Enhancing Fault Tolerance of Autonomous Mobile Robots

Didier Crestani, Karen Godary-Dejean, Lionel Lapierre

Enhancing Fault Tolerance of Autonomous Mobile Robots

D. Crestani[a], K. Godary-Dejean[a,*], L. Lapierre[a]

[a]*Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier (LIRMM), UMR 5506, Université Montpellier 2; 161 rue Ada 34095 Montpellier Cedex 5 - France*

Abstract

Experience demonstrates that autonomous mobile robots running in the field in a dynamic environment often breakdown. Generally, mobile robots are not designed to efficiently manage faulty or unforeseen situations. Even if some research studies exist, there is a lack of a global approach that really integrates dependability and particularly fault tolerance into the mobile robot design.

This paper presents an approach that aims to integrate fault tolerance principles into the design of a robot real-time control architecture. A failure mode analysis is firstly conducted to identify and characterize the most relevant faults. Then the fault detection and diagnosis mechanisms are explained. Fault detection is based on dedicated software components scanning faulty behaviors. Diagnosis is based on the residual principle and signature analysis to identify faulty software or hardware components and faulty behaviors. Finally, the recovery mechanism, based on the modality principle, proposes to adapt the robot's control loop according to the context and current operational functions of the robot.

This approach has been applied and implemented in the control architecture of a Pioneer-P3DX mobile robot.

Keywords: Dependability; Fault tolerance; Diagnosis; Recovery; Control Architecture; Mobile Robotics.

*Corresponding author : Karen Godary-Dejean, +33/0 467 41 85 62
Email addresses: crestani@lirmm.fr (D. Crestani), godary@lirmm.fr (K. Godary-Dejean), lapierre@lirmm.fr (L. Lapierre)

1. Introduction

Mobile autonomous robots operating in an unstructured dynamic environment have many practical applications, e.g. space robots for space exploration, war robots for military field operations or rescue robots for urban search and rescue missions. However, autonomous control of a robot in the field is still an open research domain.

Experimental feedbacks and mission sequence analysis of unmanned mobile robots in the field have generally demonstrated that there is a huge lack of dependability when physical or logical failures occur [1, 2, 3, 4]. Unfortunately, this limitation is an important bottleneck for the development of a real public-oriented robotics that must efficiently deal with availability, reliability, safety and maintainability aspects. To achieve this objective, it is essential to consider and introduce dependability rules as soon as possible into the robot's design life cycle. According to this principle, this paper proposes a global and structured approach to guide the introduction of fault tolerance principles into a robot's embedded control architecture.

The first part of this paper outlines what motivated this work and enumerates some of the challenges that must be overcome to implement fault tolerance. The second part summarizes the main techniques used to introduce fault detection and identification, and recovery mechanisms into the control architecture. Then the different phases we proposed for the development of a fault tolerant control architecture are explained. The next part details the implementation of our approach using the COTAMA control architecture and a Pionner-P3DX mobile robot for an office delivery mission. Finally, before concluding, the experimental results and robot behaviors are analyzed.

2. Dependable Robotics: a Critical Challenge

The design of dependable systems is a critical issue when human life, environmental pollution or expensive projects are concerned (space, aviation, nuclear,

human rescue, etc.) [5]. Therefore it is essential to develop techniques to avoid fault occurrence, and to use methods that take faults during system operation into account. Indeed, nominal behavior could not be preserved in uncontrolled and dynamic environments that represent a potential source of dysfunction. A fault tolerant system could guarantee minimal and acceptable behavior even in presence of faults.

Dependability principles have been largely studied and used in computer science [6]. In autonomous robotics, these principles must be adapted and implemented to carry out reliable missions in real and complex dynamic environments. Dependability means can be broken down into four main complementary categories:

- Fault forecasting, which performs an evaluation of the system behavior, analyzing the component failures and their effects on dependability.
- Fault removal, which aims to reduce the number of faults.
- Fault prevention concerning engineering principles, which aims to prevent faults from occurring during the system design.
- Fault tolerance, which tries to avoid service failures when a fault occurs during operation. This article mainly concerns fault tolerance issues.

