A Fault Tolerant Control Architecture Based on Fault Trees for an Underwater Robot Executing Transect Missions

Adrien Hereau, Karen Godary-Dejean, Jérémie Guiochet, Didier Crestani

A Fault Tolerant Control Architecture Based on Fault Trees for an Underwater Robot Executing Transect Missions

Adrien Hereau[1], Karen Godary-Dejean[1], Jérémie Guiochet[2], Didier Crestani[1]

***Abstract*— Robotic systems evolving in hazardous and harsh environment are prone to mission failure or system loss in presence of faults. This paper presents a fault tolerant methodology, implemented into a control architecture of an underwater robot that executes biological monitoring missions. High level constraint violations (mission, safety, energy, time and localization) and low level faults (software and hardware faults) are considered using a method based on fault trees. These undesirable events are detected and treated by a fault tolerant module that decides to recover at low level or to give a feedback to the mission manager which selects the high level reaction. This fault tolerant architecture has been tested on real field conditions, and we illustrate our methodology on a set of selected events. We conclude about reliability improvement of low cost underwater robots for complex and long missions.**

I. INTRODUCTION

Dependability is a key aspect in robotic systems ([1], [6]). For a mobile robot evolving in a hazardous and harsh environment, it is difficult to predict all the faults, especially for underwater robotics. For example, [16] noted a lot of faults on several glider underwater robots that perform long missions. 297 missions were performed across the world, 49 of them ended with a critical problem and 9 robots were lost. The faults can be of various types: sensors, mechanical, electrical, software or physical damage.

Due to the lack of low-cost efficient hardware, the underwater robot hardware is often expensive since it must be water proof while withstanding high pressure. This is also a small market where supply and demand did not lead to development of affordable and highly efficient industrial products. In practice, the main problem remains that there is no easy absolute and precise localization as the GPS signal is not available under water. The preferred approaches generally use acoustic (sonar, USBL) or vision sensors which are often imprecise, slow, expensive and/or sensitive to environmental conditions. Thus dependability is even more a challenge for underwater robotics.

In 2019, we developed an underwater robot that performed transect missions in Mayotte for biological purposes [7]. A transect consists in traveling a horizontal straight line between two points, while recording marine ecosystem data. Our robot underwent several physical constraints that were not well established at that time: pulling force on the cable, low precision of the low cost sensors... To validate the correct execution of the robot and of the mission, we designed an oracle test defining 5 property classes: mission, safety, time, energy and localization. According to this test methodology, we demonstrated in [7] that our robot was not reliable enough when performing its mission as many properties were violated.

The objective of this paper is to present a methodology that designs a fault tolerant control architecture for a low-cost underwater robot that executes autonomous missions. The first step of our methodology is inspired from common fault forecasting techniques, to describe and predict the possible occurrence of faults and their consequences using fault trees implementation. Then we implement in the embedded robot controller the classical scheme for fault tolerance: detection, diagnosis and recovery [1]. The originality of our approach is the two-levels recovery method. Indeed, robots must deal with undesired events at every level, from the violation of the high level properties cited above which may cause recovery actions at the mission management level, to the lower level events in the system (the faults) which can be resolved by local reactions.

In section II, we present different approaches proposed in the literature addressing the fault tolerance in mobile robotics. Then we describe our robotic system and the reference mission used to illustrate our methodology in section III. In section IV, we present the methodology to deal with high and low level undesired events. We present the fault tolerant module that detects in real time the presence of undesired events in the system then we focus on the recovery functionality. In section V, we propose and analyze the results of our underwater robotic mission in the field. Finally we conclude in section VI.

II. RELATED WORK

In this section, we study the existing fault tolerant techniques in the literature. As a first ascertainment, many approaches relies on strong hypotheses such as the prior knowledge needed to forecast the faults ([8] [3]). Some papers are based on the single fault assumption [13], which is very reductive in an open environment where various faults can occurs at the same time.

Most of the detection methods rely on model-based approaches. The main difficulties of these approaches are to choose the detection thresholds and to determine the robot model. [19] detects the residuals between a model and the sensor on a Roomba robot. The detection thresholds are calculated by an adaptive law. [9] prefers to use a sensor-based approach for robotic system fault detection and diagnosis using correlations between sensors. [4] chooses to detect fault on actuators for underwater robots based on the energy consumption. A widespread approach is also to use

[1] LIRMM, University of Montpellier, CNRS, Montpellier, France
[2] LAAS-CNRS, University of Toulouse, CNRS, UPS, Toulouse, France

comparison between sensors to detect invalid sensor data using multiple data fusions [2].

