

# Fault Tolerance Enhancement using Autonomy Adaptation for Autonomous Mobile Robots

B. Durand, K. Godary-Dejean, L. Lapierre, R. Passama and D. Crestani  
Laboratoire Informatique Robotique Microélectronique de Montpellier  
Université Montpellier 2 / C.N.R.S.  
{durand, godary, lapierre, passama, crestani}@lirmm.fr

*Abstract* — This paper presents how autonomy adaptation can be useful to enhance the fault tolerance of autonomous mobile robots. For that, we proposed a global and structured methodology which allows integrating specific fault tolerant mechanisms into an adaptive control architecture. When a problem is detected, the autonomous behavior of the robot is automatically adapted to overcome it. The human operator can punctually or definitively be inserted in the control loop to replace the damaged functionalities and to ensure the success of the mission. Experimental results on a mobile robot are proposed to illustrate the autonomy adaptation.

## I. INTRODUCTION

Nowadays there is a huge gap between the expected dependability of autonomous robots and the observed one during real world mission. Carlson and Murphy demonstrate in [1] that an autonomous system must face to many hardware, software or human faults, and that the mission must often be aborted. So, dependability principles [2], which are well defined and carried out in critical systems, must be applied to autonomous robotic systems.

During design or operational phases, systems are necessarily affected by internal (dormant) or external (coming from the environment) faults. A fault is a malicious entity which, once activated, is propagated through the system until a failure occurs in the system services. So the dependability of a system can be defined as "the ability to avoid service failure that are more frequent and more severe than is acceptable" [2]. Consequently two main means exist to enhance the dependability of a system: to suppress the faults or to try to deliver a correct service in spite of their occurrences.

### A. Dependability approaches for autonomic robots

1) *Fault forecasting*: Before avoiding a fault or correcting its effect an important step would be to evaluate which ones are the most harmful for the system. However, practically, few results are available for autonomous robot systems, except [1] which shows qualitative experimental results identifying, classifying and ranking the observed failure modes. The FMECA (Failure Mode Effects and Critical Analysis) [3] methodology would be interesting to determine the most relevant failures.

2) *Fault avoidance*: To minimize the amount of faults, faults prevention and removal techniques can be used.

Fault prevention mainly depends on software development methods ensuring easy maintainability, analyzability and testability. For example, modularity using software components is observed in robot control architectures such as LAAS [4], IDEA [5], CLARATy [6] or COTAMA [7].

Fault removal concerns tests and formal validation. Simulation and intensive testing can be deployed like for the RAX architecture [8] to point out system's faults. However, for autonomous robots, the test generally remains incomplete and the conclusion cannot be considered as a proof. Formal validation approaches are based on properties verification. The validation capacity of synchronous languages as Esterel has been used in different control architectures in [9] or ORCCAD [10], and Model-Based Programming Language in [11]. In [12] the system's description is translated into a formal representation for symbolic model checking. In the LAAS architecture, the formal validation of safety properties using model checking has been developed for the execution controller [13].

Unfortunately, all the faults could not be avoided, as for example sensors or effectors breakdown. Solutions must then be proposed and implemented as fault tolerance and robustness approaches.

3) *Fault tolerance and Robustness*: Depending on the location of the problem, 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) issues can be distinguished for autonomous robots [14]. In robotic control architectures, robustness and fault tolerance are mainly based on fault (or adverse situation) detection, diagnosis, and recovery (or treatment).

Fault detection can be done using timing checks, reasonableness checks, safety-bag checks, or model-based monitoring and diagnosis [14]. Some architecture focus on hardware faults, as the SFX-EH [15] which proposes to recover from sensing faults using hardware reconfiguration. Brandstötter *et al.* expose in [16] a model-based fault diagnosis and reconfiguration framework using a probabilistic hybrid automaton. IDEA [5] distributes timing checks observation over each agent. CLARATy [6] develops a resources manager to locally manage resources on affectation conflict and fault detection.