This paper focuses on the application of fault tolerance concepts to enhance the reliability of autonomous robotics.

2.1. Motivation

Today, the most efficient robot system is not able to have the perfect behavior necessary to execute all robotic missions that people could imagine. A fault may generate relatively dramatic effects such as a significant decrease in the robot's performance, an aborted mission, loss or crash of the robot, or human injuries.

Carlson [3] conducted a detailed analysis of several unmanned ground vehicle (UGV) military or rescue field missions performed by a broad range of robotic platforms. The analysis globally highlighted a major lack of reliability

of the robotic systems, frequently leading to a dead stop in the mission. Besides human-induced faults (design, mistakes), five classes of physical faults were identified, which are effector, sensor, power, communication or control system oriented.

The development of really efficient robots for public, industrial or military applications require integration of dependable principles from the beginning of the robot design process. However, this issue is very seldom addressed in robotics. In research laboratories, most autonomous robots are just built to address a specific scientific issue. Hence, there are just prototypes for which the dependable principles are not really considered. Moreover, when dependability is critical, e.g. for space robots, the economic cost is out of line with large-scale production strategies. Indeed, the industrial robotic domain has not been confronted with hard critical constraints in the past decade. Things may change with new emergent robotic applications.

This is why so few articles have been published that propose a methodology explicitly integrating the dependable perspective into the design of autonomous robots. This scientific topic is rarely addressed in major robotics conferences (ICRA, IROS). Concerning Robotics and Autonomous Systems, it seems that only one article has focused on the dependable issue [7] since 2006.

There is also a need for a methodology to consciously guide robot designers to introduce dependable principles. Concerning the fault tolerance aspects, this methodology covers the offline relevant fault identification problem, and the embedded fault detection, identification and recovery mechanisms.

2.2. Challenges for Fault Tolerance in Robotics

Many challenges must be overcome to create fault tolerant autonomous mobile robots:

- Although many research studies have focused on fault tolerant control systems, there is not yet an accepted standardization of the concepts, design methods and terminology [8].

- An autonomous mobile robot running in a complex dynamic and unstructured environment can cope with a large variety of faults. The robotic system must be able to address them.

- It is unrealistic to think that a robotic system is able to consider all possible faults and react intelligently to every situation that might be encountered. Hence it is important to identify the most relevant faults, i.e. faults that are the most crucial for the robot's behavior and the concerned mission, depending on the current context.

- The severity of a failure must be linked with the performance of the robotic mission. For the user, what are the performance criteria? They could be, for example, the mission duration, the robot availability, or just the mission success. On the other hand, failure severity depends on the recovery means. Does the user accept a loss of performance? Does the user accept to adjust the autonomy level of the robot to overcome -punctually or not- an encountered fault? All of these questions must be addressed to define the relevant faults.

- When a fault occurs and can be identified, it must be handled in real-time using the robot's embedded operational resources of the robot as well as possible.

- Many research studies concern some parts of the fault tolerance logic. However, in robotic control architectures, there is a lack of a global, transverse and structured fault tolerance approach.

- Fault tolerance principles are currently scrambled into the control architecture. These fault tolerance principles must be reified to be efficiently considered.

- The design of a fault tolerant control architecture requires collaboration between two different worlds, i.e. robotics specialists and architecture designers (a priori computer specialists). They must understand each

other to integrate robotics and architectural constraints into the fault tolerance design.

3. Fault Tolerance Principles in Robotic Systems: An Overview

3.1. Fault Tolerance Principles

Zhang [8] defines a fault tolerant control system as a control system that is able to automatically maintain the system stability and an acceptable performance when component failures occur. Generally these systems must implement four main principles to achieve these objectives:

- Fault Detection: To detect if 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 associated severity.
- Fault Recovery: According to the identified fault, to adapt the system controller structure so as to maintain the stability of the system and acceptable performance.

