Reliability improvement in control architecture for mobile robots: implementation using COTAMA

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 component oriented 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

A. CONTEXT

Mobile robotic missions are becoming increasingly 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 interesting 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].

B. State of the art

The first paragraph in this section is related to fault identification and localization with fault detection and diagnosis mechanisms. The second concerns fault recovery, and finally fault tolerance and robustness in control architectures is tackled.
1) Fault detection and diagnosis methods: Several techniques can be used for fault detection [3]: timing checks (watchdogs), reasonableness checks (valid interval value verification), safety-bag checks (command verification), and model-based monitoring and diagnosis (detection of any inconsistency between measured system data and corresponding model values) [4]. Model-based fault detection is widely used to highlight hardware faults [5], where the authors used multiple model based methods (bank of Kalman filters) to detect sensor and mechanical faults. In [6], the authors use a particle filter based state identification method to detect hardware faults such as battery voltage drops or motor encoder decoupling. Other approaches exist for example, in [7], a model based diagnosis approach is described using a probabilistic hybrid automaton to model the considered failure modes and the nominal mode, or in [8], who propose a ”generate and test approach” to check all possible sensing failure origins of current symptoms.
However, multiple model oriented approaches may be hampered by state space handling problems when the number of treated faults increases, especially in an embedded and real-time context. However in robotics, faults are not always hardware related, they can also be associated with the software. Few studies have assessed the detection of complex robot control software faults during runtime. In [9], the author develops a model based approach using Petri nets to monitor component-based systems and detect erroneous components. But there is no evidence that this approach can handle real-time constraints. Weber and Wotawa [10] describe a model-based diagnosis paradigm to detect and localize runtime control software faults. After a fault detection, the failed component is identified using a failure dependency graph. To the best of our knowledge, this proposition has not been tested in the field.
2) Fault recovery: In robotics, recovery solutions after fault detection have been proposed in some previous studies.
For example, concerning hardware faults in [7], the authors determined whether a robot should reconfigure, use a degraded mode or stop on the basis of qualitative constraints on robot components and diagnostic results. In [8], they use exception handling to recover from a

Manuscript received on April 10 2010

detected failure. Concerning software faults, the usual approach is to stop and restart either the robot, either malfunctioning components and its dependent services [10].

3) Fault tolerant control architectures: This section presents mechanisms currently used in control architectures to prevent, diagnose and/or recover from faults. Timing checks, reasonableness and safety-bag checks are usually implemented in robotic control architectures. They are usually spread over the architecture and directly embedded in the different control algorithms. In their survey [11], Duan et al. present different fault tolerant control architectures. The authors principally focused on hardware fault detection and mainly proposed software redundancy to tolerate faults. In the LAAS architecture, the R2C component [12] is in charge of propriety and assertion tests using safety bag checks. Both LAAS and CIRCA [13] architectures detect adverse situations using execution control and propose high level replanning to tolerate faults. In [14], in the IFREMER control architecture, Nana proposed to use an Intelligent Diagnosis System with a dedicated decisional module to detect and react to faults.

C. *PROPOSITION*

This short analysis concerning reliability and robustness in robotic control architectures highlights some limitations. Concerning fault detection and diagnosis, the methods mainly deal with hardware faults. They do not use methods to identify relevant faults which are essential to detect.

Concerning fault recovery, most fault recovery solutions proposed in the literature are generally very basic (often rebooting or stopping). Moreover, recovery mechanisms often only concern a limited number of faults. Finally, there is clearly no link between the identification of pertinent faults, and the fault detection methods and fault recovery works. There is also a lack of global structured approaches to efficiently integrate dependable concepts into the design of the robot control architecture.

In this paper, we first present our control architecture, and then the approach used to improve its reliability. After the presentation of the experiment context we expose the modification made on the architecture to host the approach principles. To conclude we present experiment results and discuss about new strategies presented here.

II. COTAMA CONTROL ARCHITECTURE

COTAMA (COntextual TAsk MAnagement) [15] is a modular component oriented 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.