Many works put their attention on the diagnosis of faults but few works are relating fault analysis and online diagnosis. For instance, [8] uses fault tree (FT) analysis to diagnose the possible cutsets, that are sets of component failures that could lead to a global system failure. FT are usually used to find the minimal cutsets of a system and to try to calculate the probability of an undesired event. The system is considered as safe if the probability of fault occurrence is sufficiently low. FT are a good way to inventory faults and are widely used in aerospace [18]. They have proved their efficiency in modeling faults in complex systems. Other risk analysis technique can be used, like in [3] where FMECA analysis is combined with incidence matrix using residuals from sensors. From a list of bad residuals, it is then possible to diagnose a particular fault.

Recovery is the final step of fault tolerance. Recoveries can intervene at several levels in the architecture. For example at low level, [13] and [17] deal with recovering flying drones by compensating faulty actuators with valid ones. At a higher level, [12] deals with multi-robot applications and chooses many possible recoveries: change path, reset algorithm, request or alert human, reset parameters, switch to wait state. In [11], the recovery is carried at in the decision layer, where redundant planners are used in case of failures to replan switching from one planner to another (sequentially or concurrently). The authors of [5] use FT and can trigger reactions at different level, depending on the detected faults. The reactions to preserve the safety are: emergency stop, controlled stop, restart nodes, change autonomy level. On the opposite, authors of [10] favor the mission first. They use the same breakdown of fault categories as ours and propose to use redundancy and mission replanning if needed. [14] simply recovers with a replanning of the path of a rover in order to continue the mission. Other approaches adapt the level of autonomy of the robot depending on the severity of the fault [3].

To our knowledge, no real complete architecture successfully embeds high level constraint violation management (mission, safety…) and low level fault management (sensor, actuators…) for a robot in an open field. We propose then an architecture that addresses this issue, embedded in the robotic system described in next section.

III. SYSTEM AND MISSION DESCRIPTIONS

A. System Description

The underwater robot (Fig. 1) is an improvement of a previous version described in [7]. With 4 vertical and 4 horizontal thrusters (M1 to M8), the robot is a 6 degrees of freedom (DoF) semi-AUV. It embeds 6 sensors (echosounder, IMU, pressure, camera, GPS and USBL), and 4 batteries allowing a 4-hours energy autonomy. The mission is done autonomously but a communication between the surface and the robot is provided through a 200 meters Ethernet cable. The operator can launch or interrupt an ongoing autonomous mission, and remotely control the robot. The long-term objective is to remove the cable to perform fully autonomous tasks.

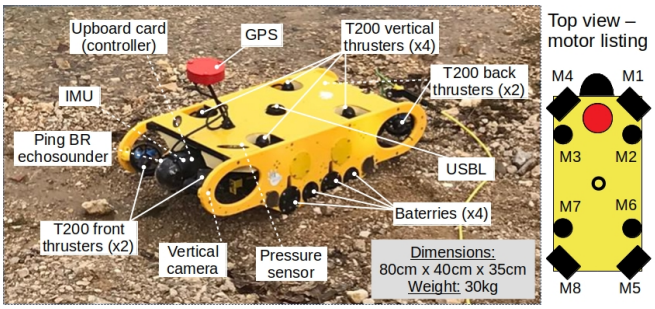

Fig. 1. The REMI robot

The software control architecture embedded on the controller is shown in the left side of figure 3. It consists of several independent modules launched by `module_manager`.

`sensors_driver` is composed of several independent modules receiving data from the USBL (3D absolute position = X, Y, depth), the echosounder (altitude), the vertical camera (surge and sway velocities), the embedded GPS (X, Y), the IMU (attitude, rotational speeds, accelerations), the pressure (depth) and the water leak sensors. Using an extended Kalman filter and a dynamic model of the robot, `state_estimator` module processes the sensor data and the thrust of the propellers to determine the state of the robot (position and speed).

