Fault Tolerance in Control Architectures for Mobile Robots: Fantasy or Reality?

Didier Crestani, Karen Godary-Dejean

Fault Tolerance in Control Architectures for Mobile Robots: Fantasy or Reality?

D. Crestani, K. Godary-Dejean
Laboratoire Informatique Robotique Microélectronique de Montpellier
Université Montpellier Sud de France - CNRS
{godary, crestani}@lirmm.fr

Abstract—. Due to the future development of robotic autonomous systems in human environment, the fault tolerance paradigm will be a central issue in robotics. This article presents a survey of fault tolerance concepts, means and implementations in robotic architectures.

I. INTRODUCTION

Step by step fault tolerance in autonomous systems becomes a key challenge for the next decades. As point out in [1], systems and control science must be able to propose in the near future efficient control strategies optimizing the system's behavior to accomplish intending functions, while satisfying performance requirements and minimizing negative effects (resources consumption, safety behavior, etc.). But the increasing complexity of autonomous systems evolving in open, complex, unstructured and dynamic environment and interacting with *vulgum pecus* humans implies that the realization of sophisticated missions in a reliable way seems to currently be a fantasy. Fault tolerance integration will be a major part of the answer to the development of reliable and secure autonomous systems. The software aspect related to control architecture will be an important constituent of the fault tolerance deployment.

This paper is based on some relevant works and analysis concerning robots reliability [2], dependable concepts [3], fault tolerance in engineering [4] [5] [6] [7] [8], fault tolerance in autonomous (robotic) systems [9] [10] [11] [12] [13], and fault tolerance issues [14] [15].

The next section presents through a wide analysis of on-the-field missions, the existing practical limitations of mobile robot reliability. Some basic definitions concerning dependability are given section III, with discussion about faults categories and location. Then section IV reminds the fault tolerance principles and section V exposes some of the main techniques used in control engineering. Next, sections VI and VII deal with fault tolerance in robotics. Methods really deployed in robotics are presented and analyzed before glancing at some robotic control architectures. Finally the conclusion highlights some important issues for fault tolerance integration for autonomous robotic systems.

II. FAULT TOLERANT ROBOTS: A FANTASY!

For many years J. Carlson works on fault tolerance in mobile robotics [2] [15] [16] [17] [18]. In [2] a detailed analysis of Unmanned Ground Vehicles in the field missions is realized. This study covers 24 different robots (15 models), from 3.6 kg (Micro Tracs from Inuktun©) to more than 27 tons (Panther army tank)! These robots are developed by 7 manufacturers. The concerned applications are Urban Search and Military Operations in Urban Terrian. The analysis concerns mainly on the field missions like during the World Trade Center disaster or for demining in Bosnia. Depending on the robot type, the communication with a distant operator is ensured using wired or wireless means.

The Urban and Rescue missions analyze shows that Inuktun© robots have a MTBF equal to 6.1 hours and an availability ratio of 90% because that encountered faults remain minors and easy to repair. Irobot© robots have approximately the same MTBF but the availability ratio was only equal to 36% because most of the failures are serious and hard to repair. The army robot mission analysis shows that there are only 1.16 failures per day, but that most of them (95%) are terminal for the mission.

It must be mention that the timing granularity of the mission data collection is not always known. For the World Trade Center missions, where a low granularity is available, 1.4 minor failures are observed per minute!

Concerning the physical failures location and impact, Figure 1 shows the synthetic diagram proposed by Carlson in [2]. It summarizes the range probability of the physical components observed

during the studied missions and their impact on the mission success.

The failures encountered during the different analyzed missions are of several types. The most common ones are the effectors' failures. For example, pinion gear collecting dirt and debris, tracks work off their wheels.

Sensors failures concern occluded camera, incorrect lighting, lack of depth perception, hard external conditions inducing moisture, dirt or mud on the lenses. Power failures are sometimes encountered due to low batteries or fuel problems (level, fuel filters). The control system failures are often encountered. It is a critical origin inducing unresponsive robot, uncontrolled behavior or system shutdown. The solution is to reboot the system. The communication is also a critical function because the wireless communication remains problematic in the field (availability, bandwidth) like for the World Trade Center and the impact of its structural steel. Another result is that failures have an important impact on the mission success since most of them are terminal.

