Belief-based fault recovery for marine robotics

Jeremy Paul Coffelt[1], Mahya Mohammadi Kashani[2], Andrzej Wąsowski[2] and Peter Kampmann[1]

[1]*ROSEN Technology and Research Center GmbH*
[2]*IT University of Copenhagen*

Abstract
We propose a framework expanding the capabilities of underwater robots to autonomously recover from anomalous situations. The framework is built around a knowledge model developed in three stages. First, we create a deterministic knowledge base to describe the "health" of hardware, software, and environment components involved in a mission. Next, we describe the same components probabilistically, defining probabilities of failures, faults, and fixes. Finally, we combine the deterministic and probabilistic knowledge into a minimal ROS package designed to detect failures, isolate the underlying faults, propose fixes for the faults, and determine which is the most likely to help. We motivate the solution with a camera fault scenario and demonstrate it with a thruster failure on a real AUV and a simulated ROV.

Keywords
ontology-based autonomy, probabilistic logic programming, fault detection and recovery, marine robotics

1. Introduction

Unmanned underwater vehicles (UUVs) are used for tasks that are impossible or too dangerous for humans: bathymetric and ecological surveys, offshore infrastructure inspection and maintenance, or clearing underwater minefields. The required autonomy and reliability for these missions grows along with their range and duration. When things go wrong, these robots cannot ask a human for help like service robots or safely stop in place like ground robots.

Remotely operated vehicles (ROVs) are UUVs that are designed to be neutrally buoyant and remain tethered to a surface vessel. Despite this, notable ROVs, such as the Kaikio, have have been lost at the bottom of the ocean [1]. Autonomous underwater vehicles (AUVs) are UUVs that operate with complete autonomy, untethered to any human operator or vessel, which puts them at an even greater risk of being lost. Although designed to be positively buoyant and float to the surface when things go wrong, AUVs can drift far away from their last known location, be struck by a passing vessel, or lost forever under an ice shelf [2, 3]. A recovery is time consuming and expensive, as a specialized ship and crew must be chartered to search for and load the robot. Several noteworthy AUVs remain lost [4].

Because of these risks and costs, it is especially important that UUV missions are not aborted unnecessarily. To this end, we propose a solution enabling compromised marine robots to autonomously recover and successfully complete their missions. Our main contributions include:

The Eighth Joint Ontology Workshops (JOWO'22), August 15-19, 2022, Jönköping University, Sweden
jcoffelt@rosen-group.com (J. P. Coffelt); mahmo@itu.dk (M. Mohammadi Kashani); wasowski@itu.dk (A. Wąsowski); pkampmann@rosen-group.com (P. Kampmann)

- *An ontology deterministically describing the "health" of hardware, software, and environment components involved in an underwater mission.* The semantics and syntax of the ontology allow information to be shared between the systems on the vehicle, the engineers that develop it, and the domain experts that guide the knowledge base.
- *A probabilistic extension of the ontology to describe the same components in terms of likelihoods of failures, faults, and fixes.* This guides which fixes are available and whether they are likely to resolve the problem or lead to new faults.
- *A knowledge service combining the deterministic and probabilistic capabilities above.* In order to simplify and accelerate development time, our solution relies only on existing, open-source resources, such as OWL, Protégé, Prolog, and ROS.
- *A demonstration of feasibility with experiments* involving a catastrophic thruster failure during a real mission and various failures during a simulated shipwreck survey.

2. Deterministic knowledge

Before proceeding, we define two key terms. Following Laprie [5] (as extended in [6]) and ISO standards [7], we consider a *failure* to be a deviation from some desired behavior and a *fault* to be the defect or anomaly that caused the failure. For instance, blurry images could be a failure due to numerous faults, including a damaged camera sensor.

Now, consider an AUV several days and hundreds of kilometers into a week-long port-to-port pipeline inspection. The AUV relies on a camera to avoid obstacles, track the pipeline, and document findings. How should the AUV behave if it suddenly experiences a camera fault? What knowledge and reasoning are required to turn a failed or aborted mission into a successful one? We investigate these issues through the following competency questions.

Q1: How is a fault detected? To detect a camera failure, the ontology must permit clear definitions of metrics for nominal camera operation so that a "health monitor" can process all camera-related data and determine whether current readings are within expected limits. For instance, metrics like *power draw* and *image brightness* can assess the health of a camera.

Q2: What faults can cause a failure? Unusual power draw can be caused by a camera hardware fault, a loose connector, or a short circuit in a cable. Dark images could be caused by a faulty sensor, or by environmental disturbances ("faults") such as high turbidity. Consequently, our ontology must capture all failures for each potential fault. For the AUV example, let $\texttt{CamFault} = \{\texttt{cam\_hardware\_fault}, \texttt{cam\_driver\_fault}, \ldots\}$ be the set of all camera faults, and $\texttt{CamFail} = \{\texttt{power\_fail, raw\_data\_fail, processed\_data\_fail}, \ldots\}$ be all camera-related failures. Then for a failure $\texttt{fail} \in \texttt{CamFail}$, the model constrains the possible faults to:

$$\texttt{fault} \in \texttt{CamFault} \wedge \texttt{hasFault}(\texttt{cam0}, \texttt{fault}) \wedge \texttt{resultsIn}(\texttt{fault}, \texttt{fail})$$

