

Using Adaptive Control Architecture to enhance Autonomous Mobile Robot Reliability

B. Durand, K. Godary-Dejean, L. Lapierre, R. Passama and D. Crestani

Laboratoire Informatique Robotique Microélectronique de Montpellier - Université Montpellier 2

{durand, godary, lapierre, passama, crestani}@lirmm.fr

Abstract— This paper details the development of an adaptive control architecture permitting to improve the reliability and robustness of autonomous mobile robot. A continuous monitoring of the significant failures allows dynamically choosing the most relevant reaction ensuring the success of the mission. This adaptive behavior is implemented into the control architecture COTAMA. The key points of the specific mechanisms added to COTAMA are addressed and explained. Experimental results are proposed to illustrate the control architecture adaptive behavior.

I. INTRODUCTION

Mobile robotic missions are becoming more complex, leading to increased robot complexity. Robots have numerous powerful sensors which provide accurate information about the robot state and its surrounding environment. They also have various locomotion and interaction capacities thanks to efficient and adapted actuators. The control architecture is the central and critical part of the robot which manages increasingly complex robot activities.

In a perfect world, a robot would succeed in completing its allocated missions whatever the encountered situation. Unfortunately, robots are hampered by numerous types of fault. The study of Carlson and al. [1] concerning unmanned ground vehicle operating in real environments demonstrates that robots are often unable to achieve their mission. The authors conclude that reliability, which is the capacity to ensure the "continuity of correct service" [2], is low due to a huge variety of failures having many origins. Hence, in the real world robots do not always succeed in dealing with some adverse situations.

To improve reliability, it is essential to design robust (capacity to deliver a suitable service in adverse situations due to uncertain system environments) and fault-tolerant (capacity to deliver a suitable service despite faults affecting system resources) robots [2]. In robotic control architectures, robustness and fault tolerance are mainly based on three principles [2]: fault or adverse situation detection, diagnosis, and recovery or treatment. The following section presents a short overview of existing works in this domain.

A. Fault detection and recovery in control architectures

Fault detection can be done using many techniques: detection uses timing checks (watchdogs), reasonableness checks (valid interval values verification), safety-bag checks (verifying commands), and model-based monitoring

and diagnosis (detection of inconsistency between the measured system's data and the corresponding model values).

The detection of adverse situation using execution control is often used in control architecture to lead to specific reaction. Similarly, replanning is one of the most common reactions proposed in robotic. For example, in the LAAS architecture, the execution controller R2C [3] detects adverse situations and erroneous requests, then the IxTeT [4] component proposes high level re-planning strategies to tolerate faults. The CIRCA architecture [5] implements execution control in order to not execute actions which could lead to identified adverse situations. It then uses actions and functionalities redundancy to recover using high level replanning. Similarly, the ORCCAD architecture [6] uses Robot-Task redundancy to switch around actions to recover from failures.

Some architectures focus on hardware faults, as the SFX-EH (Sensor Fusion Effects Exception Handling) architecture presented in [7] proposes a methodology for faults classification. Then it proposes to recover from sensing ones using hardware reconfiguration. Brandstötter et al. expose in [8] a model-based fault diagnosis and reconfiguration framework using a probabilistic hybrid automaton modeling the considered failure modes and the nominal one.

The NASA research centers propose different solutions in its various architectures. The RAX [9] architecture has a MIR (Mode Identification and Recovery) module which detects anomalous situations using model based diagnosis method and proposes to the executive module to recover from this situation. The RAX Remote Agent (RA) concept has also been employed in the IDEA architecture [10], which proposes to distribute timing checks observation on each agent. Furthermore, the CLARATy architecture [11] develops a resources manager to locally manage resources on affectation conflict and fault detection.

Finally, in the IFREMER control architecture [12], Nana proposed to use Intelligent Diagnosis System with dedicated decisional module to detect faults and evaluate their criticality. To our knowledge, the reaction based on software redundancy is initiated by the Human supervisor.

To complete this state of the art, interesting surveys with regard to detection, diagnosis and/or recovery mechanism in robotic control architectures could be found in [13], [14] and [15].

B. Conclusion

This short analysis concerning reliability and robustness in robotic control architectures highlights some limitations. Many architectures implements classic detection and recovery solutions, as timing checks, execution control and replanning. These fault-tolerant mechanisms are usually spread over the architecture and directly embedded in the different control algorithms. There is a lack of global structured approaches to efficiently integrate dependable concepts into the design of the robot control architecture.