The general principles of fault tolerance have generated a substantial corpus of scientific research concerning fault detection and isolation (FDI), and fault detection and diagnosis (FDD) (i.e. FDI with identification in addition [8]) in control engineering over the last decades. Most of them can potentially be used for robotic purposes.

3.2. Fault Tolerance in Control Engineering

Fault detection and diagnosis requires inter-disciplinary expertise. Many interesting and complete bibliographical overviews have been proposed [8, 9, 10, 11, 12, 13].

Fault tolerant control usually distinguishes passive and active techniques. Passive approaches do not require a previous fault diagnosis. Robust control techniques (quantitative feedback theory, CRONE control, H∞ synthesis, adaptive control) are used to design a controller to ensure that the control-loop remains stable and efficient from a performance standpoint. Unfortunately these methods can only deal with a few system faults. Active techniques need to know which fault affects the system (FDD methods) in order to adapt the controller design while preserving the stability and graceful performance (recovery means).

Generally two main classes of FDD methods are identified: model-based and data-based methods. In model-based methods, mainly based on analytical redundancy which provides mathematical models of the system's behavior, the emergence of a fault induces a modification of the system's state variables or model parameters. The detected deviation (residual) of the real system from the system's model can be used for fault diagnosis purposes. Widely used in robotics, many approaches performing signal processing techniques have been proposed for fault detection using state estimation observers, Kalman filters, parameter estimation, or parity spaced oriented methods. Contrary to quantitative methods, data-based methods (expert systems, trend analysis, principal component analysis) do not require 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.

Two main strategies can be used for recovery. First, some predefined control laws can be selected to correct the impact of identified faults (multiple models or variable gain command approaches) can be applied. Second, in response to the occurrence of a fault, an on-line controller synthesis can be performed.

Many advanced control engineering research strategies can be used to develop fault tolerant controllers to deal with robotic issues. However, for the development of efficient fault tolerant robotics, these strategies are not sufficient to address all of the issues that must be considered:

- It is difficult to ensure that all of the proposed approaches will always be

suitable for real-time treatment of a fault occurrence. This is a key point for mobile robots functioning in a dynamic environment.

- In the vein, fault tolerant mechanisms must be efficiently implement in an embedded context with limited resources.

- Some efficient detection techniques like data-based ones are not suited to robotics where thorough knowledge of the system can be easily used but the amount of data is limited.

- The tolerant control-loop integrating FDD and recovery solutions proposed in control engineering are efficient for dealing with sensor or actuator faults. But they generally do not consider faults related to high-level knowledge such as the contextual environment of the robot.

- Despite the efficiency of these approaches, we think that they cannot deal with the wide range of fault types nor multiple fault occurrence which must be considered in mobile robotics for complex missions in unknown environments.

The control laws are embedded into a software environment. So the software aspect is the main dimension to be considered to develop real fault tolerant mobile robots. This is where fault tolerant principles must be integrated and managed. Control architectures provide more flexibility, expendability and adaptability capabilities, which are precious in a design process for fault tolerant systems. They will therefore be considered in the sequel.

3.3. Fault Tolerance in Robotic Control Architectures

To enable efficient autonomous robotics, hybrid control architectures [14, 15, 16] are presently widely used because they allow merging of planning capabilities with reactive behavior. Conventionally, in these architectures, planning (deliberative), executive and behavioral (reactive) layers are defined. They may be used to efficiently address goal oriented and dynamic environment issues. They are clearly the main support of fault tolerance principle deployment.

3.3.1. Fault Detection and Diagnosis

In [17], three basic mechanisms are identified to implement FDD in autonomous robot system architectures.

Timing checks. are frequently introduced for temporal fault detection. They supervise the robot's functionality liveness using timers and watchdog principles. For example, they are used to manage the activity of RoboX9 [1], an autonomous museum robot tour guide, or for a robot taking part in the robot Soccer 2005 World Cup [18]. Watchdogs are also systematically associated with each BIP component used in the LAAS architecture [19].