Q3: What failures could result from a fault? If the camera sensor is damaged (a fault), multiple failures are expected to occur—power draws unusually high or low, raw camera data with unexpected distributions, and suspiciously dark or noisy processed videos. The ontology must fully describe the many-to-many relationships between faults and failures. In our example, if a $\texttt{fault} \in \texttt{CamFault}$ occurs, it produces failures satisfying the following constraint:

$$\texttt{fail} \in \texttt{CamFail} \wedge \texttt{hasFailure}(\texttt{cam0}, \texttt{fail}) \wedge \texttt{resultsIn}(\texttt{fault}, \texttt{fail})$$

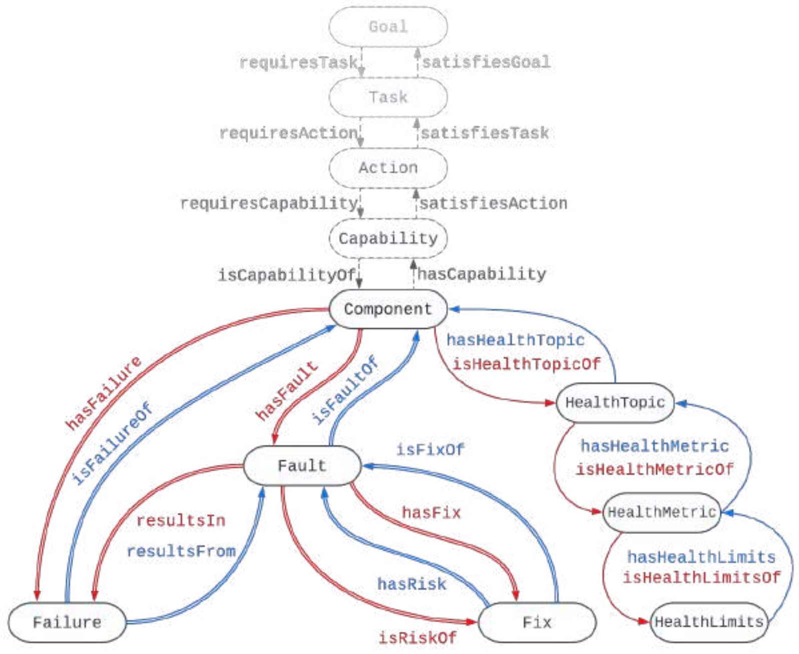


Figure 1: An overview of the REMARO ontology relating failures, faults, fixes, components, and health metrics. The single and double lines represent deterministic and probabilistic relationships, respectively.

```
% individual sensor capabilities
hasCapability(mono_camera,     2D_vision_data).
hasCapability(stereo_camera,   3D_vision_data).
hasCapability(side_scan_sonar, 2D_sonar_data).
hasCapability(mbes_sonar,      3D_sonar_data).
```

Figure 2: Deterministic knowledge relating components and capabilities.

```
% ROS topics providing health metrics for camera "cam0"
hasHealthTopic(cam0, "remaro_auv/cam0/power_stats").
hasHealthTopic(cam0, "remaro_auv/cam0/processed_data/img_stats").
% Specific fields in the ROS messages that correspond to health metrics
hasHealthMetric("remaro_auv/cam0/power_stats", cam0_current_draw).
hasHealthMetric("remaro_auv/cam0/processed_data/img_stats", cam0_img_brightness).
hasHealthMetric("remaro_auv/cam0/processed_data/img_stats", cam0_img_noise_ratio).
% Acceptable limits on each health metric
hasHealthLimits(cam0_current_draw,    [0.0, 0.2]). % amps
hasHealthLimits(cam0_img_brightness,  [0.1, 0.9]). % 0=black --> 1=white
hasHealthLimits(cam0_img_noise_ratio, [5.0, 1e6]). % 40=excellent
```

Figure 3: Deterministic knowledge modeling health metrics for an AUV camera.

Q4: Are any fixes available for a fault? Some faults, like a damaged camera sensor, are unrecoverable during mission execution, others like a buggy camera driver could have one or more fixes. The ontology should support mapping faults to potential fixes, which results in constraints and queries similar to those relating failures and faults.

```
% Potential faults common to any camera
hasFault(Comp, camera_hardware_fault)  : 0.01 :- isCamera(Comp).
hasFault(Comp, camera_firmware_fault)  : 0.05 :- isCamera(Comp).
hasFault(Comp, camera_driver_fault)    : 0.10 :- isCamera(Comp).
hasFault(Comp, camera_processor_fault) : 0.25 :- isCamera(Comp).
hasFault(Comp, camera_power_fault)     : 0.05 :- isCamera(Comp).
hasFault(Comp, camera_comm_fault)      : 0.10 :- isCamera(Comp).
% Possible camera failures
resultsIn(camera_hardware_fault, camera_current_draw_fail)    : 0.50.
resultsIn(camera_hardware_fault, camera_img_brightness_fail)  : 0.85.
resultsIn(camera_hardware_fault, camera_img_noise_ratio_fail) : 0.70.
resultsIn(camera_firmware_fault, camera_current_draw_fail)    : 0.05.
% Potential fixes for various faults
hasFix(camera_hardware_fault, camera_cycle_power_fix)    : 0.10.
hasFix(camera_firmware_fault, camera_cycle_power_fix)    : 0.30.
hasFix(camera_driver_fault,   camera_cycle_power_fix)    : 0.30.
hasFix(camera_driver_fault,   camera_restart_driver_fix) : 0.50.
% Note the dependence on whether the fault already exists
hasRisk(camera_cycle_power_fix, camera_hardware_fault) : 0.99
               :- isCamera(Comp), hasFault(Comp, camera_hardware_fault).
hasRisk(camera_cycle_power_fix, camera_hardware_fault) : 0.05
               :- isCamera(Comp), \+hasFault(Comp, camera_hardware_fault).
```

Figure 4: Probabilistic knowledge of failures, faults, and fixes for an AUV camera.