Moreover, except the replanning solutions, most of the fault recovery solutions proposed in the literature are generally very basic (often rebooting or stopping) ones, which are not always compatible with the current robot context.

So, we propose a new global methodology, detailed in [16], allowing: identification of pertinent faults, detection and diagnosis of the identified faults and suitable reactions to these faults. This paper focuses on the decisional mechanisms based on the adaptation of the control architecture, and focuses on their integration into the COTAMA architecture. This one is presented in section II, and the proposed methodology is summarized section III. Then, after the presentation of the experiment context, section V details the decisional mechanisms implemented in COTAMA. To conclude experimental results are presented and discussed.

II. COTAMA CONTROL ARCHITECTURE

COTAMA (COntextual TAsk MAnagement) [17] is a modular control architecture. It is split into two main parts: the executive and the decisional levels (Fig. 1). The executive level involves low level robotic control. The decisional level manages the executive one according to the robot mission evolution and its environment.

A. Decisional level

This level is divided into two sublevels, the global and local supervisors. The *Global Supervisor* (GS) is in charge of the mission execution. Depending on the mission, the environment and the robot state, it defines the objectives that have to be carried out by the *Local Supervisor* (LS). This last supervisor manages a given objective, splitting it into sub-objectives which are controlled by a scheduler. A sub-objective corresponds to a set of modules that have to be executed to achieve the corresponding task. The LS then decides which sub-objective has to be executed depending on the context and the events received from the executive level.

B. Executive level

This level is composed of a scheduler and low level modules. These modules embed robotic algorithm, or implement specific functionalities (for example the WiFi communication management).

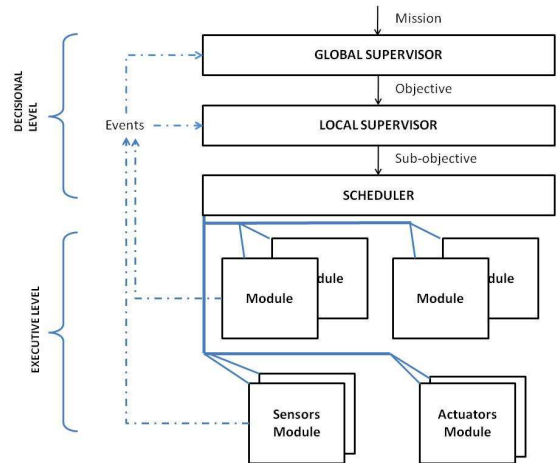


Fig. 1 COTAMA architecture

All modules are based on a specific middleware which manages real-time constraints and modules communications. These ones are made according to the consumer/producer paradigm using specific mailboxes. Using middleware allows the respect of maintainability, upgradeability and reusability concepts. Modules have a set of different types of ports:

- Data – Communication of data values between the low level modules.
- Events – Communication from the executive level to the decisional one. These communications are generated on specific contextual events (for example: the robot has reached its goal then the path following sub-objective has to be stopped).
- Requests – Three types of requests (*Activate*, *Stop* or *Kill*) are available to manage the low level modules. These ports are also used to configure Data and Events ports.
- Parameters – The requests ports can be used for an external parameterization of the module. In this case, the request message is associated with a specific setting port.

The scheduler manages the modules execution using their request port to activate or inactivate them. It also manages the real-time constraints on modules and sub-objectives execution.

III. THE PROPOSED METHODOLOGY

This section describes the methodology proposed to enhance robustness and fault tolerance in COTAMA. This methodology is divided into several steps: fault identification, fault detection and diagnosis, then reaction to the detected faults.

A. Fault identification

This step of the methodology is necessary to identify the potential faults of the system, including physical faults (effector, sensor, power, or in the control architecture), faults in environment representation and human design

faults. Human interaction faults are not yet considered. The FMECA (Failure Mode, Effects and Critical Analysis) approach [18] is used to study potential failures, and to determine the most critical ones. It begins with a functional decomposition of the system. For each identified robot's function, cause-and-effect diagrams [19] bring to light the pertinent faults to monitor. The fault severity is also analyzed according to the robot tasks and the considered autonomy level. Four severity levels are defined: weak, medium, hard and fatal, which will guide the reactions.