Reasonableness checks. are commonly used in robotics to verify the correctness of system variables according to algorithm constraints or manufacturer specifications. Few examples are reported in the literature [1], although it is a basic method that is probably often used in robotic codes. Reasonableness checks could involve a simple test of data values in a dedicated interval, or a more complex assertion or property check, which could be named "safety bag", as in the R2C controller component of the LAAS architecture [20].

Monitoring for diagnosis. has been covered in many publications (see [21] for a survey), generally inspired from the control engineering techniques discussed above. Analytical redundancy using model-based approaches is mainly dedicated to sensors and actuator fault detection and often used to detect a malfunctioning. For example, in [22], [23] or [24], probabilistic models are used to detect wheel and motion interference faults on mobile robots. Multiple-models [25] like multiple-model Kalman filters [26] are often developed for detection and diagnosis issues in mobile robotics.

However, these FDD mechanisms have a set of limitations, which must be overcome to develop efficient fault tolerance in robotics.

- Reasonableness and safety bag checks are essential for robotics issues. However, their implementation is usually embedded into the code by the

developer without any visibility, architectural logic and coherent global view.

- Monitoring for diagnosis usually focuses only on sensor and actuator faults. But other components faults, as well as linked, multiple and/or transient faults, are not considered. Furthermore, more subtle faults, which are generally induced by external adverse situations, are not considered. Moreover most of the research studies have focused mainly on the development of a sophisticated detection technique without considering its connection with the recovery part of the fault tolerance. Furthermore, the relevance of the detected faults can sometimes be questioned with respect to the real probability of occurrence.

- The FDD diagnosis phase has never been addressed in robotic control architecture publications. However, its accuracy may have a marked impact on the fault tolerance capacity of the robot, especially in the presence of complex faults when a global view or a sophisticated technique is needed for diagnosis. In such cases, it is necessary to separate the detection phase, which generally highlights a failure (a consequence of fault propagation), from the diagnosis phase which must detect the origin of this failure.

Once a fault is detected and diagnosed, recovery mechanisms must be deployed to handle the problem and maintain the robot's capacity to achieve its goals.

3.3.2. Recovery

As in computer science, recovery can be defined as the process which "transforms a system state that contains one or more errors (faults) into a state without detected errors (faults) that can be activated again" [6]. In robotics, two main approaches are implemented in control architectures, i.e. execution control and re-planning.

Execution control. , which is largely used in robotic architectures, supervises the execution of the plan generated by the task planner. In case of malfunction

detection, the execution control tries to overcome the problem. Local or more global reactions can be implemented. For example, in the CLARATY architecture [27], dedicated mechanisms are implemented locally within the functional modules to react to known faults. In the same way, malfunctioning can be resolved locally in the ORCCAD architecture [15] using robotic task reconfiguration, or can be managed more globally using robotic procedures to adapt the current task. The LAAS architecture [20] uses the dedicated R2C module at the decisional layer to deal with recovery. The modality switching principle, which identifies and uses several different tasks to carry out the same action, can also be proposed as in [28] or [29]. Execution control allows a pertinent and reactive fault detection response. However, to be efficient, it must be based on in-depth and deliberate analysis of the system's faults and corresponding severities. If it fails to ensure the current task execution, a more global solution must be found using the task planner capacities.

Re-planning. leads to the drawing up of a new plan, ensuring the mission objectives from the current situation, and solving the detected problem. It is developed, for example, into the LAAS and CIRCA architectures [30, 31]. Re-planning, which is highly time consuming, can sometimes be replaced by a plan-repair. This technique supposes that a part of the current plan remains valid and can be executed while re-planning. This approach is developed into the LAAS architecture using the IXTET module [32], whereas in CLARATY [33], iterative planning is managed using CASPER (Continous Activity Scheduling, Planning, Execution and Re-planning). Unfortunately, this high-level and time-consuming fault handling cannot be adopted when dynamic environments requiring reactive behaviors must be considered.

If the robot cannot recover autonomously from a fault detection using execution control or re-planning, it is usually turned off in a safe state, terminating the current mission. For now, this is the most current consequence of a fault occurrence. We think that distant human interaction is another way to potentially recover from an untreated fault and enable the mission to complete [34].