Adverse situation detection commonly uses execution control. The CIRCA architecture [17] implements execution control to trigger high level re-planning. Like in ORCCAD [10] redundancy permits to recover from failures. In the LAAS architecture, the execution controller R2C [4] detects adverse situations and erroneous requests, then the IxTeT [18] component proposes high level re-planning or plan repair strategies to tolerate faults.

### B. Objective of the paper

Evidently fault forecasting, fault prevention and fault removal are interesting and sometimes powerful means to limit the amount of faults. However since it is impossible to suppress all the possible internal and external faults it seems evident that fault tolerance and robustness will be essential to provide dependable robots. Adaptive control architecture is an interesting way to reach this objective. Unfortunately, autonomous robots are not able to handle all the system resources malfunctioning or adverse situations. Then, the operator help seems to be for still a long time, the unique possibility to deliver an acceptable service.

In spite of numerous works concerning dependability concepts for autonomous robots, there is a lack of global and structured approach including all these aspects of fault tolerance. We have proposed in [19] a methodology aiming to address, in a flexible and generic way, all these aspects. It is based on four successive steps allowing the enhancement of robots reliability: fault identification, fault detection and diagnosis, and fault recovery.

This paper focuses on the recovery mechanisms developed and implemented into the COTAMA (Contextual Task Management) architecture [7]. It shows how control architectures may use relevant autonomy level adaptation to ensure the success of a whole mission, even in presence of faults. This article presents the specific decisional mechanisms used to manage the autonomy adaptation.

Next section presents the main autonomy sharing principles. Section III details the fault tolerant mechanisms used in COTAMA architecture to support relevant autonomy adaptation. Before concluding, section IV explains some experiment results.

## II. AUTONOMY SHARING

Autonomy is one of the main objectives to achieve in mobile robotic. The definition of Autonomy is not unique as it depends on the point of view. However for robotics and multi-agent systems the following one seems to be acceptable: “A system (or agent) is autonomous if, alone, it is able to define and perform its action”. This definition remains wide, and needs to be refined, from the point of view of applications focusing on autonomy evaluation.

The ALFUS group proposes [20] a three-axis based evaluation: the operator independency, the mission complexity and the environmental difficulty. In [21], Clough *et al.* propose to evaluate autonomy using an Autonomous

Control Level (ACL) chart based on the degree of interaction between the robot and the human operator. Unfortunately these works remain limited due to the concepts complexity.

The most important result is that the autonomy can be leveled in function of the Human Robot Interaction (HRI). A well-known scale has been presented in [22], by Sheridan *et al.*, proposing 10 levels of autonomy. Since then, various authors have proposed variations based on this scale. For example, in [23] the authors propose a review of Human-Robot Interaction providing a synthetic point of view of the numerous fields related to HRI. It proposes a variation of the Sheridan's scale focusing on "mixed initiative interaction", which is defined as a “flexible interaction strategy in which each agent (human and [robot]) contributes what it is best suited at the most appropriate time”. This new scale is shown fig. 1. On the direct control side, the operator as to do the entire job; the user interface must be enhanced in order to reduce the operator workload. On the other extreme, peer-to-peer collaboration requires full autonomous robots which have high appropriate skills in order to interact with the operator.

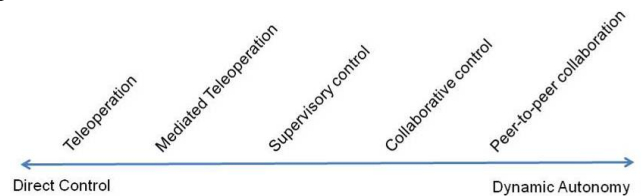


Fig. 1 Levels of autonomy proposed in [23]

To perform dynamic autonomy using mixed initiative paradigm, two questions remain central to adapt the level of autonomy. Which one holds the adaptivity decision? How the autonomy levels can be adapted during the mission?

Some works try to answer to these questions. Opermann *et al.* analyze in [24], the spectrum of adaptivity from adaptive (where the system initiates adaptivity) to adaptable (where the user initiates it). In [25], the INEEL architecture has been developed for Urban Search and Rescue context. The robot has several functioning modes and autonomous behaviour modes. A specific system suggests to the user to choose the most appropriate mode depending on the robot state and the supposed operator problem. Then the adaptivity is suggested to the user, which could choose to adapt or not.

