medellín 050031, Colombia; carlos.zuluaga@upb.edu.co (C.A.Z.); luismiguel.aristizabal@upb.edu.co (L.M.A.); diego.franco@upb.edu.co (D.A.F.); dorie.osorio@upb.edu.co (D.A.O.)

<sup>2</sup> Electronics and Telecommunications Engineering Department, Universidad de Medellín, Medellín 050026, Colombia; srua@udemedellin.edu.co

\* Correspondence: rafael.vasquez@upb.edu.co; Tel.: +57-4-4488388 (ext. 14165)

**Abstract:** This paper addresses the development of a modular software architecture for the design/construction/operation of a remotely operated vehicle (ROV), based on systems engineering. First, systems engineering and the Vee model are presented with the objective of defining the interactions of the stakeholders with the software architecture development team and establishing the baselines that must be met in each development phase. In the development stage, the definition of the architecture and its connection with the hardware is presented, taking into account the use of the actor model, which represents the high-level software architecture used to solve concurrency problems. Subsequently, the structure of the classes is defined both at high and low levels in the instruments using the object-oriented programming paradigm. Finally, unit tests are developed for each component in the software architecture, quality assessment tests are implemented for system functions fulfillment, and a field sea trial for testing different modules of the vehicle is described. This approach is well suited for the development of complex systems such as marine vehicles and those systems which require scalability and modularity to add functionalities.

**Keywords:** marine robotics; remotely operated vehicle; systems engineering; software architecture; marine engineering



**Citation:** Zuluaga, C.A.; Aristizábal, L.M.; Rúa, S.; Franco, D.A.; Osorio, D.A.; Vásquez, R.E. Development of a Modular Software Architecture for Underwater Vehicles Using Systems Engineering. *J. Mar. Sci. Eng.* **2022**, *10*, 464. <https://doi.org/10.3390/jmse10040464>

Academic Editors: Angelo Odetti, Gabriele Bruzzone and Roberta Ferretti

Received: 28 February 2022

Accepted: 22 March 2022

Published: 25 March 2022

**Publisher's Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Copyright:** © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

The development of modern technologies that have been used in ocean exploration, including unmanned marine vehicles, has allowed reaching inaccessible regions all around the world, which is helpful for the comprehensive study of the ocean [1]. These robotic vehicles have been mostly divided into three types: remotely operated (ROV) [1,2], autonomous (AUV) [3,4], and surface (USV) [5,6] vehicles, and, as is usual in different robotic devices, all of them need the combination of mechanics, hardware, software, and control strategies to be developed and appropriately integrated. This combination of disciplines has been also proven to be useful for other types of marine systems such as advanced monitoring environments [7], aerial systems [8], electric ships [9], cleaning robots [10], alarm systems [11], multirobot systems [12,13], and research platforms [14], among others. Hence, the development of modern marine systems and vehicles represents an open problem, which depends on the operational conditions and user requirements.

The development of underwater vehicles during the last five years has increased considerably for different applications, depending on the type of missions that are to be executed [15]. For AUVs, one can find reports with vehicles to be used in ice-covered oceans [16], pipeline detection [17], rapid and accurate vehicles [18], archaeological survey [19], harsh environments inspection [20], ship hull inspection [21], deep-sea exploration [3], and hydrological surveying [22], among others. For ROVs, the last five years'

developments include low-cost ROV prototypes [23], fault-tolerant control systems [2], small ROV for assessing radiation [24], deep-ocean research [25], energy-efficient vehicles [26], hybrid vehicles that can work either as ROV or AUV [27], and motion feasibility frameworks based on industry standards [28].

Systems engineering (SE) has been defined by NASA [29] as “a methodical, multi-disciplinary approach for the design, realization, technical management, operations, and retirement of a system” and has been identified to be a key discipline to enable the interactions of components that provide functionality within a complex system [30,31] that is expected to meet several requirements [29]. Some robotic systems developments that have used SE by considering functional requirements [32,33] include, among others, unmanned aerial systems [34], quadrotor aerial vehicles [35], and specific frameworks regarding the maritime domain [36].

The development of software systems, based on the functional decomposition started at the beginning of the 1990s, with object-oriented approaches proposed to replace the traditional software life cycle [37]. In the decade of the 2000s, software development for underwater vehicles was focused on the implementation of real-time platforms to be used with hierarchical control structures [38,39]. Later, in the 2010s, embedded systems were frequently used within the hardware architecture for robotic systems [40–42], which required more advanced software architectures in order to allow integrating new components [43] into the vehicles while guaranteeing real-time operation and the execution of several tasks [44] and more challenging missions [45,46]. More recently, because of the sophistication of autonomous systems, new opportunities for maritime activities have appeared in the scenario [47] and more advanced and flexible software systems are being required. Regarding this matter, Bozhinoski et al. [48] performed an extensive review from a software engineering perspective on solutions for mobile robotic systems that will need to be able to operate in uncontrollable and unknown environments. For instance, Yu et al. [49] proposed a two-layer software architecture for a hybrid underwater vehicle that exhibited a satisfactory multi-task processing performance. Nowadays, underwater cooperative robotics offers the possibility to perform challenging survey, operation, and intervention tasks [50–53] that require developments that allow the users to expand the capabilities of underwater robots.

As a response of adaptability needs for underwater vehicles, this work addresses the development of a flexible and modular software architecture developed with the use of systems engineering, considering a functional division during the design process in order to facilitate the integration of components and subsystems, which is desired for modern hardware and software architectures. Together with the work of [54], the proposed modularity for hardware/software systems represents a contribution in the field of underwater vehicles, which are complex multifunctional systems that require flexibility, depending on the tasks that they are expected to execute. Such flexibility is also important for operational stages that require simulation campaigns, which are employed before the vehicle is deployed in a realistic scenario, as has been reported in [55], who built a continuous validation framework in the scope of the EU project DexROV [56]. This work exemplifies the process for the remotely operated vehicle Pionero500, developed in order to reduce the gap in the development of marine technologies in Colombia. The organization of the paper is as follows. Section 2 describes the methodology used in the system’s software development process. Section 3 addresses the SE-based software architecture. Then, Section 4 contains some details about the first sea trials for the ROV Pionero500. Finally, conclusions are provided in Section 5.

## 2. Methodology

### 2.1. Systems Engineering Approach

Systems engineering was used to build and process all stages of the ROV life cycle, from its conception to its retirement stage, considering that it is a complex system in which multiple disciplines interact. Due to this complexity, its life cycle cannot be represented

through a horizontal timeline, since, naturally, throughout the development of the stages, iterations can occur, hence giving different dynamics that are finally better represented by methods such as the Vee model.

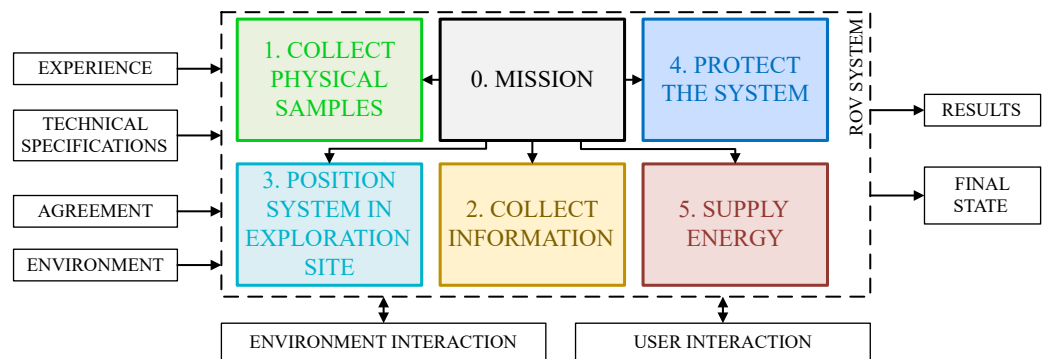
Aristizábal et al. [54] provided a description of how SE has been used to construct the functional division of the ROV system, using product design specification (PDS) and quality function deployment (QFD). Thanks to a cooperative/iterative effort between the client and the development team, extensive lists of requirements and engineering characteristics were built and evaluated through the use of the House of Quality (HoQ), a tool commonly associated with a QFD used to rank characteristics by quantifying the influence of the requirements on them. A comprehensive HoQ was built for the whole ROV system, but it is beyond the scope of this paper. In order to generate a set of engineering specifications (ES), the HoQ exercise was conducted for the hardware and software systems and reported by [54]; this ES list shows the complexity of the system that has to be developed to fulfill stakeholders' expectations and that requires the modular software architecture that is being reported:

1. Power system autonomy time.
2. Average current (and range) of electrical power transmission.
3. Number of additional ports for connecting auxiliary equipment.
4. Energy storage capacity in the vehicle.
5. Power consumed by the propulsion system.
6. Supply voltage.
7. Maximum consumption current allowed.
8. Number of devices required for the power supply system.
9. Nominal power transmission voltage.
10. Control system response time.
11. Surface station size.
12. Average power demanded by the system.
13. Longitudinal advance thrust.
14. Additional ports available on the surface for connecting devices.
15. Power consumed by the lighting system.
16. Power consumed by the launch/recovery system.
17. Cable length.
18. Up/down thrust.
19. Umbilical cable gauge.
20. Existence of an electrical protection system.
21. Number of people needed to operate the system.
22. Number of lights.
23. Total illumination intensity.
24. Total number of thrusters.
25. Power (capacity/required availability) supply.
26. Vehicle dimensions.
27. Vehicle weight in air.

#### Functional Decomposition

After the definition of the ES, the development of the functional decomposition is required to understand the connections between parts of the system which are unknown before executing the detailed design in order to fulfill the established requirements [57]. The resulting system's architecture can be represented graphically, and describes the desired functions to be performed by the designed solution. Figure 1 presents a simplified functional decomposition for the ROV Pionero500, as reported in [54,58], a complex and complicated system in which multiple functions must be performed, often with internal and external interactions. As addressed in [54], this system's architecture sets the foundations for the definition of a physical architecture, since it involves flows that connect abstract functions with a real context. Consequently, this is also the basis to define and develop a

software architecture that allows the ROV system to perform all the functions it has been designed for, while being modular and flexible.



