Sistema de Percepción para un Vehículo Autónomo Submarino

Sebastián A. Villar[1] and Gerardo G. Acosta[1,2]

[1]Grupo INTELYMEC-CIFICEN-CONICET, Facultad de Ingeniería. Universidad Nacional del Centro provincia de Buenos Aires. Av. del Valle 5737 - (B7400JWI) - Olavarría. Argentina.
[2]GEE, Dept. Física. Universidad de las Islas Baleares, Palma, España.
svillar@fio.unicen.edu.ar, ggacosta@fio.unicen.edu.ar

Abstract. La necesidad de la industria *off-shore* para compartir información y recursos de energía a través de cables y tuberías submarinas conduce a un creciente despliegue de infraestructuras sumergidas. Esto requiere de un posterior mantenimiento preventivo. Los vehículos autónomos submarinos (AUVs) representan una alternativa para llevar a cabo esta tarea. En base a la *percepción*, estos vehículos deben estar equipados con distintos dispositivos de sensores como cámaras de vídeo, dispositivo rastreador electromagnético, sonar de barrido lateral, ecosonda muti-haz, como así también dispositivos de ubicación como sistema de posicionamiento global, sistema de navegación inercial, brújula, entre otros. Cada uno de estos dispositivos hay que tratarlos por separado para la captura e interpretación de datos, pero en conjunto para la búsqueda de conocimiento útil que modifique el comportamiento *on-line* de un AUV. Este documento presenta el diseño y desarrollo de una arquitectura de software de un *sistema de percepción* para un AUV (PS-AUV). La arquitectura emplea como entrada datos provenientes de dispositivos de sensores interconectados aplicando distintos procesos, y como salida, conocimiento que alimentará al modelo del mundo del robot que se encuentra implementado en forma de un sistema basado en conocimiento.

Keywords: AUV, sistema de percepción, arquitectura de software.

1 Introducción

Los océanos cubren un área aproximada del 70% de la superficie de nuestro planeta, son un factor importante para la regulación del clima, y proporcionan un gran número de recursos esenciales para el hombre. Sin embargo, el mundo submarino es un medio desconocido debido a sus condiciones adversas. La explotación sostenible de recursos oceánicos y la exploración del lecho marino motivan el uso de una mejor tecnología.

Los robots autónomos vienen experimentando un sostenido grado de avance en los últimos años, tanto en lo que respecta a algoritmos de procesamiento de señales a bordo y su control. Este tipo de dispositivos se han vuelto esenciales para explorar en detalle nuevos ambientes, como el desconocido océano o la exploración de planetas cercanos. En especial, existe gran interés en emplear esta tecnología en inspecciones

de "*estructuras planificadas"* y *"estructuras accidentales"* [9]. Las *estructuras planificadas*, representan manifestaciones intencionales que incluyen tuberías, cables, accesos de embarcaciones, explotación de recursos, etc. Las *estructuras accidentales* se centran en actividades como arqueología de naufragios, efectos derivados de la pesca, contaminación marina, por citar algunos ejemplos [7, 16-18].

La necesidad de comunicación en telefonía, internet y televisión, y el transporte de energía como electricidad, gas y petróleo, han llevado a un incremento de infraestructuras submarinas con cables, oleoductos y tuberías. A fin de mantener esta infraestructura en condiciones seguras de operación, se hace necesaria una inspección periódica preventiva. Es de vital importancia evitar derrames y mantener operativa toda infraestructura sumergida el mayor tiempo posible. Daños producidos por anclajes de las embarcaciones, *free span*[1], rozamiento con objetos o *residuos marinos*, son algunas de las causas de deterioro de las tuberías que al dañarse pueden causar un fuerte impacto ambiental y cortar suministros críticos. Una solución cada vez más empleada para realizar este mantenimiento preventivo de infraestructura sumergida es el empleo de *Vehículos Autónomos Submarinos* (AUVs) para realizar inspecciones periódicas del estado de la infraestructura. Un AUV representa un robot inteligente que navega bajo el agua sin intervención humana. Este vehículo permite obtener datos de mejor calidad y de un modo más económico que la alternativa por control remoto (ROV) [7, 17]. Los AUV se sumergen permitiendo una navegación más rápida y una adquisición de datos más confiables. Durante los últimos años exitosas pruebas se han llevado a cabo utilizando AUVs [6, 7, 19, 21, 28]. Entre ellas las derivadas de los proyectos europeos *AUTOTRACKER* (FP5) y *AUVI* (FP6) [1, 2, 3]. Ellos fueron los antecedentes del prototipo ICTIOBOT[2]. Este prototipo constituye la actual plataforma utilizada para la realización pruebas de investigación [4].