The following section details the COTAMA architecture initial concepts and the fault tolerant mechanisms we add to enhance robot reliability.

## III. COTAMA CONTROL ARCHITECTURE

COTAMA is a modular control architecture initially exposed in [7]. To enhance reliability and robustness of robotic systems, we propose to integrate in this architecture fault tolerant mechanisms for fault detection and fault recovery. These mechanisms are included in a global methodology detailed in [19].

The architecture decomposition is presented Fig. 2. COTAMA is split into two main parts: the executive and the decisional levels. The executive level involves low level robotic control, as well as specific modules dedicated to fault detection and diagnosis. The decisional level manages the executive one according to the robot mission evolution and its environment, and implements fault recovery mechanisms.

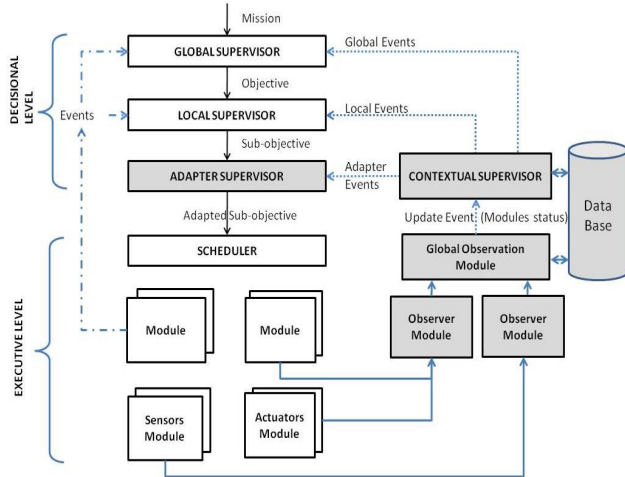


Fig. 2 COTAMA architecture

### A. Executive level

This level is composed of a scheduler and low level modules. There is three types of low level modules: control modules, which embed robotic algorithms (for example Monte-Carlo Localization MCL); functional modules which implement specific functionalities (for example the WiFi communication management); and the specific Observer modules which implement fault detection. All modules are based on a specific middleware which manages real-time constraints and modules communications. Using middleware allows the respect of maintainability, upgradeability and reusability concepts. The modularity concept is also a useful one for fault prevention, insuring independent design and test of the robotic algorithms.

The Observer modules produce observation information which are retrieved by the *Global Observation Module* (GOM). This specific module uses these 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 *Modules status*. Indeed, 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 as a *Modules status vector*, which is updated each time a modification of the context is detected.

Finally, the *Scheduler* manages the modules, activating or inactivating them using specific events. It also manages the

real-time constraints on modules and sub-objectives execution. In one hand it allocates predefined execution time to each module and verifies using watchdogs that these constraints are not violated. If a module is too often late, the scheduler detects a real time fault for this module. In other hand, it verifies that all the modules of the current sub-objective could be executed within a predefined duration. If not, the scheduler detects a real time problem on this sub-objective execution. In both cases, it reports the problem to the decisional level.

An important characteristic of the COTAMA architecture is the ability to dynamically reconfigure modules parameters, interconnections and scheduling, allowing so to adapt control algorithms related observers to the current robot and mission states. Adaptation decision is taken by the decisional level.

### B. Decisional level

This level was initially divided into two sublevels, the *Global and Local Supervisors*. The *Contextual Supervisor*, as well as the *Adapter Supervisor*, have been added to implement fault recovery mechanisms. All the supervisors react on events received from their superior supervisor or from the Contextual one.

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). The GS also implements a specific security objective which leads the robot in a safe state in case of fatal failures.

The main task of the *Local Supervisor* (LS) is to manage 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 has also to consider the different autonomy modes: it decides which sub-objective has to be executed depending on the context. It manages human-robot interactions, in order to provide fault tolerance at the objective level.