Q5: Are there any potential risks with the proposed fixes? Some fixes come with no risk. For instance, if an image processing service has crashed, restarting it can do no further damage. Other fixes come with substantial risk. As an example, consider a central communication service that has become unresponsive due to a large message queue. If left alone, the service might recover and resume normal operation. However, attempting to kill and reinitialize the service could have disastrous consequences as obstacle avoidance processes lose critical sensor data. For these reasons, it is important the ontology includes not only a list of fixes for each fault, but also a list of risks (additional faults) associated with each fix.

Motivated by the competency questions above, we propose the **RE**liable **MA**rine **RO**botics (REMARO) ontology outlined in Fig. 1. We have elaborated it and translated into OWL syntax [8] using the Protégé editor [9]. The resulting OWL file, which is available in the REMARO Project repository,[1] is used to define the basic classes and relationships in the knowledge base.

The deterministic capabilities of the ontology are ideally suited for storing mission-agnostic knowledge, such as relationships between goals, tasks, capabilities, and components. As an example, consider the Prolog snippet in Fig. 2, which relates components and capabilities. Another Prolog file includes the health information specific to a particular vehicle configuration. The snippet in Fig. 3 models the reliability of an AUV camera. Among other things, Fig. 3 includes health data directly and indirectly measuring camera performance. The availability of such data is a limiting factor in determining which failures (and, therefore, faults) are detectable.

[1]https://github.com/remaro-network/remaro_ontologies

3. Probabilistic knowledge

Suppose the AUV has an image quality failure. As previously discussed, this could be due to numerous faults, such as a broken sensor (hardware), a crashed processing script (software), or turbid water (environment). In general, a dead camera is less likely than a software bug. And, depending on the environment, murky water could be near impossible or almost certain. For these reasons, it is valuable to consider not only faults and failures, but also their likelihoods.

In addition, some recovery "fixes" might resolve the original issue but produce new faults as unintended side effects. Again, probabilities could be attached to both the proposed fixes and their potential risks. If a fix is likely to restore a key capability and has low probability of detrimental side effects, it should likely be executed. However, if the fix involves cycling power to some mission-critical system/section of the vehicle (compromising the safety of the entire vehicle if power cannot be restored), then it should be avoided. With probabilities, these pros and cons can be weighed to guide and explain efforts towards capability restoration and mission adaptation.

As shown in Fig. 2 and 3, we represent deterministic knowledge in Prolog. To add probabilistic knowledge while remaining in Prolog, we use the open-source `cplint`[2] framework. This allows knowledge to be interpreted as a probability distribution over deterministic logic programs. `cplint` includes the PITA and MCINTYRE modules for exact and approximate causal inference, respectively. With them, queries to the knowledge base can be calculated by considering a joint distribution over possible words [10].

These probabilistic capabilities are ideal for describing failures, faults, and fixes specific to the vehicle and environment in the current mission. For instance, suppose priors such as

$$\Pr(\mathtt{fault}_i) = p_i \quad \text{and} \quad \Pr(\mathtt{fail}_j \mid \mathtt{fault}_i) = q_{j,i} \tag{1}$$

are known for all $\mathtt{fault}_i \in \mathtt{Faults}$ and $\mathtt{fail}_j \in \mathtt{Fails}$. Then `cplint` provides a reasoning engine for the Bayes' Theorem calculations required to compute posteriors of the form

$$\Pr(\mathtt{fault}_i \mid \mathtt{fail}_1, \ldots, \mathtt{fail}_n), \tag{2}$$

which can be used to determine the fault most likely to cause the detected failure(s). Fig. 4 lists a few such probabilistic facts for an AUV camera. To clarify the notation in Fig. 4, note that

$$\Pr(\text{coin shows heads} \mid \text{coin is tossed} \ \wedge \ \overline{\text{coin is fair}}) = 0.6$$

would be translated as:

```
showsHeads(coin) : 0.6 :- isTossed(coin), \+isFair(coin).
```

Note that faults, such as those given in Fig. 4, are carefully defined in broad terms to cover all internal, external, and interface aspects of each component. Also, unless stated otherwise, probabilities are assumed independent of each other and based on some consistent time interval (e.g., per mission, per hour of mission, per device lifetime).

As much of the power in the proposed solution lies in its probabilistic capabilities, it is worth discussing potential sources of priors like those shown in Equations 1-2 and Fig. 4. We believe the ideal approach involves a combination of the following:

[2] https://github.com/friguzzi/cplint

- *Device manufacturers*: Data sheets and manufacturer engineers/technicians are perhaps the simplest source of information regarding component-related faults. Manufacturers can also provide details about power, communication, and environmental limitations that are otherwise difficult to determine without tedious and expensive experimentation.
- *Past missions*: If available, results from past missions can provide improved estimates for a particular vehicle configuration or mission environment. For instance, suppose data sheets give reliability estimates that are contradicted by the observed frequency of faults for similar vehicles and environments. Then, these observed probabilities should be prioritized over the theoretical probabilities provided by the manufacturer.
- *Simulated missions*: When a new vehicle configuration or mission environment is being considered, the cheapest and quickest source of information is usually simulation. Unfortunately, some faults and failures are difficult or impossible to simulate.
- *Domain experts*: With their unique knowledge and experience, domain experts can combine all of the above into improved estimates. Unfortunately, domain experts can be difficult to find, expensive to hire, and still susceptible to human error.