El mantenimiento de infraestructuras sumergida por medio de un AUV contiene tres etapas [5]: *búsqueda y detección del objetivo*, *seguimiento* e *inspección*. Para resolver estas tareas un AUV debe estar equipado con dispositivos de percepción. Una *cámara de vídeo* [22, 25, 26], un *dispositivo rastreador electromagnéticos* (MAG) [4], un *sonar de barrido lateral* (SSS) [11, 18, 20, 21], un *sonar de apertura sintética* (SAS) [15, 23], una *ecosonda muti-haz* (MBE) [14, 21], entre otros, pueden ser utilizados para estos fines. Asimismo, se necesitan dispositivos de posicionamiento del vehículo como *sistema de posición global* (GPS), *sistema de navegación inercial* (INS), medidor de velocidad relativa (DVL), brújula, sensores de profundidad, entre otros.

Para dotar al robot de autonomía en las decisiones, es necesario contar a bordo con la capacidad de procesar los datos provenientes de estos sensores en un tiempo eficiente. De este modo el AUV podrá replanificar tareas y trayectorias durante la inspección. Este trabajo, que presenta el diseño y desarrollo de una arquitectura de software de un *sistema de percepción para un AUV* (PS-AUV) dedicado al mantenimien-

[1] *Free Span:* La tubería no se encuentra apoyada totalmente sobre el fondo del mar.

[2] *ICTIOBOT:* Vehículo Autónomo Submarino ganador del primer premio en robótica del Concurso Nacional Innovar 2012- Ministerio de Ciencia, Tecnología e Innovación Productiva de la Nación.

to de infraestructuras sumergidas, se organiza de la siguiente manera. En la Sección 2 se muestra brevemente la arquitectura general del AUV-ICTIOBOT donde residirá el PS-AUV. Posteriormente en la Sección 3 se presenta el diseño, construcción e implementación de la arquitectura de software del PS-AUV. Finalmente, en la Sección 4 se presentan las conclusiones obtenidas en el trabajo.

2 Arquitectura General del AUV-ICTIOBOT

Los AUVs, al igual que todos los robots móviles autónomos, requieren tener resuelto en forma robusta el problema general de *Navegación, Guiado y Control* (NGC, **N**avigation, **G**uidance and **C**ontrol) [12], como así también la re-planificación *on-line* de los objetivos de la misión. En términos generales un sistema de NGC está compuesto por un conjunto de subsistemas: el *Sistema de Navegación* (NS, **N**avigation **S**ystem), el *Sistema de Guiado* (GS, **G**uidance **S**ystem), el *Sistema de Control* (CS, **C**ontrol **S**ystem) y el *Sistema Planificador de Misión Dinámico* (DMPS, **D**ynamic **M**ission **P**lanning **S**ystem). El NS abarca a todos los sistemas sensores de a bordo específicos para a llevar cabo la misión (GPS, INS, brújula, sensor de profundidad, altímetros, etc). El GS es el responsable de la generación de trayectorias, una vez fijados los puntos por los que debe pasar el robot o también denominados *waypoints*, mientras que el CS comprende los lazos de realimentación que le permiten al robot describir una trayectoria lo más cercana posible a la propuesta por el GS [13]. También se necesita un DMPS, de acuerdo a la aplicación específica que se pretenda para el robot [1-4], lo que completa el sistema de control jerárquico en tiempo eficiente.

El *modulo de fusión de sensores* (SFM, **S**ensor **F**usión **M**odule) se utiliza para estimar la trayectoria de la tubería a partir de distintos sensores (SSS, MBE, MAG, etc). Toda la información, provenientes de los sensores, se combina en el SFM dando una estimación de la posición y la dirección del objetivo a inspeccionar.

Un *planificador de misión* (MP, **M**ission **P**lanner), de acuerdo con la aplicación del robot, es necesario para realizar las tareas de manera autónoma. La adquisición de datos por los sensores o acciones especiales puede ser también considerada dentro del MP. El MP se compone de dos tipos de objetivos: objetivos a largo plazo estables, y los objetivos a corto plazo más cambiantes. Los primeros, se dan de antemano, en una forma rígida a través de la interfaz hombre-máquina y conforman el *plan estático de la misión* (SMP, **S**tatic **M**ission **P**lanner). Los objetivos a corto plazo pueden ser modificados constantemente, y constituyen el *plan dinámico de la misión* (DMP, **D**ynamic **M**ission **P**lanner), y varía según el movimiento del vehículo que avanza en el mundo real (*re-planificar* la misión).

A partir de la información brindada por el NS (datos de la posición de AUV y del objetivo), SFM (datos de la estimación de la posición del objetivo) y las *zonas de exclusión* (datos de zonas prohibidas), el DMPS debe ser capaz de decidir una trayectoria para diferentes situaciones como puede ser la búsqueda de una tubería, seguimiento, navegando más cerca del objetivo, entre otros. Esta trayectoria deseada se define como una colección *waypoints* para ser alcanzado por el vehículo.