B. Fault detection

For the detection of each identified fault, dedicated monitoring modules named *Observers* are integrated into the control architecture. Their role is to set a flag when a module malfunctioning is detected or when an inconsistency in the robot behavior is observed. Into an *Observer*, the most adapted existing fault detection algorithm is used to detect the faults or inconsistencies occurrence.

The modular approach of the control architecture allows flexible management of these *Observers* so that the fault detection capacity will be adapted as a function of the robot mission, its environment or its available resources. *Observers* monitor the executive level of the architecture and convey information on faults to dedicated decisional modules.

C. Reaction to detected faults

The reaction to a fault uses the knowledge of the current robot available capacities to define a solution to pursue the mission. The software control architecture enables a broad range of reactions depending on the four severity levels used. For weak or medium faults the architecture adaptation concerns only the current autonomy level which can be adapted to consume fewer or different resources. For hard faults an architecture adaptation is required to switch to a different autonomy level which can involve the operator's capacities. For example information can be requested from a human operator, or the robot can adjust its autonomy mode. Finally for fatal faults the robot mission can not be pursued and must be neatly ended.

IV. EXPERIMENTAL CONTEXT

A. Mission and robot characteristics

1) Experimental Mission

The proposed robot mission is to deliver objects in the laboratory upon users' request. The delivery mission is carried out in a known environment, from which an a priori map is available. However, the environment remains dynamic since, for example, some humans can interact in the neighborhood of the robot.

The robot delivery mission involves four different objectives: waiting for a mission, driving into the laboratory, and receiving or delivering objects (interactive

tasks with users). This paper only deals with the most significant one for a mobile robot: the Drive objective. This objective can be decomposed into two sub-objectives: path planning and path following.

2) Robot characteristics

The experiments were carried out with a Pioneer-3DX from MobileRobots with two driving wheels using reversible DC motors. To perceive the environment, the robot has the following embedded sensors: two sonar arrays, two bumpers rows and a camera. An embedded laptop hosts the control architecture COTAMA, under a Linux RTAI real-time operating system, and communicates, with a serial connection, with the robot integrated microcontroller. It also communicates with a WiFi network with a remote PC which manages the overall mission and human-robot interactions.

B. Low Level Modules

In our experiment, autonomous, teleprogrammed and teleoperated autonomy modes are available. Each one requires different control laws, functional and observer modules.

1) Control Modules

Table I lists the robotic algorithms integrated into the control modules at the architecture executive level. The last column represents the name of these modules in the architecture implementation.

TABLE I
ROBOTIC ALGORITHM IN CONTROL MODULES

Robotic tasks	Algorithms	Name
Path planning	Lazy PRM	PPL
Localization	Monte Carlo Localization Robot's odometry	MCL ODO
Obstacle avoidance	Deformable Virtual Zone Safe Maneuvering Zone	DVZ SMZ
Guidance	Path following	GUI
Control	Asymptotic control with actuator velocity saturation	CTR

2) Functional Modules

Others low level modules ensuring non robotic tasks can be used: the functional modules. For example, the communications management is implemented into these modules: the LAN module to communicate with WiFi with the remote PC, the P3D module for the USB communication between the embedded laptop, where the control architecture is executed, and the robot microcontroller which collects sensors values and applies commands to motors.

The SIM module allows performing HIL simulation. Our experiment is HIL: the real odometric and bumpers sensors are used, whereas the sonars values are simulated. Then, the UST module receives the raw sonars values and produces suitable data for the robotic algorithms.

3) Observer Modules

The following Observers have been developed to monitor and diagnosis as much as possible the identified failures for the Drive function:

- Hardware or collision failures observation: The robot's micro-controller embedded hardware monitors bumper sensors, battery voltage and motor stall. Those observations are included in the P3D module.
- A Sensor Observer verifies sensors data with reasonable checking methods as valid interval verification and for sonar sensors timing checks.
- An Effectors Observer uses model based approach: a multiple-model Kalman filter [20], to diagnosis failures on motors or wheels.
- A Communication Observer retrieves data on external communication (WiFi) status (link, level, noise) and proceeds to valid interval verification on these data.
- The Scheduler module verifies modules and sub-objectives real time constraints using watchdog. It acts as an observer detecting real time faults.
- A Localization Observer monitors the localization data using valid interval verification.
- A Path Following Observer verifies that the corresponding algorithm respects uniform convergence properties. The observer monitors the coherence of the robot moving along its path, and monitors the asymptotic control convergence of the algorithm.