The third supervisor, the *Adapter Supervisor* (AS), manages the different functioning modes for a given sub-objective and a given autonomy level. 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. For example for the path following sub-objective in autonomous level, it could define optimal path following or degraded (for example without obstacle avoidance) ones.

Finally, the *Contextual Supervisor* (CS) is a specific module dedicated to fault recovery. It determines the robot context depending on the current robot state, the functioning mode and the available functionalities. It then manages the correlation between the current sub-objective and the robot context. Moreover, this module chooses the most suitable

reaction, depending on the modules status updated by the *Global Observation Module*.

The severity of the defined robot context will be the base of the CS decision, which alerts 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 unavailable (fatal failure).

The next section illustrates the use of the COTAMA adaptative mechanisms in a delivery mission.

#### IV. EXPERIMENT

##### A. Experimental context

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.

In the following subsections we focus only on the design of supervisors, since they are the entities that take the autonomy adaptation decision.

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 two bumpers rows and a camera. An embedded laptop hosts the control architecture COTAMA, under a real-time OS, Linux RTAI, 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.

3) *Autonomy levels and functioning modes*: The experimental mission is basically executed in an autonomous mode. But for reliability purpose, all the mission objectives have been defined for three autonomy levels: autonomous, teleprogrammed and teleoperated. Indeed, the robot could have to face to failures and to adjust the autonomy level. For a given objective, each autonomy level requires different low level modules, as the control law, the needed functionalities and then the related observers.

For example, Fig. 3 shows the Petri net of the LS for the Drive objective of the robot mission. This objective is composed of two sub-objectives: Path Generation and Path

Following. The LS manages those autonomous sub-objectives, but also the teleprogrammed and teleoperation ones. In teleprogramming mode, the operator can restart the autonomous path generation with new way points, or can give a new path to be followed autonomously. A specific sub-objective is dedicated for "Human-Robot Interactions", in which the Robot waits for Human decision.

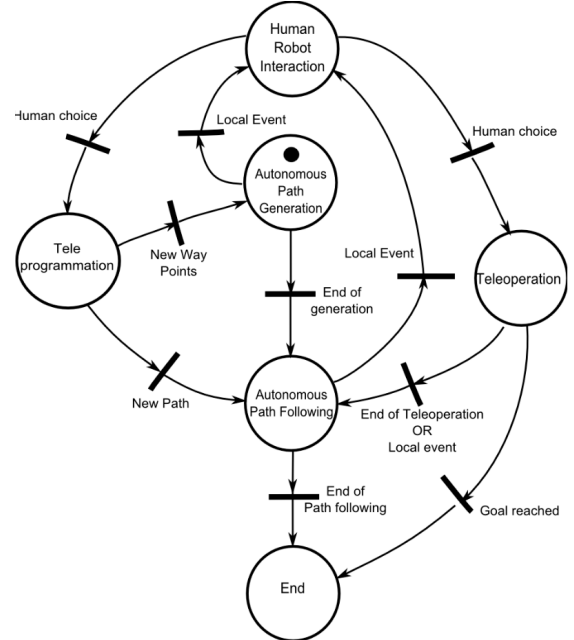


Fig. 3 Autonomy level management for the Drive objective

Furthermore, each of the autonomy level can be decline in several functioning modes depending on the context and the robot available resources.

For example, Fig. 4 presents a simplified Petri net managing three functioning modes for the autonomous Path following sub-objective: optimal path following with efficient localization and obstacle avoidance algorithms; and two degraded functioning mode, one using only the imprecise odometric localization, and another one without obstacle avoidance. The AS proposes two kinds of reactions to recover on failures: an adaptation of the parameters of the MCL module (staying in the optimal functioning mode), and a switch to a degraded Path following sub-objective.