**Figure 1.** Pionero500 system's simplified functional architecture [54,58].

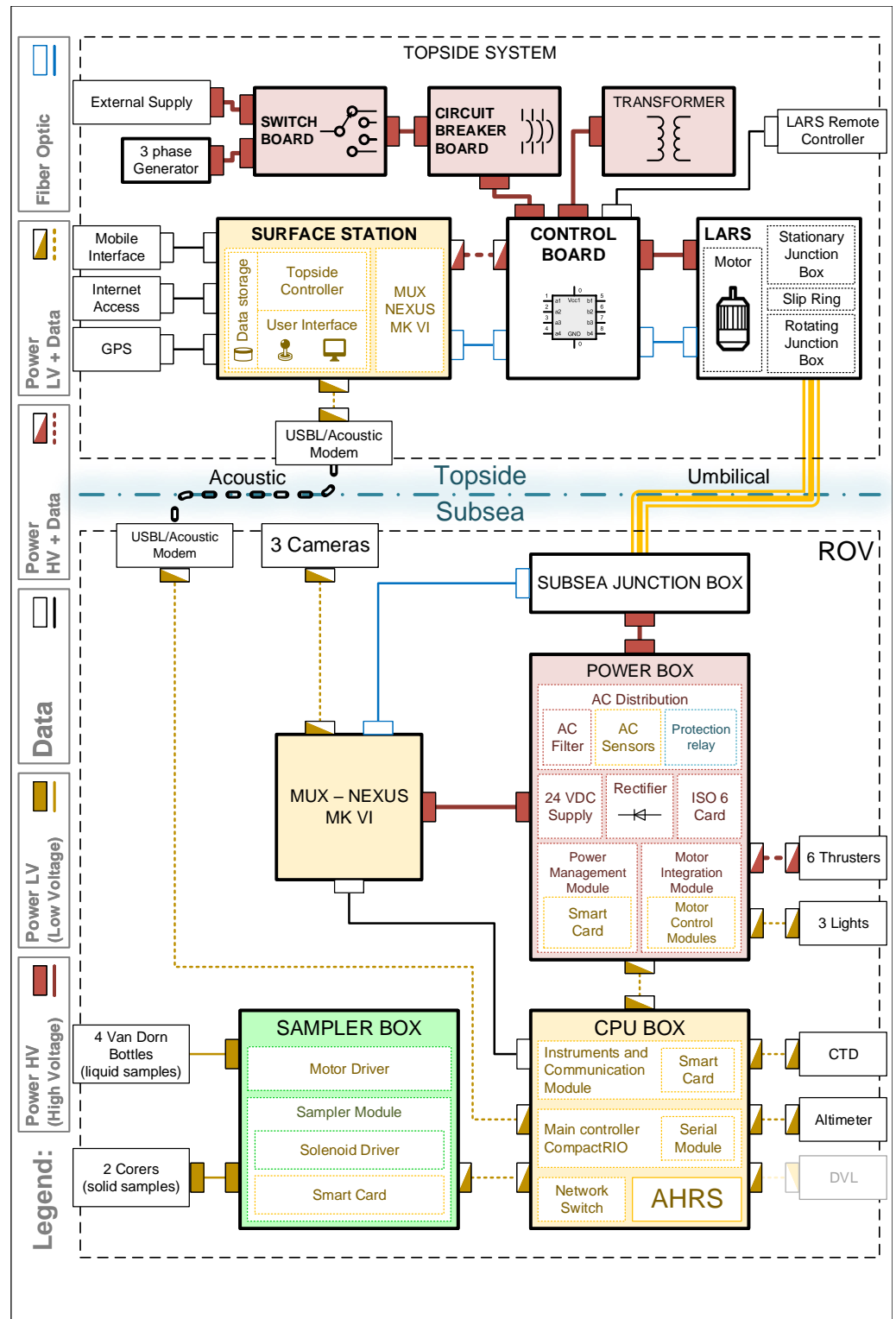
### 2.2. Modular Hardware Architecture

Figure 2 shows the hardware architecture and physical components of the ROV that were developed by Aristizábal et al. [54]. Three main groups can be identified: subsea systems (which include the vehicle itself and all its components), topside systems (components that are at the surface station), and the umbilical cable that connects the first two systems and guarantees the flow of data and energy.

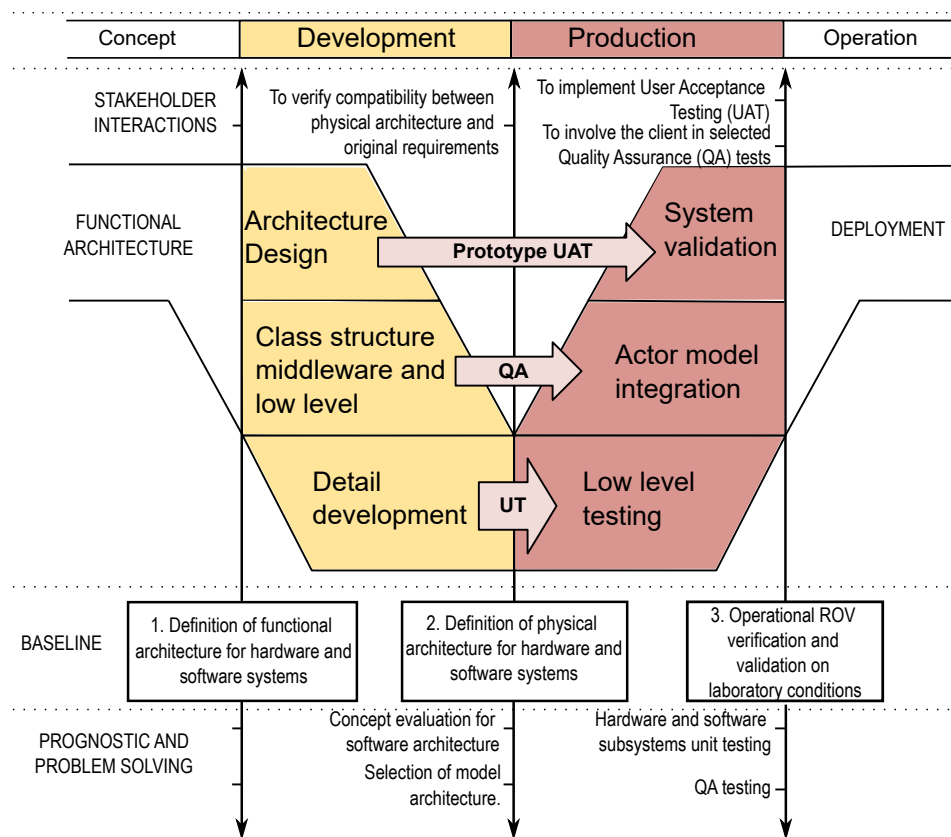
In order to achieve modularity, subsea systems were approached by modules in compartments called boxes, which were named according to the main function they perform. Thus, there are three functional boxes, or modules, that are required for basic functionality: power box (functions of the power system), CPU box (main processing module of the vehicle), and mux box (multiplexer responsible for communications). In addition, thanks to the modularity, there is another box called the sampler box that is in charge of operating the sediment and liquid samplers for the ROV Pionero500. In the same way, other boxes can be used to add functions to the vehicle, as long as the design meets the power and data constraints.

### 2.3. Vee Model

The V model is a basic model of traditional systems engineering. According to NASA [29] and INCOSE [57] systems engineering handbooks, technical processes can be organized and managed according to this model [59]. The V model shows the stages of the life cycle that are executed through levels, but carrying out verification and validations at the end of each stage in order to ensure that failures are not propagated in later stages, to finally verify that the user's requirements have been fulfilled. In this way, then, the diagram begins in the upper left part with the definition of user requirements and ends in the upper right part with the confirmation of such requirements. The left side explains the requirement-based design process from top to bottom, while the right side explains the validation and integration process from bottom to top [60]. Part of the model used on the ROV Pionero500 is shown in Figure 3, focusing on development and production stages, while concept and operation stages are addressed deeply in [54]. The outcome of such stages is mostly common for both software and hardware systems.



**Figure 2.** Pioneer500 modular hardware architecture presented in [54]. Three main groups can be identified: subsea systems (which include the vehicle itself and all its components), topside systems (components that are at the surface station), and the umbilical cable that connects the first two systems. Power box, CPU box, and mux are required for basic operation, while modularity allows using more boxes, i.e., sampler box, to expand the ROV capabilities.



**Figure 3.** Development and production stages of the Vee model, based on the general model presented in [54].

The model defines stakeholders interactions (Figure 3). This is useful for validation on both stages, development and production, and at every implementation level vertically; for instance, think of architectural design that must be validated with stakeholders, as well as its implementation in production stage. One of the main elements of the model are the baselines, that are established during the early stages of the project and must be fulfilled at the end of each stage, indicating the beginning of the next one, as it happens for hardware and software systems. Prognostic and problem-solving actions and tools are specified as detailed as possible with the resources available when developing the model (lower part of the diagram in Figure 3). Regarding software development, engineering design selection processes applied to choose an architecture and then producing a prototype were specified among such activities at the development stage. Production stage tools in this context comprise typical software tests applied to both hardware and software systems, such as unit testing (UT) and quality assurance (QA).

#### 2.4. Integration, Verification, and Validation (IV&V)

The IV&V activities provide a plan that indicates how to integrate, verify, and validate the results of the Vee stages. They are represented horizontally and are key in the development and production stages. Additionally, they are reinforced during prototyping activities in order to avoid reprocesses and errors in integration stages.

The Vee model is widely used in the development of complex products because it uses iterative processes such as design–redesign and think–rethink, which are essential to prevent problems from spreading from one stage to another by using IV&V activities; this allows avoiding transferring errors and incomplete results to subsequent stages which will lead to failure at the end of the project [60]. Regarding software development for Pionero500, IV&V activities involve mainly typical software tests, applied to early models and prototypes of the results expected at the baseline. An appropriate example is an early user acceptance test (UAT) activity executed for software architecture selection: the team de-



signed and implemented a semi-functional mockup software made to test different models during architecture design; it reused both hardware and software elements from previous developments [61] while incorporating a novel distribution. The mockup was presented to the stakeholders, aiming to receive acceptance and validate the selected architecture.

