Opportunistic (Re)planning for Long-Term Deep-Ocean Inspection

An Autonomous Underwater Architecture

By Elisa Tosello, Paolo Bonel, Alberto Buranello, Marco Carraro, Alessandro Cimatti, Lorenzo Granelli, Stefan Panjkovic, and Andrea Micheli

Robots are increasingly used in subsea environments because of their positive impact on human safety and operational capabilities in the deep ocean. However, achieving full autonomy remains challenging because of the extreme conditions they encounter. In this context, we propose an autonomous underwater architecture for long-term deep-ocean inspection that robustly plans activities and efficiently deliberates with no human help. It combines the innovative Saipem's Hydrone-R subsea vehicle with an advanced planning architecture, resulting in a robot that autonomously perceives its surroundings, plans a mission, and adapts in real time to contingencies and opportunities. After describing the robot hardware, we present the technological advancements achieved in building its software, along with several compelling use cases. We explore scenarios where the robot conducts long-term underwater missions operating under resource constraints while remaining responsive to opportunities, such as new inspection points. Our solution gained significant reliability and acceptance within the oil and gas community as evidenced by its current deployment on a real field in Norway.

Digital Object Identifier 10.1109/MRA.2024.3352810
Date of current version: 6 February 2024

INTRODUCTION

Subsea pipelines play a crucial role in carrying oil and gas: any damage or malfunction can result in significant environmental and financial repercussions. Consequently, regular inspections are essential to guarantee the assets' integrity. These inspections can include internal checks conducted by intelligent pegs and crawling robots or external examinations utilizing electromagnetic, radiographic, acoustic, and fiber-optic sensors [1]. In this article, we focus on external inspection and aim to address the needs of asset owners and operators to optimize procedures and offer adaptable technologies.

Unmanned underwater vehicles are pivotal in this regard as they can operate for an extended duration in deep waters, surpassing the depths reachable by human divers. Their usage

©SHUTTERSTOCK.COM/CRYSTAL EYE STUDIO

is independent of the field, and they can promptly react to changes in the setup or assignment. We distinguish remotely operated vehicles (ROVs), which rely on remote control by pilots and require connections to supporting vessels for continuous power supply, and autonomous underwater vehicles (AUVs), which instead operate autonomously. AUVs are particularly compelling for our specific application as they enhance efficiency and safety by minimizing human involvement in demanding and high-risk tasks. Additionally, they help energy companies in reducing their ecological footprint.

For optimal task performance, AUVs must exhibit deliberate behavior. Deliberation involves undertaking actions driven by specific objectives backed by rational reasoning aligned with these objectives [2]. For instance, this capability allows a vehicle to dynamically monitor the available resources, like the battery level, and optimize their consumption. It enables the robot to be aware of its surroundings and adjust its parameters according to environmental changes, e.g., minimizing trajectory deviation while navigating. It guarantees the system to make informative decisions when handling unforeseen events.

We propose a hardware and software architecture that enables deliberative deep-sea inspection for extended periods. The robot is Saipem's Hydrone-R: a drone equipped with innovative hardware designed to withstand the extreme conditions of the deep ocean and make the system a subsea *resident* (see Figure 1). Indeed, Hydrone-R has been operating autonomously for more than 200 days (since June 2023) on a real offshore oil and gas field in Norway—the goal is to achieve 10 years of service (see https://www.saipem.com/en/projects/hydrone-njord-field-development). It is equipped with a range of specialized sensors, computing power, advanced control systems, and artificial intelligence (AI)-driven navigation and inspection modules that leverage feature recognition. Such components empower the robot with the ability to autonomously perceive its surroundings, detect pipelines, precisely localize itself, and accurately estimate its internal state. We put on board a task planner that effectively

utilizes this information to plan safe inspection missions while adhering to resource constraints. An Orchestrator manages the mission and triggers replanning in real time in case of contingencies, e.g., an unexpected decrease in available battery charge, and emerging opportunities, e.g., the inspection of not previously considered locations (see Figure 2). We perform planning via the Unified Planning (UP) library (available at https://github.com/aiplan4eu/unified-planning): the Python3 open source library developed by the AIPlan4EU project (see https://www.aiplan4eu-project.eu/). It allows users to model and manipulate classical, numerical, and temporal assignments [3] to the point of solving complex problems, such as multiagent and task and motion planning ones. In our case, we exploit the UP library to enable Hydrone-R in planning and optimizing numerical and temporal missions. This includes managing resource constraints, prioritizing targets, and defining ordering rules among the goals. These features promote the vehicle's long-term stay in the deep ocean.

> “AUVs are particularly compelling for our specific application as they enhance efficiency and safety by minimizing human involvement in demanding and high-risk tasks.”

The rest of the article is organized as follows. In the next section, we summarize related work. In the section "The Underwater Vehicle," we present the robot hardware, and in the section "Problem Formulation," we define the planning problem arising from autonomous deep-sea inspection. We describe the UP library, the planning architecture, and the methods implemented to handle contingencies and new opportunities in the sections "The UP Library," "Our Advanced Planning Architecture," and "Handling Contingencies and Opportunities," respectively. In the section "Experimental Experience," we present our ongoing experimental experience. Finally, we draw some conclusions and discuss future work in the section "Conclusions."

THE STATE OF THE ART

The number of available AUVs is continually increasing. Among others, KM's HUGIN robot [4], although a general-purpose AUV, has proven effective in subsea pipeline inspection, even if its torpedo shape necessitates constant forward motion. On the other hand, Saab's Seaeye Sabertooth [5] and the Girona 500 [6] are purpose-built for asset inspection, with the latter tailor-made and widely employed in academic projects. However, they lack *true* autonomy, missing the capability to dynamically adjust missions in real time to face unexpected or changed conditions. Moreover, they cannot be subsea *residents* for long periods (months or years). Our goal is to equip our robot with both capabilities. Indeed, full autonomy and subsea residency minimize reliance on support vessels, boosting safety, reducing offshore personnel needs, and cutting the carbon footprint.

FIGURE 1. The Saipem Hydrone-R AUV.

In this context, Saipem started developing underwater robots back in 2015. Its Hydrone family currently consists of three main vehicles: Hydrone-W, FlatFish, and Hydrone-R. They share the whole control system (proprietary and internally developed) and most of the algorithms but are tailored to answer to specific offshore energy needs. Hydrone-W is a fully electric ROV integrated with autonomous capabilities designed only to assist human pilots when operating in extremely demanding conditions. FlatFish is a survey vehicle able to perform long fully autonomous missions. Finally, Hydrone-R is both a ROV and an AUV. It can be teleoperated, or it can autonomously track and follow pipelines, inspect risers, and accomplish complicated survey patterns and visual inspections. Most of all, it is proving to be a subsea resident: it has been operating in Norwegian waters for more than 200 days, intending to stay in the water for 10 years.