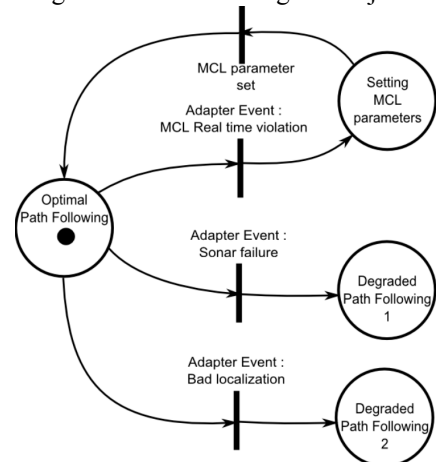


Fig. 4 Functioning mode management for the Path Following sub-objective

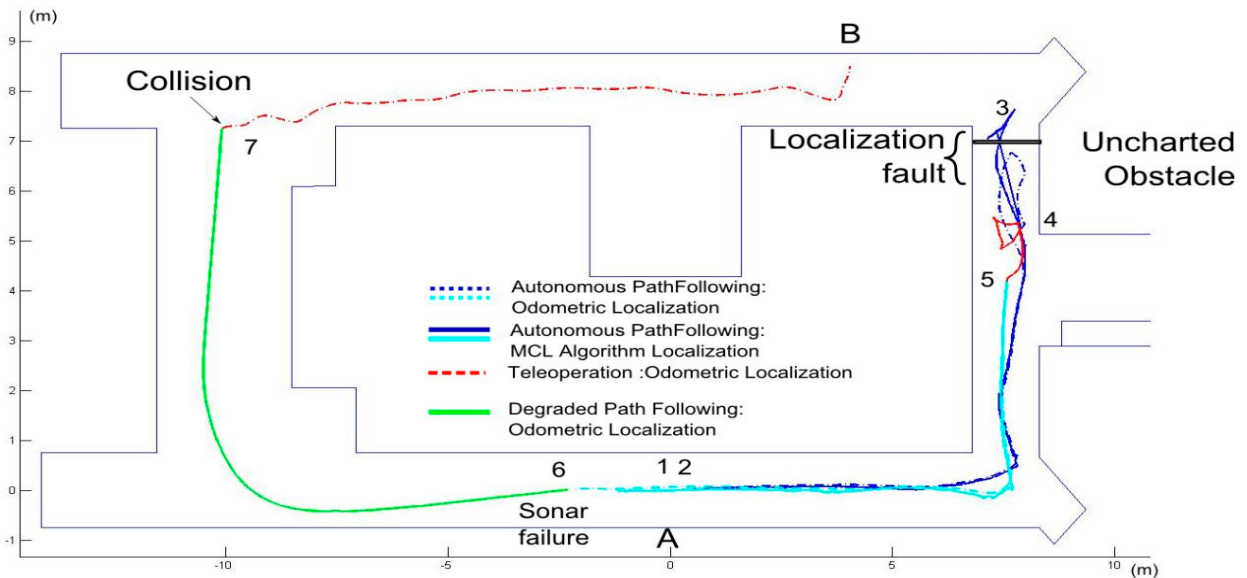


Fig. 5 Experimental mission scenario

### B. Experimental results

This part illustrates our methodology. It highlights the fault detections and the involved reactions in the architecture. This experiment is realized Hardware In the Loop (HIL). Faults as actuators, sensors, communication or real-time faults can be observed and managed. Some of these 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.

1) *Description of the mission scenario:* 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. It is the maximal reaction time to a detected fault.

The interesting points of the recorded mission are:

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. In this optimal functioning mode, represented in blue continuous line, the localization is performed using Monte Carlo algorithm [26].

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 in order to reduce its execution time.

Point 3: The robot is confronted to an uncharted obstacle. It then considers that it could have a localization problem, and suspects an uncharted obstacle as it was forced to go back on its path. This situation is complex, so it requests human help.

Point 4: The human operator decides to observe the robot environment with the on-board camera in teleoperated mode (red dashed line) to validate the presence of this obstacle.

Point 5: The operator detects the uncharted obstacle and decides to change the path. The robot restarts the path

following optimal sub-objective with this new path (cyan continuous line).