4. Health monitor implementation

As shown in Fig. 5, the proposed health monitor consists of a publisher/subscriber node connected to a knowledge base. Before detailing the implementation, we offer the following justifications for basing our solution in ROS: (1) it is the de facto standard for robotics middleware, (2) real-world data is available in ROS "bags", (3) a ROS-based UUV simulator is available for experimentation, and (4) a popular ROS-based knowledge base already exists.

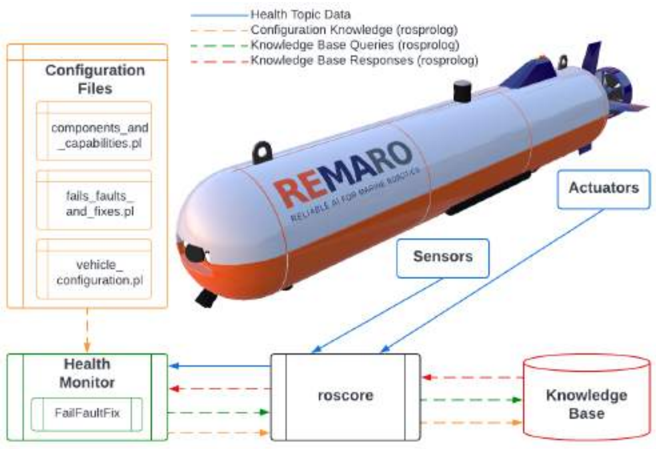
4.1. Knowledge base

The knowledge base provides a means of storing existing knowledge and reasoning to generate new knowledge. A priori knowledge like that shown in Fig. 2-4 is mostly static and reusable between missions. For instance, a team will likely have an inventory of actuators and sensors that it shares across multiple robots. Because the capabilities of these components are deterministic and remain fixed, the team could have a single `components_and_capabilities.pl` file, like the one shown in Fig. 2. Similarly, the potential faults, failures, and fixes for these components do not change between missions. So, the team could also have a single `fails_faults_and_fixes.pl` file containing probabilistic knowledge similar to that shown in Fig. 4. These files can be extended as inventory and beliefs evolve, but should not change for regular day-to-day testing.

Other knowledge will vary on a mission-by-mission basis. For instance, for each vehicle configuration, there should be a corresponding `health_topics.pl` file, like the one shown in Fig. 3. This information should match the current vehicle and environment. For instance, for an AUV camera, nominal `img_brightness` could depend on whether the mission occurs near the surface or at depth, during day or night, in a turbid river or in clear ocean water.

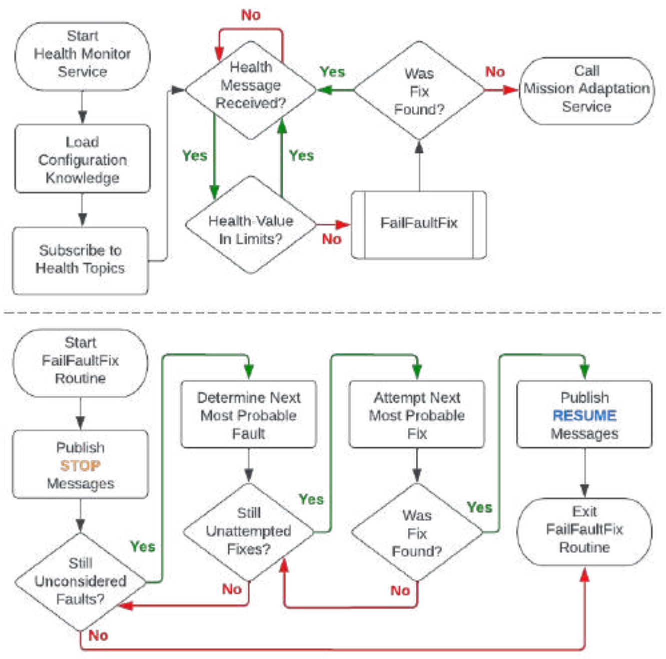
Currently, information contained in these files is loaded during initialization of the ROS node discussed in Sec. 4.2. In the future, we plan to fully integrate our solution with KnowRob[3],

[3]https://github.com/knowrob/knowrob



Health Topic Data
Configuration Knowledge (rosprolog)
Knowledge Base Queries (rosprolog)
Knowledge Base Responses (rosprolog)
Configuration Files
components_and_capabilities.pl
fails_faults_and_fixes.pl
vehicle_configuration.pl
REMARO
RELIABLE AI FOR MARINE ROBOTICS
Actuators
Sensors
Health Monitor
FailFaultFix
roscore
Knowledge Base

Figure 5: Overview of the proposed health monitor architecture.



Start Health Monitor Service
Load Configuration Knowledge
Subscribe to Health Topics
Health Message Received?
No
Yes
Yes
Yes
Health Value In Limits?
No
FailFaultFix
Was Fix Found?
Yes
No
Call Mission Adaptation Service
Start FailFaultFix Routine
Publish STOP Messages
Still Unconsidered Faults?
Yes
No
Determine Next Most Probable Fault
Still Unattempted Fixes?
Yes
No
Attempt Next Most Probable Fix
Was Fix Found?
Yes
No
Publish RESUME Messages
Exit FailFaultFix Routine

Figure 6: Flowchart for the proposed health monitor ROS node.

which is an open-source, ROS-based knowledge processing system for acquiring, grounding, representing, and reasoning with heterogeneous knowledge [11]. For now, our solution relies only on the `rosprolog`[4] functionality within KnowRob, which provides an interface between Prolog and other ROS nodes. Its primary usage is extracting knowledge from the Prolog files mentioned above and using this knowledge to answer queries from other ROS nodes, such as the proposed health monitor node.