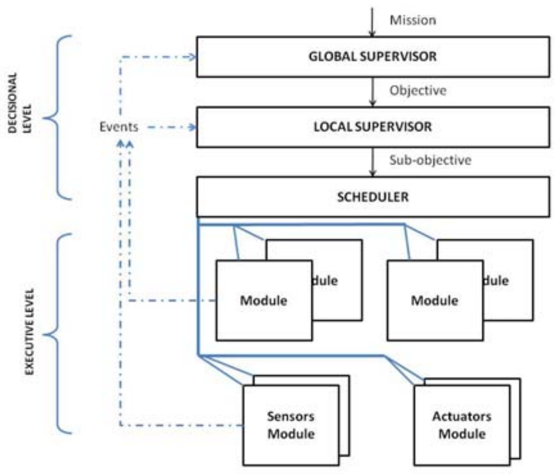

Fig. 1 COTAMA architecture

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 can be control ones, which implement a robotic algorithm, or modules which implement a specific functionality (for example the WiFi communication management). 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 a 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 current 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 setting of the module parameters. In this case, the request message is associated with a specific setting port.

The scheduler manages the modules execution using their requests 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 reliability in COTAMA. This methodology is divided into several steps: fault identification, fault detection and diagnosis, and 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), environment faults and human design faults. Human interaction faults are not yet considered. The FMECA (Failure Mode, Effects and Critical Analysis) approach [16] 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 [17] bring to light the pertinent failures 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 suspected. 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 on the still operational robot functionalities to define a solution to manage the encountered problem and pursue the mission with the current robot available capacities. 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 the 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 pursue 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 so a laboratory 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 by 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, 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.

Robotic tasks	Algorithms	Name
Path planning	Lazy PRM [18]	PPL
Localization	Monte Carlo Localization [19] Robot's odometry	MCL ODO
Obstacle avoidance	Deformable Virtual Zone [18] Safe Maneuvering Zone [20]	DVZ SMZ
Guidance	Path following [21]	GUI
Control	Asymptotic control with actuator velocity saturation [21]	CTR

Tab. 1: Robotic algorithms in control modules

2) *Functional Modules*

Moreover, 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 in 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 effectors.

A module called SIM simulates the robot sensors in order to make some tests without the robot. It could be parameterized to simulate one or all of the odometric, bumper and sonar sensors. Our experiment is HIL: the real odometric and bumpers sensors are used, whereas the sonars values are simulated. Then, the UST module receives the gross 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. So 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 employs model based approach: a multiple-model Kalman filter [22], 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. So 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 some 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 principles to integrate the previous proposed concepts to enhance robustness and fault tolerance. These modifications are represented in grey in Fig. 3. 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 to determine 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 adapt the functioning mode of a given sub-objective.

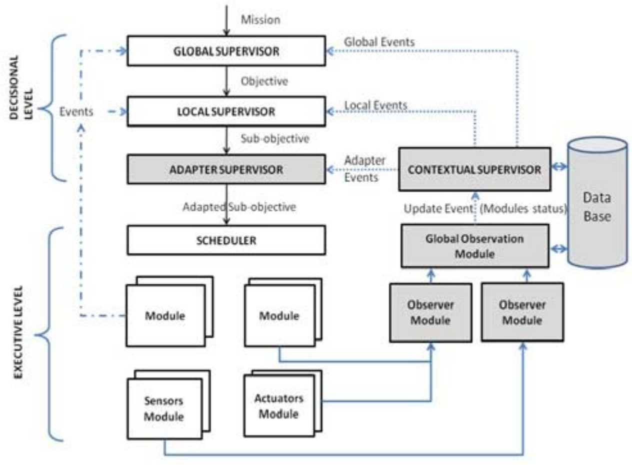

Fig. 3 Modified COTAMA architecture

A. *Observation and detection*

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

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

2) *Diagnosis:* The GOM used these information to diagnosis 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 step 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 available in this 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.