### 2.5. Actor Model

Concurrency problems are common in systems that execute several tasks at the same time, i.e., parallel, distributed, and mobile systems [62]. The actor model addresses these issues, being a concurrent computing model where the actors are autonomous objects that operate simultaneously and asynchronously [63], communicate by means of messages, and have a clearly established communication infrastructure [64,65]. The actor model introduces some simple rules to control state sharing and cooperation between execution units and directly attack concurrency problems from two fronts: there is no deadlock in the system since each actor is independent and there is always a replacement in case their answer is missing [62,66–68]. The independence of the actors favors modularity, because they make it possible to add knowledge as the system grows without having to completely rewrite it and to obtain maximum parallelism by participating in several conversations at the same time [69,70]. Furthermore, they can be used as a framework to model, understand, and gain reason about a wide range of concurrent systems [70]. Modularity, independence of processes, and concurrent work capability within a development team made the actor model appropriate for structuring high-level software systems in Pionero500. A model that requires parallel execution at its core has specific performance requirements and, usually, its implementation is not appropriate in platforms with limited resources, such as microcontrollers. For these devices, simplified modular architecture approaches based on object-oriented programming can be used instead.

### 2.6. Object-Oriented Programming

Object-oriented programming (OOP) is a programming paradigm that allows gathering data and functionality on a piece of code (object) with unique attributes and behavior. OOP is focused on the objects and its functions rather than the logic required to work with them. OOP is a well-suited programming paradigm for the development of software architectures for robotic systems due to reusability, scalability, and efficiency characteristics of code. For example, Hien et al. [35] presented an object-oriented system to develop controllers on unmanned aerial vehicles. Mouelhi et al. [71] presented a software design approach for cooperative autonomous systems based on Ada standards. Xu and Li [72] developed an object-oriented software architecture for a modular agricultural robotic system; the software was designed using a robot operating system (ROS) using three object-oriented modules: control, navigation, and vision module. For underwater vehicles, OOP is useful for low-level systems that involve microcontrollers up to high-level systems, such as human–machine interfaces. In low-level systems, modularity allows different sensors to be connected using the same software architecture. In high-level software components, scalability allows third-party software to be integrated into the mission.

### Reconfigurable Input–Output Architecture

For the ROV Pionero500, the National Instruments™ CompactRIO® system, which is an industrial embedded controller, was selected because it meets the requirements of the reconfigurable architecture, since it is a processing system with high-performance capabilities, sensor input and output systems, and programming tools directly related to LabVIEW®. In addition, it responds to real-time processing needs with reliable and predictable behavior. Finally, the field programmable gate array (FPGA) contained in its chassis provides high-speed processing and precise timing.

Given the requirements, it was decided to use LabVIEW given its high performance in parallel processing, where the actors are conceived in independent virtual instruments (VI), launched through the LabVIEW base tools for parallel execution, following the RIO

(reconfigurable input output) architecture, given the potential that offers to connect hardware and software modules [73,74]. LabVIEW queues were implemented as a means of communication, so that the actors can communicate asynchronously, sending and receiving messages at any time [68].

### 3. Modular Software Architecture Definition

Modularity means that the design is based on a set of independent blocks, called modules, whose structure and behavior follow certain rules. The modular design allows developing scalable systems and makes components of the system to be reusable in a collaborative development [75]. For Pionero500, this modularity has been addressed hardware-wise in [54]. To introduce the corresponding software architecture, a customized unified modeling language (UML) deployment diagram is presented in Figure 4 to establish its relationship with physical systems.

Physical modules are represented as UML nodes, and are differentiated in the architecture according to its dependability. Some modules are required for the architecture to work; others can be removed or added, including original equipment manufacturer (OEM) modules that are usually closed software platforms; custom devices are developed from scratch. An example of required modules are those associated with communication tasks and data processing, such as topside computer, Ethernet switch, multiplexer units, and the National Instruments (NI) CompactRIO (cRIO) vehicle's computer. Modules with software that has been developed or modified by the team are distinguished from the rest by having UML components inside. Finally, this structure considers other components that can be added in the future, which gives advantages in terms of modularity and scalability.

Components that are hosted in NI MyRIO, computer, and NI cRIO have been developed in LabVIEW. Real-time capabilities of the RIO-based devices are leveraged to improve the prioritized engineering specifications, such as control system's time response. Gigabit Ethernet compatibility was also established as a requirement for communication devices on the main link between topside and subsea devices, such as computer, Ethernet switch, multiplexers, and NI cRIO. The NI cRIO serves as the main processing unit of the vehicle and is equipped with a communication module that allows it to exchange data with every instrument on board: altimeter, AHRS, CTD, and USBL modem, regardless of the physical standard. For example, while the CTD uses RS-232, the AHRS uses RS-422. It also communicates with every box's SmartCard in a master-slave scheme, using an RS-422 full-duplex bus.

Each SmartCard processing unit is a 32-bit microcontroller. One of the key characteristics taken into account for microcontroller platform selection was developing an environment and language supporting OOP. In this particular implementation, Atmel® SAM3X series complied with this specification. Arduino™ provided a simplified framework for implementing functionality while reducing development time by using third-party libraries provided by device manufacturers and developers community.

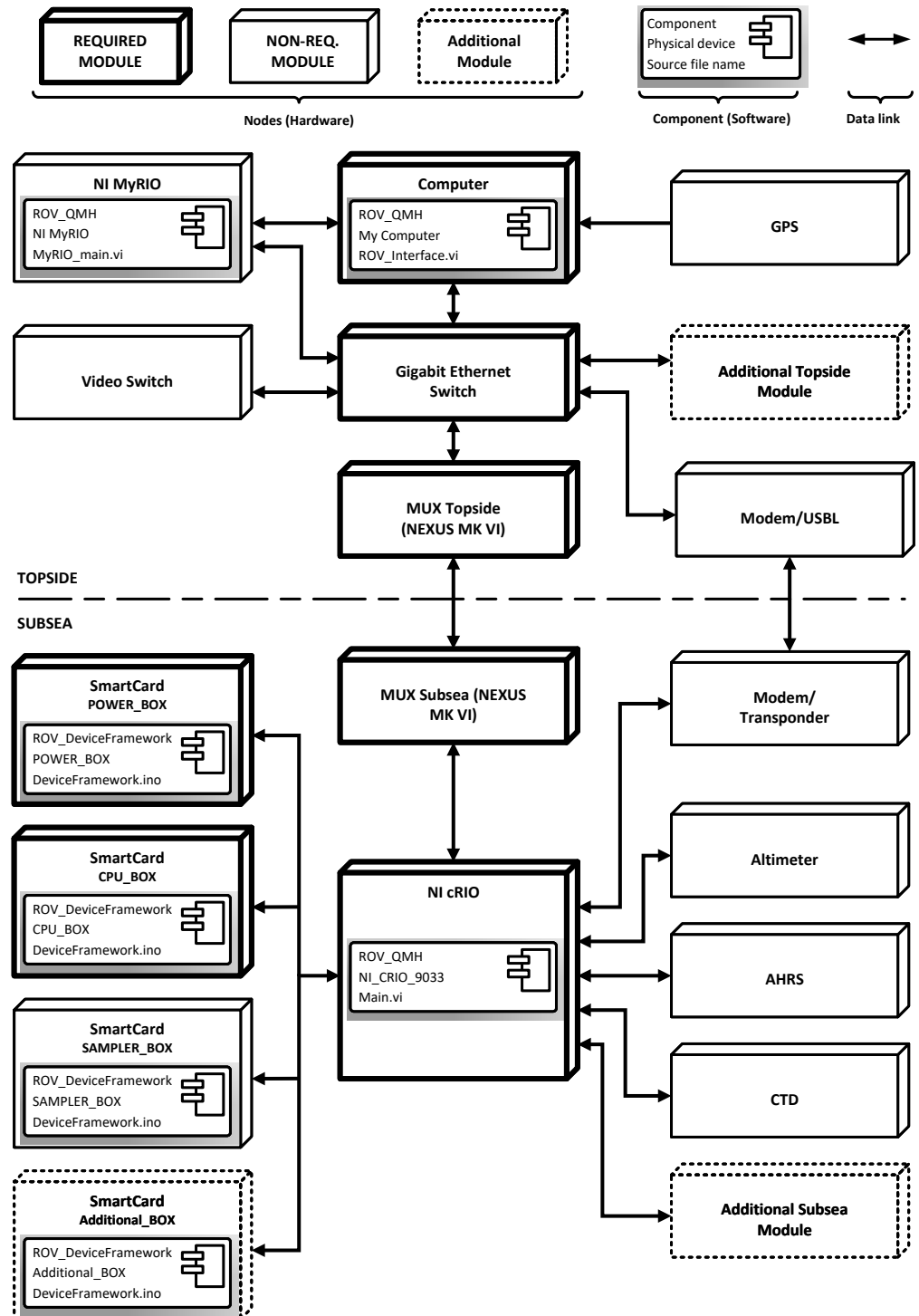
#### 3.1. Software Components Architecture

A general view of the designed software architecture is displayed by using a modified UML components diagram. Figure 5 represents Pionero500's software component architecture. The design is layered, with top levels being close to the user, and bottom levels addressing low-level signals and devices. The user level defines interactions with users through interfaces. The topside level involves components hosted mostly on the topside computer, where the actor model is implemented. The instrument level regards the vehicle main computer and instruments components. Both the topside and instrument levels' coding language is National Instruments LabVIEW. Finally, the device level comprises firmware components, hosted on SmartCard devices. Further details on SmartCards can be found in [54].