[4]https://github.com/knowrob/rosprolog

4.2. ROS node

As shown in Fig. 6, the ROS node is both a subscriber and publisher. On initialization, the node extracts health topics, metrics, and limits from `health_topics.pl`. It then subscribes to health topics for all hardware, software, and environment components defined for the current mission.

During nominal operations, the node simply awaits published messages on the monitored topics. When a new message is received, a callback is prompted, the current health value is extracted, and that value is compared to the predefined limits. If the value is within range, the callback terminates and the subscriber resumes waiting for the next message.

If a value is non-nominal, the failure triggers the FailFaultFix subroutine. First, a STOP procedure is initiated to protect the affected component from further damage. These STOP procedures consists of ROS messages that can, for example, unpower a sensor, return an actuator to default position, or kill a software service.

Next, the FailFaultFix subroutine searches for faults capable of causing the detected failure. The most likely fault is determined (via `cplint` queries involving the knowledge in `fails_faults_and_fixes.pl`). For this fault, its most likely fix (determined similarly) is then attempted. Like the emergency STOP procedures, each FIX publishes a sequence of ROS messages. For instance, the messages could restart a camera driver, adjust perception algorithms to account for visibility changes, or pulse the thruster to clear a blockage. The last step in the FIX procedure essentially undoes the STOP procedure. If the failure is no longer detected, the fix is assumed successful and the FailFaultFix subroutine exits. If the failure remains, the next most likely fix is considered until all have been exhausted. Once all possible fixes for a fault have been attempted, the next most likely fault is considered along with its fixes. This process is repeated until either the failure has been resolved or no more potential fixes remain.

Although the current implementation can compute risks (likelihoods of side-effect faults) of candidate fixes, it can also attempt fixes for those faults should they be realized. As such, the current priority is fixing present faults, not avoiding potential faults. In future work, we intend to consider the upper half of Fig. 1 and how such risks might jeopardize mission success. For now, should the FailFaultFix subroutine exists unsuccessfully, then higher-level services must be involved to adapt mission plans for any compromised capabilities. The most notable frameworks for this include the Cognitve Robotics Abstract Machine (CRAM)[12] and Metacontrol for ROS (MROS)[13]. We consider our current solution a complement to such systems, ensuring faults are truly unrecoverable before resorting to component substitutions and controller adaptations.

5. Experiments

5.1. Real thruster fault on an AUV

In our first scenario, data is analyzed from a real field trial where an AUV suffered a catastrophic thruster failure when a cable became tangled around the rotor blades. As Fig. 7 shows, several failures occurred in quick succession. Almost simultaneously, the difference jumped between desired and measured speeds, and the temperature began rapidly increasing in the motor. Around 10 seconds later, three power issues occurred near synchronously.

Since these power-related metrics/messages may have different publication rates, all of them

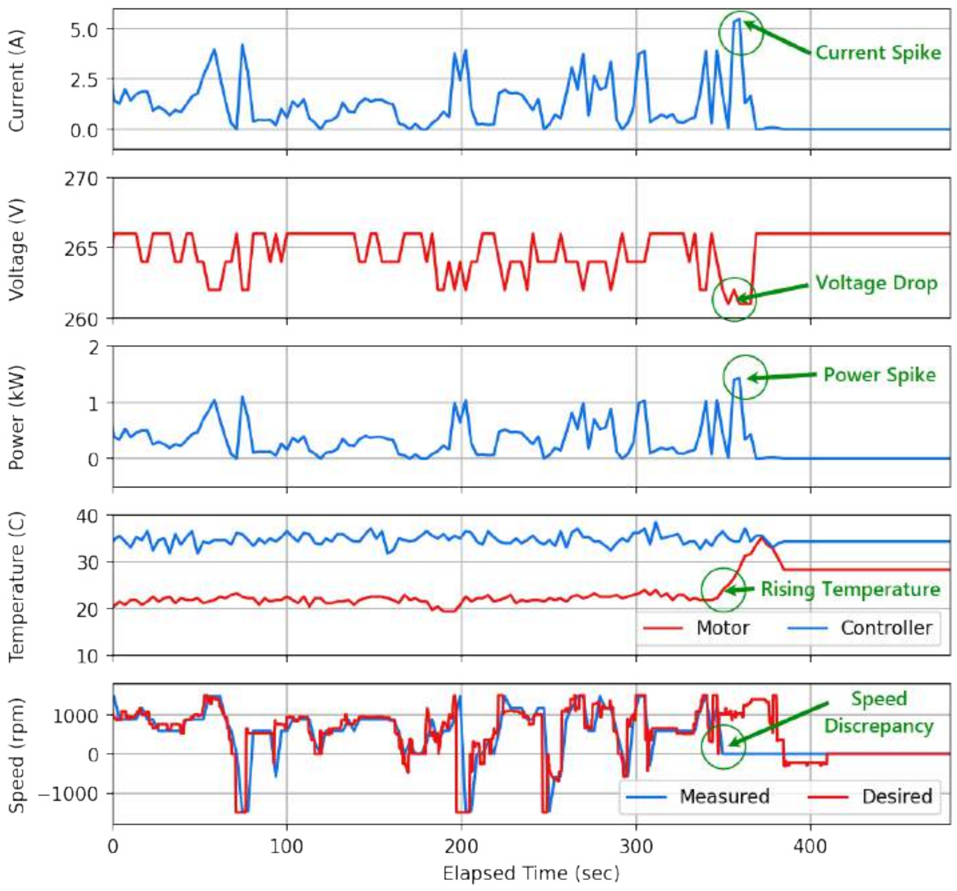


Figure 7: Failure analysis for real AUV thruster fault.

```
hasHealthTopic(port_thruster, "remaro_auv/current_draw").
hasHealthTopic(port_thruster, "remaro_auv/speed_diff").
hasHealthTopic(port_thruster, "remaro_auv/temp_gain").
hasHealthMetric("remaro_auv/port_thruster/current_draw", current_draw).
hasHealthMetric("remaro_auv/port_thruster/speed_diff",   speed_diff).
hasHealthMetric("remaro_auv/port_thruster/temp_gain",    temp_gain).
hasHealthLimits(current_draw, [0, 5]).   % amps
hasHealthLimits(speed_diff,   [0, 0.1]). % a unitless ratio
hasHealthLimits(temp_gain,    [0, 2]).   % 2 C/sec (6 C over 3 sec)
```