V. APPLICATION OF THE METHODOLOGY TO COTAMA.

This section presents the architectural modifications added to COTAMA in order to integrate the previous proposed concepts and enhance robustness and fault tolerance. These modifications are represented in grey in Fig. 2. At the executive level *Observer Modules* and a *Global Observation Module (GOM)* are integrated to detect and diagnosis fault occurrences. At the decisional level a new *Contextual Supervisor (CS)* is added. It is in charge of determining the robot context depending on the current robot state, the functioning mode and the available functionalities. The Contextual Supervisor then manages the correlation between the current sub-objective and the robot context. Moreover it chooses the most suitable reaction, sending specific events to the concerned supervisor. Finally, an *Adapter Supervisor (AS)* is also introduced. It can select the most suitable functioning mode of a given sub-objective.

A. Observation and detection

The detection-diagnosis paradigm is implemented in COTAMA with the following steps:

1) *Information collecting*: the observation information, produced by the Observer Modules, are retrieved by the Global Observation Module.

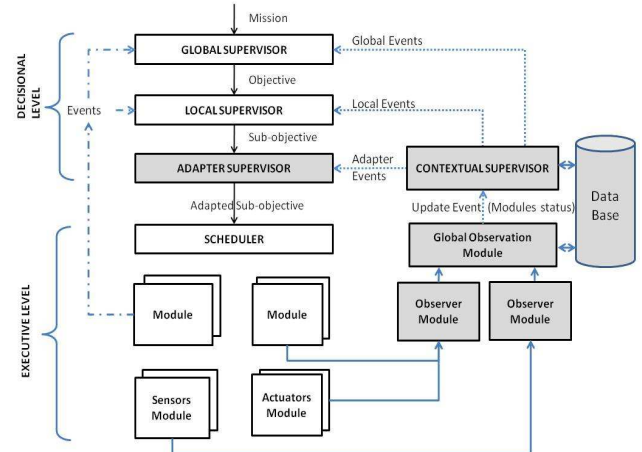


Fig. 2 Modified COTAMA architecture

2) *Diagnosis*: the GOM uses this information to diagnose the original fault and to identify the actual faulty modules (as for example a corrupted data provided by sensors can produce faulty behaviors in all the control modules). The diagnosis results depend on the detected fault but also on the current *module status*.

3) *Module status*: at this stage the GOM can estimate which functionalities, and then which modules (functional or control ones), remain active or become unavailable. The availability of the modules functionalities are represented has a *module status vector*, which is updated each time a modification of the context is detected. On such an update, an event is generated to the *Contextual Supervisor*.

B. Contextual Supervisor

Depending on the current state, the new module status, and the identified fault severity (stored in the database according to the FMECA analysis), the CS defines if the current sub-objective remains suitable to the new context. The severity of the defined context will be the base of the CS decision to alert the different supervisors using dedicated event:

- An *adapter event* is produced if the severity of the failure is weak or medium, to continue the current sub-objective with an adapted configuration of the low level modules.
- A *local event* is emitted to the local supervisor when the sub-objective cannot be pursued (hard failure).
- A *global event* is generated to the global supervisor when the objective can not be managed or if vital capacities of the robot are not available anymore (fatal failure).

C. Decisional level

Previous sections show how a fault is detected and diagnosed and how the *Contextual supervisor* propagates the decision as an event to the other supervisors according to the failure severity for the robot and its mission. Now we present the different supervisors.

1) Global supervisor

The *GS* manages the overall mission, i.e. the different objectives of the ongoing mission. In our delivery mission, it manages the driving objectives and interactions with users to receive and deliver objects. A specific security objective is also added to manage the fatal failure reported by the global events. This security objective leads the robot in a safe state (it stops), and warning signals are generated (as WiFi messages or alarms).

2) Local supervisor

The main task of the *LS* is to decompose objectives in sub-objectives, as for example the Driving objective is decomposed in Path planning and Path following sub-objectives. But the *Local Supervisor* has also to consider the different autonomy modes: when a hard failure is detected (local event reception) the *LS* switches to another autonomy mode. It manages human-robot interactions, in order to provide fault tolerance at the objective level.