La información almacenada en *zonas de exclusión* representa datos sobre la ubicación de las infraestructuras, profundidades, etc. Esta información es accedida constantemente por el DMPS y por el *sistema de prevención de obstáculos* (OAS, **O**bstacle **A**voidance **S**ystem). El OAS recibe datos del *sonar frontal* (FLS, **F**orward-**L**ooking **S**onar). Cuando se detecta un obstáculo cerca de un *waypoint*, el OAS propone corregir la trayectoria deseada del DMPS. El *planificador de trayectoria* (PP, **P**ath **P**lanner), en esta arquitectura, sólo decide la trayectoria dada por el DMPS es o no posible, de acuerdo con los resultados de OAS. A continuación, los *waypoints* pertenecientes a la trayectoria deseada son la entrada al GS. El GS es el responsable de la generación de trayectorias, una vez fijados los *waypoints*, por los que debe pasar el robot, mientras que el CS comprende los lazos de realimentación que le permiten al robot describir una trayectoria lo más cercana posible a la propuesta por el GS [13], comandando los actuadores del vehículo (hélices, timones, etc.). La descripción completa de esta arquitectura implementada en el prototipo ICTIOBOT se encuentra en [4].

3 Sistema de Percepción (PS-AUV)

En base a la arquitectura general presentada en la sección anterior, el PS-AUV se integra con el NS y SFM. En particular, con la adquisición y procesamiento de aquellos datos brindados por los sensores que ayuden a obtener el mayor conocimiento para retroalimentar el DMPS de modo tal de modificar el comportamiento del vehículo en tiempo eficiente.

3.1 Metodología de desarrollo

La metodología de desarrollo denominada diseño dirigido por atributos (ADD) [29] se utilizó para el diseño y construcción de la arquitectura de software del PS-AUV. La metodología ADD representa una aproximación iterativa basada en la recursiva descomposición de procesos donde tácticas y patrones arquitecturales son escogidos para satisfacer un conjunto de *escenarios*[3]. Este método recibe como entrada una lista de *architectural drivers*[4] y produce como salida estructuras que conforman al diseño de la arquitectura. ADD de manera resumida tiene los siguientes ítems: **(1)** Revisar la información de los *architectural drivers*; **(2)** Elegir un elemento a descomponer; **(3)** Elegir un subconjunto de *architectural drivers* a satisfacer; **(4)** Elegir conceptos de diseño para satisfacer los *architectural drivers*; **(5)** Asignar responsabilidades a los elementos; **(6)** Definir interfaces; **(7)** Verificar la satisfacción de los *drivers* seleccionados en el paso **(3)**. **(8)** Repetir los pasos anteriores para elementos que requieran un mayor refinamiento hasta cubrir la mayoría de los *architectural drivers*.

[3] Los *escenarios* representan cambios posibles ante los que puede estar expuesto el sistema.

[4] Los *architectural drivers* describen los principales beneficios priorizados que deberían adecuarse en el diseño de la arquitectura software.

Para el ítem **(4)** se eligen distintos conceptos de diseño arquitectónicos que pueden referirse a textos sobre arquitectura de software [8, 24]. Generalmente ADD finaliza cuando se han satisfecho la mayor cantidad de los *architectural drivers*.

3.2 Principales requerimientos del PS-AUV

Se detallan los principales requerimientos funcionales y no funcionales del PS-AUV: **(1)** El PS-AUV deberá poder seleccionar el o los dispositivos a trabajar; **(2)** El PS-AUV deberá obtener e interpretar los datos brindados por los dispositivos de sensores; **(3)** El PS-AUV deberá tener la alternativa de seleccionar *procesos*[5] y *cadenas de procesos*[6] que se apliquen a los datos brindados por un sensor, como así también a los datos en conjunto formados por distintos sensores; **(4)** El PS-AUV deberá tener la posibilidad realizar pruebas *off-line* sobre un entorno de simulación de un *proceso* o *cadena de procesos* para verificar su aplicabilidad en pruebas *on-line*; **(5)** El PS-AUV debe tener la capacidad agregar fácilmente distintos dispositivos de sensores con sus características en particular; **(6)** El PS-AUV debe integrarse fácilmente a la tecnología hardware y software que contiene actualmente el AUV-ICTIOBOT; **(7)** El PS-AUV deberá adaptarse fácilmente a la modificación o agregado de nuevos *procesos*.

3.3 Diseño de Arquitectura de Software para PS-AUV

La arquitectura de software resultante para PS-AUV se observa en la Fig. 1. La primera decisión arquitectónica satisface *architectural driver* de prioridad alta: *"el PS-AUV debe poder integrarse fácilmente al software/hardware que contiene actualmente el AUV-ICTIOBOT"*. Para ello se utilizó la táctica de separación de la interfaz de usuario del resto de la aplicación [8]. El resto de la aplicación representa el PS-AUV que será embebido dentro del AUV-ICTIOBOT. Por otro lado, la interfaz de usuario se encarga de las preferencias de visualización soportando cambios sin afectar al resto del sistema. Asimismo, la interfaz puede adecuarse a los gustos del usuario para facilitar el aprendizaje y uso. Luego de distintas pruebas de simulación *off-line* se consigue una configuración precisa que será migrada al ICTIOBOT.