Along with cutting-edge hardware, we aim to give the robot decision-making and managerial skills that allow it to act deliberately and *truly* autonomously. In this sense, Albiez et al. utilize behavior-based methods to manage the tasks and a plan manager to control the deployment, activation, and deactivation of the behaviors to maintain progress, enable AUVs to fulfill the missions, and handle underinformed situations [7]. In [8], the authors compose deliberative and reactive layers that operate concurrently and exchange feedback on the environmental conditions, goals, plans, and situations. The former manages the execution of scheduled tasks, while the latter manages real-time reactions to critical events. In [9], Cashmore et al. consider AUV assignments where opportunities to achieve additional utility can arise during execution. The authors frame the missions as temporal planning problems, treating the opportunities as soft goals with high utility. These goals are dynamically addressed as they arise, while ensuring the achievement of the problem's hard goals. Finally, [10] proposes a fully automated task allocation algorithm for AUVs that optimizes mission planning for threat detection. The task assignment problem is modeled as a traveling salesman problem with distinct start and

end points. The authors optimize the total sailing distance and AUV turning angle while traversing threat points in a specific order, considering constraints like the minimum radius of turn and speed. The problem is solved using an improved ant colony optimization algorithm with fuzzy logic and dynamic pheromone volatilization.

In contrast to existing approaches, our planning architecture gives generality to the problem addressed, allowing the robot to solve planning problems of different types—numerical and temporal—thanks to the exploitation of the UP library developed by AIPlan4EU. Our goal definition allows us to assign resource constraints to each mission, like restrictions on the residual battery, time allocation, and disk space usage. Moreover, we can sort the inspection targets in sequences, temporally concatenate them, or arrange them in a partial order fashion, facilitating the incorporation of optional goals and the definition of a custom priority hierarchy among them. Finally, we can replan in case of new opportunities and contingencies via a close interaction between the UP library and the robot's Orchestrator. Such a system can operate across the entire Saipem Hydrone family, with its maximum utility being demonstrated when deployed on Hydrone-R.

THE UNDERWATER VEHICLE

In this section, we introduce Hydrone-R and the tools that enable it to support our planning module and act deliberately.

Hydrone-R serves as both a ROV and an AUV, capable of either teleoperation or autonomous functioning. It can track and inspect pipelines and risers and accomplish complicated survey tasks. A detailed description of its components follows.

- ***Optical communication.*** An optical modem enables wireless control of the vehicle from its docking station or locations where operators need to receive real-time feedback.

- ***Acoustic communication.*** A low-bandwidth, long-range acoustic modem enables operators to receive diagnostic data

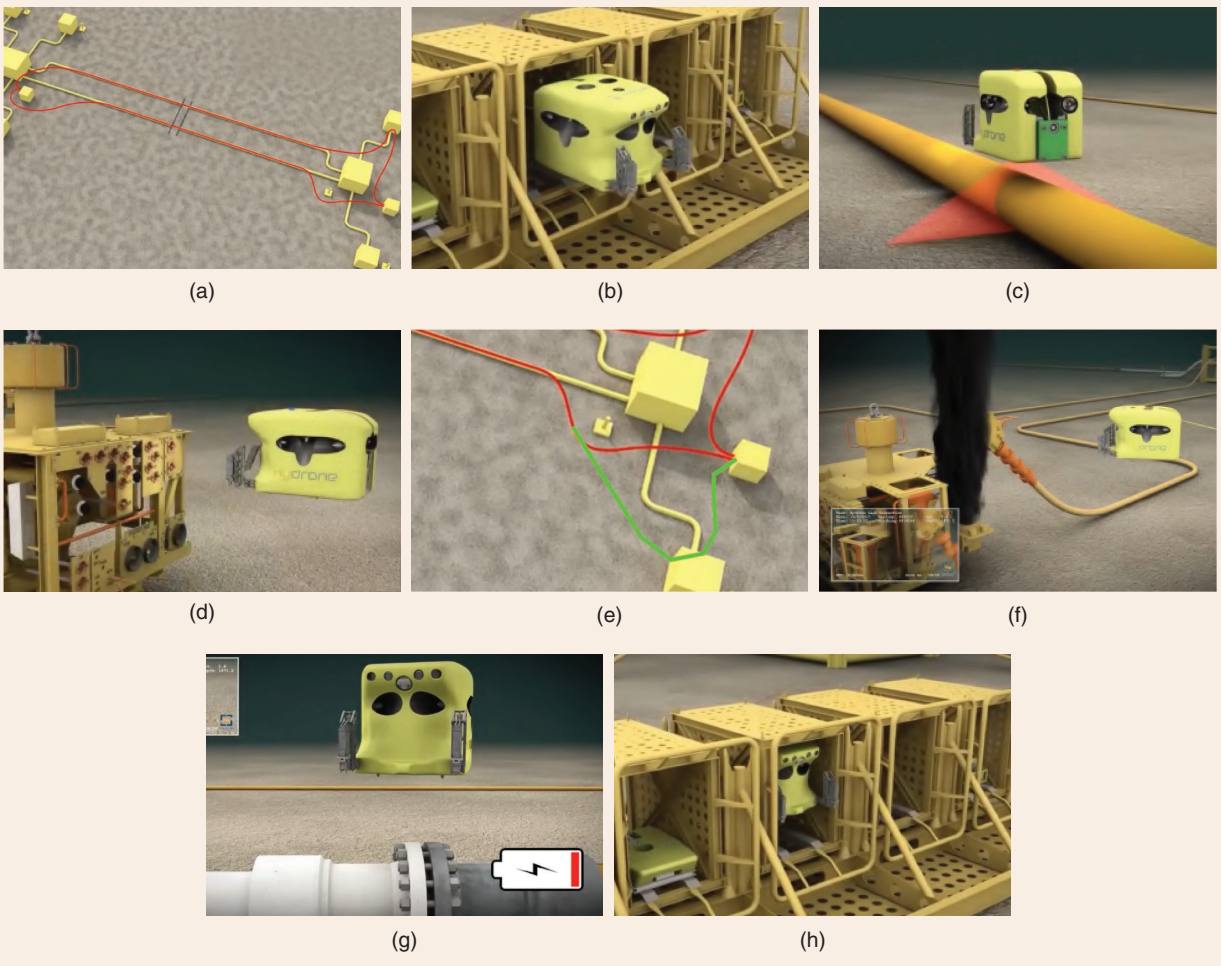

FIGURE 2. An overview of autonomous robotic pipeline inspection: Hydrone-R (a) plans the mission, (b) leaves its dock station, and (c) starts inspecting. (d) While surveying a cluster, it detects an anomaly in another cluster and (e) adapts on the fly its plan (f) to include the new location (g) The robot inspects the pipeline until its battery enforces (h) returning to the recharge station.

and send task updates to the robot during missions. It also works as an ultrashort baseline positioning system.

- ***Advanced underwater vision system.*** Hydrone-R mounts two frontal cameras and one bottom camera used in conjunction with a laser projector to generate 3D point clouds. Lights maximize scene illumination and improve robot visibility. The system enables pipeline tracking and following by recognizing structures and markers and providing position data. It can detect and avoid obstacles and catch anomalies.