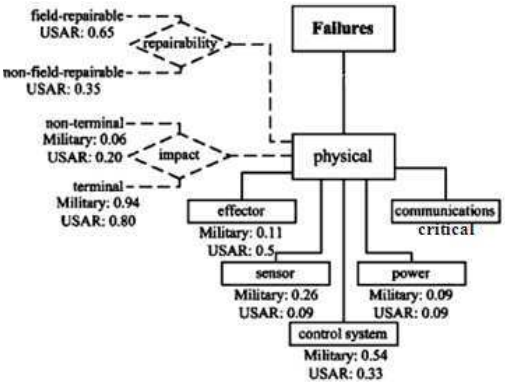

Fig. 1 Component failure probability (from [2])

This detailed analysis shows that the overall robot reliability is quite low with 6 to 20 hours of MTBF. Moreover it demonstrates that mobile robots must cope with many faults and are not very autonomously fault tolerant.

III. BASIC CONCEPTS

This part of the article reminds the basic concepts related to dependability, and discusses about fault and failure definition, taxonomy and location.

A. Basic definitions of dependable concepts (from [3])

This section presents some accepted definitions used in the dependability and security communities [3], and then clarifies the robotic point of view.

1) *Concepts in computing engineering*

- **Dependability**: Ability to avoid service failures that are more frequent and more severe than acceptable.
- **Reliability**: Continuity of correct service. It is an attribute of dependability.

The means of dependability have been grouped into 4 categories:

- **Fault prevention** to prevent the occurrence of the introduction of faults. Fault prevention mainly depends on software development methods ensuring easy maintainability, analyzability and testability.
- **Fault removal** to reduce the number and presence of faults. Fault removal concerns mainly tests and formal validation.
- **Fault forecasting** to estimate the present number, the future incidence, and the likely consequences of faults.
- **Fault tolerance** to avoid service failures in the presence of faults.

2) *Concepts in robotics*

Depending on the fault location Lussier in [9] [11] proposes to distinguish two concepts:

- **Robustness**: Restrict the fault tolerance of the treatment of adverse situations arising due to the variability of the autonomous system environment (unexpected obstacle, lighting variability, etc.).
- **Fault tolerance**: Concerns only the ability to deliver a correct service according to faults affecting the autonomous system itself (hardware or software fault).

B. Failure and fault: Definition, taxonomy and location

1) *Definitions*

- **Failure** (service failure): Event that occurs when the delivered service deviates from the correct service.
- **Error**: Part of the system's state that may lead to a failure.
- **Fault**: Cause of an error.

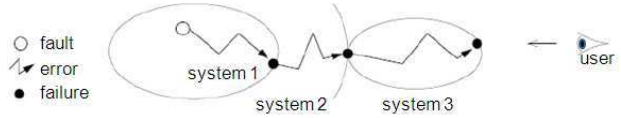

Fig. 2. From fault to failure (from [3])

Figure 2 explains the way leading from a fault to a failure. During design or operational phases, systems are necessarily affected by internal or external (coming from the environment) faults. So a fault is a malicious entity which, once activated, creates an error. This error is propagated through the system until a failure occurs in the system service. If the system is composed of several synchronized sub-systems, the failure of one sub-system service could become a fault for the following sub-system.

2) *Taxonomy*

Many quite similar faults taxonomies have been proposed in the literature [2] [3] [13] [17].

In robotics, human and physical faults are usually distinguished (Figure 3). Human faults concern system development and interaction faults, whereas physical faults are decomposed into effector, sensor, power, control system and communication faults.

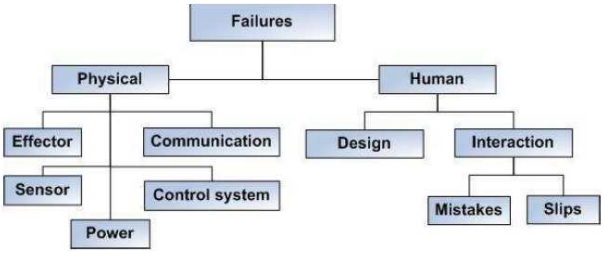

Fig. 3 Murphy faults taxonomy (from [17])

It is important to point out that a fault occurrence can only be observed by the user when a failure is detected. So, from Figure 2 it is easy to understand that many (sub-)systems can be affected during the error propagation without external signal of failure. This observation puts in light that a failure is not necessarily a fault. Thus, it is crucial to be able to detect (detectability) as soon as possible a fault activation and to deploy efficient diagnosis mechanisms to get back to the original fault.

3) *Fault Location*

In [14] an in-depth analysis of autonomous systems faults is proposed. Faults are spread through the environment, the autonomous system and its environment awareness (Figure 4). The following types can be distinguished:

Concerning the ***environment***:

- **Exogenous event** fault: The environment might change in an unpredictable way. For example the change can be caused by another agent, indirectly by the autonomous system itself or an inadequate model of the environment.