Figure 8: Minimal health knowledge needed for proposed thruster health monitor.

should be included in the health monitor to ensure the earliest possible failure indicator is caught. Other particularly useful metrics can be derived from standard topics. For instance, monitoring the relative difference between desired and measured values is usually more effective than measuring the values individually. For this reason, we consider the scaled speed differential:

$$\texttt{speed\_diff} = |\texttt{speed\_measured} - \texttt{speed\_desired}| \;/\; \texttt{speed\_max}$$

In addition, we suggest limits on both temperature and its time derivative. More specifically, we consider an averaged derivative over an interval of $\Delta t = 5$ seconds, which provides a nice compromise between slow response time and frequent false alarms from noisy readings:

$$\texttt{temp\_gain}(t) = [\texttt{motor\_temperature}(t) - \texttt{motor\_temperature}(t - \Delta t)] \ / \ \Delta t$$

To avoid similar future faults, we propose the settings shown in Fig. 8. With this configuration, the health monitor (limited only by the 0.3 Hz publication rates of the `desired_speed` and `motor_temperature` topics) detects within 5 seconds both the temperature gain and speed discrepancy. This results in a STOP message (setting speed to zero) long before the current spike that ultimately destroyed internal circuitry. Notice the catastrophic fault could have been prevented thanks in part to the internal motor temperature sensor. A cheaper thruster might not provide such data causing the same failure to go undetected in time to save the device.

Although not testable with the available ROS bag, we believe several fixes were possible. First, a "wait-and-see" fix could have allowed the cable to drift out on its own. Other possibilities include alternating pulses to loosen a jam or reversing to unwind an entanglement. Each of these attempts would be immediately aborted if health metrics continued to worsen.

5.2. Simulated thruster faults on an ROV

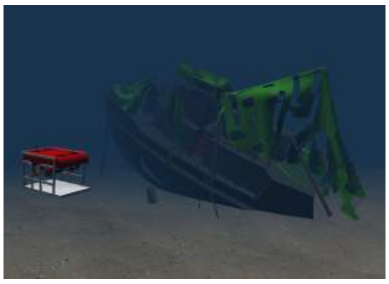
Our second scenario considers a simulated ROV suffering a thruster fault while performing a shipwreck survey. More specifically, we consider the REXROV surveying the Hercules wreckage shown in the Gazebo simulator screenshot in Fig. 9(a). The ROV, ship, and terrain model are all included in the `uuv_simulator`[5] developed during the SWARMs Project [14].

Because we do not have access to low-level hardware data in the simulation, like temperature, power usage, and control signals, we instead use trajectory deviations as our primary health metric. For demonstrations purposes, we configured the vehicle to follow a tight helical path as it descends around the Hercules. With the shipwreck at 60 meters depth and the ROV starting 15 meters above, we began three trials, each with the ROV completing five loops of radius 14 meters and a vertical displacement 2.6 meters per loop. During the first scenario, no fault was present and the ROV followed the desired trajectory. During the second, a fault was forced at $t = 540$ seconds by manually disabling a subset of the sternside thrusters. This result is shown in RViz in Fig. 9(b). The final scenario recreated the same fault, but followed with a successful `thruster_cycle_power_fix` at $t = 850$ seconds. As can be seen in Fig. 10, the ROV successfully recovered and reacquired the desired trajectory within 30 seconds.

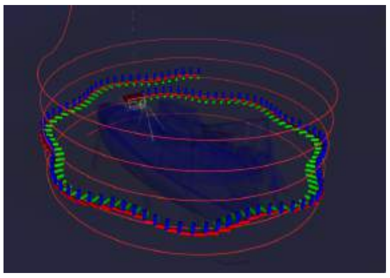
6. Related work

A variety of approaches are common for fault detection, including analytical, knowledge-based, and data-driven models [15]. Analytical models are often derived from first principles to describe differences between observed and modeled behavior [16, 17]. Such systems can be accurate, but also difficult to develop. Knowledge-based methods, which rely on domain experts and ontological formulations, can offer simpler and more expressive solutions [18, 19]. Data-driven

[5]https://github.com/uuvsimulator/uuv_simulator



(a) Simulated scenario in Gazebo.



(b) Faulty trajectory in RViz.

Figure 9: Simulation of REXROV surveying the Hercules shipwreck.