- ***Sonars.*** In subsea environments, light propagation is limited, and water particles make vision effective only within a few meters of the robot. Long-range-view sonars become mandatory. Hydrone-R is equipped with a frontal multibeam imaging sonar (an acoustic camera) and a top-mounted mechanical scanning imaging sonar that ensures a 360° field of view to enhance situational awareness around the vehicle.

- ***Advanced navigation system.*** Hydrone-R has an inertial measurement unit sensor that combines optical fiber gyroscopes and solid-state accelerometers aided, through data fusion algorithms, by a Doppler velocity logger, an ultrashort baseline, and the aforementioned vision system. The system limits the navigation error to less than 0.05% of the traveled distance.

- ***Internal computational power and storage resources.*** Computing systems host sufficient power and storage resources for the robot to safely carry out assigned missions.

- ***Manipulators.*** Hydrone-R is equipped with a light manipulator and a grabber. At the time of writing, manipulation operations are performed in a teleoperated fashion.

- ***Integrated Tether Management System.*** To work in ROV mode, a tether and a tether management system provide real-time data exchange and battery-charging power when connected to the host base of the subsea system. The robot can connect to multiple bases, according to mission needs.

- ***Skid integration and vehicle capabilities extension.*** A skid is a set of sensors and electronics that one can attach to the bottom of the vehicle to extend its capabilities. Data and power flows occur through the use of inductive connectors.

- ***Inductive connectors.*** To connect and disconnect tethers and skids, we use an inductive connector offering up to a 100-MB/s data exchange bit rate and 2 kW of electrical power for charging. This contactless connector allows significantly more mating cycles compared to wet-mateable connectors.

> “WE CAN REPLAN IN CASE OF NEW OPPORTUNITIES AND CONTINGENCIES VIA A CLOSE INTERACTION BETWEEN THE UP LIBRARY AND THE ROBOT'S ORCHESTRATOR.”

PROBLEM FORMULATION

To autonomously accomplish pipeline inspection, Hydrone-R must compute the sequence of actions required to achieve the goal while effectively managing the available resources and minimizing makespan. If new opportunities or contingencies occur during execution, e.g., a new cluster needs to be examined or the battery level drops unexpectedly, the robot replans to maximize the final outcome while preserving its safety; e.g., it may decide to return to the dock station (see Figure 2).

We define this problem as an *automated planning* problem, where *automated planning* is a branch of AI that, given a model of a system (the robot and its capabilities) and a goal to reach (inspection of a set of locations), finds a course of actions to drive the system from its current state to the target. When state variables include only rational numerical values, like the battery level, the problem to be solved is a *numeric planning* problem.

DEFINITION 1

A numeric problem Ψ is a tuple $\langle V, I, A, G \rangle$, where the following hold.

- $V = \{f_1, .., f_n\}$ is a finite set of variables (or fluents) $f \in V$, each with a domain $Dom(f)$.
- I is the initial state, which assigns a value $I(f) \in Dom(f)$ to each variable $f \in V$.
- A is a set of actions $a = \langle P, E \rangle \in A$ such that
  - P is a set of preconditions, each being a combination of atoms of the form $f \bowtie v$, with $\bowtie = \{=, <, \leq, >, \geq\}$, $f \in V$, and $v \in Dom(f)$.
  - E is a set of instantaneous effects of the form $f := v$, with $f \in V$ and $v \in Dom(f)$.
- G is the set of goal conditions, each being a combination of atoms of the form $f = v$, with $f \in V$ and $v \in Dom(f)$.

A plan π that solves Ψ is a sequence of actions $\{a_0, \ldots, a_m\}$ that brings the system from I to G by linking the effects of a_i with the preconditions of a_{i+1}.

When including timed goals, time constraints, or durative actions, we enter the field of *temporal planning*.

DEFINITION 2

A temporal planning problem Φ is a tuple $\langle V, T, I, A, G \rangle$, where

- V, I, and G are defined as before.
- T is a set of timed initial literals, each of the form $\langle [t] f := v \rangle$ where $f \in V$, $v \in Dom(f)$, and $t \in \mathbb{R}_{>0}$ is the time at which f will be assigned the value v.
- A is the set of actions $a = \langle [l, u], C, E \rangle$, where $[l, u]$ are the duration bounds, C is the set of conditions, each one

with start and end points, and E is the set of effects, each one becoming true at certain time instants [3].

π that solves Φ is a sequence $\{\langle t,a,d\rangle\}$, where a is paired with a start time instant $t \in \mathbb{R}_{\geq 0}$ and a duration $d \in \mathbb{R}_{>0}$.

We aim to solve both Ψ and Φ in the context of subsea pipeline inspection. Moreover, the robot must autonomously decide which location to inspect based on its significance and available resources. The goal becomes $G = G_h \cup G_s$, where G_h is the set of hard goals that must be achieved in any goal state. G_s, instead, is the set of soft goals $\langle g_s, c(g_s)\rangle$ that the system may achieve: for each $g_s \in G_s$, there exists a cost $c(g_s) \in \mathbb{R}_{>0}$ to be paid if g_s is not achieved by π. The problem becomes a *planning problem with oversubscription* [11]. Finally, we allow the defining of an ordering rule $o = g_i < .. < g_j$ among the goals in G so as to prioritize inspection points. Our goal becomes to find π that complies with o, achieves all of the hard goals, and pays the minimum cost $c(\pi) = \min \sum c(g_s)$.

> “TO AUTONOMOUSLY ACCOMPLISH PIPELINE INSPECTION, HYDRONE-R MUST COMPUTE THE SEQUENCE OF ACTIONS REQUIRED TO ACHIEVE THE GOAL WHILE EFFECTIVELY MANAGING THE AVAILABLE RESOURCES AND MINIMIZING MAKESPAN.”

In this context, a *new opportunity* (e.g., inspection of a new cluster) is a soft goal $\langle g_s, c(g_s)\rangle$ created at runtime by an event $E = \langle [t] f := v\rangle$ (i.e., an action effect or external occurrence), which instantiates a fluent $f \in V$ via $v \in Dom(f)$ [12]. The problem becomes an *opportunistic planning problem with oversubscription* $\langle \Psi, A', G'\rangle$ (or $\langle \Phi, A', G'\rangle$), where $A' \in A$ is the subset of actions that are preemptable and G' is the set of new opportunities. If an opportunity is discovered, the initial state I is updated, and replanning is triggered. Being in a nondeterministic world, contingencies result in the same procedure, and A' collects the set of actions that are preemptible to handle both opportunities and contingencies.

In the following sections, we present the planning library used to solve the problem and the planning architecture implemented to find π and monitor its execution.

THE UP LIBRARY

To provide our robot with autonomous decision-making capabilities, we adopted the UP library (https://github.com/aiplan4eu/unified-planning). The UP library is open source, reusable, and planner agnostic. It allows one to easily model, manipulate, and solve several planning problems, including classical, numerical, and temporal assignments, multiagent tasks, and task and motion planning. The UP library is part of AIPlan4EU (https://www.aiplan4eu-project.eu/), an H2020 project that aims to enhance accessibility, reuse, and integration of planning algorithms and data.