`task_controller` calculates the desired motor Pulse Width Modulation (PWM) commands to perform the current task and sends them to `motors_driver` which commands the motors through power-switches and speed controllers. `mission_manager` decides the task to launch from its knowledge of the actual task, of the desired mission and of the robot state. `surface_comm` gets the user command (desired mission or remote control command) and sends back the supervision information to the surface.

We add `fault_manager` (right side of figure 3) module that detects and diagnoses the encountered undesired events. This module is also able to launch low level recovery actions that do not impact directly the mission execution. Its description is detailed in section IV.

B. Mission Description

We focus on the transect mission in autonomous mode. A transect (described in Fig. 2) is basically defined by its Start Point (SP) and End Point (EP). A virtual straight line called Transect Line (TL) links SP and EP. A transect can be executed either at constant depth or constant altitude according to user needs.

We identify 4 Mission Phases (MP) during the mission. In initial phase 0 (MP0), the robot dives vertically until a certain depth and checks

The goal of our work is to allow our robot to execute this mission autonomously while remaining safe.

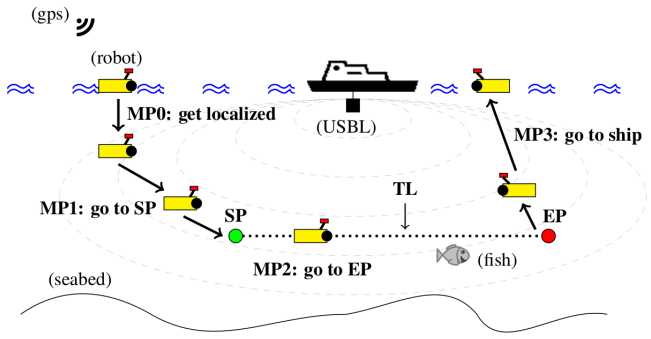

Fig. 2. Transect mission description: The robot dives and looks for the USBL signal. Then its goes to SP, performs the transect and finally goes back to ship.

IV. METHODOLOGY OF FAULT TOLERANCE

We developed a methodology based on the classical scheme of fault tolerance [1]: detection-diagnosis-recovery, preceded by a fault forecasting phase. In this phase, we perform an analysis of the properties of our system over several axes and we list them using fault trees (FT). During the mission execution, we detect undesired events with classical methods, then we use the FT to diagnose the events and we recover at two levels: low level for local reactions and high level for a task change (Fig. 3). This methodology takes into account all the variability of events that underwater robots can encounter.

In this section, we describe each step of our methodology, illustrating them on our case study, but focusing mainly on the recovery aspect. For that reason, we choose as example simple events to detect and diagnose, but with different recovery possibilities.

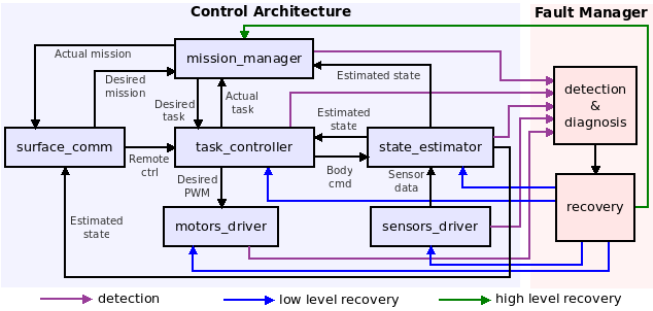

Fig. 3. Architecture for fault tolerance

A. Fault Forecasting

The first step of our methodology is inspired from fault forecasting methods, allowing *to estimate the present number, the future incidence, and the likely consequences of faults* [1]. We list the properties the system has to validate to guarantee a safe and correct execution of the mission, and we determine the undesirable events that could provoke the violation of these properties.

List of the properties and the undesirable events of our case study: To operate a correct and safe mission, the robot must satisfy several properties. As in our previous works ([10] [7]), we divide these properties into 5 classes. The *time* class imposes constraints over the duration of the mission phases. The *safety* class imposes constraints to preserve the integrity of the robot and its environment. The *energy* class imposes constraints to make sure there is enough remaining energy to finish the mission. The *localization* class imposes constraints on the precision of the localization. Finally, the *mission* class deals with the user-oriented constraints over the whole mission. In table I, we present the short list of the properties we focus on in this paper, restricted to classes Mission, Energy and Safety.