La segunda decisión arquitectónica se centró en satisfacer el *architectural driver*: *"el PS-AUV deberá implementarse en un entorno de simulación para pruebas y análisis de resultados de distintos procesos y cadenas de procesos en tiempo off-line"*. Para una mejor separación de responsabilidades se aplicó el patrón modelo, vista y controlador (MVC) [8, 24]. Un MVC divide una aplicación en tres áreas: modelo, vista y controlador. Esta táctica separa la interfaz de usuario en vista y controlador (ver Fig. 1). El modelo contiene el núcleo funcional de la aplicación encapsulando los datos apropiados y el procesamiento específico, es decir, el PS-AUV. El PS-AUV desconoce las interfaces asociados a él obteniendo independencia con cualquier vista o controlador, por otro lado, las vistas y controladores si conocen al PS-AUV. Asimismo, el entorno de simulación queda a cargo de las vistas y controladores, pudien-

[5] Definimos *proceso* a una técnica, algoritmo o refinamiento aplicado a datos en particular.
[6] Definimos *cadena de procesos* a un conjunto anidados y secuenciales de *procesos*.

do simular distintas configuraciones y pruebas de procesos y cadenas de procesos. De esta manera, las diferentes vistas presentan la información del PS-AUV de diferentes maneras. Los controladores aceptan entradas del usuario como eventos. Los eventos son traducidos en peticiones para el modelo o las vistas asociadas.

La tercera decisión arquitectónica se centra en refinar el PS-AUV. El *architectural driver* seleccionado es: *"el PS-AUV debe adaptarse a distintos hardware de dispositivos de sensores"*. Para satisfacer el driver se utiliza la táctica arquitectónica que encapsula las responsabilidades de adaptabilidad al hardware en un único componente denominado capa de portabilidad o *nivel de dispositivos* [8]. Este nivel ofrece a los distintos sensores una interfaz abstracta a su entorno. Esta interfaz permanece sin variaciones, aislando al PS-AUV, debido a que el sistema se adapta a cualquier tipo de dispositivos, aunque el nivel de dispositivos pueda modificarse.

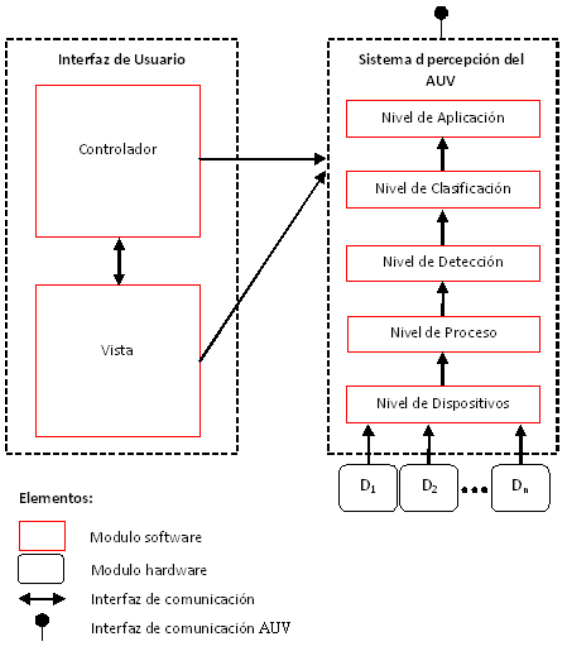

Fig. 1. Arquitectura general del PS-AUV.

La cuarta decisión arquitectónica se centra en satisfacer *architectural driver*: *"el PS-AUV debe poder aplicar distintos procesos o cadenas de procesos en base a los datos brindados por los dispositivos de sensores"*. Para satisfacer el *driver* se aplicará el patrón arquitectónico de capas [8, 24]. Las capas ayudan a estructurar aplicaciones y ser descompuestas en grupos de sub-tareas. Cada capa tiene un nivel particular de abstracción, donde un cambio realizado dentro de una capa no afecta a las demás. De esta manera, el PS-AUV se descompone en niveles de proceso, detección, clasificación y aplicación. El *nivel de proceso* se encarga del procesamiento de datos de los sensores de bajo nivel (correcciones de errores, problema de lecturas, ruido, etc.). El *nivel de detección* tiene la responsabilidad de detectar los objetos de estudio. El *nivel*

de clasificación se encarga de clasificar las detecciones en un grupo finito de clases. El *nivel de aplicación* propone escenarios que alimentarán al sistema basado en conocimiento, el cual en definitiva será el responsable de modificar el comportamiento del robot, replanificando trayectorias de navegación y tareas a ejecutar.