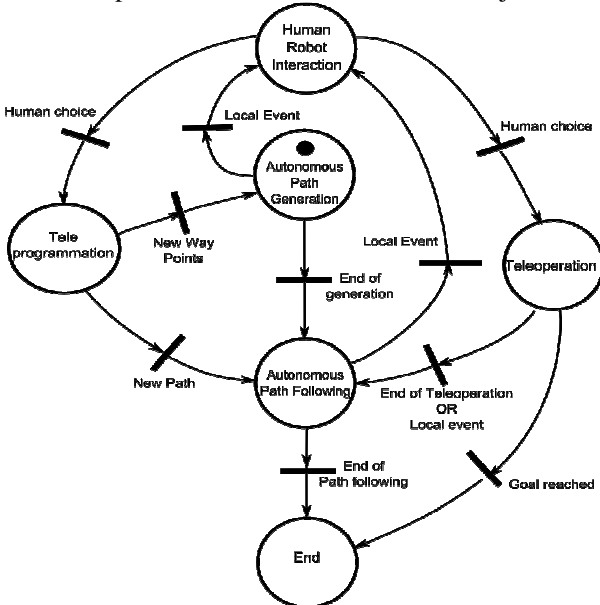


Fig. 3 Sub-objectives management for the Drive objective

Fig. 3 shows the Petri net of the *LS* for the Drive objective. For example, in the autonomous Path following sub-objective, if a *local event* is received, the current sub-objective can not be pursued and a human robot interaction mode begins. The Human operator takes hand on the decision level of the Robot. For the Drive objective, two possibilities are implemented: teleprogramming or teleoperation of the robot. In teleprogramming mode, the operator can restart the autonomous path generation with new way points, or can give a new path to be followed

3) Adapter supervisor

The *Adapter supervisor* receives adapter events on low or medium faults. Thus, for a given sub-objective, it can propose two types of adjustments: modifying parameters of some modules to modify the behavior of the corresponding embedded algorithm, or switching from the current sub-objective to a degraded version of it.

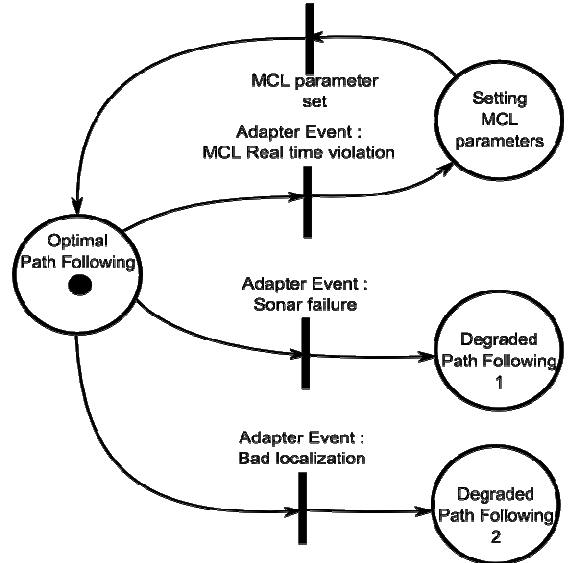


Fig. 4 Adapted Sub-objectives management for the Path following sub-objective.

For example Fig. 4 presents a simplified Petri net managing three failures in the autonomous Path following sub-objective. It shows two kinds of reactions: an adaption of the parameters of the MCL module, and a switch to a degraded Path following sub-objective.

VI. EXPERIMENTS

This experiment illustrates our methodology. It highlights the fault detections and the involved reactions in the architecture. This experiment is realized HIL (Hardware In the Loop). Some of the observed faults were deliberately created to test the detection of unusual faults (like sonar failure). The considered mission is to deliver an object from office A to office B. Fig. 5 presents the recorded experimental robot trajectory and lists the different map points where relevant events were observed. When moving, the robot speed is 0.3 m/s. The control loop of each sub-objective must be executed in less than 0.1 s. Description of the mission scenario

Point 1: The mission objective is received and the corresponding path is generated. The path following sub-objective is then engaged to reach point B. (The localization is performed by MCL. Blue line)

Point 2: A real-time fault on the MCL module is rapidly observed at the beginning of the path following task. As the complexity of this algorithm depends on its particles number, this number is decreased setting the parameters of the module to reduce its execution time.