Concerning the ***autonomous system***:

- **Hardware fault**: Effectors, sensors, embedded control hardware faults.
- **Software fault**: Design and implementation flaws, run-time faults.
- **Knowledge fault**: The embedded knowledge about the world (initial or updated) is inappropriate with the mission, leading to inadequate decisions.

Concerning the ***environment awareness***:

- **Execution fault**: The real outcome of an action is not the expected one.
- **Sensing fault**: Sensors observations are erroneous due to sensors shortcomings or perception algorithms limitations.

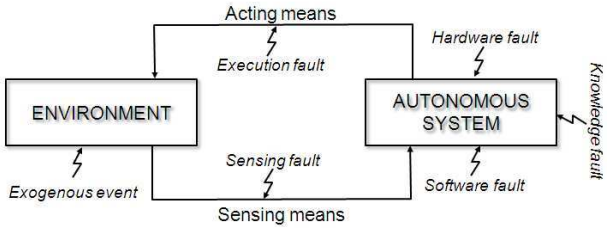

Fig. 4 Autonomous system faults (adapted from [18])

A fault tolerant autonomous system would be able to detect and react to all these possible encountered faults.

IV. FAULT TOLERANCE PRINCIPLES

Zhang [4] defines a Fault Tolerant Control System (FTCS) as a control system, which is able to automatically maintain the system stability and an acceptable performance when component failures occur.

To reach these objectives, FTCS must generally implement 4 main principles:

- **Fault Detection**: To detect that there is something wrong in the system and that a fault has occurred somewhere.
- **Fault Isolation**: To decide which component is faulty and its location in the system.
- **Fault Identification**: To identify the fault and its severity.
- **Fault Recovery**: To adapt the system controller structure, according to the identified fault, to maintain the system stability and an acceptable performance.

In the literature the fault isolation and identification steps are often merged into the single word **diagnosis**.

These last principles allow the fault tolerance execution. However, it must be notice that to be able to detect a fault, it is necessary to be conscious that this fault can occur. Therefore, **fault forecasting** is an inescapable and preliminary step, which cannot be avoided before undertaking a fault tolerance process.

These general principles have generated an important corpus of scientific researches concerning Fault Detection and Diagnosis (FDD) in control engineering over the last decades. Most of them can potentially be used for robotic purpose.

V. FAULT TOLERANCE IN CONTROL ENGINEERING: EFFICIENT BUT UNSUFFICIENT

Fault Detection and Diagnosis require an inter-disciplinary expertise. Many interesting and complete bibliographical overviews have been proposed [4] [5] [6] [7] [8].

A. Fault detection and diagnosis

Generally two main classes of FDD methods are identified: model-based and data-based methods.

The quantitative ***model-based methods***, widely used in robotics, are mainly based on analytical software redundancy. Many approaches performing signal processing techniques have been proposed for fault detection: state estimation observers or Kalman filters based approaches, parameter estimation oriented techniques or parity spaced oriented methods.

The ***data-based oriented methods*** don't need an a priori knowledge about the system behavior. They only require a large amount of historical data to extract a knowledge base which is used by the diagnosis system. Expert systems, neural networks using pattern recognition, or multivariate statistical techniques like principal component analysis can be used for example. The data-based methods can easily handle a large amount of data and are mainly used to control industrial processes. They are rarely employed in robotics where a model-based knowledge of the system can be easily addressed and the amount of data is limited.

B. Recovery

Usually, passive and active fault tolerant approaches can be distinguished for recovery.

Passive approaches don't need to previously have a fault diagnosis. Robust control techniques like the quantitative feedback theory, the CRONE control, the H_∞ synthesis or the adaptive control are used to design a controller ensuring that the control-loop remain stable and efficient from a performance point of view. Unfortunately these methods can only deal with few system's faults.

Active techniques need to know which fault affects the system to adapt the controller design while preserving the stability and a graceful performance. Two main reconfiguration strategies can be used. On the first hand some predefined control laws can be selected to correct the impact of identified faults. Then approaches using multiple models and variable gain commands can be applied. On the second hand, to react to the occurrence of a fault, an on-line controller synthesis can be realized. Active recovery approaches allow a better fault coverage than passive methods.

C. Conclusion

For the development of fault tolerant robotics, the works dedicated to control engineering are not able to address all issues that must be considered:

- It is difficult to ensure that all the proposed approaches always satisfy the real time treatment of a fault occurrence. This point is central for mobile robotic in a dynamic environment.
- The tolerant control-loops integrating FDD and recovery issues are efficient to deal with sensors or actuators faults. But they are not able to consider faults related to high-level knowledge.
- Despite the efficiency of the control engineering recovery mechanisms, the proposed answers are not flexible enough to manage the situations which must be handled during complex missions.