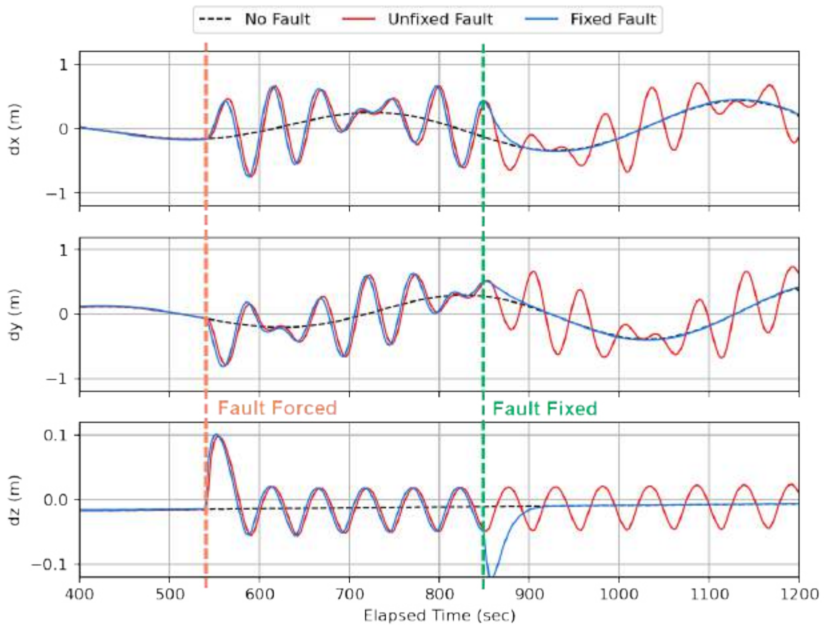


Figure 10: Comparison of trajectory deviations with and without faults and fixes.

methods have gained increasing popularity due to the rapid growth of AI-based techniques [20]. For example, in [21], a neural network was trained on empirical data to provide a soft-fault model for marine thrusters. An interesting hybrid approach in [22] is used to develop an energy-aware architecture for fault mitigation in AUVs from both data-driven actuator characterizations and model-based anomaly diagnoses. We believe our proposed solution offers a complement to such methods, while adding probabilistic capabilities to existing knowledge-based approaches.

Several existing frameworks rely on semantic knowledge representation for collaborative underwater robots [23]. In the SWARMs Project [14], a probabilistic ontology was combined with a multi-entity Bayesian network (MEBN) to reason with uncertainty for chemical spill monitoring. Cordova [24] developed an ontology to describe robotic devices and skills to enable collaborative knowledge acquisition between real and virtual worlds. Our ontology is also designed to be shareable between robots and environments, but at the component level.

Diab et al. [25] presented an ontology for overcoming task execution failures in service robotics. In [26], a framework was developed for fault-tolerant mission adaptations in underwater robotics by relating plans, actions, and capabilities. Such solutions attempt to overcome faults by substituting components and capabilities, or altering plans. Our solution takes a different approach by attempting to exhaust all possible fixes to allow tasks to be executed as planned.

Self-awareness, self-adaptation, and metacontrol offer additional perspective for overcoming faults and failures. Patrón et al. [27, 28] used Boyd's Cycle [29] of *Observe* $\rightarrow$ *Orient* $\rightarrow$ *Decide* $\rightarrow$ *Act* to combine situational awareness with analytical planners to repair and modify plans. In [13], TOMASys[6] was used as part of a ROS-based metacontrol system for self-adapting systems that combined multiple levels of abstraction and autonomy. TOMASys is also used in [30] to enable self-diagnosis and autonomous reconfiguration to preserve mission-mandated capabilities. Papadimitriou [31] demonstrated a semantic-based knowledge framework to simulate and overcome a hardware fault on a REMUS100 AUV by performing mission and plan adaptation during mine countermeasure missions (MCM). In each of these cases, the solutions require complete frameworks in order to achieve their higher-level functionalities. Our lower-level solution is designed to be an add-on to existing frameworks, including these, ensuring first that components cannot be repaired before they are replaced.

7. Conclusion

This paper introduces the REMARO ontology for detecting failures, isolating their underlying faults, and attempting fixes for their recovery. The ontology supports both deterministic and probabilistic knowledge representation and reasoning. These probabilities allow domain experts, past and simulated missions, and manufacturer-provided specifications to guide the order in which faults and fixes are considered.

To demonstrate the ontology, we propose a minimal ROS node designed to add fault recovery capabilities to existing UUV frameworks. This node subscribes to health topics specified in a mission configuration file. Whenever a new message is received that reveals non-nominal health values, a subroutine is triggered to consider the most probable faults and their most likely fixes. If a fix is found, monitoring resumes. Otherwise, higher-level services must be used.

In future work, we hope to extend the health monitor node to provide component substitutions when fixes fail and mission reductions when substitutions are unavailable. For now, our solution serves as a complement to mission adaptation frameworks like CRAM [12] and metacontrol architectures like MROS [13]. Other future plans include field testing of the presented solution and extension of the solution to include more complex probabilistic representations. Such extensions could consider recurring faults and temporary fixes, updating probabilities based on outcomes from recently attempted fixes, and standardization of STOP and FIX subroutines.

Acknowledgments

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie SkłodowskaCurie grant agreement No. 956200. For more info, please visit https://remaro.eu.

[6]Teleological and Ontological Model of an Autonomous System

References

[1] S. Ishibashi, H. Yoshida, Developing a sediment sampling ROV for the deepest ocean, Sea Technology 49 (2008) 43–46.

[2] P. Norgren, R. Lubbad, R. Skjetne, Unmanned underwater vehicles in Arctic operations, in: Proceedings of the 22nd IAHR International Symposium on Ice. Singapore, 2014, pp. 89–101.

[3] J. Strutt, Report of the inquiry into the loss of AutoSub2 under the Fimbulisen, 2006. URL: https://eprints.soton.ac.uk/41098/.

[4] J. Copely, Onwards & downwards: when ROVs or AUVs are lost in ocean exploration, 2014. URL: http://www.joncopley.com/blog_may14a.html.