La quinta decisión arquitectónica se enfocó en satisfacer el *architectural driver*: *"el PS-AUV debe procesar los datos de los sensores on-line en un tiempo útil para el AUV-ICTIOBOT"*. Para satisfacer el *driver* se aplicará el patrón *pipe&filter* [8, 24]. Este patrón divide la tarea de un sistema en varias etapas de procesamiento secuencial, donde los datos de salida de una etapa representan la entrada a la siguiente etapa. Cada etapa de procesamiento está implementada con un componente *filter*, que enriquece, refina o transforma los datos de entrada. Los *filters* están conectados por medio de *pipes*. Cada *pipe* implementa el flujo de datos entre etapas de procesamiento adyacentes. Los *filters* del PS-AUV representan los niveles de proceso independientes o *threads* siguientes: detección, clasificación, aplicación y dispositivos, y los *pipes*, la manera de comunicación entre estos niveles de proceso. El *pipe* sincroniza los filtros mediante un buffer FIFO (*first in-first out*). La ventaja fundamental de aplicar este patrón arquitectónico es la eficiencia en procesamiento paralelo cuando se implementa en un sistema multiprocesador.

La sexta decisión arquitectónica se centró en refinar cada nivel del PS-AUV. El *architectural driver* seleccionado es: *"el PS-AUV debe poder adaptarse a modificaciones y agregado de nuevos procesos"*. Para satisfacer el *driver* reduciendo costo y tiempo de las modificaciones se aplican las tácticas arquitectónicas *localizar los cambios*, *coherencia semántica* y g*eneralizar módulos* [8, 24]. Para ello, se toma cada nivel por separado creando una herencia de *procesos* específicos.

3.4 Modelo Lógico de Clases para PS-AUV

El modelo lógico de clases para el PS-AUV se observa en la Fig. 2. El diseño consiste en un MVC (modelo o PS-AUV, vista y controlador). Para implementar la comunicación entre el PS-AUV y las vistas se utilizó el mecanismo *change-propagation* con el diseño del patrón *Observer* o *Publish-Suscribe* [24]. Este patrón define una dependencia de uno a muchos entre el PS-AUV y las vistas. De esta manera, cuando el PS-AUV cambia de estado, sus vistas asociadas son notificadas automáticamente. La clase *observable* conoce a todos sus observadores del PS-AUV representadas por las vistas. La clase *observer* define una interfaz de actualización para los objetos a ser notificados en base a cambios del PS-AUV. La clase *DataProcessing* representa el núcleo funcional de la aplicación, donde se encapsula los datos apropiados y el procesamiento específico. La clase *View* presenta la información al usuario del estado del PS-AUV de diferentes maneras. La clase *Controller*, quien hace de intermediario entre el PS-AUV y las vistas, acepta entradas del usuario como eventos que son traducidos en peticiones para el PS-AUV o las vistas asociadas. Asimismo, las clases abstractas *Controller* y *View* contienen *puntos de extensión* (EP, **E**xtention **P**oint) para la escalabilidad del sistema. El EP 1 (Fig. 2) permite el agregado de nuevas vistas de cualquier elemento observable del PS-AUV. Asimismo, el EP 2 (Fig. 2) permite el agregado de nuevos controladores de una vista particular.

Cada nivel de abstracción del PS-AUV representa un proceso independiente, tal como se había establecido en el diseño. La comunicación entre los procesos es por medio de un *buffer* quien es responsable de la sincronización. En la Fig. 2 también se aprecia la clase *ProcessingFilter* que representa un proceso independiente heredando el comportamiento de la clase *Threads* (implementa los métodos *run* y *stop*). Cada *ProcessingFilter* contiene dos *DataBuffer* que conectan los *filters* adyacentes. El *DataBuffer* realiza la sincronización y almacenado de la combinación de los datos de sensores. Estos datos se almacenan en clase *SensorData*. El *DataBuffer* contiene dos métodos, *send* y *receive*, que permiten enviar y recibir objetos del tipo *SensorData*. Cada *ProcessingFilter* contiene una clase del tipo *DataProcessing*. Esta clase contiene un EP 3 (Fig. 2) para agregar cualquier nivel de abstracción. Asimismo, el *DataProcessing* posee una clase concreta denominada *CompositeDataProcessing*, responsable de construir objetos complejos mediante composición recursiva de objetos similares. Por este motivo, un compuesto *DataProcessing* es conformado por un conjunto de niveles de abstracción. Esta alternativa permite que los niveles de abstracción se ejecuten en un sistema multiprocesador.