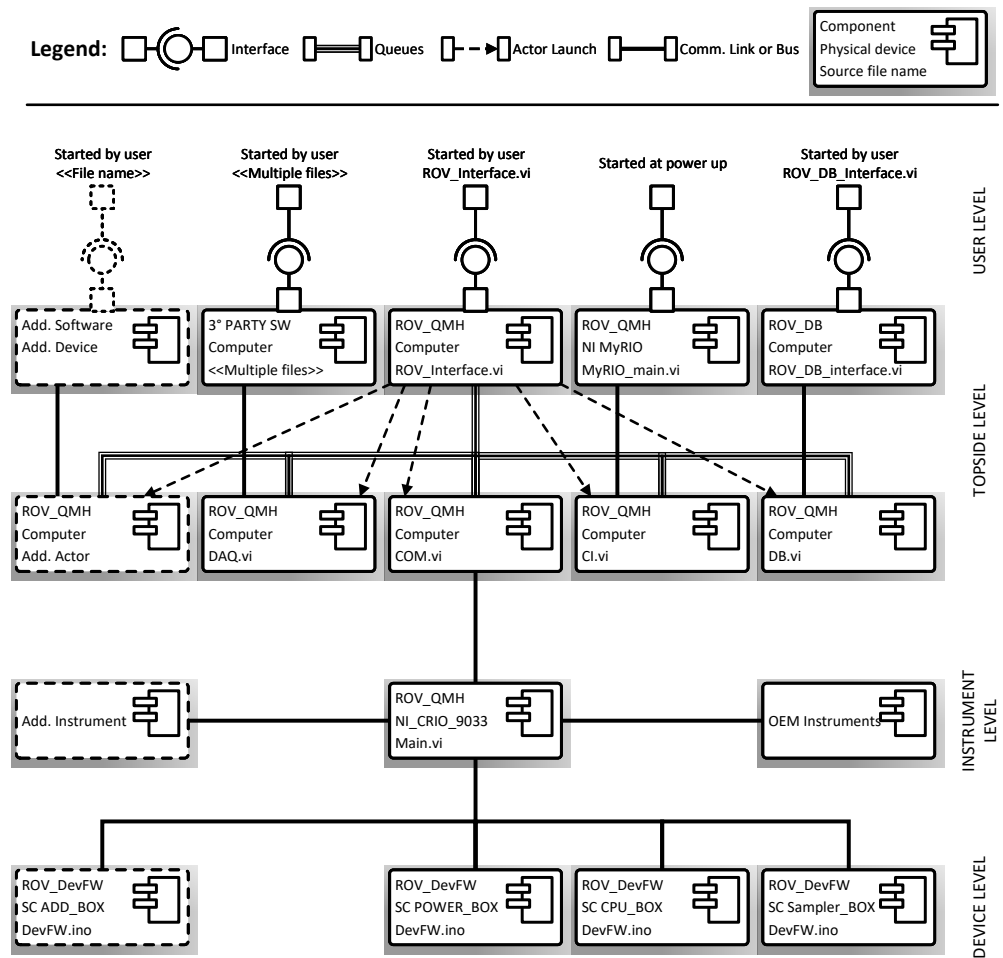
Regarding modularity, components identified with dashed lines are named as additional. This indicates where to include future modules and how they would interact with



others in the architecture. For example, an additional actor included in the topside level needs to be launched by ROV\Interface.vi, and communicates through queues with other actors.



**Figure 4.** Deployment diagram specifying relationships between hardware nodes and software components. Similar to the hardware architecture, there are components running in the surface station and others in the ROV. The software architecture is implemented using virtual instruments (.vi) for middle-ware and high-level components, and custom software based on Arduino libraries (.ino) for low-level components.



**Figure 5.** Component architecture for the ROV Pionero500. The topside and user levels run in the surface station, while the instrument and device levels run on the ROV. Dashed lines correspond with potential components that can be connected to the software.

### Actor Model Implementation

The last example serves as an introduction to the specific implementation of the actor model in this architecture. As mentioned in Section 2.5, actors essentially need to be independent and a communication channel. Considering this, the development team created its own implementation of an actor-based model in LabVIEW, as third-party approaches added unnecessary complexity to the solution. LabVIEW queues served as communication channels, as they offer asynchronous lossless message exchange, in a first-in first-out (FIFO) manner. Each actor must be created with its own queue, which also must be accessible to every other actor and the actor launcher. For actor execution and termination, the main interface application `ROV\_Interface.vi` serves as launcher, by running each actor VI using LabVIEW asynchronous VI execution. When running, each actor’s specific user interface, also known in LabVIEW jargon as a front panel, can be accessed through the launcher’s front panel. This feature enables modular and concurrent user interface development, as each actor’s interface can be modified simultaneously without modifying the launcher’s front panel.

Essential actors include `COM.vi`, which handles communication between topside and instrument level; `CI.vi`, named after Command Interface, which contains indicators and controls needed for human–machine interaction, and also establishes communication with the ROV command peripherals controller, in this case, an NI MyRIO. Non-essential actors implemented can be removed without affecting basic operation of the whole system,

although it involves removing useful features such as instrument data acquisition (DAQ.vi) and mission data storage (DB.vi).

Each actor can interact with multiple external third-party applications. One implementation of this feature is the data acquisition actor DAQ.vi, which establishes communication channels with topside instrument's own applications, e.g., Ultra Short Baseline (USBL) server or Global Positioning System (GPS) serial data. Regardless of the application, DAQ.vi obtains, processes, and distributes data to other actors or external applications. Another case of interaction with third-party software is the database actor DB.vi, which connects to databases, stores vehicle and mission data, executes queries, and displays information as requested by the operator.

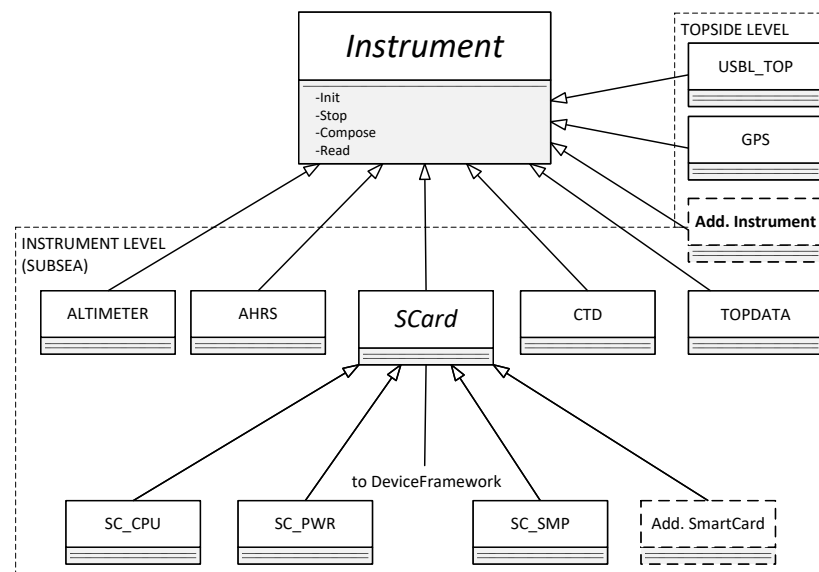
It is worth mentioning that this model has made it easier to extend the system's functionality, as actors are defined with clear communication schemes: internally, using LabVIEW queues' asynchronous messaging capability; and externally, exchanging data with additional applications, thanks to the extensive and diverse communication tools provided by the integrated development environment (IDE) and third-party applications developers. On the other hand, not every feature of the actor model is required in lower levels of the architecture, e.g., user interfaces are only present at the topside level components. A simplified model was necessary at the instrument level, without losing modularity and concurrent development capabilities. As described in Section 2.6, the OOP paradigm favors high cohesion, low coupling, modularity, and reusability of code; therefore, it was considered for instrument level software design. The resulting architectural design highlights can be expressed through class diagrams in the following section.

### 3.2. Class Structures

Pionero500, as an exploration system, mainly benefits from modularity by increasing functions regarding data acquisition from its surrounding environment, thus making instrumentation fundamental for the ROV. This premise oriented the development of a class model that uses the concept of instrument as an abstraction, with functionality added by specification, while sharing some common members. Figure 6 shows a simplified version of the main class structure. An abstract class Instrument is at the top, with multiple children classes inheriting from it. As an abstract class, Instrument functionality is overridden, serving as a template for every child, providing them with essential methods that must be present in all instruments. These generic methods are:

- **Init:** runs instrument initialization routines, usually for establishing communication with the main processor. It should be called at least once per operation of the vehicle.
- **Stop:** terminates the operation of the instrument as intended, according to its requirements. Frequent routines implemented by this method are serial buffer clearing and controlled power-off.
- **Read:** handles data exchange between the instrument and processing unit. Its main function is to accumulate and parse incoming data, but also can implement specific routines, such as requests and process responses.
- **Compose:** consolidates information in a standardized manner by using a type-defined data structure. The structure is the same for every instrument. This is necessary to facilitate data exchange between the instrument and the whole system.

After inheriting Instrument abstract class, each instrument class implements specific functionality to the inherited methods, while preserving the parent's calling properties, allowing the main application to call overridden methods iteratively. This is a key aspect of a modular model, as the addition of modules only requires code development of its own implementation, with little to no intervention of main code.



**Figure 6.** Topside and instrument levels class structure. The SmartCards (SCard) are classified as instrument since they are monitoring the health status of each box.

### 3.2.1. Topside Level Classes

Topside level classes are instantiated in the topside computer, implemented in NI LabVIEW, aided by its OOP tools. Derived classes in this level implement functionality to GPS and USBL instruments. An example of the derived methods and specific implementation can be illustrated with the GPS class; Init method establishes connection with a marine GPS through serial port; Stop closes the port and flushes buffers; Read receives data from the instrument and parses NMEA 0183 frames; Compose writes the type-defined data structure that holds the topside data, making the acquired data available to the whole application at every level. Additional instruments can be easily incorporated to the system at this level by inheriting Instrument class, writing the code needed by the instrument, and instantiating the new derived class in the DAQ.vi actor.