The library's application programming interfaces (APIs) provide users with the capability to model planning problems manually, parsing a formal language (e.g., PDDL [13] or ANML [14]), exploiting the interoperability interfaces with other frameworks (e.g., Tarski [15]), or mixing such approaches. It allows users to guide the plan search via *custom heuristics*, which provide functions to evaluate the goodness of a given state. It supports the specification of *quality metrics* to impose optimization criteria. For example, users can minimize the number of steps in the resulting plans or the action cost. It allows them to define optional goals, each with an associated cost that is paid if the goal is not achieved by the plan (*oversubscription*). Once all specifications have been outlined, the UP library passes them to the connected planning engines via *operation modes*. Each planning engine solves a specific planning problem (e.g., numerical, temporal, etc.), and each operation mode offers an abstract interface to such engines and gives access to some of their functionalities. The simplest operation mode is *OneshotPlanner*, which allows an engine to be used as a one-time plan generator. We also provide ways to *validate* a plan, *simulate* it (with associated resource consumption), or trigger a *replanning* routine.

Such capabilities make the UP library suitable to solve planning problems across diverse domains, like autonomous ocean inspection. In the next section, we detail its usage in this regard.

OUR ADVANCED PLANNING ARCHITECTURE

To grant our robot autonomous deliberation, we designed a new goal-specification grammar. Then, we developed a planning architecture that takes goals as input, retrieves the available resources, plans the mission, and imprints the robot with the commands needed to execute it.

The grammar allows one to represent a set of goals to be reached (see Figure 3). Each `<single_goal>` asks to **reach** an `<expression>` defined as an internal state of the system (e.g., “*switch from ROV to AUV*”) or a condition of the environment in which it operates (e.g., “*valve 1 must be closed*”). Each target can be equipped with a set of resources (**while** `<expression>`) or time (**within** `<interval>`) constraints that must be observed while achieving it. For instance, we can limit the battery usage and makespan. Goals can be structured in sequences (each goal has a unique `<id>`), ordered (`<ord_constrs>` ::= `<id>` **<** `<id>`), and concatenated (`<ord_constrs>` **and** `<ord_constrs>`). This flexibility empowers the incorporation of optional goals (**optionally**) and the definition of a custom priority hierarchy among them (**optionally with priority** `<number>`). For instance, we can ask the robot to “*reach valve1-closed within [0, 30] while battery* $\geq$ *50*”, which means finding a plan to reach *valve1-closed* within 30 min, always maintaining the battery level above 50%. The *optionally* keyword marks desirable goals that are not mandatory: “*optionally reach valve2-photographed*” specifies a desire to take a picture of *valve2*, but if this is not possible, the mission will be satisfied anyway.

Given a set of goals, we implemented an Online (Re)planner and an Orchestrator that cooperate to successfully reach them (see Figure 4). The former models the planning problem via Problem Encoder, finds a plan through the UP library, and sends it to the robot via Solution Decoder (see Algorithm 1). The latter manages the execution, monitors the robot state, and handles unforeseen situations by triggering safety operations or asking for replanning in real time. A description of each component of the Online (Re)planner follows, while the next section describes the ability of the Orchestrator to handle unforeseen events.

```
<opt> ::= <none>
        | optionally
        | optionally with priority <number>
<single_goal> ::= reach <expression>
                | reach <expression> within <interval>
                | reach <expression> while <expression>
                | reach <expression> within <interval> while <expression>
<opt_single_goal> ::= <opt> <single_goal>
<goals_sequence> ::= <opt_single_goal>
                   | <opt_single_goal> ; goals_sequence
<id_goals_sequences> ::= <id> : <opt_single_goal>
                       | <id> : <opt_single_goal> ; <id_goals_sequence>
<ord_constrs> ::= <id> < <id>
                        | <ord_constrs> and <ord_constrs>
<goal_deordering> ::= <id_goals_sequences> ; <ord_constrs>
<goal_spec> ::= <goal_deordering>
              | <goals_sequence>
```

FIGURE 3. Our goal grammar. It allows the users to represent a set of objectives defined as internal states of the system or conditions of the environment. Each goal can be equipped with resource and time constraints. We can apply ordering, optionality, and priority rules among goals. For example, "*reach valve1-closed within [0, 30] while battery ≥ 50*" asks to close *valve1* within 30 min while keeping the battery level above 50%. "*optionally reach valve2-photographed*" specifies a desire to take a picture of *valve2*, but if this task is unfeasible, the mission will be satisfied anyway.

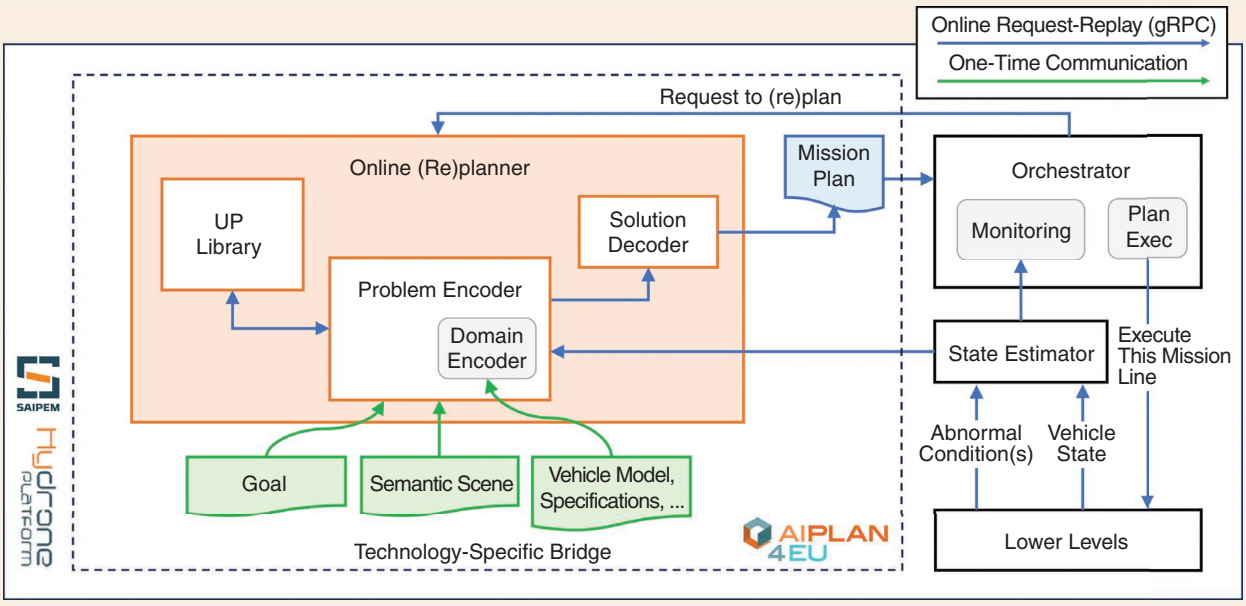

FIGURE 4. Our software architecture. The Online (Re)planner takes as input the robot model, its current state, the scene, and the goal. It combines these data and models the planning problem via Problem Encoder, finds a plan through the UP library, and sends it to the Orchestrator via Solution Decoder. The Orchestrator sends the plan to the lower levels, manages its execution, monitors the robot state via State Estimator, and eventually asks for replanning in real time.