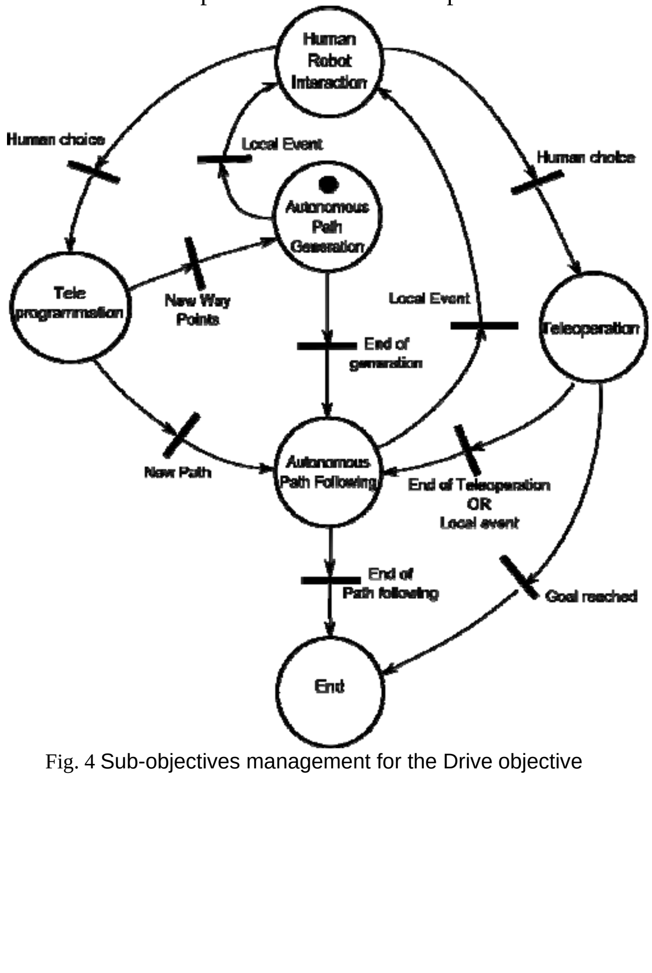

Fig. 4 Sub-objectives management for the Drive objective

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 Drive objective and interactions with users to receive and give 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 Drive 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 switch to another autonomy mode. It manages human-robot interactions, in order to provide fault tolerance at the objective level.

Fig. 4 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 then takes the decision that the Robot does not succeed to find. For the Drive objective, two possibilities are implemented: teleprogrammation or teleoperation of the robot. In teleprogrammation 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 if possible from the optimal sub-objective (the most efficient) to a degraded one.

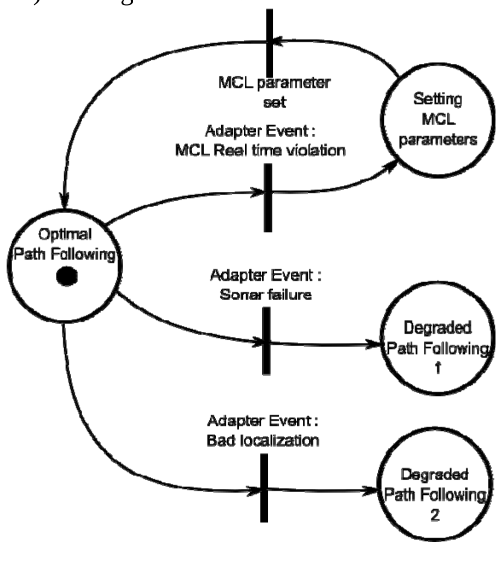
MCL parameter set
Setting MCL parameters
Adapter Event : MCL Real time violation
Optimal Path Following
Adapter Event : Sonar failure
Degraded Path Following 1
Adapter Event : Bad localization
Degraded Path Following 2

Fig. 5 Adapted Sub-objectives management for the Path following sub-objective.

For example Fig. 5 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 in action. 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. 6 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.

A. Description of the mission scenario

Point 1: The mission objective is received and the corresponding path is generated. The path following optimal sub-objective is then launched to reach point B.
Point 2: A real-time fault on the MCL module is rapidly observed at the beginning of the path following task. The particles number of the algorithm is decreased setting the parameters of the module.
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.
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.
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 by decreasing the robot speed.
Point 7: The robot bumps something. So the human operator decides to complete the Drive objective using degraded teleoperation (without obstacle avoidance).