So, the software aspect is the main dimension to be considered to develop real fault tolerant mobile robots. This is the place where the fault tolerant principles must be integrated and managed. The control architecture provides more flexibility, expendability and adaptability capabilities than a pure oriented computer engineering approach.

VI. FAULT TOLERANCE IN CONTROL ARCHITECTURES: A FANTASY

Efficient autonomous robotics needs to merge planning capabilities with reactive behaviors. Hybrid control architectures decomposed in planning (deliberative), executive and behavioral (reactive) layers are often used. Component

oriented development permits to address fault prevention issues.

A. Fault Tolerance principles in control architectures

1) *Fault Detection and Diagnosis principles*

In [9] [10] [11], four basic mechanisms are identified to implement FDD in autonomous robots architectures.

- **Timing checks** are frequently introduced for timing fault detection. They supervise robot's functionalities liveness using timers and watch-dog principles.
- **Reasonableness checks** are commonly used in robotics to verify the correctness of system's variables according to algorithms constraints or manufacturer specifications
- **Safety bags checks** are also frequently implemented in robotic architectures to monitor the system outputs and blocking erroneous values to avoid unsafe behavior.
- **Monitoring for Diagnosis** is the most attractive axis for fault detection leading to many publications (see a survey in [19]). This domain is strongly connected to the control engineering techniques previously presented section V. Mainly dedicated to hardware fault detection, software redundancy using model-based approaches is often used to detect sensors or effectors failure.

2) *Recovery principles*

[3] presents a large analyze of the recovery mechanisms proposed in computer science which could be classified in error and fault handling. Some of them are used in robotics. Focusing on robotic control architectures, [11] and [13] identify the following recovery means:

For the executive level:

- **Specific treatment**: When a known situation is encountered, the hardware context or the concerned algorithm is changed to treat the identified fault.
- **Action retry**: The failed action is restarted expecting that its execution context will get better.
- **Action retry using functional redundancy**: The failed action is restarted using a functional alternative (if it exists).
- **Modality switching**: The failed action is restarted using the most efficient and operational functioning mode. It is an extension of the previous principle, as it needs to have a degree of redundancy. The redundancy can be software or hardware.

For the planning level:

- **Re-planning**: It consists of developing a new plan from the current situation facing adverse situation, to the expected final situation. This method is time consuming and the robot must be placed into a safe state during re-planning.
- **Plan repair**: This is a faster solution than re-planning, where the initial plan is locally modified to overcome the encountered adverse situation.

For the mission level:

- **Autonomy adaptation**: When recovery fails at executive and planning levels, a human assistance can be envisaged to pursue the mission. In this last case, an interesting solution is the adaptation of the autonomy level of the robot [13].

B. Control architectures and fault tolerance

From [12] [13], this part analyses the fault tolerance aspect in some classical robotic control architectures and frameworks. The main results are summarized in Table1.

CLARATY [20] has been developed by the NASA to propose an efficient robotic software for space mission. It is organized in two layers: the Decision layer, which integrates the planning and executive functionalities; and a Functional layer for interfacing with hardware platform. For fault tolerance purpose, each component may integrate a state machine to monitor the state evolution, and uses internal or external estimators for state variables estimation. These means allow implementing diagnostic and recovery mechanisms. If the conflict cannot be handled locally, it is considered by higher-level modules up to the planner level. The CLEaR (Closed-Loop Execution and Recovery) system is a hybrid controller coordinating goal-driven and event-driven approach. CLEaR integrates CASPER (Continous Activity Scheduling, Planning, Execution and Re-planning) and TDL (Task Description Language) systems. TDL can implement exception handling like action retry. CASPER implements plan repair recovery.

CoolBOT [21] is a component-oriented system dedicated to robotic purposes. Each component implements a states machine and communicates by

forwarding information via event and data ports. Exception handling permits to switch into a set of error states. Hierarchical component variables adaptation, backward error recovery and functional redundancy can be used to manage error handling.

COTAMA FFT (COTAMA For Fault Tolerance) [13] is a component-oriented architecture dealing with fault tolerance. Observation modules are used to monitor the identified faults occurrence. A dedicated diagnosis module, using residues analysis, identifies the fault and updates a database of the modules' state. According to the detected fault severity and the available resources, the recovery is engaged using the modality switching principle and autonomy adaptation.