TABLE I
SOME PROPERTIES OF THE TRANSECT MISSION

Class	Property	Description
Mission	P1	The robot must remain close to TL
Safety	P2	The robot must remain at safe distance from the seabed
	P3	The robot must be able to control its 6DoF
Energy	P4	The robot must have enough energy to finish the mission

We consider that the violation of a property is a failure of the system, and faults are the direct or indirect causes of a failure.

Building fault trees: We use FT to inventory the undesirable events and represent the relations between them, i.e. between the violation of the properties and the faults. As seen in section II, FT are often used to forecast undesirable events and improve the reliability of a system. We use them in a different way, as we note FT are easy to implement on an embedded device to run in real time, which will be useful for the diagnosis phase. For simplicity, we use in the FT only the basic gates (AND, OR, Transfer) and the basic events [18].

We build the global FT of the system with 4 sub-fault trees representing the failures including mission, safety, energy and localization classes. The time constraint violations are integrated in the mission sub-fault tree, and the undesirable events of the system can be present in all the sub-fault trees.

The *mission sub-fault tree* is built by cutting the mission into mission phases as explained section III-B. Basically, there is a failure in the mission if there is a failure in one of its mission phases. In each phase, a failure is a violation of a property defined by the user, as for example the violation of P1.

For the *safety sub-fault tree* (Fig. 4), we list as failures all the undesired events that could have important consequences. For example, we want to avoid any collisions with the seabed. There are several steps we can consider before a collision. For that reason, we define a *critical zone* very close to the seabed (0.5m) in which the robot must not enter (this is property P2), and a *warning zone* (1.1m) before the critical zone (0.5m). If the robot enters into one of these zones,

an undesired event is detected, but with different recovery actions. These two cases are represented in Fig. 4 with the events SAFE_1 and SAFE_1_1.

Regarding safety, we can cite another example as the violation of P3, which leads to the failure SAFE_2 in Fig. 4. This could be the consequence of a system fault, as for example a problem on the actuators.

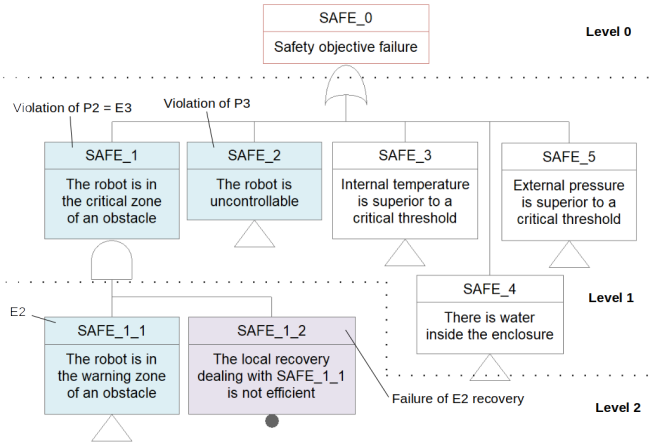

Fig. 4. Extract of the safety sub-fault tree

B. Detection and Diagnosis

In this article, we use basic fault detection methods which consist in comparing a data value with a threshold. Also, we do not illustrate the diagnosis part, supposing the detected undesired event is sufficient to engage a recovery action.

Example for our case study: In this paper, we focus on 5 undesired events linked with the failures of the properties described in table I:

$E1$: violation of $P1$. We check when the distance between TL and the estimated position of the robot is superior to a specific threshold ($1m$) during more than a certain time ($10s$). Basically, we compute the distance between a point and a line in 3D space.

$E2$: related to $P2$: the robot enters the warning zone of the seabed. We detect this event by checking if the estimation of the altitude is below a threshold (1.1m).

$E3$: violation of $P2$: the robot enters the critical zone. We detect it as $E2$ but with a different threshold (0.5m).

$E4$: M6 (see Fig. 1) is not operational. It can lead to a violation of $P3$ if a specific set of other motors are out of service. We can detect that a motor is down by looking at its consumption. As shown in Fig. 5, a motor which is blocked[1], consumes more intensity than an operational one. This figure presents experimental results we have done on our robot, individually actuating the motors in different situations and measuring the current intensity thanks to a power-switch.

[1]In submarine robotics, this problem is quite common due for example to corrosion because of the salt residue, or to seaweeds stuck in the screws.