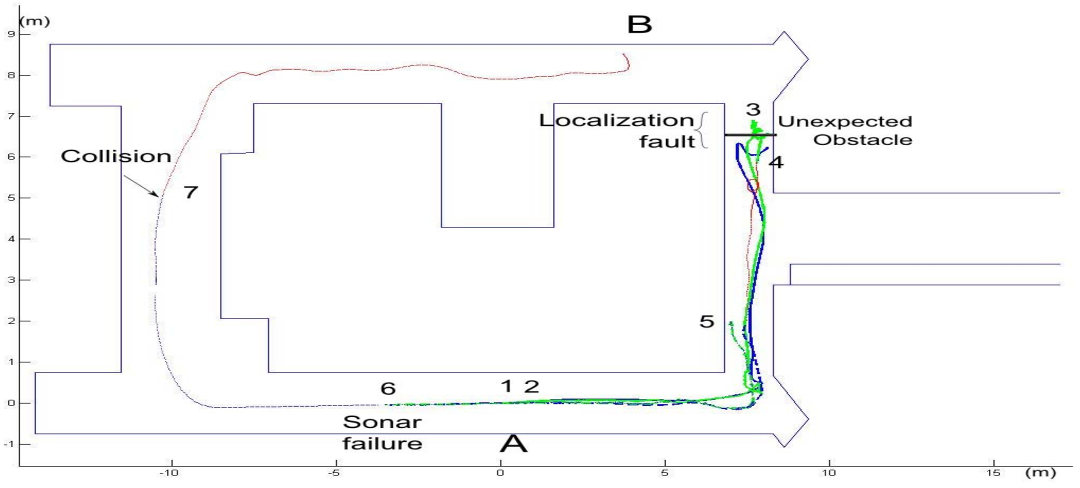

Fig. 6 Experimental mission scenario

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.

B. Interesting Point

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

1) Detection of real time failure.

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 thus sent to notify a real time failure.
At point 1 in fig. 6 the scheduler detects a persistent real-time failure in the Localization algorithm of MCL. As only one module is faulty, two solutions are available: enlarge real time constraints 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 diagnosis 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 derive from the environment map if it does not represent the actual environment, for example when there is an uncharted obstacle.

In our experiment, two Observers are employed 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.

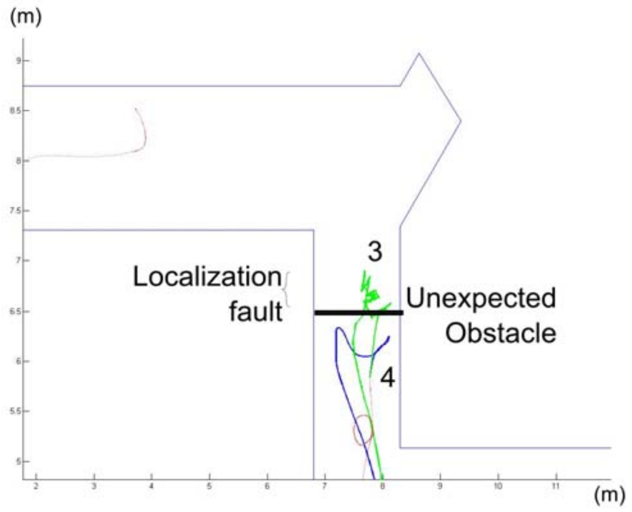

Fig. 7 Detection of localization fault

Thus, a localization problem is encountered within point 3 in Fig. 7. The two previous observers detect a problem but precise 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 corrupt 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 3 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 sonar 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 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 4 in Fig. 6) is made using bumper sensors by the robot microcontroller. The odometric localization seems not to be enough sufficient. 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 and robustness 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 COTAMA control architecture dedicated mechanisms for fault detection and diagnosis, and for adapted reactions. Thus, specific Observer Modules are added to monitor the relevant faults. Depending on their severity and on the robot context, the current functioning mode is adapted to face to the failure occurrence and to pursue the mission. This adaptation may be internal to the robot which chooses autonomously the suitable reaction, or may involve Human-robot interactions switching to teleprogrammed or teleoperated functioning modes.

This article describes a case study experiment and the obtain results to show the feasibility and the efficiency of our methodology implemented in COTAMA. 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 more complete series of tests will allow an estimation of the efficiency of the proposed approach and mechanisms and consequently of the impact on robot's reliability. They are necessary to improve our approach, to make it useful on different missions and more complex environments.