The **LAAS** architecture [22] is dedicated for autonomous robots. It is a 3 layers architecture. The decisional level is dedicated to plans generation and execution. At the execution control level, the R2C component checks the validity of the requests of the decisional layer [23]. Finally, the functional level proposes modules generated with the GenoM tool, and offers services to hardware and software resources. The BIP software framework [24] allows the design of a controller enforcing low-level safety properties (ordering or synchronization violations, data freshness). R2C checks that modules correctly interact. It can verify some safety rules and may stop unacceptable actions. IxTeT planner component [25] implements re-planning and plan-repair facilities. Moreover, FTplan [9] manages some planning faults and proposes re-planning alternatives.

Name	Fault Forecasting	Detection Mean	Diagnosis Mean	Recovery Mean
CLARATY	No	State monitoring State estimators	Unknown if it exist	**E.L.**: Exception handling **P.L.**: Plan repair
COOLBOT	No	Exceptions list	Unknown if it exist	**E.L.**: Exception handling using component variables adaptation, backward error recovery or functional redundancy
COTAMA FFT	Yes	Observation modules (All types of detection means)	Residue based diagnosis	**E.L.**: Execution control using modality switching **M.L.** : Autonomy adaptation
LAAS	No	Safety rules verification at module and execution control level. Planning faults detection	Unknown if it exist	**E.L.**: Stop action **P.L.**: Re-Planning – Plan-repair
MIRO	No	Exceptions list Guard	Unknown if it exist	**E.L.**: Dynamic reconfiguration
ORCA	No	Action realization check	Unknown if it exist	**E.L.**: Action retry - Dynamic reconfiguration and redundancy
ORCCAD	No	Properties verification Exceptions handling Model-based detection?	Unknown	**E.L.**: Parameterization – dynamic reconfiguration – safe mode
IDEA	No	Variables timelines monitoring Non- nominal conditions detection	Unknown if it exist	**E.L.**: Ad-hoc recovery – Pre-compiled scripts using **P.L.**: Local Re-Planning
PROCOSA	No	Disruptive events detection (engine failure, payload failure, etc.)	Unknown if it exist	**E.L.**: Dynamic reconfiguration **P.L.**: Re-Planning (proposed) **M.L.**: Possible
RA	Perhaps	Failure mode identification module	Model based diagnosis	**E.L.**: Module repairing - Reconfiguration **P.L.**: Re-Planning with degraded capabilities **M.L.**: Possible
3T	No	Safety verification Watchdog	Unknown if it exist	**E.L.**: Resetting – dynamic reconfiguration **P.L.**: Re-Planning

E.L.: Executive Level – **P. L.:** Planning Level – **M.L.:** Mission Level

Tab. 1 Fault tolerance and robotic control architectures

MIRO [26] is an object-oriented middleware dedicated for mobile robot applications. It is structured into 3 architectural layers, interwoven with two layers of the CORBA middleware. The device layer provides interface for robots sensors and actuators. The service layer provides active service abstractions for sensors and actuators. The framework layer specifies the robotic functionalities. In this architecture there are some handling capabilities and a list of exceptions indicating hardware, service or load problems. Miro also defines action patterns organized in behavior hierarchies and guards. They notify external event occurrences which can potentially induce a dynamic reconfiguration.

ORCA [27] is a component-oriented open-source framework for robotics. Components implement algorithms and hardware interfaces. They communicate using push, pull or query patterns via TCP/IP or UDP protocols. Simple checks can be done at low-level, like correct action realization, If an action fails, a cleanup is simply realized. At application level some dynamic reconfiguration capabilities using redundant components are also possible.

ORCCAD (Open Robotic Controller CAD) [28] is a dedicated framework for the development of robotic applications. The two basics concepts allow the definition of a robotic action. The Robotic Task (RT) which models basic robotic actions and where control aspects are dominants. Robotic Procedure (RP) merges Robotic Tasks to construct more complex actions. RT and RP can be composed hierarchically in structure of increasing complexity. In RT the user can explicitly detailed the exceptions he wants to deal with and the corresponding recovery procedures: Parameters adaptation of a control law, new RT application, safe mode engaged. In [29] dedicated diagnosis modules are used to detect sensors and motor faults. Moreover, synchronous programming allows to verify crucial properties like safety (any fatal exception handled), liveness (RP reaches its goal in a nominal situation), conflict detection, behavior verification.