STATE ESTIMATOR

The State Estimator retrieves the current state of the robot, that is, its current pose p, the available resources R (e.g., battery charge), and the goals $\overline{G}$ already visited (line 4). $<p, R>$ forms the initial state I of the system (line 5). Together with $\overline{G}$, they will be updated at each step of the mission.

PROBLEM ENCODER

The Encoder takes as input the initial state I, the current goal G, and the robot specifications, i.e., the logical and temporal expressions representing the set of actions A that the AUV can perform, like moving from a start location s to a goal g. It combines this information with the set of fluents V needed for the problem (e.g., variables representing the resources under consideration) and—eventually—the timed initial literals T. Then, it models the problem Φ (or Ψ) using the UP library API (line 6). It sends Φ to the UP library and waits for a plan π, if one exists (line 7). Once received, the Encoder decorates π with *check* events that aim to verify whether the real value of a resource aligns with the one estimated via the UP library's *Simulator* for the corresponding time instant. This approach ensures the identification of any deviation and promptly triggers replanning in case of unforeseen events. In case of numeric planning, π is an ordered sequence of actions, and we can add checkpoints at the end of each of them. When considering time, instead, actions may overlap: we must first transform π into an ordered list of events (the effects of the actions), wherein the order reflects the time instants at which these events are scheduled. We enrich this list with the checkpoints and transform it back into a plan (lines 9–19). The Encoder sends the decorated plan π' to the Solution Decoder for its translation into a robot-compliant mission ρ. If the Orchestrator triggers a replan request while executing ρ, the Encoder retrieves the new initial state I, the available opportunities G', and the goals $\overline{G}$ already achieved, and it sends these data to the UP library asking for replanning.

UP LIBRARY

When the UP library receives a problem to be solved, it calls *OneshotPlanner* to solve it—with *Oversubscription* in case of optionality between goals (line 7 with *mode* = *OneshotPlanner*). In the case of unforeseen events, instead, the UP library triggers *Replanner*, which updates the initial state of the robot and the goals to be reached according to the current situation and tries to solve the new problem (lines 22–27). In the case of numeric planning, *OneshotPlanner* solves the problem via Expressive Numeric Heuristic Search Planner (ENHSP) [16]. When including time, we use Tamer [3].

SOLUTION DECODER

If π exists, the Decoder translates it into a robot-compliant mission ρ (line 20) and sends it to the Orchestrator (line 21).

ORCHESTRATOR

This block consists of a Plan Executor, executing the mission, and a Monitor, retrieving and monitoring the vehicle state. If contingencies or new opportunities occur and affect either the vehicle state or the mission success, the Orchestrator preempts the execution of the current assignment in real time, sends the useful information to the Problem Encoder, and allows it to online replan (lines 22–32).

Our planning architecture is hosted on board the robot to increase its level of autonomy, guarantee real-time responsiveness, and ensure the proper condition of the robot despite the unpredictable and nondeterministic nature of the deep ocean.

HANDLING CONTINGENCIES AND OPPORTUNITIES

Failures result from accidents, such as instrument faults, and require the immediate stop of the vehicle. If we are not to lose it at sea, we must abandon the mission and

ALGORITHM 1: Online (Re)planner.

```
1  connect StateEstimator
2  connect Orchestrator
3  procedure REPLAN(V, T, A, G, G', mode)
4      p, R, G ← StateEstimator.getCurrentState()
5      I ← < p, R >
6      Φ ← < V, I, T, A, G >
7      π ← Planner.solve(Φ, mode)
8      if π ≠ None then
9          E' ← []
10         E ← getEvents(π)
11         for each e ∈ E do
12             E'.push(e)
13             I ← Simulator.apply(I,e)
14             for each r ∈ R do
15                 v ← I.getValue (r)
16                 E'.push(check(r, v))
17             end for
18         end for
19         π' ← getPlan(E')
20         ρ ← decode(π')
21         out ← Orchestrator.startMission(ρ)
22         if out == replan then
23             if replan == contingency then
24                 return REPLAN(V, T, A, G - G, G', replanner)
25             else if replan == opportunity then
26                 return REPLAN(V, T, A, G - G + G', ∅, replanner)
27             end if
28         else if out == success then
29             return True
30         else
31             return False
32         end if
33     else
34         return False
35     end if
36 end procedure
```

perform procedures that bring the system to the surface. Contingencies and new opportunities, instead, will pause the assignment. Examples include the detection of a battery level that is lower or higher than expected. In both cases, we need to replan to recover from the drop in system performance or to maximize the outcome.

```
{
  "nodes" : [
    {
      "id" : "AUV",
      "kind" : "preemptable",
      "plan_file" : "",
      "emergency" : "EMERGENCY_SURFACING",
    },
    ...

    {
      "id" : "EMERGENCY_SURFACING",
      "kind" : "not_preemptable",
      "executable" : "../emergency_surfacing",
    }
  "transitions" : [
    {
      "id" : "c1",
      "source" : "AUV",
      "command" : "abort_mission",
      "to" : "ROV",
      "preemption_policy" : "abort"
    },
    ...

    {
      "id" : "t1",
      "source" : "AUV",
      "trigger" : "imminent_collision",
      "to" : "GET_OFF_MY_BACK",
      "preemption_policy" : "suspend"
    }
}
```

FIGURE 5. Examples of nodes and transitions of the Orchestrator's FSM.

The Orchestrator addresses these situations by exploiting a finite-state machine (FSM) where each node is a mode of the robot. For instance, the *AUV* mode allows the vehicle to autonomously plan and execute a mission, while the *ROV* mode enables a human user to teleoperate the robot. *standby* enables the system to remain ready for further instructions, while *all_stop* brings the robot to a complete halt. A node can be preemptable or nonpreemptable. A preemptable node can be interrupted by an observation or a command leading to another node. A nonpreemptable node is a stand-alone external program that is invoked and takes control of the entire system. The Orchestrator becomes insensible to any stimulus and waits for the termination of the external program to resume operation according to the transitions being specified. As an example, Figure 5 shows the *AUV* and *emergency_surfacing* nodes: the *AUV* node is preemptable, while *emergency_surfacing* is nonpreemptable and executes a precomputed routine that safely ascends the robot to the surface.

The edges of the FSM are the controls that allow the robot to change modality. We have three types of edges: *command*, *trigger*, and *replan*. The first is an external command received from a human operator or from the Online (Re)planner. For example, *abort_mission* aborts any mission running while in *AUV* mode and brings the robot to the *ROV* modality (see Figure 5). *trigger* applies the transitions triggered by conditions that become true from the current node. For instance, *imminent_collision* suspends the autonomous modality of the robot and brings it to a safe location (see Figure 5). Once the transition is completed, the robot becomes autonomous again. Finally, *replan* is an edge that cycles on the *AUV* mode. It sends a replan request to the Online (Re) planner and makes the robot wait for a new plan. If received, *replan* keeps the robot in the *AUV* modality but suspends the ongoing activities and starts executing the new plan from the current state of the robot.