### 3.2.2. Instrument Level Classes

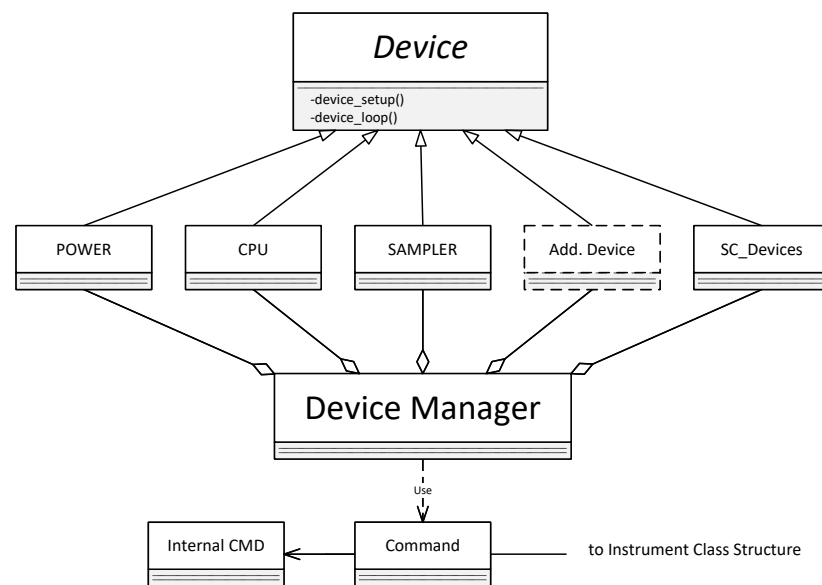
Topside level classes are instantiated in the NI cRIO processor onboard the vehicle, using the same developing environment, tools, and methods from the topside level. The main difference with the latter relies on real-time capability of RIO devices; this specific implementation uses real-time communication channels, called RT-FIFOs. These operate in a similar way to regular LabVIEW queues, with some restrictions on data types, e.g., fixed size arrays and static type variables, but with the benefit of time-controlled data access operations. Strict actor model implementation was compromised, mainly because of these restricted communication channels, yet benefiting in reliability and deterministic message delivery.

Derived classes at instrument level are shown in Figure 6. Altimeter, AHRS, and CTD are associated with physical OEM instruments integrated to the vehicle. SCard refers to an abstract class that implements similar features of Instrument, as described in Section 3.2. This is made to extend modularity to custom hardware and devices in the form of additional SmartCards. At the time of writing this document, three SmartCards were deployed in the vehicle, each one supporting functional watertight enclosures called Boxes; see [54] for details on hardware aspects. Children of Instrument class can also be associated with non-physical entities or instruments. TOPDATA is a derived class that represents topside data to the vehicle computer as an instrument, standardizing the access method to variables originated at the surface level, such as vessel position and heading, joysticks, and user interface controls.

### 3.2.3. Device Level Class Structure for Firmware

Firmware development followed the same principles of modularity and concurrent development as the higher levels. Previous successful experiences from the hardware development team in modular, OOP-based architectures running on microcontrollers [61] supported this approach, and established the starting point for the device level class structure design. Figure 7 presents a simplified class diagram that implements an adapted version of the modular class models developed for upper levels of the architecture. Here, an abstract class Device serves a similar purpose to Instrument class of Figure 6, as functionality is implemented by deriving devices. In this case, calling of devices is in charge of the class Device Manager, which instantiates every device and handles execution of its inherited methods. Such methods are device\_setup, similar to Instruments Init, this is meant to be executed once at startup and contains routines to initialize devices, and device\_loop, which contains device routines that have to execute periodically.

In this design, external communication with the main vehicle processor is handled by Command class, which is instantiated once in global scope, and used by Device Manager to enable interaction of devices with the rest of the ROV. As an internal communication channel between devices, Command uses the Internal Command class, which is a simplified and lighter version of the former.



**Figure 7.** Device framework class diagram. The device class diagram is based on the same concept of an Arduino program, where the device\_setup() method is called once while the device\_loop() executes periodically.

Regarding specific devices, the ones displayed in Figure 7 actually represent a collection of devices. SC\_Devices is a derived class which implements functionality of internal SmartCard devices such as temperature, humidity and pressure sensor, status indicator, voltage, and current sensor. POWER, CPU, and SAMPLER group distinguish devices according to each box, e.g., thruster signal generation is performed in a device class that is only implemented in POWER Box. Detailed firmware class structure exceeds the scope of this document.

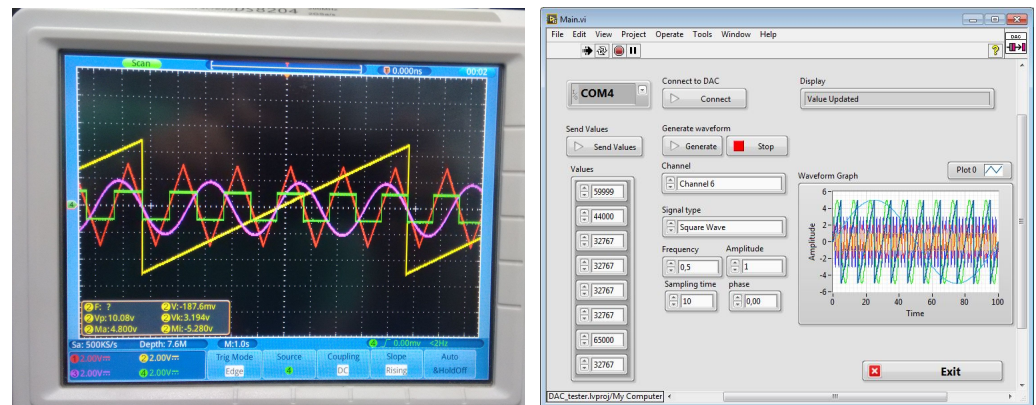
## 4. Testing

Tests covered in this section regard production and operation stages. The type of test used is closely related to the location in the timeline of the project, both horizontally to the life cycle stage, and vertically to the Vee specification component. Unit tests are useful during implementation of low-level components and specific functions of the system. QA

is composed mainly of integration tests in controlled conditions. UAT is carried out in both laboratory and field tests.

#### 4.1. Unit Testing

Unit testing is a practice in software development which helps to maintain code and bug detection. It allows correcting any errors before the deployment of the system in an uncontrolled environment. Since the developed architecture was based on actor model, each element is tested by launching it independently, sending messages, and waiting for the correct answer. For example, for the database actor, some messages with dummy data (similar to data provided by sensors) were produced, and it was verified that they were stored. Most of the test coverage was performed from a functional criteria taking into account the functional architecture given in the previous sections. Another successful example is the thruster driving system, that benefited from early unit tests, as control signal generation was prototyped by integrating OEM demonstration modules for digital-to-analog converters (DAC), Arduino™ libraries, and a simple software interface developed in LabVIEW. The latter was also used to test fundamental programming tools, such as LabVIEW queues and latency in serial communication between the microcontroller and software interface. Both generated signals and LabVIEW interface are shown in Figure 8.

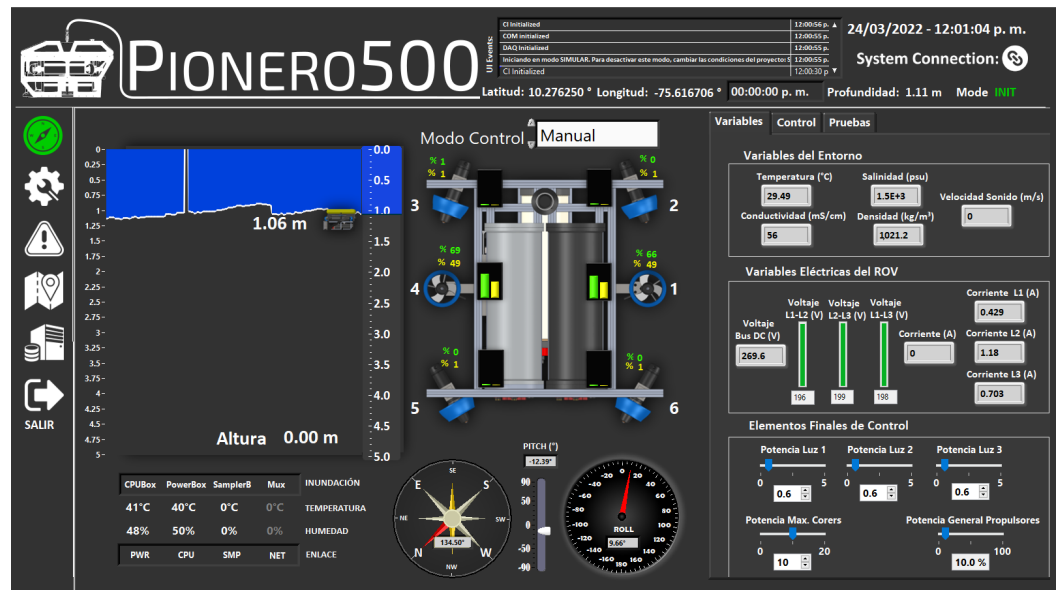


**Figure 8.** Testing of signal generation system for thrusters. Measured signals with an oscilloscope are on the left; software interface developed for the test is on the right.

#### 4.2. Quality Assurance

These tests verified the integration of elements to the designed architecture, aiming to fulfill at least one of the functions referred in the product design specifications (PDS) [54]. This is better explained by expanding a previous database example: one of the main functions presented in the PDS is to collect information; to fulfill this, real variables must be measured by instruments, collected by the NI cRIO, transmitted to the topside computer and, finally, stored in the database by the corresponding actor. It is worth mentioning that such tests can be performed in laboratory conditions, as long as the instruments can operate, e.g., CTD, USBL, and Altimeter have to be underwater to work properly. As QA tests require element integration to allow the pilot to interact with the vehicle, a user interface must be provided. Figure 9 shows a screenshot of the front panel used by the ROV pilot, and is currently displaying the navigation tab. A tabbed design was selected regarding front-end layout, as it is compatible with the modular approach, allowing the development team to add functionality through different tabs. Most of the tabs content is essentially each actor's individual front panel, integrated to the same interface through LabVIEW subpanel function, which enables real concurrent development among the team.