IDEA (Intelligent Distributed Execution Architecture) [30] is an agent based architecture. All agents possess the same structure and behavior (reactive planner, plan data-base, domain model, plan runner). Agent communicates using the communication services of the underlying middleware. More recently a Remote Agent functionality including mode recovery has been introduced. IDEA introduces timelines for the variables monitored. An inconsistency is reported to the Plan Data base which can engaged recovery action within the specified time boundaries. Non-nominal conditions handling can be introduced in the agent's central model. The flexibility of planning capacities (Minimal/long horizon and deliberative planning) can improve the system's responsiveness to external adverse events using local re-planning Recovery using pre-compiled alternative solutions (scripts) is possible.

PROCOSA [31] is an asynchronous Petri net-based software package dedicated for autonomous systems control and monitoring. Petri nets allow the description of the execution logic of the autonomous systems behaviors including nominal and disruptive phases. A Petri net player executes the associated procedures and manages the communications. The Petri net model permits to verify some structural and behavioral properties. PROCOSA takes care of unique faults occurrence and engages adapted recovery means including operator help. However the treatment remains limited and not systematic.

RA (Remote Agent) architecture [32] has been developed for autonomous spacecraft missions at NASA. RA is based is based on a 3 layered abstract machine: a planner/scheduler which plans actions and schedules resources; an executive layer which executes the defined plan; and a mode identification (MI) and mode reconfiguration (MR) layer used for diagnosis and fault recovery. MI is dedicated for observation and MR for recovery. MR is responsible for exiting from a failure mode using component repairing or reconfiguration. Re-planning with degraded capabilities can also be envisaged. So RA was design for fault tolerance.

The NASA **3T** [33] is a 3 layered architecture. The decisional layer controls the long term capabilities. A mediating layer achieves transitions toward the reactive skills layer. Depending on the experimental applications different fault tolerant capacities have been developed. Remote Agent principle has also been integrated in some 3T architecture experiments. Concerning task scheduling faults a generalized scheduler/reschedule is able to reschedule a task (start, stop, resources, time constraints) if a disruptive event occurs. For the

skill layer it is possible to detect that an expected state has not be reached. The skills layer is also able to manage a loss of communication using watchdogs and keep some sub-systems in safe mode. The mediating layer is able to handle non-nominal situations and uses reconfiguring skills or re-setting to manage their detection. Moreover the 3T architecture achieves safety behavior for different autonomy modes (teleoperated, semi-autonomous and autonomous).

C. Fault tolerance in control architectures: Current limitations

First of all it must be noticed that all the presented control architectures propose some means to implement fault tolerance. However many limitations can be identified:

- None of the proposed architectures but COTAMA FFT seem to integrate explicitly a preliminary fault forecasting step into the design process. Without a systematic off line approach of fault identification the fault tolerance will be drastically limited.
- In most of the architectures, fault detection and identification are roughly spread into the code. There is no structured reasoning guiding their implementation.
- Except for RA and COTAMA FFT the diagnosis mechanisms are not explained in the literature. This opacity denotes that the diagnosis phase remains basic. It seems that two important shortcomings are implicitly realized. First, there is no distinction made between faults and failures detection. Secondly, only unique fault occurrence is considered. These limitations hardly limit the relevance of the recovery phase.
- Generally, few types of recovery methods are proposed in a given architecture. No architecture proposes recovery solutions at executive, planning and mission levels. Recovery mechanisms are rarely formalized like in COTAMA FFT, RA, PROCOSA or ORCCAD. Furthermore, the most common recovery solution is to put the robot in a safe state, without proposing most efficient recovery solutions.
- Finally, few architecture complete fault tolerance using fault removal techniques (LAAS, ORCCAD).

Few research works have addressed the fault tolerance as a central issue in control architecture. There is a lake of global integration of fault tolerance principles which are generally spread into the software. There is a need of detection, diagnosis and recovery reification in the control architecture design. A framework for fault tolerance integration would be useful.

VII. FAULT TOLERANCE FOR MOBILE ROBOTS: FROM FANTASY TO REALITY

Most of the time, the fault tolerance paradigm is not the central issue that is considered in the research works. Thus, to develop fault tolerant robotics, a set of open issues using Fault Detection Isolation and Recovery principles (FDIR) can be identified.

Application domains for FDIR [14] [15]: The development of fault tolerant robotics is clearly domain (environment), safety and cost dependant. The set of possible encountered faults is quite different for airplanes, robotic manipulators or autonomous mobile robots. The safety consideration is different when FDIR concerns a domestic or a companion robot. The associated development costs are not comparable. Actually due to a lack of self-confidence in architectural FDIR principles, critical systems [34] (space, avionic) often implement a large hardware redundancy to deal with the fault tolerance issue. Fault tolerant control architectures would be a nice and flexible answer for FDIR. The design of such architecture needs to be able to propose and develop efficient and reliable methodologies, concepts and software.