According to the decisional relation between the robot and the operator, there are several classes of human-robot interactions, ranging from teleoperation, supervisory control [35] to full autonomy [36]. Rarely used for fault tolerance, adjustable autonomy could, however, be a solution for systems where sporadic human intervention is conceivable. For example, this recovery strategy is used for urban search and rescue robots via shared control [37] or an error handler [38], allowing human-robot interactions to solve the problem in the SFX-EH architecture [39].

3.4. Conclusion

This analysis highlights some strong points used to develop fault tolerance into robotic control architectures:

- Modular architectures are well adapted for implementing the flexible, reactive and adaptive behaviors needed for fault tolerance.
- Detection mechanisms could be easily used to detect specific faults such as sensor or actuator faults.
- Fault recovery mechanisms can be addressed, including executive control, re-planning or human-robot interactions.

Unfortunately, this analysis also revealed many important drawbacks which must be overcome to develop efficient fault tolerant control architectures:

- During the architecture design phase, a detailed analysis of the main faults that can affect the robot during its mission, and their corresponding severity, has to be carried out. This preliminary fault analysis step has never been addressed in robotics.
- Detection mechanisms are embedded into the code by the developer without any global development logic.
- The diagnosis phase has hardly ever been addressed to allow the detection of more complex and subtle failure origins. Proposed diagnosis mechanisms are too simple, based on too simple fault hypotheses (unique,

independent or permanent faults, for example), and could thus lead to erroneous recovery actions.

- Executive control and re-planning are efficient recovery mechanisms but Human-Robot interactions could also be considered as a way to implement fault tolerance to compensate for a lack of robot autonomy.

- Independent design of FDD techniques without considering recovery, and vice versa, will lead to a lack of efficiency for both parts.

- There is currently no existing approach to structure fault tolerance mechanism integration into robotic control architectures.

- Globally, there is a lack of a framework to guide the development of a fault tolerance architecture while considering all fault tolerance fields from a robotic standpoint.

To overcome some of these limitations, hereafter we present our approach, which structures and guides the development of robotic fault tolerant control architectures.

4. An Approach to Develop a Robotic Fault Tolerant Control Architecture

Here, it is important to specify the type of control architecture considered in this paper. It is assumed to be a two layered and component oriented architecture. The decisional layer, implementing the execution control mechanisms, manages the mission objectives and sub-objectives according to the robot's state and the detected faults. Each sub-objective describes a robotic task and corresponds, in the functional layer, to a set of software components (modules) that implement the concerned robotic functionalities. These software components are managed at the functional layer with real-time control of their execution. This decomposition is usually employed in robotic architectures. The mission planning layer is not considered in the present work.

The proposed fault tolerant architecture design approach can be decomposed into four main steps, which are presented in figure 1. First, fault forecasting must be conducted to evaluate the system behavior with respect to fault occurrence, and to produce a database with all information concerning the possible system failures. Based on this analysis, the three fault tolerance phases -detection, diagnosis and recovery- must be clearly integrated into the architecture structure. The detection step determines which architecture modules could be faulty. Then, according to the fault forecasting and fault identification information, the diagnosis phase diagnoses the faults and updates the state of the architecture modules (faulty or not) and determines the most severe fault. Finally, depending on the operational modules and the failure severity, the most adapted fault recovery action is engaged.

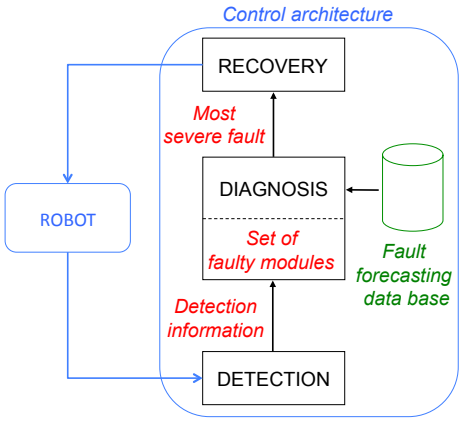

Figure 1: Fault tolerance diagram

4.1. Step 1: Fault forecasting using a fault tolerance oriented FMECA

This phase is essential for fault tolerance implementation because, before reacting, the failure affecting the system must first be identified before the active recovery treatment. A detailed analysis of the different possible failures and their consequences on the system's behavior must therefore be conducted.