Cada nivel representa una interfaz unificada para un conjunto de interfaces en un subsistema. Esta interfaz se diseña mediante el patrón de diseño *Facade* [24]. Este patrón define una interfaz de alto nivel que permite la comunicación con el subsistema. Por ejemplo, el *ProcessLevel* representa la fachada de la implementación del nivel de proceso. La clase demanda la realización del proceso sin conocer como este lo realiza. Además, la clase *ProcessLevel* representa el único punto de acceso para comunicarse con la clase *Process.* Asimismo, la clase *Process* contiene un EP (4) (Fig 2) para el agregado de cualquier tratamiento a aplicar sobre el objeto *SensorData*. Tener en cuenta, que la clase *Process* contiene un método abstracto denominado *process* que debe ser implementado por cada una de las clases concretas. Este método recibe como parámetro un *SensorData* y retorna su transformación aplicando un procesamiento particular. Además, la clase *CompositeProcess* permite generar un objeto complejo de procesamientos. De esta manera, el usuario puede instanciar un proceso compuesto por una serie de procesos particulares aplicados en cadena. De igual manera que la clase *Process*, se tiene la clase *Detection*, *Classification* y *Application* con sus respectivos EP (5), (6) y (7) (Fig 2), que permiten el agregado de cualquier algoritmo o técnica de detección, clasificación o aplicación a ser utilizada.

La clase *DeviceLevel* representa la fachada de implementación del nivel de dispositivo. Esta clase demanda el tratamiento de los dispositivos de sensores sin conocer cómo éste lo realiza, fabricando un *SensorData*. Este patrón de diseño es conocido como *Abstract Factory* [24]. La clase *DeviceLevel* recibe dispositivos de sensores (*SensorDevice*) y los envía a procesar (*ProcessDevice*). La clase *SensorDevice* contiene la trama de datos e información que brinda el dispositivo. Se define un período de procesamiento que comprende N_i muestras de los i dispositivos sensores que están siendo utilizados por el robot. De este modo, en cada período de procesamiento se reciben $N=N_1+N_2+\ldots+N_i$ objetos *SensorDevice* que representan los datos brindados por los dispositivos sensores conectados. Asimismo, la clase *SensorDevice* contiene un EP (8) Fig. 2 para el agregado de cualquier otro dispositivo. Además, se tiene una clase concreta denominada *CompositeSensorDevice* que permite crear un objeto com-

plejo de un grupo de sensores concretos. Por ejemplo un *CompositeSensorDevice* puede contener dos dispositivos relacionados semánticamente que producirán una fusión de datos.

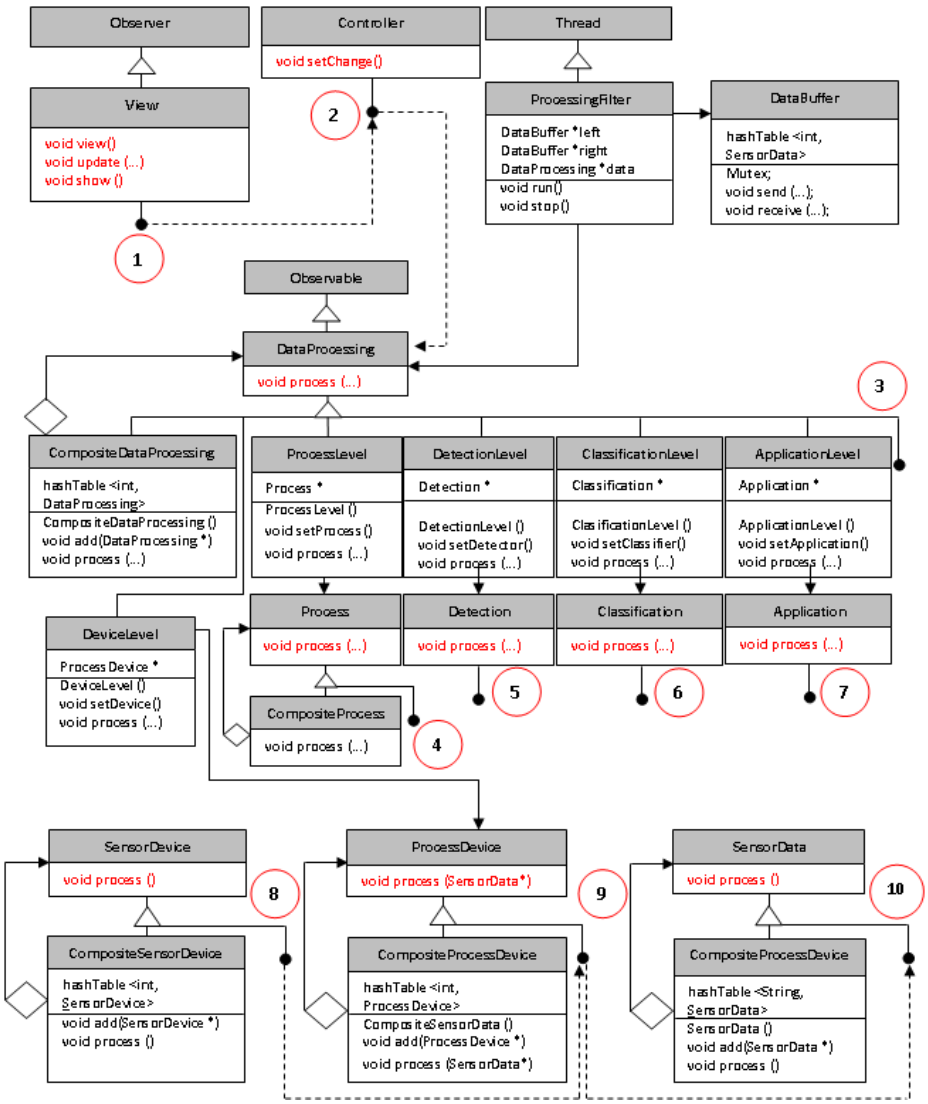

Fig. 2**.** Modelo lógico de clases del PS-AUV.