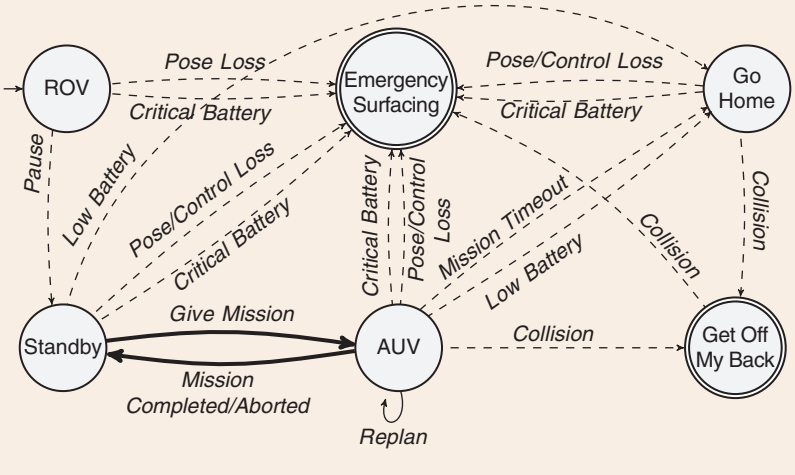

FIGURE 6. One segment of the FSM used by the Orchestrator. Nodes are preemptable (single circles) or nonpreemptable (double circles). Edges are commands (bold lines), triggers (dashed lines), or replan requests (solid lines).

Figure 6 shows part of the resulting FSM. The robot initially operates in *ROV* mode, entering *standby* until receiving a plan execution request from the Online (Re)planner through the Orchestrator, transitioning the FSM to *AUV* mode to begin execution. Plans may include *check* actions that require feedback from the State Estimator or replanning. In all cases, the FSM remains in the *AUV* mode until mission completion or abortion. During low-battery scenarios, initial replanning occurs to maintain the AUV mode. If it fails, the robot could shut down in an unreachable area. Then, we hold two threshold values, α for low battery and β for critical

battery, that trigger mode changes. α prompts navigation to the charging station (*go home*), while β initiates *emergency surfacing* to the water's surface, crucial also for avoiding system loss in cases of localization or control loss. In the case of imminent collision, instead, we suddenly suspend the AUV mode and switch to the nonpreemptable *get off my back* mode. If, even in this case, collision remains imminent, we enter the *emergency surfacing* state. Other nodes and edges exist, but for brevity, we showed the most interesting one concerning our use case.

EXPERIMENTAL EXPERIENCE

We are engaged in a comprehensive and ongoing testing program, proving our hardware and software contributions.

The hardware standpoint sees Hydrone-R operating on a real field in Norway for more than 200 days (since June 2023) uninterruptedly at a depth of 325 m. The goal is to achieve 10 years of service as a subsea resident, that is, a permanent subsea presence except for infrequent scheduled maintenance periods spread out over time. At present, the robot has started inspection activities and is being closely monitored to ensure its optimal performance and condition. Figure 7 shows the vehicle being deployed in the waters in June 2023, while Figure 8 depicts the robot at work. As proof of Saipem's efforts in developing resident robots for subsea pipe inspection, the FlatFish robot is also being tested. It is currently operating at a depth of approximately 1,700 m off the Brazilian coast. Figure 9 depicts FlatFish executing its autonomous docking routine (see the video available at https://www.youtube.com/watch?v=3ni6kLWgvAk), while Figure 10 shows FlatFish inspecting a pipeline.

Saipem offers also a simulated environment based on the Gazebo simulator [17]. It consists of a Hydrone-R robot (faithful to the real one), pipelines, and clusters to be inspected. In this context, we equipped the robot with the capability of performing an action $a = move(s, g)$ that brings the system from s to g. To be executed, the robot must be at s with a battery level at least equal to that required to complete the action itself according to the robot consumption model. As an effect, *move* brings the robot to t, marking this location as visited, and consumes the necessary battery. Then, we asked the robot to visit a set of targets. In the case of numeric planning, *move* is instantaneous, and, among its effects, it increases the plan duration d by the elapsed time computed according to the consumption model of the robot. In the case of temporal planning, instead, *move* is a durative action with fixed duration d. In both cases, we tested the capability of our planning module to automatically generate valid plans, optimize resources, accept mandatory and optional goals, respect ordering rules, and replan. Details about the performed tests and

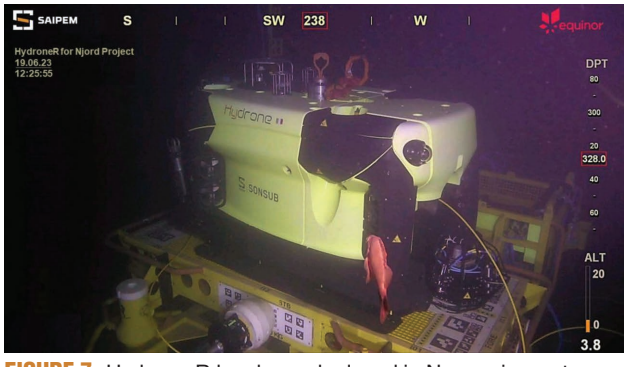

FIGURE 7. Hydrone-R has been deployed in Norwegian waters since June 2023.

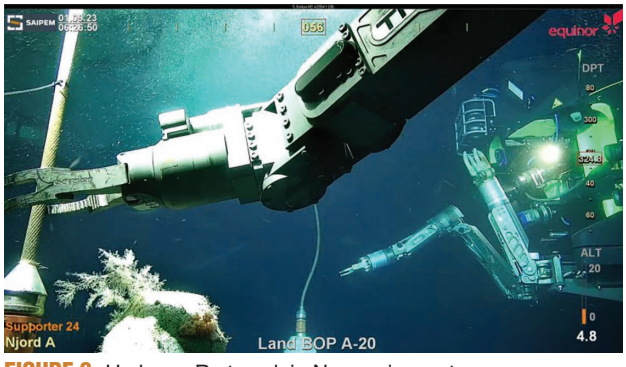

FIGURE 8. Hydrone-R at work in Norwegian waters.

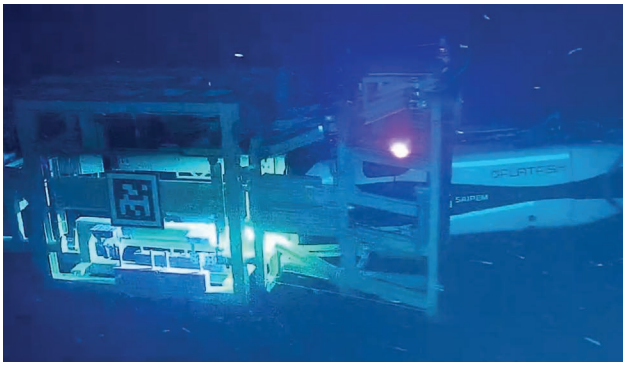

FIGURE 9. FlatFish autonomous docking.