**Figure 9.** Pionero500 navigation software front panel, navigation tab. On the upper side of the screen, a summary of current ROV deployment is displayed permanently: time, simplified event log, ROV coordinates, depth, operation mode, and link status. The left side arranges icons representing tabs for implementation of multiple functions, such as settings, event log, map, and database.

### 4.3. User Acceptance Testing

The first sea trials for Pionero500 were carried out in Cartagena, Colombia as described in [54], with the cooperation of the General Maritime Directorate of Colombia (DIMAR) and its Caribbean Research Center for Oceanographic and Hydrographic Research (CIOH), using the research vessel ARC Roncador. These field tests were performed within an academia–government industry that included several stakeholders, and were very useful to perform not only the UAT but to gain feedback of operational characteristics of the ROV system, which is undergoing a software update nowadays, together with the development of a navigation system. The main objective of the partnership is to develop a generalized framework that integrates modern tools and methods that are useful for the characterization of underwater ecosystems in order to strengthen decision-making processes in Colombia.

The field exercise was conducted at two national parks: Corales de Profundidad National Natural Park, which is located next to an oil and gas exploration block, and Parque Nacional Natural Corales del Rosario y San Bernardo. Figure 10 shows ROV Pionero500’s system on the R/V ARC Roncador, which is managed by the General Maritime Directorate of Colombia (DIMAR) and its Caribbean Research Center for Oceanographic and Hydrographic Research (CIOH); this image shows the vehicle and a twenty-foot container that carries the tether management device and a backup power supply. The field exercise was directed in such a way that the use of modern methods for the characterization of underwater ecosystems were applied, bringing a particular focus to supporting collaborations among several stakeholders of data users and producers to implement collective action solutions to improve the availability and quality of data for development of different offshore activities in the country.

Figure 11 shows different elements of the ROV’s operational stage during a ground truthing session, where the vehicle is used to visually verify selected places from the initial sea bottom model built with the use of bathymetry information of the surveyed area. As an example of the software modularity, the actor DAQ.vi and the USBL class USBL (see Section 3.2.1) were modified during the test in order to connect the ROV system data to OpenCPN, a third-party application that is used to create a chart plotter that displays live ROV position and heading over mission-specific map provided in the vessel.



Figure 10. ROV Pionero500 (front) on board R/V ARC Roncador, with container with tether and backup power supply on the back.

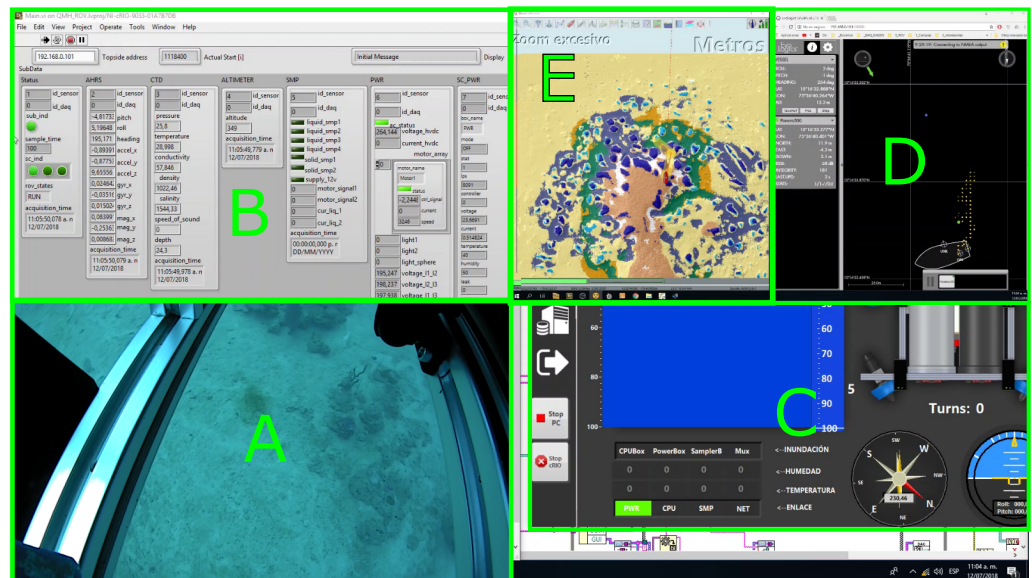


Figure 11. Screenshot of user interfaces of Pionero500 software during field test. (A) bottom camera, (B) NI cRIO front panel, (C) topside computer ROV interface, (D) USBL third-party application, (E) OpenCPN with live ROV position and heading displayed over mission-specific map.

Validation in this stage included performing tests on all the ROV system, with integrated hardware and software architectures. Figure 12 shows the interface that allows the user to interact with the control module. This module gives the possibility to change parameters for manual control (thrust allocation), depth control, altitude control, heading control, and thrust control. Control algorithms have been implemented in the NI cRIO, taking advantage of the real-time system; details of the control systems are beyond of the scope of this work.

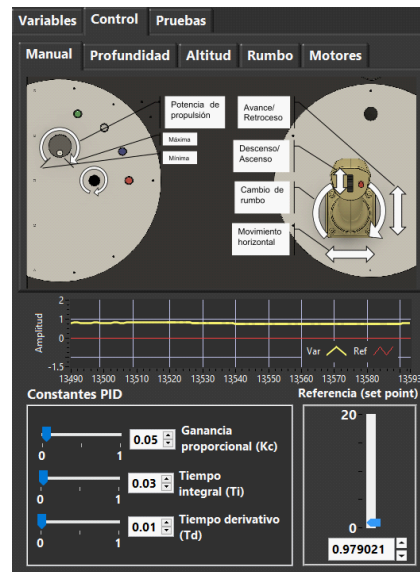


Figure 12. Control setup tab of the navigation interface.

## 5. Conclusions

This paper addressed the development of a modular/flexible software architecture for a Class-II remotely operated vehicle, named Pionero500, based on system engineering. This methodology is suited for the development of complex systems (including underwater vehicles and maritime systems), which starts with the understanding of user requirements from a functional level to provide a solution.

The actor model was implemented as a solution to reduce time and complexity in the development of the vehicle as a high-level software architecture. Such an architecture is useful to solve concurrency problems since the actors can operate simultaneously and independently. Object-oriented programming was used in instrument-level software design as a solution to decrease coupling in code, and increase cohesion, modularity, and reusability, which are highly desired characteristics in a complex underwater exploration system.

Finally, IV&V activities helped reducing time for software design at each level. These activities were applied to early models in order to generate a baseline, and in the end they were compared with the final integrated solution. The complete system was tested at sea, and allowed the development team to add required functions during the trials, validating the proposed architecture and proving to be a useful tool for software design for underwater vehicles and maritime systems.

**Author Contributions:** Conceptualization, C.A.Z., L.M.A., S.R. and R.E.V.; methodology, C.A.Z. and L.M.A.; validation, C.A.Z., L.M.A., S.R., D.A.F. and D.A.O.; investigation, L.M.A., C.A.Z., S.R., D.A.F., D.A.O. and R.E.V.; writing—original draft preparation, L.M.A., C.A.Z., S.R., D.A.F., D.A.O. and R.E.V.; writing—review and editing, R.E.V.; supervision, C.A.Z. and R.E.V. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was developed with the funding of the Fondo Nacional de Financiamiento para la Ciencia, la Tecnología y la Innovación, Francisco José de Caldas; the Colombian petroleum company, ECOPETROL; the Universidad Pontificia Bolivariana—Medellín, UPB; the Universidad Nacional de Colombia—Sede Medellín, UNALMED; through the “Strategic Program for the Development of Robotic Technology for Offshore Exploration of the Colombian Seabed”, project 1210-531-30550, contract 0265-2013. Sea trials were funded by the Newton Fund/Royal Academy of Engineering Industry/Academy Partnership Program “Development of a technology-based methodology for the characterization of underwater ecosystems as tool towards marine spatial planning decisions of marine areas in the Colombian seas”, that included ECOPETROL; UPB; UNALMED; Newcastle University; the General Maritime Directorate of Colombia (DIMAR)—Caribbean Research Center for Oceanographic and Hydrographic Research (CIOH); and Parques Nacionales Naturales de Colombia; Project IAPP1617\69.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Not applicable.

**Conflicts of Interest:** The authors declare no conflict of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

AHRS	Attitude and Heading and Reference System
AUV	Autonomous Underwater Vehicle
CTD	Conductivity–Temperature–Depth
ES	Engineering Specifications
GPS	Global Positioning System
HoQ	House of Quality
IV&V	Integration, Verification and Validation
NMEA	National Marine Electronics Association
QA	Quality Assurance
OEM	Original Equipment Manufacturer
OOP	Object Oriented Programming
PDS	Product Design Specification
QFD	Quality Function Deployment
ROS	Robot Operating System
ROV	Remotely Operated Vehicle
SE	Systems Engineering
UML	Unified Modeling Language
UT	Unit Testing
UAT	User Acceptance Tests
USBL	Ultra Short Baseline
USV	Unmanned Surface Vehicle
VI	Virtual Instrument