Design for FDIR [14]: The development of fault tolerant robotics needs to consider the fault tolerance issue from the beginning of the system design. A fault tolerance control architecture cannot be the unique answer. Software, but also hardware and control aspects must be considered together to implement the most efficient solutions for reliability. So specific approaches for FDIR integration and design would be proposed.

Fault tolerant control architecture design [13]: The previous analysis demonstrates that, generally, there is an important lack of fault tolerance consideration into control architecture.

Fault forecasting is an essential and initial step which must be realized to develop fault tolerant architectures. It is central to identify the most

significant faults the robotic systems could encounter within its application domain.

Fault detection and particularly monitoring for diagnosis must consider more than actuators and sensors faults to deal with situation awareness. Moreover, noisy sensors problem (false or missed detection) must be overcome to enhance sensors' data reliability.

Fault diagnosis advanced techniques must be really implemented in control architectures. Moreover, these methods have to be more relevant to prepare an efficient recovery. They must be able to distinguish faults, errors and failures detection and to deal with single and multiple faults occurrences.

Fault recovery must be spread over the mission, executive and planning architectural layers using for example adaptive autonomy, modality or re-planning capacities. Furthermore, the development of adaptive behaviors could also consider the mission performance aspects (trajectory accuracy, speed, safety, etc.).

For us, the design of a fault tolerant control architecture needs to reify the fault detection, diagnosis and recovery principles. Moreover, as we present in [13] and develop further, it is also crucial to propose a fault tolerant methodology integrating the forecasting step, and guiding the developer during the control architecture design.

FDIR metrics and benchmarks [14]: A key point to achieve fault tolerance is to be able to define, like for integrated circuits "a certified" fault coverage. So domain, objective, and comparable public metrics must be defined. As well, realistic and reproductive scenarios must be elaborate considering pure simulation, hardware-in-the-loop, laboratory or in-the-field environments.

VIII. CONCLUSION

This article proposes a general survey of the fault tolerance issue in mobile robotics, focusing on control architectures. It puts in light that this objective needs to develop design methodologies for FDIR, merging hardware, control engineering and software knowledge. Due to its flexibility and its adaptability, the design of fault tolerant control architectures would be a central answer for fault tolerance needs. However, nowadays, the fault tolerance principles are neglected in the control architecture design. So an important effort for fault tolerance integration must be realized. Moreover, to be industrially accepted, fault tolerant metrics must be proposed and evaluate for specific application domains and contexts of mobile robots.

REFERENCE

[1] HYCON2 – NoE leaders, "Position Paper on Systems and Control in FP8", June 2011.

[2] 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.

[3] 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.

[4] Y. Zhang, J. Jiang, J., "Bibliographical review on reconfigurable fault-tolerant control systems". Annual Reviews in Control, 32(2), pp. 229–252, 2008.

[5] R. Isermann,. "Fault-Diagnosis Systems : An Introduction from Fault Detection to Fault Tolerance". Springer, 2006.

[6] Venkatasubramanian, V., Rengaswamy, R., and Kavuri, S. N., "A review of process fault detection and diagnosis: Part I: Quantitative model-based methods", Computers and Chemical Engineering, 27(3), pp. 293–311, 2003.

[7] Venkatasubramanian, V., Rengaswamy, R., and Kavuri, S. N., "A review of process fault detection and diagnosis: Part II: Qualitative models and search strategies", Computers and Chemical Engineering, 27(3), pp. 313–326, 2003.

[8] Venkatasubramanian, V., Rengaswamy, R., and Kavuri, S. N., "A review of process fault detection and diagnosis: Part III: Process history based methods", Computers and Chemical Engineering, 27(3), pp. 327–346, 2003.

[9] B. Lussier, B., "Tolérance aux fautes dans les systèmes autonomes", PhD Thesis, Institut National Polytechnique de Toulouse, 2007.

[10] B. Lussier, R. Chatila, F. Ingrand, M.-O. Killijian, and D. Powell, “On Fault tolerance and robustness in autonomous systems” in proc. 3rd IARP IEEE/RAS EURON Joint Workshop on Technical Challenges for Dependable Robots in Human Environments Manchester UK (2004).

[11] 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.