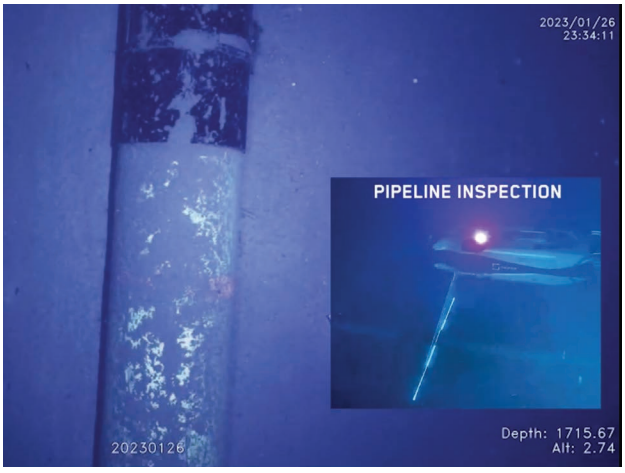

FIGURE 10. FlatFish inspecting a pipeline in the Brazilian coastal area.

obtained results follow, subdivided according to the features we want to prove. Table 1 reports the planning time t (s), consumed battery b (%), number n of targets reached, duration d of the plan (s), and cost c of the plan (see the section "Problem Formulation") both in the case of numeric (left side of each result) and temporal planning (right side). The UP library automatically selects ENHSP in the case of numeric planning and Tamer in the case of temporal planning.

> "WE TESTED THE CAPABILITY OF OUR PLANNING MODULE TO AUTOMATICALLY GENERATE VALID PLANS, OPTIMIZE RESOURCES, ACCEPT MANDATORY AND OPTIONAL GOALS, RESPECT ORDERING RULES, AND REPLAN."

AUTOMATIC PLAN GENERATION AND VALIDATION

The robot has to visit $N = 6$ targets randomly sampled in a cube of side $\sqrt[3]{A} = 100$ m around the initial pose of the robot. The robot has enough battery to visit them all. We do not apply any additional optimization metrics or constraints. All goals are mandatory. The result is one sequence of *move* actions that enables the robot to visit all of the locations in an order that differs depending on whether the problem is numerical or temporal—see battery consumption (92% versus 89%) and plan duration (279.9 s versus 274.2 s) of Test 0 of Table 1.

PLAN AND OPTIMIZE RESOURCES

We repeat the same test while optimizing the battery usage. The UP library still does not provide an optimal temporal planner; thus, results are available only for the numeric use case. ENHSP outputs a plan that consumes 86% of the battery instead of 92% (Test 1).

ACCEPT MANDATORY AND OPTIONAL GOALS

We ask the robot to visit $N = 8$ random targets, each with an associated cost: $\{t_0{:}\,7,\ t_1{:}\,6,\ t_2{:}\,3,\ t_3{:}\,1,\ t_4{:}\,5,\ t_5{:}\,8,\ t_6{:}\,2,\ t_7{:}\,4\}$—for a total cost of 36 if no target is reached. Given the robot's consumption models, it cannot visit all assigned locations, but such points are optional, which means the robot will pay the cost of missed goals. We performed the test with no constraint, a "*while battery > 10%*" constraint for each goal, and a "*within 180 seconds*" constraint for each goal. If no constraint is applied, ENHSP and Tamer output different plans (see b and d of Test 2), but they both discard t_3 (with $c(t_3) = 1$) so as to pay the lowest possible price. When applying the numerical constraint, instead, they discard t_6. In this case, they have to find a good balance between the cost to pay and the fact that the battery should be greater than 10% at each step. Indeed, they visited all of the targets, saving about 10% of the battery compared with the unconstrained test (Test 3). When we ask to reach each goal within 180 s, both engines compute a path that passes through four out of eight goals (Test 4). In both cases, the plan duration is almost equal to our upper time (175.7 s in the case of numeric planning and 177.9 s in the case of temporal planning), and the cost is the minimum payable when discarding four goals from the weighted list ($c = 1 + 2 + 3 + 4$).

ACCEPT ORDERING RULES

The robot should visit an ordered set of $N = 6$ mandatory targets $[t_0 < t_4,\ t_1 < t_5,\ t_2 < t_3,\ t_3 < t_5,\ t_4 < t_5]$ ($N = 4$ with $[t_0 < t_3,\ t_1 < t_2,\ t_2 < t_3]$ when we apply the time constraint). ENHSP and Tamer find a plan whether or not numerical and/or temporal constraints are applied (Tests 5–7). For example, if no constraint is applied, ENHSP outputs $(s \to t_1 \to t_2 \to t_0 \to t_4 \to t_3 \to t_5)$, which consumes 84% of the battery, while Tamer outputs $(s \to t_1 \to t_0 \to t_4 \to t_2 \to t_3 \to t_5)$, which consumes 87% of the battery. Plans are valid as they meet the constraints imposed.

TABLE 1. Performance of our planning module.

TEST	OPTIMAL	OPTIONAL	ORDERED	$\sqrt[3]{A}$	N	t (s)	b (%)	n ($\leq N$)	d (s)	c
0	–	–	–	100	6	0.68 | 0.16	92 | 89	6 | 6	279.9 | 274.2	–
1	[]	–		100	6	0.74 | –	86 | –	6 | –	268.46 | –	–
2	–	[]	–	100	8	10.27 | 767.52	99 | 97	7 | 7	292.1 | 287.96	1 | 1 (out of 36)
3	–	[while]	–	100	8	2.96 | 221.22	86 | 86	7 | 7	266.7 | 266.7	2 | 2 (out of 36)
4	–	[within]	–	100	8	2.5 | 186.84	55 | 57	4 | 4	175.7 | 177.9	10 | 10 (out of 36)
5	–	–	[]	100	6	0.72 | 0.21	84 | 87	6 | 6	265.2 | 269.51	–
6	–	–	[while]	100	6	0.76 | 51.49	89 | 89	6 | 6	274.3 | 274.56	–
7	–	–	[within]	100	4	0.49 | 33.19	54 | 54	4 | 4	174.1 | 174.3	–

Hydrone-R must visit N targets randomly sampled on a cube of side $\sqrt[3]{A}$. We ask for *optimal* plans and plans including *optional* or *ordered* targets. We impose no constraints ([]), numeric constraints ([*while battery > 10%*]), and temporal constraints ([*within 180 s*]). Results show the obtained planning time t (s), consumed battery b (%), number n of targets reached, duration d of the plan (s), and its cost c in the case of numeric (left) and temporal (right) planning.

REPLAN

We ask the robot to reach t_0 while keeping in memory an ordered set of opportunities $G' = \{t_1, t_2, t_3\}$. We decorate the plan with *check* actions verifying the battery level is consistent with the simulated one. If greater than expected or enough to keep the robot operating, we add the first target of G' not yet inspected. Once t_0 is reached, *check* estimates that the battery level should be at least equal to 93.63%. It is currently 99.34%: we add t_1. The procedure continues until all targets in G' are visited as the robot's consumption model permits it. Without replanning, the robot would have visited only t_0, leaving G' for future missions.