## References

1. Macreadie, P.I.; McLean, D.L.; Thomson, P.G.; Partridge, J.C.; Jones, D.O.; Gates, A.R.; Benfield, M.C.; Collin, S.P.; Booth, D.J.; Smith, L.L.; et al. Eyes in the sea: Unlocking the mysteries of the ocean using industrial, remotely operated vehicles (ROVs). *Sci. Total Environ.* **2018**, *634*, 1077–1091. [[CrossRef](#)] [[PubMed](#)]
2. Capocci, R.; Omerdic, E.; Dooly, G.; Toal, D. Fault-Tolerant Control for ROVs Using Control Reallocation and Power Isolation. *J. Mar. Sci. Eng.* **2018**, *6*, 40. [[CrossRef](#)]
3. Ignacio, L.C.; Victor, R.R.; Francisco, D.R.R.; Pascoal, A. Optimized design of an autonomous underwater vehicle, for exploration in the Caribbean Sea. *Ocean Eng.* **2019**, *187*, 106184. [[CrossRef](#)]
4. Marini, S.; Gjerci, N.; Govindaraj, S.; But, A.; Sportich, B.; Ottaviani, E.; Márquez, F.P.G.; Sanchez, P.J.B.; Pedersen, J.; Clausen, C.V.; et al. ENDURUNS: An Integrated and Flexible Approach for Seabed Survey Through Autonomous Mobile Vehicles. *J. Mar. Sci. Eng.* **2020**, *8*, 633. [[CrossRef](#)]
5. Braginsky, B.; Baruch, A.; Guterman, H. Development of an Autonomous Surface Vehicle capable of tracking Autonomous Underwater Vehicles. *Ocean Eng.* **2020**, *197*, 106868. [[CrossRef](#)]
6. Guardado, R.; López, M.J.; Sánchez, J.; Consegliere, A. AutoTuning Environment for Static Obstacle Avoidance Methods Applied to USVs. *J. Mar. Sci. Eng.* **2020**, *8*, 300. [[CrossRef](#)]
7. Nilssen, I.; Ødegård, Ø.; Sørensen, A.J.; Johnsen, G.; Moline, M.A.; Berge, J. Integrated environmental mapping and monitoring, a methodological approach to optimise knowledge gathering and sampling strategy. *Mar. Pollut. Bull.* **2015**, *96*, 374–383. [[CrossRef](#)]
8. Zolich, A.; Johansen, T.A.; Cisek, K.; Klausen, K. Unmanned aerial system architecture for maritime missions. Design & hardware description. In *2015 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*; IEEE: Cancun, Mexico, 2015. [[CrossRef](#)]
9. Sulligoi, G.; Vicenzutti, A.; Menis, R. All-Electric Ship Design: From Electrical Propulsion to Integrated Electrical and Electronic Power Systems. *IEEE Trans. Transp. Electrification.* **2016**, *2*, 507–521. [[CrossRef](#)]
10. Hachicha, S.; Zaoui, C.; Dallagi, H.; Nejim, S.; Maalej, A. Innovative design of an underwater cleaning robot with a two arm manipulator for hull cleaning. *Ocean Eng.* **2019**, *181*, 303–313. [[CrossRef](#)]
11. Saravanan, K.; Aswini, S.; Kumar, R.; Son, L.H. How to prevent maritime border collision for fisheries?—A design of Real-Time Automatic Identification System. *Earth Sci. Inform.* **2019**, *12*, 241–252. [[CrossRef](#)]



12. Ma, T.; Liu, S.; Xiao, H. Location of natural gas leakage sources on offshore platform by a multi-robot system using particle swarm optimization algorithm. *J. Nat. Gas Sci. Eng.* **2020**, *84*, 103636. [[CrossRef](#)]
13. Singh, Y.; Bibuli, M.; Zereik, E.; Sharma, S.; Khan, A.; Sutton, R. A Novel Double Layered Hybrid Multi-Robot Framework for Guidance and Navigation of Unmanned Surface Vehicles in a Practical Maritime Environment. *J. Mar. Sci. Eng.* **2020**, *8*, 624. [[CrossRef](#)]
14. Utter, B.; Brown, A. Open-source five degree of freedom motion platform for investigating fish-robot interaction. *HardwareX* **2020**, *7*, e00107. [[CrossRef](#)]
15. Capocci, R.; Dooly, G.; Omerdić, E.; Coleman, J.; Newe, T.; Toal, D. Inspection-Class Remotely Operated Vehicles—A Review. *J. Mar. Sci. Eng.* **2017**, *5*, 13. [[CrossRef](#)]
16. Spears, A.; West, M.; Meister, M.; Buffo, J.; Walker, C.; Collins, T.R.; Howard, A.; Schmidt, B. Under Ice in Antarctica: The Icefin Unmanned Underwater Vehicle Development and Deployment. *IEEE Robot. Autom. Mag.* **2016**, *23*, 30–41. [[CrossRef](#)]
17. Jiang, C.M.; Wan, L.; Sun, Y.S. Design of motion control system of pipeline detection AUV. *J. Cent. South Univ.* **2017**, *24*, 637–646. [[CrossRef](#)]
18. Li, Y.; Guo, S.; Wang, Y. Design and characteristics evaluation of a novel spherical underwater robot. *Robot. Auton. Syst.* **2017**, *94*, 61–74. [[CrossRef](#)]
19. Gelli, J.; Meschini, A.; Monni, N.; Pagliai, M.; Ridolfi, A.; Marini, L.; Allotta, B. Development and Design of a Compact Autonomous Underwater Vehicle: Zeno AUV. *IFAC-PapersOnLine* **2018**, *51*, 20–25. [[CrossRef](#)]
20. Pugi, L.; Allotta, B.; Pagliai, M. Redundant and reconfigurable propulsion systems to improve motion capability of underwater vehicles. *Ocean Eng.* **2018**, *148*, 376–385. [[CrossRef](#)]
21. Hong, S.; Chung, D.; Kim, J.; Kim, Y.; Kim, A.; Yoon, H.K. In-water visual ship hull inspection using a hover-capable underwater vehicle with stereo vision. *J. Field Robot.* **2018**, *36*, 531–546. [[CrossRef](#)]
22. Xu, H.; Zhang, G.C.; Sun, Y.S.; Pang, S.; Ran, X.R.; Wang, X.B. Design and Experiment of a Plateau Data-Gathering AUV. *J. Mar. Sci. Eng.* **2019**, *7*, 376. [[CrossRef](#)]
23. Pinjare, N.S.; Chaitra, S.; Shraavan, S.; Naveen, I.G. Underwater remotely operated vehicle for surveillance and marine study. In Proceedings of the 2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICECCOT), Mysuru, India, 15–16 December 2017. [[CrossRef](#)]
24. Rozman, B.Y.; Elkin, A.V.; Kaptsov, A.S.; Ermakov, I.D.; Ermakov, D.I.; Krasnov, V.G.; Kondrashov, L.S. Upgrade of ROV Super GNOME Pro for Underwater Monitoring in the Caspian Sea. *Oceanology* **2018**, *58*, 144–147. [[CrossRef](#)]
25. Zhang, Q.; Wang, H.; Li, B.; Cui, S.; Zhao, Y.; Zhu, P.; Sun, B.; Zhang, Z.; Li, Z.; Li, S. Development and Sea Trials of a 6000 m Class ROV for Marine Scientific Research. In Proceedings of the 2018 OCEANS—MTS/IEEE Kobe Techno-Oceans (OTO), Kobe, Japan, 28–31 May 2018. [[CrossRef](#)]
26. Kadiyam, J.; Mohan, S. Conceptual design of a hybrid propulsion underwater robotic vehicle with different propulsion systems for ocean observations. *Ocean Eng.* **2019**, *182*, 112–125. [[CrossRef](#)]
27. Kong, F.; Guo, Y.; Lyu, W. Dynamics Modeling and Motion Control of a New Unmanned Underwater Vehicle. *IEEE Access* **2020**, *8*, 30119–30126. [[CrossRef](#)]
28. Ramírez-Macías, J.A.; Vásquez, R.E.; Sørensen, A.J.; Sævik, S. Motion Feasibility Framework for Remotely Operated Vehicles Based on Dynamic Positioning Capability. *J. Offshore Mech. Arct. Eng.* **2021**, *143*, 011201. [[CrossRef](#)]
29. Nasa. *NASA Systems Engineering Handbook: NASA/SP-2016-6105 Rev2—Full Color Version*; 12th Media Services: Washington, DC, USA, 2017.
30. Madni, A.M.; Sievers, M. Systems Integration: Key Perspectives, Experiences, and Challenges. *Syst. Eng.* **2013**, *17*, 37–51. [[CrossRef](#)]
31. Madni, A.M.; Sievers, M. Model-based systems engineering: Motivation, current status, and research opportunities. *Syst. Eng.* **2018**, *21*, 172–190. [[CrossRef](#)]
32. Dove, R.; Schindel, B.; Scrapper, C. Agile Systems Engineering Process Features Collective Culture, Consciousness, and Conscience at SSC Pacific Unmanned Systems Group. *INCOSE Int. Symp.* **2016**, *26*, 982–1001. [[CrossRef](#)]
33. Freire, L.O.; Oliveira, L.M.; Vale, R.T.; Medeiros, M.; Diana, R.E.; Lopes, R.M.; Pellini, E.L.; de Barros, E.A. Development of an AUV control architecture based on systems engineering concepts. *Ocean Eng.* **2018**, *151*, 157–169. [[CrossRef](#)]
34. Eaton, C.; Chong, E.; Maciejewski, A. Multiple-Scenario Unmanned Aerial System Control: A Systems Engineering Approach and Review of Existing Control Methods. *Aerospace* **2016**, *3*, 1. [[CrossRef](#)]
35. Hien, N.V.; Truong, V.T.; Bui, N.T. An Object-Oriented Systems Engineering Point of View to Develop Controllers of Quadrotor Unmanned Aerial Vehicles. *Int. J. Aerosp. Eng.* **2020**, *2020*, 8862864. [[CrossRef](#)]
36. Weinert, B.; Hahn, A.; Norkus, O. A domain-specific architecture framework for the maritime domain. In *Informatik 2016*, P-259 ed.; Heinrich, C., Mayr, M.P., Eds.; Lecture Notes in Informatics; Gesellschaft für Informatik: Bonn, Germany, 2016.
37. Henderson-Sellers, B.; Edwards, J.M. The object-oriented systems life cycle. *Commun. ACM* **1990**, *33*, 142–159. [[CrossRef](#)]
38. Kim, T.; Yuh, J. Development of a real-time control architecture for a semi-autonomous underwater vehicle for intervention missions. *Control Eng. Pract.* **2004**, *12*, 1521–1530. [[CrossRef](#)]
39. Li, J.H.; Jun, B.H.; Lee, P.M.; Hong, S.W. A hierarchical real-time control architecture for a semi-autonomous underwater vehicle. *Ocean Eng.* **2005**, *32*, 1631–1641. [[CrossRef](#)]