FMEA and FMECA [40] are classical techniques of fault forecasting, which can be used to analyze failure modes, identify the potential causes of the failure and evaluate the corresponding effects and severity on the system's behavior. These approaches are largely used in different industrial fields such as aviation, automotive, chemical or aerospace domains. However, in robotics, only FMEA is sometimes employed to find hardware answers to material malfunctioning [41, 42, 43], but never used for software issues. This risk analysis technique, which takes multiple experts' viewpoints into account, is broken down into several steps, from identification of the system's limits to failure severity estimation and solution suggestions.

4.1.1. Identification of the system's limits

The FMECA process identifies the system's limits to consider for analysis. A global viewpoint must be integrated in a mobile robotic context: an autonomous mobile robot must carry out a given mission which is broken down into a set of concurrent and/or sequential objectives, which in turn correspond to different sub-objectives and robotic tasks. A first approach is thus to consider that each sub-objective constitutes a particular system, which must be individually analyzed from a FMECA perspective. Overcoming a failure can lead to a modification of the current sub-objective, or to a change of the robot autonomy level in an adjustable autonomy context. In that case, a sub-objective can be adapted into several functioning modes, depending on the available (non-faulty) functionalities. This can lead to the emergence of new system limits, which have not been yet assessed. Finally, a set of different sub-systems is identified depending on the concerned functioning mode.

FMECA methods are then used to list these sub-systems, their functioning and failure modes. The origin of each of these failure modes must be identified, as well as the importance of each failure on the basis of the severity of the consequences on the system. A complete FMECA analysis also involves an analysis of the fault occurrence probability, but our method does not yet use this information.

Figure 2 describes a mission that can be decomposed into n sequential sub-objectives. Alternative functioning modes for the different sub-objectives can potentially be identified and executed, depending on the impact of a considered failure, with or without changing the autonomy level.

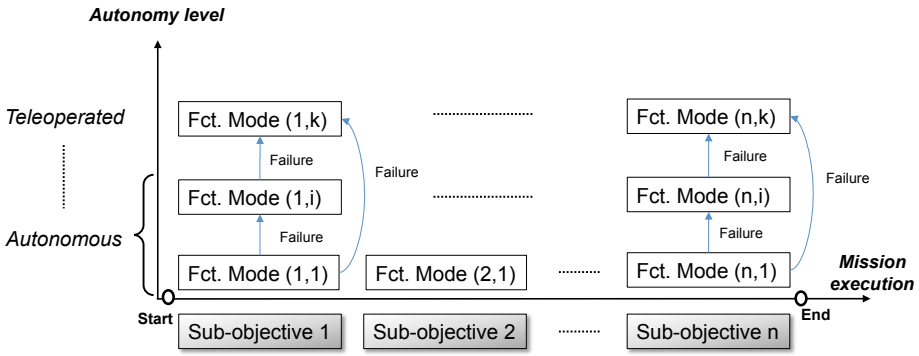

Figure 2: Identification of the limits of the sub-systems

4.1.2. Functional decomposition

In the FMECA method, the functional analysis is carried out to determine the different functionalities composing the analyzed system. For example, SADT/IDEF0 [44] is a classical methodology which allows a top-down hierarchical decomposition of system's functions and of the data associated with those functions.

To conduct this process in our context, it is important to map this decomposition in an architectural point of view based on the modular organization of the architecture. The decomposition process is carried out for a given functioning mode until each identified function can be merged into a software module.

4.1.3. Failure modes

Once the different functionalities and their corresponding architectural component modules have been identified, the following phase of the FMECA approach concerns the failure mode identification. A failure mode characterizes the availability and quality of the service delivered by a component module.