$E5$: violation of $P4$. Thanks to a model of the energy consumption of our system[2], we estimate the energy needed for the robot to finish its mission. We also estimate the energy remaining in the robot. The undesired event is present if the remaining energy is below the needed energy to finish the mission (plus a safety margin).

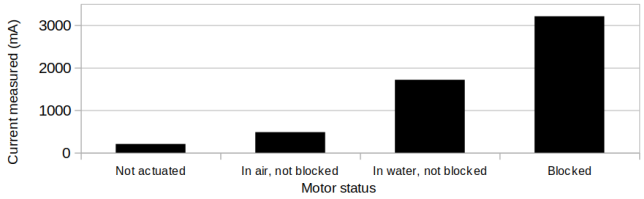

Fig. 5. Mean power-switch current measured for a motor actuated with a constant PWM input value ($1600\mu s$)

C. Recovery

The objective of a recovery is to suppress an undesirable event or to stabilize it in order to prevent the failure to occur. Depending on the event detected or diagnosed, there are two levels of recovery actions: either at local level to prevent more serious consequences, or at decisional level when it is necessary to act at the mission level. These recovery levels are directly linked with the sub-fault trees levels (Fig. 4). Basically, the occurrence of event at level 1 of a sub-fault tree is directly sent to the mission manager. This last one will take the decision to change the task the robot has to execute, in order to prevent disastrous consequences. On the other hand, if the detected event is at a lower level, we try to carry out local recovery actions to prevent this event to propagate to higher levels. Such recoveries could be changing parameters of algorithms, restarting software modules, managing the redundancy...

Examples for our case study:

$E1$ is a level 1 event in the mission sub-fault tree. As a consequence, the reaction is at high level. The biologists estimate that being too far from the desired ecosystem they want to monitor is without interest, so in that case the mission manager will stop the transect mission.

$E2$ is a level 2 event (SAFE_1_1). As it exists a low level recovery (its failure is represented by event SAFE_1_2 on Fig. 4), $E2$ is then treated locally. The recovery consists in replacing the desired heave command calculated by the task_controller by a command that will bring the robot towards the surface (here -40 Newtons).

$E3$ is a level 1 event in the safety sub-fault tree (SAFE_1). If this event occurs, the previous local recovery planned to move away from the warning zone has failed. The recovery is thus at high level and the mission manager will for example stop the mission, considering that the risk of collision with the seabed is too high.

$E4$ is located in the safety sub-fault tree but not at level 1. The local recovery here is the management of the actuators redundancy, as our robot is over-actuated [15]. It

[2]Ongoing works adapted from [10] for submarine robots.

consists in rebuilding the actuator configuration matrix (A), to use only the valid actuators. We also check that all the DoF may be reached by verifying if the rank of A is 6. Basically for our robot, there must be 1 or less vertical and 1 or less horizontal motor broken. In that case, we recalculate the actuator distribution matrix (A^+) as the Moore–Penrose pseudo-inverse of A. If the rank of the A is inferior to 6, then the robot cannot control all its degrees of freedom and becomes uncontrollable (the robot is under-actuated). It is then a violation of P3 (SAFE_2) that triggers a high level reaction. In that case, depending on the available DoF, the mission manager will decide which tasks could still be done by the robot.

$E5$ is at level 1 of the energy sub-fault tree, so the reaction is at high level. Different reactions could be imagined here, depending on the remaining energy and the possible tasks. Typically, if the robot could not execute the core of the transect (MP2), the mission manager will interrupt the current task and order to go back to the ship immediately if possible. This could allow for example to change the batteries of the robot and reprogram the mission.

V. EXPERIMENTAL RESULTS

In 2020, we performed transect missions in several experimental sites. The goal is to illustrate the behavior of the robot when it undergoes the previously listed events. For some of them, we deliberately put the robot in specific situations leading to these events. The others were simulated.

E1 - Distance to the transect line TL: We perform a normal transect then pull the cable at t=112s to deflect the robot from TL during more than 10s. We plot the distance between the robot and TL over time on Fig. 6.