Point 3: The robot considers that it has a localization problem and so requests human help and solution. **Point 4:** The human operator decides to observe the robot environment with the on-board camera in teleoperated mode. (The localization is performed by odometers and operator. Dashed red line)

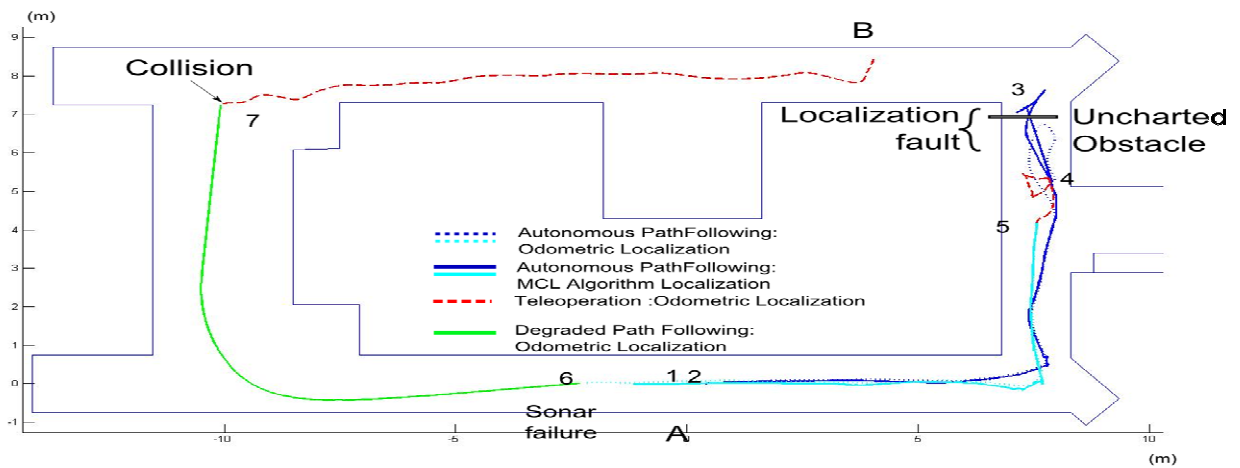


Fig. 5 Experimental mission scenario

Point 5: The operator detects that an unforeseen obstacle is present and decides to change the followed path. The robot restarts the path following optimal sub-objective with this new path. (The localization is performed by MCL. cyan line)

Point 6: A permanent fault is observed on sonars which cannot be used anymore. The degraded autonomous path following sub-objective with neither obstacle avoidance nor Monte-Carlo localization is chosen. The mission could be pursued anyway, relying on odometry for the localization, and decreasing the robot velocity in order to decrease the eventual damages caused by the collision with an obstacle. (The localization is performed by odometers and. Green line)

Point 7: The robot bumps into an obstacle. So the human operator decides to complete the drive objective using degraded teleoperation (without obstacle avoidance). (The localization is performed by odometers and operator. Dashed red line)

This experimental mission shows that the robot is able to detect the different faults that occurred, and to react depending on the current context. Sometimes the robot opts to ask the human operator for help, and sometimes it solves the problem on its own. Finally, the mission is achieved despite fault occurrences.

A. Interesting Point

This section focuses on specific detection, diagnosis and reaction of the experiments.

1) Real time failure detection.

The scheduler manages the real time execution of modules. It uses watchdog techniques to detect real time constraints violation. To tolerate transient real time violation, the scheduler gives extra-time to the faulty module in order to let it finish its job. But if a module has several consecutive violations it is considered to be faulty in a persistent way. An event is then sent to notify a real time failure.

At point 1 in fig. 5 the scheduler detects a persistent real-time failure in the Localization algorithm of MCL. As only one module is faulty, two solutions are available: release the module temporal constraint, or reduce the execution time of

this module. In this case the Contextual supervisor chooses the second one and sends an event to the Adapter supervisor to set the MCL particles number.

2) Localization failure.

It exists numerous ways to detect or observe that "the robot is lost". This question is not so easy to resolve as it is very hard to diagnose the real origin of a localization loss. For example, the loss can be due to noisy odometric sensors or due to the localization algorithm. But it can also be triggered by a default (uncharted obstacle) in the a priori map of the environment, inducing a non-suited reference for the localization algorithm, the particle filter in our case.