In our context, we consider two main **Classes** of failure modes:

- a failure mode is **Complete** when the concerned service can no longer be delivered. A motor breakdown can be an example.
- a failure is **Partial** when the service remains available but with a loss of quality. A network overload can illustrate this failure mode.

Furthermore these two classes can be split into two **Types** according to the activation duration of the failure mode:

- a failure mode is **Permanent** when the service quality is definitively affected.
- a failure mode is **Transient** if the service quality can be recovered.

4.1.4. Failure cause identification

The aim is now to determine the potential causes (faults) of the detected failures. To achieve this goal and guide the analysis of each component module belonging to a given functioning mode, we propose to use a classical effect-cause methodology based on the Ishikawa diagrams [45].

The Ishikawa diagrams are causal diagrams developed to identify potential factors contributing to a failure. Causes are usually grouped into major domains of potential sources of the failure. These major domains or analysis axes depend on the applicative domain. For fault tolerance related to the control architecture in mobile robotics, these domains can be inspired from different fault taxonomies [3, 6]. We then propose the following analysis axes (Figure 3):

- **Sensor**: if the fault concerns the hardware sensors or their interface devices.
- **Actuator**: if the fault concerns the hardware actuators or their interface devices.
- **Energy**: if the fault is linked up to energy devices and management problems.

- **Design**: if the fault is due to a problem of design or control algorithm implementation.
- **Environment**: if the fault is a problem of perception of the environment by the robot.
- **Control architecture**: if the fault relates to a software management problem, e.g. a scheduling problem.

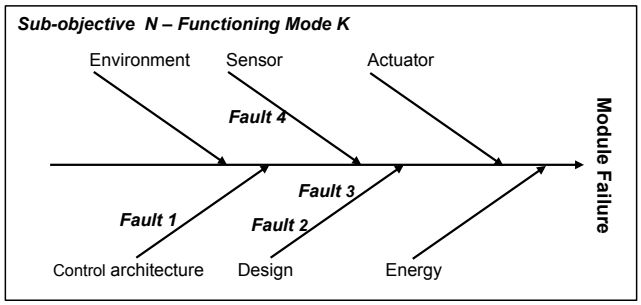

Figure 3: An Ishikawa diagram structure

4.1.5. Failure severity

Conventionally, in the FMECA approach, the notion of risk estimation is used by considering the probability of the failure occurrence (criticality) and its effects on the system (severity). Nevertheless, during the first design of a mobile robot and mission, it is impossible to have any statistical data to enable evaluation of the probability that a specific fault will occur. Even if some information could be obtained (such as the failure probability depending on the wear state, etc.), we decided to not yet consider failure criticality in our methodology, although it could easily be added in the future.

On the contrary, determination of the component failure severity is essential for fault tolerance efficiency. Its accuracy is strongly connected to the relevance of the recovery mechanism management. Firstly, note that the severity depends on the performance goals that are expected for the robotic mission. For example,

the goal could be just to finish the mission (without constraining requirements such as time or precision), and then only faults that abort the mission are serious, whereas if a safe behavior for the robot and/or its environment is imposed, the severity of the different failures must be refined. Moreover, timing or precision constraints can still be added. Secondly, the failure severity definition also largely depends on the availability of redundancy. The functional redundancy can compensate for the failure of a module if an alternative functionality is operational. This is usually an embedded redundancy when alternative hardware or software resources are available. However, without any embedded alternative, distant functional redundancy involving human assistance can be engaged.

Like many existing studies [46, 47], we define four levels of failure severity according to the available recovery capacities (Figure 4):

- **Weak** when the mission can continue using the same functioning mode while adapting some sub-objective parameters.
- **Medium** when the mission can continue using an alternative (potentially degraded) sub-objective while remaining at the current autonomy level.
- **Serious** when the robot cannot continue the mission with the current autonomy level.
- **Fatal** when the mission can no longer be continued.

Sometimes, an identified failure can be neglected within a specific experimental context. In this case, the failure is considered to be **Ignored** and is supposed to have no impact on the mission progress.