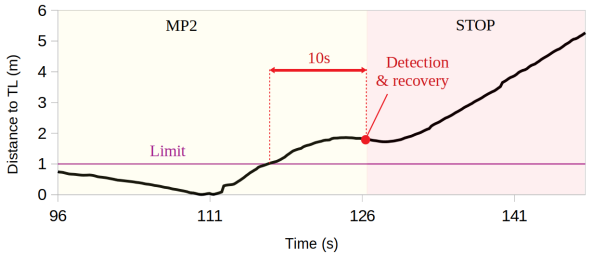

Fig. 6. Distance to TL during a transect with a detection of E1. We start pulling the cable around t=112s. At t=127s, the fault manager detects that the robot is too far from TL for more than 10 seconds. The mission manager decides then to stop the mission.

As expected, the robot detects that the robot is far from TL during 10s. This property needs a high-level recovery action: the information must be given to the mission manager which decides to stop the mission and the motors. As a consequence, the robot naturally rises to the surface with the positive buoyancy and so moves away from TL.

E2 - Distance from the seabed $< 1.1m$*:* We run a transect with a fixed desired depth of 4.5m. The total water column starts from 6.0m and decreases to 4.5m along the transect path. As a consequence, when staying at a constant depth, the robot gets closer to the seabed and finally triggers E2. On Fig. 7, we plot the depth and the altitude of the robot, with the targeted depth (which depends on the mission phase) and a minimal altitude of 1.1m provoking the event.

The robot tries to reach the targeted depth (MP0 and MP1). Then at the beginning of MP2, it stabilizes around the targeted depth while the altitude decreases. At t=237s, the robot reaches the warning zone of 1.1m: an event is detected, with an available low level recovery.

Thus the robot starts to react by sending a heave command in order to go up (Fig. 8). Once the robot returns above the limit altitude, the behavior becomes normal again and the robot tries to reach the desired depth. Then this behavior is repeated until the end of MP2 as the water column remains inferior to (4.5+1.1)m.

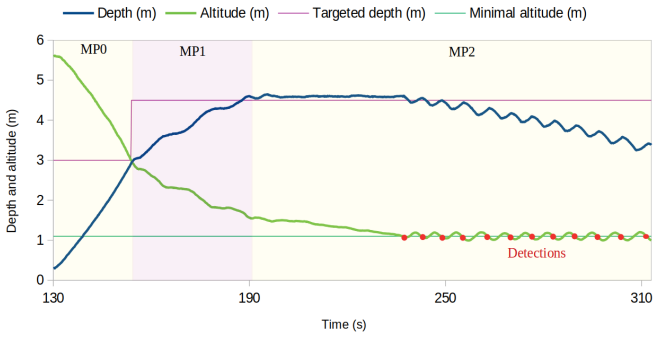

Fig. 7. Altitude and depth during a transect with a detection of E2

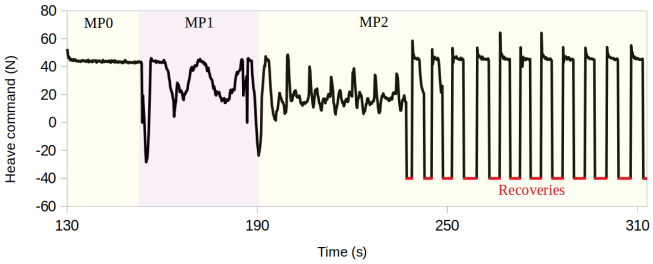

Fig. 8. Heave command during a transect with a detection of E2. Positive heave command will make the robot to dive whereas negative command will make the robot float back to the surface.

E3 - Distance from the seabed $< 0.5m$*:* We run a third transect similar to the second one except that the desired depth was 7.5m. We also deactivate the detection of E2 in order to pass below the 0.5m. When the robot reaches the critical altitude of 0.5m, E3 is detected. Through the mission manager, the robot decides to stop the mission and to go up to the surface.

E4 - Controllability of the robot: We run a fourth transect with a targeted depth of 3m where we simulate that M6 breaks down 30s after the beginning of MP2. The PWM commands of the vertical motors are plotted in Fig. 9 and the depth of the robot in Fig. 10.