Point 6: A permanent fault is observed on sonar sensors which cannot be used anymore. As a degraded autonomous Path Following sub-objective, since neither obstacle avoidance nor Monte-Carlo localization is available, the robot chooses to pursue the mission. Now, the localization is performed by odometers (green line). It decreases its velocity in order to reduce the eventual damages caused by the collision with an obstacle.

Point 7: The robot bumps into an obstacle. So a human operator help is asked. To complete the drive objective he decides to use degraded teleoperation (without obstacle avoidance). (The localization is performed by odometers; dashed red line).

As we could see, this experimental mission includes autonomy levels adaptation to ensure the success of the mission despite faults occurrence.

2) *Autonomy adaptation:* In fig. 6 the evolution of the autonomy levels is presented functions of the mission scenario points.

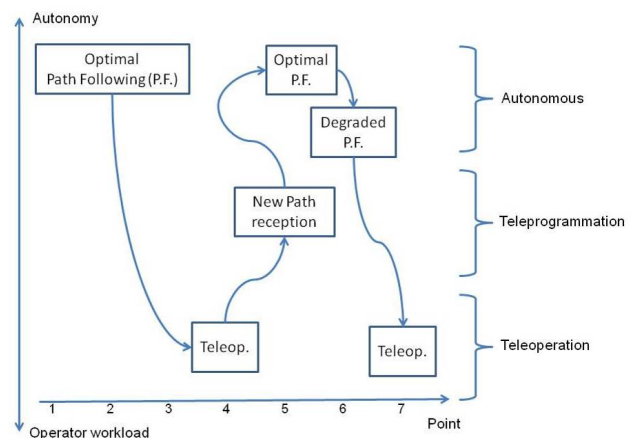


Fig. 6 Autonomy adjustment during the mission

During this mission scenario three autonomy levels are used: autonomous, teleprogramming and teleoperation modes. Moreover, the autonomous mode is decomposed into two functioning modes: an optimal and a degraded one. The human help request is not done systematically but only on relevant faults and contexts. When it is possible the control architecture adaptation is realized autonomously.

It can be noticed that, without the human help the mission would be aborted at point 3. Thanks to the human capacities to handle non consistent situation, the encountered problem is identified and the mission can continue from a new robot coherent state. At this point the cooperation between the operator and the robot is just punctual. However, at point 7, the robot possibilities are too limited to pursue the mission: there are no sonars anymore, nor localization and obstacle avoidance. As the robot bumps into something, either it is lost or it encounters an uncharted obstacle, then it concludes that it could not resolve this situation by itself.

## V. CONCLUSION

Due to the increasing complexity of autonomous mobile robots and its difficulty to face to unknown environment and situations, the potential faults sources remain very large. Autonomy sharing, between the robot and the operator, will be, for still a long time, a robust solution to ensure the success of a mission. Adaptive control architecture implementing fault tolerance principles will be essential to address this issue. This paper presents an adaptive control architecture with specific mechanisms dedicated to the enhancement of robot reliability and robustness using autonomy adaptation. However this last concept needs to answer a central question "Which entity takes the decision to adapt the autonomy level?". This paper considers only a part of the answer since, when the robot encounter an unsolved problem, it asks for human help. The symmetric answer needs to be considered in future works. When the operator seems to have a problem, the robot could suggest helping him and proposed an adaptation of its autonomy level.

## REFERENCES

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

[4] S. Bensalem, M. Galien, F. Ingrand, I. Kahlou, T. H. Nguyen, "Toward a More Dependable Software Architecture for Autonomous Robots", *Special issue on Software Engineering for Robotics of the IEEE Robotics and Automation Magazine*, vol. 16, no. 1, 2009.

[5] 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.

[6] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das, "The CLARATy architecture for robotic autonomy", *IEEE Aerospace Conference*, vol. 1, pp. 121-132, 2001.

[7] 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 (IROS'06)*, Beijing, China, 2006.