[5] J. Laprie, Dependable computing and fault-tolerance: concepts and terminology, in: Proceedings of the 25th International Symposium on Fault-Tolerant Computing, IEEE, 1995, pp. 27–30.

[6] J. Carlson, R. R. Murphy, How UGVs physically fail in the field, IEEE Transactions on Robotics 21 (2005) 423–437.

[7] ISO 10303:2021(E), Industrial automation systems and integration — Product data representation and exchange, Standard, International Organization for Standardization, Geneva, CH, 2021.

[8] G. Antoniou, F. v. Harmelen, Web Ontology Language: OWL, in: Handbook on Ontologies, Springer, 2004, pp. 67–92.

[9] M. A. Musen, The Protégé Project: a look back and a look forward, AI Matters 1 (2015) 4–12.

[10] F. Riguzzi, G. Cota, E. Bellodi, R. Zese, Causal inference in cplint, International Journal of Approximate Reasoning 91 (2017) 216–232.

[11] M. Beetz, D. Beßler, A. Haidu, M. Pomarlan, A. K. Bozcuoğlu, G. Bartels, KnowRob 2.0—a 2nd generation knowledge processing framework for cognition-enabled robotic agents, in: 2018 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2018, pp. 512–519.

[12] M. Beetz, L. Mösenlechner, M. Tenorth, CRAM—A Cognitive Robot Abstract Machine for everyday manipulation in human environments, in: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2010, pp. 1012–1017.

[13] C. H. Corbato, D. Bozhinoski, M. G. Oviedo, G. van der Hoorn, N. H. Garcia, H. Deshpande, J. Tjerngren, A. Wąsowski, MROS: Runtime adaptation for robot control architectures, arXiv preprint arXiv:2010.09145 (2020).

[14] X. Li, S. Bilbao, T. Martín-Wanton, J. Bastos, J. Rodriguez, SWARMs Ontology: A common information model for the cooperation of underwater robots, Sensors 17 (2017) 569.

[15] L. H. Chiang, E. L. Russell, R. D. Braatz, Fault detection and diagnosis in industrial systems, Springer Science & Business Media, 2000.

[16] M. L. Leuschen, I. D. Walker, J. R. Cavallaro, Fault residual generation via nonlinear analytical redundancy, IEEE transactions on control systems technology 13 (2005) 452–458.

[17] A. Shumsky, A. Zhirabok, C. Hajiyev, Observer-based fault diagnosis in thrusters of autonomous underwater vehicles, in: 2010 Conference on Control and Fault-Tolerant

Systems (SysTol), IEEE, 2010, pp. 11–16.

[18] B. Liu, M. Duan, G. Zhao, An object frame knowledge representation approach for fault diagnosis expert system, in: 2011 International Conference on Future Computer Sciences and Application, IEEE, 2011, pp. 74–77.

[19] X. Deng, R. Luo, J. Li, Similarity matching algorithm of equipment fault diagnosis based on CBR, in: 2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS), IEEE, 2015, pp. 998–1002.

[20] X. Dai, Z. Gao, From model, signal to knowledge: A data-driven perspective of fault detection and diagnosis, IEEE Transactions on Industrial Informatics 9 (2013) 2226–2238.

[21] S. Nascimento, M. Valdenegro-Toro, Modeling and soft-fault diagnosis of underwater thrusters with recurrent neural networks, IFAC-PapersOnLine 51 (2018) 80–85.

[22] V. De Carolis, F. Maurelli, K. E. Brown, D. M. Lane, Energy-aware fault-mitigation architecture for underwater vehicles, Autonomous Robots 41 (2017) 1083–1105.

[23] P. Patrón, Y. Petillot, The underwater environment: a challenge for planning, UK PlanSIG Edinburgh 2008, 2008.

[24] A. J. Cordova, Semantic web for robots : applying semantic web technologies for interoperability between virtual worlds and real robots, Technische Universiteit Eindhoven, 2012.

[25] M. Diab, M. Pomarlan, S. Borgo, D. Bebler, J. Rosell Gratacòs, J. Bateman, M. Beetz, FailRecOnt-an ontology-based framework for failure interpretation and recovery in planning and execution, in: Proceedings of the 2nd International Workshop on Ontologies for Autonomous Robotics, 2021, pp. 1–14.

[26] P. Patrón, E. Miguelanez, J. Cartwright, Y. R. Petillot, Semantic knowledge-based representation for improving situation awareness in service oriented agents of autonomous underwater vehicles, OCEANS, 2008.

[27] P. Patrón, E. Miguelanez, Y. R. Petillot, D. M. Lane, Fault-tolerant adaptive mission planning with semantic knowledge representation for autonomous underwater vehicles, in: 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2008, pp. 2593–2598.

[28] P. Patrón, E. Miguelanez, Y. R. Petillot, D. M. Lane, J. Salvi, Adaptive mission plan diagnosis and repair for fault recovery in autonomous underwater vehicles, in: OCEANS 2008, IEEE, 2008, pp. 1–9.

[29] H. Hillaker, Tribute to John R. Boyd, Code One Magazine 12 (1997).

[30] E. Aguado, Z. Milosevic, C. Hernández, R. Sanz, M. Garzon, D. Bozhinoski, C. Rossi, Functional self-awareness and metacontrol for underwater robot autonomy, Sensors 21 (2021) 1210.

[31] G. Papadimitriou, D. Lane, Semantic-based knowledge representation and adaptive mission planning for MCM missions using AUVs, in: OCEANS 2014-TAIPEI, IEEE, 2014, pp. 1–8.