In our experiment fig. 6, two Observers are used to detect a localization problem. The first one analyzes the distance between the localization values from the odometric data and the MCL algorithm. It generates a flag of suspicious fault when the distance between these values grows up rapidly. The second Observer module detects that the robot moves back a too long time on its path. This duration has been experimentally tuned. This situation occurs when the robot cannot follow its path, due to an uncharted obstacle. The obstacle avoidance algorithm drives the robot to move around the obstacle and, if there is no solution to pursue its nominal path, induces the system to go back

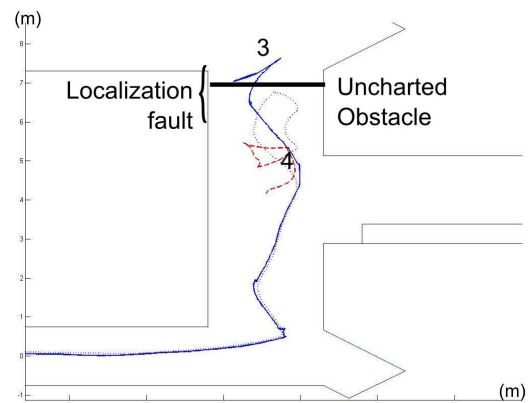


Fig. 6 Focus on the detection of localization failure
Thus, a localization problem is encountered within point 3 in Fig. 6. The two previous observers detect a problem but

unique diagnosis cannot be realized. So, the Contextual Supervisor decides to ask operator's help to diagnosis the original faults.

The operator with the onboard camera detects an uncharted obstacle. The diagnosis is relevant, the obstacle forced the robot to go backwards and the difference between the map and the real environment corrupts the MCL algorithm.

3) Sonar failure

When the real distance to obstacle can not be retrieve, or when the sonar is broken, the microcontroller gives the maximum value of sonar (5m). So, when this maximum value is observed for a long time while the robot is moving, this denotes that the concerned sensor is broken.

In our experiment, one faulty sonar leads to consider all the sonars arrays as faulty from point 6 in Fig. 6. This could be refined considering that the robot could use only a part of active sonars. So an adapter event is sent to the Adapter supervisor which chooses the most degraded path following algorithm, without obstacle avoidance and MCL localization.

Before the sonar failure detection, the robot executes the optimal autonomous path following sub-objective. In this experiment, the optimal strategy in the autonomous mode for the Path Following sub-objective is composed of the Monte-Carlo localization, the SMZ avoiding obstacle added to guidance and control. Thus, considering neither the Observer modules nor the GOM, the control loop of this sub-objective implies the following low level modules:

P3D-SIM-UST-MCL-NAV-SMZ-LAN

If the sonars are faulty, all the modules needing the proximity values become unavailable: UST, MCL, SMZ, and of course SIM which is not useful anymore.

The Contextual supervisor then decides to send a local event to the Adapter supervisor to switch in a degraded path following mode, using only the odometric estimation of the position. In the degraded sub-objective only the following modules are executed:

P3D-NAV-GUI-LAN

The GUI module is used in the degraded sub-objective to execute the guidance functionality. Indeed, the SMZ module contains both the obstacle avoidance and the guidance functionalities, but only the obstacle avoidance one is not available anymore.

4) Collision

The detection of collision (Point 7 in Fig. 6) is made using bumper sensors by the robot microcontroller. Since the actual degraded navigation is of the dead reckoning type, a drift in the estimation of the position is expected. The occurrence of a collision implies that this global navigation is no more suitable. As the robot is ever in a degraded sub-objective (without sonar), this failure leads the *Contextual*

supervisor to create a local event. Human help is needed to decide what to do.

VII. CONCLUSION

Recent studies demonstrate the low reliability of autonomous mobile robots. To improve this important weakness, robot's control architecture must integrate fault tolerance capacities. Based on a global approach analyzing the robot system to detect potential failures, this paper proposes to include in the control architecture dedicated Observer Modules monitoring the relevant ones. Depending on their severity the current functioning mode is adapted to face to the failure occurrence and to pursue the mission. This adaptation may involve limited Human-robot interaction or may need to switch from autonomous mode to teleprogrammed or teleoperated ones.