[12] A. Shakhimardanov, "Research and development of fault tolerance and robustness in robotics: A survey", B-IT Master Studies in Autonomous Systems, University of Applied Sciences Bonn-Rhein-Sieg, Fraunhofer Institute for Autonomous Intelligent Systems, August 2006.
[13] B. Durand, "Proposition d'une architecture de contrôle adaptative pour la tolérance aux fautes", PhD Thesis, University of Montpellier 2, June 2011.
[14] G. Steinbauer, F. Wotawa, "On the evaluation and certification of the robustness of autonomous intelligent systems", 22nd International Workshop of Principles of diagnosis (DX-2011), Murnau, October 4-7, 2011.
[15] J. Carlson, "Technical report for the safety security rescue research center", Robot Fault Diagnosis Workshop, ICRA 2004, pp. 1-26, 2004.
[16] J. Carlson, "Analysis of how mobile robots fail in field environments", Master's Thesis, University of South California, 2004.
[17] J. Carlson, R. Murphy, "Reliability analysis of mobile robots", ICRA'2003, pp. 274-281, 2003.
[18] J. Carlson, R. Murphy, A. Nelson, "Follow-up analysis of mobile robot failures", ICRA'2004, pp. 4987-4994, 2004.
[19] Z. Duan, Z. Cai, J. Yu, "Fault diagnosis and fault tolerant control for wheeled mobile robots under unknown environments: A survey", IEEE International Conference on Robotics and Automation, pp. 3439–3444, Barcelona, Spain, 2005.
[20] I.A. Nesnas, R. Simmons, D. Gaines, C. Kunz, A. Diaz-Calderon, T. Estlin, R. Madison, J. Guineau, M. McHenry, I. Shu, and D. Apfelbaum, "CLARAty: Challenges and Steps Toward Reusable Robotic Software," International Journal of Advanced Robotic Systems, Vol. 3, No. 1, pp. 023-030, 2006.
[21] A. C. Domingez-Brito, "CoolBOT: A Component-Oriented Programming Framework for Robotics" PhD Thesis, University of Las Plamas, Gran Canaria, 2003.
[22] R. Alami, R. Chatila, S. Fleury, M. Gallab, F. Ingrand, "An architecture for autonomy", International Journal of Robotic Research, vol. 17, 1998.
[23] F. Py, F. Ingrand, "Real-time execution control for autonomous systems", 2nd Embedded Real Time Software, Toulouse, 2004.
[24] S. Bensalem, M. Gallien, F. Ingrand, I. Kahloul, T. H. Nguyen, "Toward a more dependable software architecture for autonomous robots", Special issue on Software Engineering for Robotics, IEEE Robotics and Automation Magazine, 2009.
[25] S. Lemai-Chenevier, "IxTeT-eXeC: Plani fication, reparation de plan et contrôle d'exécution avec la gestion du temps et des ressources", PhD Thesis, Institut National Polytechnique de Toulouse, 2004.
[26] H. Utz, S. Sablatnög, S. Enderle, G. Kraetzschmar, "Miro – Middleware for mobile robot applications", IEEE Transaction on Robotics and Automation, vol. 18, n°4, August 2002.
[27] A. Makarenko, A. Brooks, T. Kaupp, "ORCA: Components for Robotics", IROS 2006, Workshop on Robotic Standardization, Beijing, China, 2006.
[28] D. Simon, R. Pissard-Gibollet, S. Arias, "ORCCAD, a framework for safe robot control design and implementation", 1st National Workshop on Control Architectures of Robots, pp. 131-144, Montpellier, April 2006.
[29] D. Simon, "Hardware-in-the-loop test-bed of an Unmanned Aerial Vehicle using Orccad", 6th National Workshop on Control Architectures of Robots, Grenoble, May 2011.
[30] N. Muscettola, G. A. Dorais, C. Fry, R. Levinson, C. Plaunt, "IDEA: Planning at the core of autonomous reactive agents",. Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space, October 2002
[31] M. Barbier, J.F. Gabard, D. Vizcaino, O. Bonnet-Torrès, "PROCOSA: A software package for autonomous system supervision", 1st National Workshop on Control Architectures of Robots, pp. 37-47, Montpellier, April 2006.
[32] N. Muscettola, P. Pandurang Nayak, B. C. Williams, "Remote Agent: to boldly go where no AI system has gone before", Artificial Intelligence, Vol. 130(1-2), pp. 5-48, August 1998.
[33] D. Schreckenghost, P. Bonnasso, D. Kortenkamp, D. Ryan "Three tier architecture for controlling space life support systems", IEEE Symposium on Intelligence in Automation and Robotics, 1998.
[34] X. Olive "FDI(R) for satellite at Thalès Alenia space: How to deal with high availability and robustness in space domain?", International Conference on Control and Fault Tolerant Systems, pp. 837-842, 2010.