When the event is detected, the robot reacts and recovers at low level. As described in the previous section, the robot

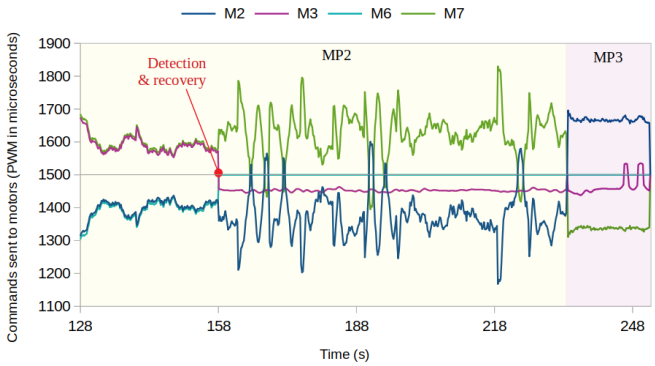

Fig. 9. PWM command of vertical motors during a transect with a detection of E4. After the recovery, the robot stop sending commands to M6 which is considered to be out of order. To compensate the loss of M6, the robot allocates the forces to the other vertical motors M2 and M7.

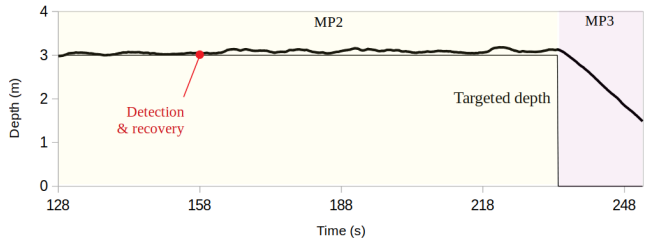

Fig. 10. Depth during a transect with a detection of E4. The depth remains stable despite the loss of M6.

reallocates the remaining thrusters. M6 is no longer actuated, its PWM input command becomes equal to the neutral value 1500μs. As a consequence, the diametrically opposite motor (M3) is not used for heave command anymore but only for the robot roll and pitch stabilization. The remaining vertical motors M2 and M7 deal with the heave command. We also see on Fig. 10 that the depth remains stable around 3m despite the loss of one motor, which proves that the fault tolerance mechanism is efficient.

E5 - Energy management: At this stage, the energy model of the robot is not yet available. The generation of this event is then only possible in simulation, with a low interest to show here.

VI. CONCLUSION

In this paper, we present an original methodology for the improvement of dependability of an underwater robot executing autonomous missions. The first step is the forecasting of properties defining a correct mission, and of the events that could prevent these properties to be respected. We used the well known Fault Trees (FT) to analyse events leading to the violation of 5 classes of properties: mission, energy, time, localization, and safety of the robot and its environment. Then we implement a detection-diagnosis-recovery scheme in the fault tolerant control architecture embedded on the robot, using FT for the diagnosis and low level recovery. Indeed, the recovery management is done at two levels: the low local level which does not influence the execution of the mission, and the high decisional level through a mission manager which changes the task of the robot (not described in this paper).

We implement this fault tolerant architecture in our robotic underwater platform and perform some experiments to illustrate our methodology. We present and explain experimental results, performing an in-the-field test campaign of transect missions, using fault injection to trigger the recovery mechanisms. Our approach seems to be able to manage faults efficiently preventing catastrophic damage while preserving the mission objective as far as possible.

Many works are needed to completely validate the fault tolerant architecture on our low-cost underwater robot REMI for the next monitoring campaign planned in 2021 in Mayotte with the marine biologists for the BUBOT project[3]. In the short term, all the event detection algorithms and low-level recovery must be implemented, the robot energy model must be finalized to manage energy-related undesired events, and the links with the mission manager must be tested in the field. Moreover, the Mayotte campaign feedback will be useful to identify new unplanned faults and to identify the limitations of our approach.

In the long term, we plan to tackle more deeply the management of the high-level event implementing performance guaranty using embedded hardware and software resources allocation or mission modifications. We also envisage to link the fault management to more classical planning techniques to enhance the richness, the reliability and the efficiency of the mission management.

Depending on the mission, the environment and the robotic system, we can easily adapt our methodology to broader applicative contexts than submarine robotics, as any autonomous mobile robot is subject to mission, safety and system failures. More than the technical tools (e.g. adapting the fault trees to other contexts and events), there are two important elements of our works that could be useful to extend in other robotic contexts. First, the consideration of a two levels recovery mechanism combining low level reactive management to higher decionnal techniques. Second, the formalisation of the properties that express the success of a mission. The importance and the difficulty of this step is often neglected while it is a key point to design a reliable system providing the performances desired by the users.