Cada clase *SensorData* tiene su forma de almacenar la información, interfaz de agregado y consulta de datos. Los procesamientos parciales de cada proceso se almacenan dentro de la estructura de datos. Asimismo, se tiene una clase *CompositeSensorData* que permite realizar objetos compuestos, es decir, datos compuestos de dis-

tintos sensores concretos. Esta alternativa permite enviar un solo objeto *SensorData* fusionado. Además, la clase *SensorData* contiene un EP (10) (Fig. 2) que permite el agregado de una nueva estructura de almacenamiento de datos de sensores.

3.5 Implementación del PS-AUV

El modelo lógico se implementó con el entorno de programación (IDE) *QtCreator* de Nokia para sistemas GNU/Linux con código de implementación C++. Además, el agregado de librerías OpenCV 2.3 [10] para utilizar sus estructuras de datos, técnicas de procesamiento digital de imágenes y técnicas de clasificación. Este diseño de clases fue implementado sobre un PC con CPU 2-GHz Intel(R) Core(TM) 2 Duo, y 2-GB de memoria RAM, con SO Linux. El entorno de desarrollo así implementado, servirá de soporte para la generación de distintas instancias con distintas características y potencialidades, que variarán en función de la aplicación esperada para el robot.
Una instanciación de esta arquitectura presentada se encuentra en desarrollo realizando pruebas *off-line*. El AUV-ICTIOBOT con una CPU FitPC-2 a bordo será la plataforma utilizada para validar el funcionamiento de esta arquitectura. Las pruebas se realizarán con el AUV-ICTIOBOT navegando por superficie para la inspección de un ducto emisario instalado en las costas de Mar del Plata. El AUV se encuentra equipado con: GPS Garmin, SSS StarFish 450F, brújula Philips KMZ51, MBE Tritech Seaking y INS marca XSens. Estos dispositivos transmiten por separado datos y se encapsulan en un objeto *SensorDevice*. La interpretación o *parseo* de estos datos los realiza un objeto *ProcessDevice* fabricando un objeto *SensorData*. El *nivel de proceso* aplica suavizado, correcciones geométricas, eliminación de información irrelevante, entre otros, como se presentó en [27]. El *nivel de detección* y *clasificación* aplica una técnica denominada CA-CFAR para la detección de la tubería. Por último, en el *nivel de aplicación* proporciona la determinación de estados o escenarios, del tipo “ducto detectado en situación normal”, “ducto detectado en free-span”, “ducto perdido”, y otros, que alimentan al sistema basado en conocimiento que se detalla en [3].

4 Conclusiones

En este trabajo se presentó el diseño, construcción e implementación de una arquitectura de software de un *sistema de percepción* para el mantenimiento de infraestructuras sumergidas por medio de un AUV. La arquitectura obtiene como entrada datos de los dispositivos de sensores interconectados aplicando distintos procesos, y proporciona como salida la determinación del escenario en el que se encuentra realizando la inspección el robot. De esta forma, el PS-AUV constituye una fuente de conocimiento privilegiado para el modelo del mundo del robot, que se encuentra implementado en forma de un sistema basado en conocimiento. Una de las principales ventajas del PS-AUV es la posibilidad de manejar distintos dispositivos de sensores interpretando los datos brindados, aplicando procesos y cadenas de procesos que añaden conocimiento a los datos crudos provenientes de los sensores. Otra gran ventaja es que le permite integrarse a la arquitectura general (Hw/Sw) del AUV-ICTIOBOT, adaptarse fácil-

mente a la modificación o agregado de nuevos *procesos* y la posibilidad de realizar pruebas *off-line* sobre un entorno de simulación completamente asimilable al entorno de funcionamiento, para verificar su aplicabilidad en pruebas *on-line*.

Agradecimientos.

Este trabajo se realizó gracias al financiamiento de los proyectos DPI2009-11298, MICINN de España, CONICET – PIP 11420090100238, y ANPCyT – PICT-2009-0142 de Argentina. El autor Sebastián Aldo Villar quiere reconocer a la IEEE/OES Student Scholarships.

Bibliografía.