The results are promising. Before exploiting the UP library, plans were manually calculated. For medium and complicated missions, the total time to write a valid, non-optimized task was around three to 10 times its duration (for writing, simulation, and correction), which exceeds the average planning time we obtained in our tests. The error rate was linear with the complexity and length of the mission. The human could not formulate optimal plans, and the ability to assign opportunities and manage abnormal conditions was limited to a restricted number of conditions decided a priori with the customer. The goal was to handle those faults that make the robot inoperative via a priori-defined rescue operations. Now, contingencies can be arbitrarily defined and managed in real time via replanning.

CONCLUSIONS

In this article, we presented Hydrone-R, an AUV designed for autonomous deep-ocean inspection that can optimize available resources and minimize makespan. Moreover, it adapts to unforeseen situations and capitalizes on new opportunities. The planning module exploits the advanced UP library, which enhances decision making and enables efficient (re)planning during underwater inspections. The library is open source and was developed as part of the AIPlan4EU project. We described the hardware and software architecture of our proposal and discussed its practical applications in the offshore energy industry. We believe that the proposed solution represents a step forward in the state of the art of underwater robots, enabling *true* autonomy and ocean *residence*.

ACKNOWLEDGMENT

This work has been partly supported by the AIPlan4EU project funded by the EU Horizon 2020 research and innovation programme under GA 101016442, the AI@TN project funded by the Autonomous Province of Trento, and by the Interconnected Nord-Est Innovation Ecosystem funded by the European Union Next-GenerationEU (Piano Nazionale di Ripresa e Resilienza – missione 4 componente 2, investimento 1.5 – D.D. 1058 23/06/2022, ECS00000043).

AUTHORS

Elisa Tosello, Fondazione Bruno Kessler, 38123 Trento, Italy. E-mail: etosello@fbk.eu.

Paolo Bonel, Saipem s.p.a. - Sonsub Robotics, 35129 Venice, Italy. E-mail: paolo.bonel@saipem.com.

Alberto Buranello, Saipem s.p.a. - Sonsub Robotics, 35129 Venice, Italy. E-mail: alberto.buranello@saipem.com.

Marco Carraro, Saipem s.p.a. - Sonsub Robotics, 35129 Venice, Italy. E-mail: marco.carraro@saipem.com.

Alessandro Cimatti, Fondazione Bruno Kessler, 38123 Trento, Italy. E-mail: cimatti@fbk.eu.

Lorenzo Granelli, Saipem s.p.a. - Sonsub Robotics, 35129 Venice, Italy. E-mail: lorenzo.granelli@saipem.com.

Stefan Panjkovic, Fondazione Bruno Kessler, 38123 Trento, Italy. E-mail: spanjkovic@fbk.eu.

Andrea Micheli, Fondazione Bruno Kessler, 38123 Trento, Italy. E-mail: amicheli@fbk.eu.

REFERENCES

[1] A. G. Rumson, "The application of fully unmanned robotic systems for inspection of subsea pipelines," *Ocean Eng.*, vol. 235, Sep. 2021, Art. no. 109214, doi: 10.1016/j.oceaneng.2021.109214.

[2] F. Ingrand and M. Ghallab, "Deliberation for autonomous robots: A survey," *Artif. Intell.*, vol. 247, pp. 10–44, Jun. 2017, doi: 10.1016/j.artint.2014.11.003.

[3] A. Valentini, A. Micheli, and A. Cimatti, "Temporal planning with intermediate conditions and effects," *Proc. AAAI Conf. Artif. Intell.,* vol. 34, no. 6, pp. 9975–9982, 2020, doi: 10.1609/aaai.v34i06.6553.

[4] K. B. Ånonsen, O. K. Hagen, Ø. Hegrenæs, and P. E. Hagen, "The HUGIN AUV terrain navigation module," in *Proc. OCEANS-San Diego*, Piscataway, NJ, USA: IEEE Press, 2013, pp. 1–8.

[5] B. Johansson, J. Siesjö, and M. Furuholmen, "Seaeye Sabertooth, a hybrid AUV/ROV offshore system," in *presented at the SPE Offshore Eur. Conf. Exhib.*, 2011, Paper SPE-146121-MS.

[6] D. Ribas, N. Palomeras, P. Ridao, M. Carreras, and A. Mallios, "Girona 500 AUV: From survey to intervention," *IEEE/ASME Trans. Mechatronics*, vol. 17, no. 1, pp. 46–53, Feb. 2012, doi: 10.1109/TMECH.2011.2174065.

[7] J. Albiez, S. Joyeux, and M. Hildebrandt, "Adaptive AUV mission management in under-informed situations," in *Proc. OCEANS MTS/IEEE SEATTLE*, 2010, pp. 1–10, doi: 10.1109/OCEANS.2010.5664350.

[8] S. MahmoudZadeh, D. M. W. Powers, K. Sammut, A. Atyabi, and A. Yazdani, "A hierarchal planning framework for AUV mission management in a spatiotemporal varying ocean," *Comput. Elect. Eng.*, vol. 67, pp. 741–760, Apr. 2018, doi: 10.1016/j.compeleceng.2017.12.035.

[9] M. Cashmore, M. Fox, D. Long, D. Magazzeni, and B. Ridder, "Opportunistic planning in autonomous underwater missions," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 2, pp. 519–530, Apr. 2018, doi: 10.1109/TASE.2016.2636662.

[10] Z. Yan, W. Liu, W. Xing, and E. Herrera-Viedma, "A multi-objective mission planning method for AUV target search," *J. Mar. Sci. Eng.*, vol. 11, no. 1, 2023, Art. no. 144, doi: 10.3390/jmse11010144.

[11] D. E. Smith, "Choosing objectives in over-subscription planning," in *Proc. Int. Conf. Automated Planning Scheduling*, 2004, pp. 393–401.

[12] D. M. Gaines, T. Estlin, F. Fisher, C. Chouinard, R. Castano, and R. C. Anderson, *Planning for Rover Opportunistic Science*. Pasadena, CA, USA: Jet Propulsion Laboratory, National Aeronautics and Space, 2004.

[13] M. Fox and D. Long, "PDDL2.1: An extension to PDDL for expressing temporal planning domains," *J. Artif. Intell. Res.*, vol. 20, pp. 61–124, Dec. 2003, doi: 10.1613/jair.1129.

[14] D. E. Smith, J. Frank, and W. Cushing, "The ANML language," in *Proc. ICAPS Workshop Knowl. Eng. Planning Scheduling (KEPS)*, 2008, vol. 31.

[15] G. Francés and M. Ramirez. "Tarski: An AI planning modeling framework." GitHub. Accessed: Jan. 18, 2023. [Online]. Available: https://github.com/aig-upf/tarski

[16] E. Scala, P. Haslum, and S. Thiébaux, "Heuristics for numeric planning via subgoaling," in *Proc. 25th Int. Joint Conf. Artif. Intell.*, 2016, pp. 3228–3234.

[17] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2004, vol. 3, pp. 2149–2154, doi: 10.1109/IROS.2004.1389727.