ACKNOWLEDGMENT

This work was partially funded through ANR (the French National Research Agency) under the "Investissements d'avenir" program (PIA) with the reference ANR-16-IDEX-0006, and by the Occitanie region.

REFERENCES

[1] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing, 1(1):11–33, January 2004.

[2] K. Bader, B. Lussier, and W. Schön. A fault tolerant architecture for data fusion: A real application of Kalman filters for mobile robot localization. Robotics and Autonomous Systems, 88:11–23, February 2017.

[3] www.lirmm.fr/bubot/

[3] D. Crestani, K. Godary-Dejean, and L. Lapierre. Enhancing fault tolerance of autonomous mobile robots. Robotics and Autonomous Systems, 68:140–155, June 2015.

[4] V. De Carolis, F. Maurelli, K. E. Brown, and D. M. Lane. Energy-aware fault-mitigation architecture for underwater vehicles. Autonomous Robots, 41(5):1083–1105, June 2017.

[5] A. Favier, A. Messioux, J. Guiochet, J.-C. Fabre, and C. Lesire. A hierarchical fault tolerant architecture for an autonomous robot. In International Conference on Dependable Systems and Networks Workshops, Valencia, Spain, June 2020.

[6] J. Guiochet, M. Machin, and H. Waeselynck. Safety-critical advanced robots: A survey. Robotics and Autonomous Systems, 94:43–52, August 2017.

[7] A. Hereau, K. Godary-Dejean, J. Guiochet, C. Robert, T. Claverie, and D. Crestani. Testing an Underwater Robot Executing Transect Missions in Mayotte. In 21st Towards Autonomous Robotic Systems Conference, Nottingham, United Kingdom, September 2020.

[8] E. E. Hurdle, L. M. Bartlett, and J. D. Andrews. System fault diagnostics using fault tree analysis. Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability, 221(1):43–55, March 2007.

[9] E. Khalastchi and M. Kalesh. A sensor-based approach for fault detection and diagnosis for robotic systems. Autonomous Robots, 42:1231–1248, 2018.

[10] P. Lambert, L. Lapierre, and D. Crestani. An Approach for Fault Tolerant and Performance Guarantee Autonomous Robotic Mission. In 2019 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), pages 87–94, Colchester, United Kingdom, July 2019. IEEE.

[11] B. Lussier, M. Gallien, J. Guiochet, F. Ingrand, M.-O. Killijian, and D. Powell. Planning with Diversified Models for Fault-Tolerant Robots. In International Conference on Automated Planning and Scheduling (ICAPS), pages 216–223, Providence, RI, United States, September 2007.

[12] L. E. Parker and B. Kannan. Adaptive Causal Models for Fault Diagnosis and Recovery in Multi-Robot Teams. In International Conference on Intelligent Robots and Systems, pages 2703–2710, Beijing, China, 2006.

[13] M. Ranjbaran and K. Khorasani. Fault recovery of an under-actuated quadrotor Aerial Vehicle. In 49th IEEE Conference on Decision and Control (CDC), pages 4385–4392, Atlanta, GA, USA, December 2010. IEEE.

[14] P. Robertson and B. C. Williams. Automatic recovery from software failure. Communications of the ACM, 49(3):41, March 2006.

[15] Benoit Ropars, Lionel Lapierre, Adrien Lasbouygues, David Andreu, and René Zapata. Redundant actuation system of an underwater vehicle. Elsevier, 151:276–289, March 2018.

[16] D. L. Rudnick, R. E. Davis, and J. T. Sherman. Spray Underwater Glider Operations. Journal of Atmospheric and Oceanic Technology, 33(6):1113–1122, June 2016.

[17] M. Saied, B. Lussier, I. Fantoni, C. Francis, H. Shraim, and G. Sanahuja. Fault diagnosis and fault-tolerant control strategy for rotor failure in an octorotor. In 2015 IEEE International Conference on Robotics and Automation (ICRA), pages 5266–5271, Seattle, WA, USA, May 2015. IEEE.

[18] M Stamatelatos, W. Vesely, J. Dugan, J. Fragola, J. Minarick, and J. Railsback. Fault Tree Handbook with Aerospace Applications. Nasa edition, 2002.

[19] D. Stavrou, D. G. Eliades, C. G. Panayiotou, and M. M. Polycarpou. Fault detection for service mobile robots using model-based method. Autonomous Robots, 40(2):383–394, February 2016.