1. Acosta, G.G.: State of the art on trajectory generation for AUV with artificial intelligence techniques AUVI. Project Internal Report Nº 3027-1 (2005).
2. Acosta, G.G., Curti, H., Calvo, O.: Autonomous underwater pipeline inspection in Autotracker Project: the navigation module. In IEEE OCEANS (2005) 1: 389-394.
3. Acosta, G.G., Curti, H.J., Calvo, O.A., Rossi, S.R.: A knowledge-based approach for AUV path planner development. In: WSEAS Transactions on Systems (2006) 5(6):1417-1424.
4. Acosta, G.G., Curti, H., Calvo, O., Rossi, S.: Some Issues on the Design of a Low-Cost Autonomous Underwater Vehicle with an Intelligent Dynamic Mission Planner for Pipeline and Cable Tracking, in Underwater Vehicles, Ed. InTech, (2009) 1-18.
5. Ageev, M.: AUV Equipment and Control While Conducting Investigation of Underwater Pipeline (in Russian). In: Underwater Technologies (2005) 1:68-72.
6. Asai, T., Kojima, J., Asakawa, K., Iso, T.: Inspection of submarine cable of over 400 km by AUV. In: International Symposium on Underwater Technology (2000) 133-135.
7. Balasuriya, A., Ura, T.: Multi-sensor fusion for autonomous underwater cable tracking. In: IEEE OCEANS (1999) 1:209-215.
8. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, 2ed. Addison-Wesley (2003).
9. Blondel, P.H.: The Hand book of Sidescan Sonar. Springer Praxis Books (2007).
10. Bradski, G., Kaehler, A.: Learning OpenCV: Computer Vision with the OpenCV (2008).
11. Celik, T., Tjahjadi, T.: A Novel Method for Sidescan Sonar Image Segmentation. In: IEEE J. Ocean. Eng. (2011) 36(2):186-194.
12. Fossen, T.: Marine Control Systems, Ed. Marine Cybernetics, Norway (2002).
13. Fossen, T.: Guidance and control of Ocean Vehicles, Chichester, UK, Wiley Ed (1994).
14. Greenaway, S.F., Weber, T.C.: Test methodology for evaluation of linearity of multibeam echosounder backscatter performance. In: OCEANS (2010) 20-23.
15. Hagen, P.E., Hansen, R.E.: Synthetic aperture sonar challenges. In: Hydro International (2008) 12(4): 26-31.
16. Hellequin, L., Boucher, J.M., Lurton, X.: Processing of high-frequency multibeam echo sounder data for seafloor characterization. In: IEEE J. Ocean. Eng. (2003) 28(1):78-89.
17. Ishøy, I., Bjerrum, A., Calvo, O., Acosta, G.G., Petillot, Y., Evans, J., Kyruakopoulos, K., Lionis, G., Slater, T., Nunn, R.: New challenges for AUTOTRACKER. In: Unmanned Underwater Vehicle Showcase, Southampton, UK (2002).
18. Lurton, X.: An introduction to Underwater Acoustics, Ed. Springer Verlag Inc (2003).

19. Matsumoto, S., Ito, Y.: Real-Time Vision-Based Tracking of Submarine-Cables for AUV/ROV. In: OCEANS (1995).
20. Perry, S.W., Guan, L.: A Recurrent Neural Network for Detecting Objects in Sequences of Sector-Scan Sonar Images. In: IEEE J. Ocean. Eng. (2004) 29(3): 857-871.
21. Petillot, Y., Reed, S., Bell, J.: Real time AUV pipeline detection and tracking using side scan sonar and multi-beam echosounder. In: OCEANS MTS/IEEE (2002) 1:217-222.
22. Rives, P., Borrelly, J.: Underwater pipe inspection task using visual servoing techniques. In: IEEE Internatinal Conference on Robotics and Systems (1997) 1:63-68.
23. Sæbø, T.O., Callen, H.J., Hansen, R.E.: Bathymetric capabilities of the HISAS interferometric synthetic aperture sonar. In: OCEANS MTS/IEEE (2007) 1-10.
24. Shaw, M., Garlan, D.: Software architecture: perspectives on an emerging discipline, Prentice-Hall, Inc., Upper Saddle River, NJ (1996).
25. Shi, J., Tomasi, C.: Good feature to track. In: CVPR IEEE (1994).
26. Tomasi, C., Kanade, T.: Shape and motion from image streams under orthography. In: International Journal of Computer Vision (1992) 9:137-154.
27. Villar, S. A., Acosta, G. G. Sousa Senna, A. L., Rozenfeld A. F.: Evaluation of an efficient approach for target tracking from acoustic imagery for the perception system of an AUV. *submitted to* J. Adv Robotics Syst Underwater Vehicles (2013).
28. Wang, Y., Lane, D.M.: Subsea vehicle path planning using nonlinear programming and constructive solid geometry. In: IEEE Control Theory Applications (1997) 144:143-152.
29. Wojcik, R., Bachmann, F., Bass, L., Clements, P., Merson, P., Nord, R.L., Wood, B.: Attribute-Driven Design (ADD), Version 2.0, Technical Report (2006).