The proposed experiment realized "Hardware in the loop" will be soon realized in our laboratory. In the future the global control architecture and the remote PC functions will have to be enriched to address more complex missions and situations. A series of tests will allow an estimation of the efficiency of the proposed approach and mechanisms and consequently the impact on robot's reliability.

REFERENCE

- [1] J. Carlson and R. Murphy, "How UGVs physically fail in the field," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 423–437, June 2005.
- [2] 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*, vol. 1, no. 1, pp. 11–33, January-March 2004.
- [3] Py, F. & Ingrand, F. "Real-Time Execution Control for Autonomous Systems", *Proc. of the 2nd European Congress ERTS, Embedded Real Time Software*, 2004.
- [4] S. Lemai-Chenevier, "TXET-EXEC: planning, plan repair and execution control with time and resource management", PhD thesis, LAAS-CNRS, 2004. [in French].
- [5] R. P. Goldman, D. J. Musliner, and M. J. Pelican, "Using model-checking to plan hard real-time controllers," in *AIPS Workshop on Model-Theoretic Approaches to Planning*, April 2000.
- [6] J. Borrelly, E. Coste-Manière, B. Espiau, K. Kapellos, R. Pissard-Gibollet, D.Simon and N. Turro, "The Orcad Architecture", *International Journal of Robotics Research, special issue on Integrated Architectures for Robot Control and Programming*, vol. 18, pp 338–359, 1998.
- [7] R. R. Murphy and D. Hershberger, "Classifying and recovering from sensing failures in autonomous mobile robots," in *Proc. of the Nat. Conf. on Artificial Intelligence*, pp. 922–929, 1996.
- [8] M. Brandstötter, M.W. Hofbaur, G. Steinbauer, and F. Wotawa, "Model based fault diagnosis and reconfiguration of robot drives," in *proc. of the Int. Conf. of Intelligent Robots and Systems*, pp. 1203–1209, 2007.
- [9] N. Muscettola, "Remote Agent: To Boldly Go Where No AI System Has Gone Before", *Artificial Intelligence*, vol.103 no.(1-2), pp 5-48, 1998.
- [10] N. Muscettola, G. A. Dorais , C. Fry , R. Levinson , C. Plaunt, "IDEA: Planning at the Core of Autonomous Reactive Agents", in *Proc. of the 3rd Int. NASA Workshop on Planning and Scheduling for Space*, 2002.

- [11] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das, "The CLARATy architecture for robotic autonomy", *IEEE Aerospace Conference, Aerospace Conference*, vol. 1, pp. 121-132, 2001.
- [12] L. T. Nana, "Investigating software dependability mechanisms for robotics applications", *The IPSI BgD Transactions on Internet Research*, vol. 3, no 1, pp. 50-55, 2007.
- [13] Z. Duan, Z. Cai, and J. Yu, "Fault diagnosis and fault tolerant control for wheeled mobile robots under unknown environments: A survey," in *Proc. of the Int. Conf. on Robotics and Automation*, pp. 3439-3444, 2005.
- [14] B. Lussier, "Fault Tolerance in autonomous systems", Phd Thesis, LAAS-CNRS, 2007.[in French].
- [15] L.T. Nana, L. Marcé, J. Opderbecke, M. Perrier and V. Rigaud, "Investigation of safety mechanisms for oceanographic AUV missions programming", *Proc. of the IEEE OCEAN 05 Europe conference*, 2005.
- [16] B. Durand, K. Godary-Dejean, L. Lapierre and D. Crestani,"Global methodology in control architecture to improve mobile robot reliability", in *Proc. of the Int. Conf. of Intelligent Robots and Systems*, 2010. (to be published)
- [17] A. El Jalaoui, D. Andreu, B. Jouvencel,"Contextual Management of Tasks and Instrumentation within an Auv control software architecture.", in *Proc. of the Int. Conf. on Intelligent Robots and Systems*, Beijing, China, October 9-15, 2006.
- [18] *Guide to failure modes, effects and criticality analysis (FMEA and FMECA)*, British Standard Std. 5760-5, 1991.
- [19] K. Ishikawa, *What is Total Quality Control? The japanese Way*, Prentice-Hall, 1985.
- [20] S. I. Roumeliotis, G. S. Sukhatme, and G. A. Bekey, "Fault detection and identification in a mobile robot using multiple model estimation," in *Proc. of the Int. Conf. on Robotics and Automation*, vol. 3, pp. 2223-2228, May 1998.