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] B. Lussier, A. Lampe, R. Chatila, J. Guiochet, F. Ingrand, M.-O. Killijian, and D. Powell, “Fault tolerance in autonomous systems: How and how much?” in proc. of the 4th IARP EURON Joint Workshop on Technical Challenges for Dependable Robots in Human Environments. Nagoya, Japan: IEEE/RAS, June 2005.

[4] M. Hofbaur, J. Köb, G. Steinbauer, and F. Wotawa, “Improvingrobustness of mobile robots using model-based reasoning,” J. Intell. Robotics Syst., vol. 48, no. 1, pp. 37–54, 2007.

[5] P. Goel, G. Dedeoglu, S. I. Roumeliotis, and G. S. Sukhatme, “Fault detection and identifcation in a mobile robot using multiple model estimation and neural network,” in proc. of the IEEE International Conference on Robotics and Automation, vol. 3, 2000, pp. 2302–2309.

[6] C. Valdiviezo and A. Capriano, “Fault detection and isolation system design for omnidirectional soccer-playing robots,” in proc. of the IEEE Conference on Computer Aided Control Systems Design”, 2006, pp.2641–2646.

[7] M. Brandstötter, M. W. Hofbaur, G. Steinbauer, and F. Wotawa, “Model-based fault diagnosis and reconfiguration of robot drives,” in proc. of Intelligent Robots and Systems, 2007, pp. 1203–1209.

[8] R. R. Murphy and D. Hershberger, “Handling sensing failures in autonomous mobile robots,” The International Journal of Robotics Research, vol. 18, no. 4, pp. 382–400, April 1999.

[9] I. Grosclaude, “Model-based monitoring of component-based software systems,” in proc. of the 15th International Workshop on Principles of Diagnosis, 2004, pp. 155–160.

[10] J. Weber and F. Wotawa, “Diagnosis and repair of dependent failures in the control system of a mobile autonomous robot,” Applied Intelligence, vol. 29, pp. 1–18, 2008.

[11] 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 IEEE International Conference on Robotics and Automation, 2005, pp. 3439–3444.

[12] F. Py and F. Ingrand, “Real-time execution control for autonomous systems.” in proc. of the 2nd European Congress ERTS, Embedded Real Time Software, Toulouse, France, January 21-23 2004.

[13] 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.

[14] L. T. Nana, “Investigating software dependability mechanisms for robotics applications,” The IPSI BgD Transactions on Internet Research N1, vol. 3, pp. 50–55, Jan 2007.

[15] A. El Jalaoui, D. Andreu, B. Jouvencel, "Contextual Management of Tasks and Instrumentation within an AUV control software architecture", In The 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'06), Beijing, China, October 9-15, 2006

[16] Guide to failure modes, effects and criticality analysis (FMEA and FMECA), British Standard Std. 5760-5, 1991.

[17] K. Ishikawa, What is Total Quality Control? The japanese Way. Prentice-Hall, 1985.

[18] A. Sanchez, R. Cuautle, R. Zapata, and M. Osorio, Advances in Artificial Intelligence, Springer Berlin, October 2006, vol. 4140/2006.chapter. “A Reactive Lazy PRM Approach for Non holonomic Motion Planning”, pp. 542–551.

[19] L. Zhang, R. Zapata, "A Three-step Localization Method for Mobile Robots", In Proc. of International Conference on Automation, Robotics and Control Systems (ARCS-09), Orlando, Florida, USA, July 13-16, 2009, ISRST, p50-56.

[20] L. Lapierre, R. Zapata and M. Bibuli, "Guidance of a flotilla of wheeled robots: a practical solution", accepted at the 7th IFAC Symposium on Intelligent Autonomous Vehicles conference (IAV 2010), 6-8 September 2010, Lecce Italy.

[21] L. Lapierre and G. Indiveri, “Non-Singular Path-Following, Control of Wheeled Robots with velocity actuator saturations,” in Proc. of the 6th IFAC Symposium on Intelligent Autonomous Vehicles, Toulouse, France, September 2007.

[22] 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 International Conference on Robotics and Automation, vol. 3, May 1998, pp. 2223 – 2228.