40. De Assis, F.H.; Takase, F.K.; Maruyama, N.; Miyagi, P.E. Developing an ROV software control architecture: A formal specification approach. In Proceedings of the 38th Annual Conference on IEEE Industrial Electronics Society IECON 2012, Montreal, QC, Canada, 25–28 October 2012. [[CrossRef](#)]
41. Sun, Y.S.; Wan, L.; Gan, Y.; Wang, J.G.; Jiang, C.M. Design of motion control of dam safety inspection underwater vehicle. *J. Cent. South Univ.* **2012**, *19*, 1522–1529. [[CrossRef](#)]
42. Freitas, R.S.; Xaud, M.F.; Marcovistz, I.; Neves, A.F.; Faria, R.O.; Carvalho, G.P.; Hsu, L.; Nunes, E.V.; Peixoto, A.J.; Lizarralde, F.; et al. The embedded electronics and software of DORIS offshore robot. *IFAC-PapersOnLine* **2015**, *48*, 208–213. [[CrossRef](#)]
43. Bonin-Font, F.; Oliver, G.; Wirth, S.; Massot, M.; Negre, P.L.; Beltran, J.P. Visual sensing for autonomous underwater exploration and intervention tasks. *Ocean Eng.* **2015**, *93*, 25–44. [[CrossRef](#)]
44. Choyekh, M.; Kato, N.; Yamaguchi, Y.; Dewantara, R.; Chiba, H.; Senga, H.; Yoshie, M.; Tanaka, T.; Kobayashi, E.; Short, T. Development and Operation of Underwater Robot for Autonomous Tracking and Monitoring of Subsea Plumes After Oil Spill and Gas Leak from Seabed and Analyses of Measured Data. In *Applications to Marine Disaster Prevention*; Springer: Tokyo, Japan, 2017; pp. 17–93. [[CrossRef](#)]
45. Gerasimou, S.; Calinescu, R.; Shevtsov, S.; Weyns, D. UNDERSEA: An Exemplar for Engineering Self-Adaptive Unmanned Underwater Vehicles. In Proceedings of the 2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), Buenos Aires, Argentina, 22–23 May 2017. [[CrossRef](#)]
46. Skjong, S.; Rindarøy, M.; Kyllingstad, L.T.; Æsøy, V.; Pedersen, E. Virtual prototyping of maritime systems and operations: Applications of distributed co-simulations. *J. Mar. Sci. Technol.* **2017**, *23*, 835–853. [[CrossRef](#)]
47. Zolich, A.; Palma, D.; Kansanen, K.; Fjørtoft, K.; Sousa, J.; Johansson, K.H.; Jiang, Y.; Dong, H.; Johansen, T.A. Survey on Communication and Networks for Autonomous Marine Systems. *J. Intell. Robot. Syst.* **2018**, *95*, 789–813. [[CrossRef](#)]
48. Bozhinoski, D.; Ruscio, D.D.; Malavolta, I.; Pelliccione, P.; Crnkovic, I. Safety for mobile robotic systems: A systematic mapping study from a software engineering perspective. *J. Syst. Softw.* **2019**, *151*, 150–179. [[CrossRef](#)]
49. Yu, C.; Xiang, X.; Maurelli, F.; Zhang, Q.; Zhao, R.; Xu, G. Onboard system of hybrid underwater robotic vehicles: Integrated software architecture and control algorithm. *Ocean Eng.* **2019**, *187*, 106121. [[CrossRef](#)]
50. Centelles, D.; Soriano, A.; Marin, R.; Sanz, P.J. Wireless HROV Control with Compressed Visual Feedback Using Acoustic and RF Links. *J. Intell. Robot. Syst.* **2020**, *99*, 713–728. [[CrossRef](#)]
51. Simetti, E.; Indiveri, G.; Pascoal, A.M. WiMUST: A cooperative marine robotic system for autonomous geotechnical surveys. *J. Field Robot.* **2020**, *38*, 268–288. [[CrossRef](#)]
52. Chen, G.; Shen, Y.; Qu, N.; Wang, D.; He, B. Control architecture of autonomous underwater vehicle for coverage mission in irregular region. *Ocean Eng.* **2021**, *236*, 109407. [[CrossRef](#)]
53. Rúa, S.; Vásquez, R.E.; Crasta, N.; Betancur, M.J.; Pascoal, A. Observability analysis for a cooperative range-based navigation system that uses a rotating single beacon. *Ocean Eng.* **2022**, *248*, 110697. [[CrossRef](#)]
54. Aristizábal, L.M.; Zuluaga, C.A.; Rúa, S.; Vásquez, R.E. Modular Hardware Architecture for the Development of Underwater Vehicles Based on Systems Engineering. *J. Mar. Sci. Eng.* **2021**, *9*, 516. [[CrossRef](#)]
55. Fromm, T.; Mueller, C.A.; Pflingsthor, M.; Birk, A.; Di Lillo, P. Efficient continuous system integration and validation for deep-sea robotics applications. In Proceedings of the OCEANS 2017—Aberdeen, Aberdeen, UK, 19–22 June 2017; pp. 1–6. [[CrossRef](#)]
56. Di Lillo, P.; Simetti, E.; Wanderlingh, F.; Casalino, G.; Antonelli, G. Underwater Intervention With Remote Supervision via Satellite Communication: Developed Control Architecture and Experimental Results Within the Dexrov Project. *IEEE Trans. Control Syst. Technol.* **2021**, *29*, 108–123. [[CrossRef](#)]
57. INCOSE. *INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*, 4th ed.; Wiley: Hoboken, NJ, USA, 2015.
58. Aristizabal, L.M.; Rúa, S.; Zuluaga, C.A.; Posada, N.L.; Vasquez, R.E. Hardware and software development for the navigation, guidance, and control system of a remotely operated vehicle. In Proceedings of the 2017 IEEE 3rd Colombian Conference on Automatic Control (CCAC), Cartagena, Colombia, 18–20 October 2017. [[CrossRef](#)]
59. Zhang, H.; Huang, B.; Ju, H. An Improved SoSE Model—The ‘V+’ Model. In Proceedings of the SOSE 2020—IEEE 15th International Conference of System of Systems Engineering, Budapest, Hungary, 2–4 June 2020; pp. 403–409. [[CrossRef](#)]
60. Yu, C.; Li, Q.; Liu, K.; Chen, Y.; Wei, H. Industrial Design and Development Software System Architecture Based on Model-Based Systems Engineering and Cloud Computing. *Annu. Rev. Control* **2021**, *51*, 401–423. [[CrossRef](#)]
61. Aristizábal, L.M.; Rúa, S.; Gaviria, C.E.; Osorio, S.P.; Zuluaga, C.A.; Posada, N.L.; Vásquez, R.E. Design of an open source-based control platform for an underwater remotely operated vehicle. *DYNA* **2016**, *83*, 198–205. [[CrossRef](#)]
62. Agha, G.; Thati, P. An Algebraic Theory of Actors and Its Application to a Simple Object-Based Language. In *From Object-Oriented to Formal Methods*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 26–57. [[CrossRef](#)]
63. Karmani, R.K.; Agha, G.; Squillante, M.S.; Seiferas, J.; Brezina, M.; Hu, J.; Tuminaro, R.; Sanders, P.; Träffe, J.L.; Geijn, R.A.; et al. Actors. In *Encyclopedia of Parallel Computing*; Springer: Boston, MA USA, 2011; pp. 1–11. [[CrossRef](#)]
64. Burgin, M. Systems, Actors and Agents: Operation in a multicomponent environment. *arXiv* **2017**, arXiv:1711.08319.
65. Agha, G. Concurrent Object-Oriented Programming. *Commun. ACM* **1990**, *33*, 125–141. [[CrossRef](#)]
66. Higuera-Toledano, M.T. About 15 years of real-time Java. In *Proceedings of the 10th International Workshop on Java Technologies for Real-Time and Embedded Systems—JTRES ‘12*; ACM Press: New York, NY, USA, 2012. [[CrossRef](#)]



67. Khamespanah, E.; Sirjani, M.; Mechitov, K.; Agha, G. Modeling and analyzing real-time wireless sensor and actuator networks using actors and model checking. *Int. J. Softw. Tools Technol. Transf.* **2018**, *20*, 547–561. [[CrossRef](#)]
68. Nigro, L. Parallel Theatre: An actor framework in Java for high performance computing. *Simul. Model. Pract. Theory* **2021**, *106*, 102189. [[CrossRef](#)]
69. Latoschik, M.E.; Fischbach, M. Chapter Engineering Variance: Software Techniques for Scalable, Customizable, and Reusable Multimodal Processing. In *Human-Computer Interaction. Theories, Methods, and Tools*; Lecture Notes in Computer Science; Springer International Publishing: Cham, Switzerland, 2014; pp. 308–319. [[CrossRef](#)]
70. Hewitt, C. Actor model of computation: Scalable robust information systems. *arXiv* **2017**, arXiv:1008.1459.
71. Mouelhi, S.; Cancila, D.; Ramdane-Cherif, A. Distributed Object-Oriented Design of Autonomous Control Systems for Connected Vehicle Platoons. In Proceedings of the 2017 22nd International Conference on Engineering of Complex Computer Systems (ICECCS), Fukuoka, Japan, 5–8 November 2017. [[CrossRef](#)]
72. Xu, R.; Li, C. A modular agricultural robotic system (MARS) for precision farming: Concept and implementation. *J. Field Robot.* **2022**. [[CrossRef](#)]
73. Czerwinski, F.; Oddershede, L.B. TimeSeriesStreaming.vi: LabVIEW program for reliable data streaming of large analog time series. *Comput. Phys. Commun.* **2011**, *182*, 485–489. [[CrossRef](#)]
74. Morlock, M.; Meyer, N.; Pick, M.A.; Seifried, R. Real-time trajectory tracking control of a parallel robot with flexible links. *Mech. Mach. Theory* **2021**, *158*, 104220. [[CrossRef](#)]
75. Prokhorenko, L.; Klimov, D.; Mishchenkov, D.; Poduraev, Y. Surgeon–robot interface development framework. *Comput. Biol. Med.* **2020**, *120*, 103717. [[CrossRef](#)]