[8] D. E. Bernard, E. B. Gamble, N. F. Rouquette, B. Smith, Y. W. Tung, N. Muscettola, G. A. Dorais, B. Kanefsky, J. Kurien, W. Millar, P. Nayal, K. Rajan, W. Taylor, "Remote Agent Experiment DS1 Technology Validation Report", *Ames Research Center and JPL*, 2000.

[9] B. Espiau, K. Kapellos, M. Jourdan, "Formal Verification in Robotics: Why and How", in *proc. of the 7th Int. Symposium of Robotics Research*, Munich, Germany, Cambridge Press, pp. 201-213, October 1995.

[10] J. Borrelly, E. Coste-Manière, B. Espiau, K. Kapellos, R. Pissard-Gibollet, D. Simon and N. Turro, "The ORCCAD Architecture", *Int. Journal of Robotics Research, special issue on Integrated Architectures for Robot Control and Programming*, vol. 18, pp. 338-359, 1998.

[11] B. C. Williams, M. D. Ingham, S. H. Chung, P. H. Elliott, M. Hofbaur, T. Sullivan, "Model-Based Programming of Fault-Aware Systems", *AI Magazine*, Vol. 24, no 4, pp. 61-75, 2003.

[12] R. Simmons, C. Pecheur, G. Srinivasan, "Towards Automatic Verification of Autonomous Systems", in *IEEE/RSJ Int. conf. on Intelligent Robots & Systems*, 2000.

[13] A. Basu, M. Gallien, C. Lesire, T. H. Nguyen, S. Bensalem, F. Ingrand, J. Sifakis, "Incremental Component-Based Construction and Verification of a Robotic System", *ECAI 2008 18th European Conference on Artificial Intelligence*, Greece, 2008.

[14] B. Lussier, A. Lampe, R. Chatila, J. Guiochet, F. Ingrand, M. O. Killijian, D. Powel, "Fault Tolerance in Autonomous Systems: How and How Much?", in *proc. of the 4th IARP - IEEE/RAS - EURON Joint Workshop on Technical Challenges for Dependable Robots in Human Environments*, Japan, 2005.

[15] R. Murphy and D. Hershberger, "Classifying and recovering from sensing failures in autonomous mobile robots," in *proc. of the National Conference on Artificial Intelligence*, pp. 922-929, 1996.

[16] 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.

[17] 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.

[18] S. Lemai-Chenevier, "IXTET-EXEC: Planning, Plan Repair and Execution Control with Time and Resource Management", PhD thesis, LAAS-CNRS, 2004. [in French].

[19] B. Durand, K. Godary-Dejean, L. Lapiere and D. Crestani, "Global methodology in control architecture to improve mobile robot reliability", in *proc. of the Conf. of Intelligent Robots and Systems, (IROS'10)*, Taipei, Taiwan, 2010. (to be published)

[20] H-M, Huang, K. Pavek, B. Novak, J. Albus, and E. Messina, "A Framework For Autonomy Levels For Unmanned Systems (ALFUS)," *Proceedings of the AUVSI's Unmanned Systems North America*, June 2005, Baltimore, Maryland.

[21] B. T. Clough, "Metrics, Schmetrics! How The Heck Do You Determine a UAV's Autonomy Anyway?", *Performance Metrics for Intelligent Systems Workshop*, Gaithersburg, MA, USA, 2002.

[22] T. B. Sheridan and W. L. Verplank, "Human and Computer Control for Undersea Teleoperators". MIT Man-Machine Systems Laboratory, 1978.

[23] M. A Goodrich, and A. C. Schultz, "Human-Robot Interaction: A Survey", *Foundations and Trends @ In Human-Computer Interaction*, 2007, no.3, pp. 203-275.

[24] Oppermann, R. and Simm, H. "Adaptability: user-initiated individualization". In *Oppermann, R. (Ed.), Adaptive User Support: Ergonomic Design of Manually and Automatically Adaptable Software*, Lawrence Erlbaum Associates, pp. 14-64, 1994.

[25] M. Baker and H. A. Yanco, "Autonomy Mode Suggestions for Improving Human-Robot Interaction", in *proc. of the IEEE Conf. on Systems, Man and Cybernetics*, 2004.