4.1.6. FMECA table

At the end of the FMECA process, all the collected information is usually summarized in a dedicated table. Moreover, this table specifies some potential recovery actions to address the identified faults.

Then we propose to establish, for each functioning mode, a FMECA table (Table 1) involving the name and failure modes for each module. Then, for

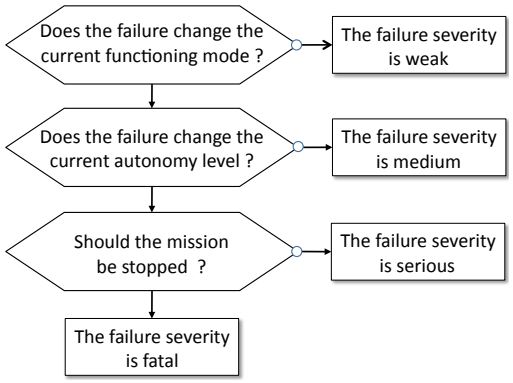

Figure 4: Determination of the failure severity

each failure mode, the potential failure causes (information on the fault) and the corresponding severity level are given. To facilitate the diagnosis phase for potentially related failures (failure of a component could trigger other component failures), information on the logical relations between failures are also given as a failure cause element. Moreover, more dedicated information is added (in the Actions column), which can be used to guide the fault tolerant process deployment within the control architecture. First, for the detection phase, this table identifies what kind of detection techniques can be used to detect the associated fault. Secondly, information related to the recovery phase are also specified.

Module name	Failure mode		Failure causes		Severity	Actions	
	Class	Type	Fault identifier	Logical relation		Detection	Recovery

Table 1: Basic structure of our FMECA table

This table is used to integrate fault tolerance principles into the robotic control architecture according to Figure 1: the FMECA tables are used by the diagnosis component to identify the fault and its corresponding severity, thus allowing the recovery component to engage the best adapted recovery action.

4.2. Step 2: Failure detection

The detection process (when exists) is now embedded in the control algorithm code. Our proposal is to defined dedicated modules to monitor robotic functionalities and material, thus reifying the detection process. Only a few studies have implemented this kind of detection mechanism, e.g. [18] for real-time faults.

Several types of detection modules (DM) can be identified (in grey in Figure 5): DM1 which monitors only one data stream (inputs or outputs of a module), implementing reasonableness and safety bag checks; DM2 or DM3 which monitor one or a set of modules by model-based detection mechanisms; finally, one DM is dedicated to real-time fault detection (not represented in the figure).

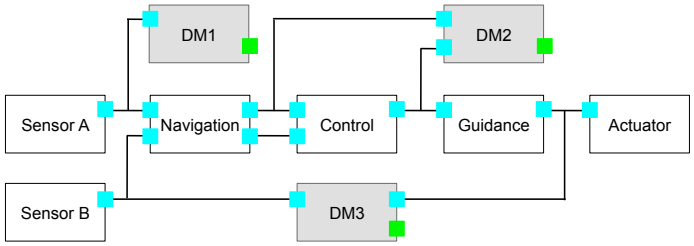

Figure 5: Detection module types

The use of detection modules to implement fault detection has two main advantages. Firstly, the most relevant one is that this method structures and reifies the detection process in a flexible manner. Detection could then be adapted depending, for example, on the reliability goals or the embedded resources, for the most relevant faults. Secondly, the use of independent modules provides more observation access, allowing the detection of faults that could not be detected inside the code. Furthermore, adding specific detection points will facilitate the diagnosis process. Finally, these new detection modules will naturally be scheduled into the robotic control loop. So their integration must verify the imposed real-time constraints. Their integration thus has some drawbacks. Firstly, execution of these new component modules is time consuming, so detection must be

controlled to fulfill the real-time constraints. Fortunately, detection tasks like reasonableness and safety bag checks are easy to implement and require little execution time. Secondly, these new detection modules must also be monitored and fault tolerant. We could consider that, with the exception of complex detection (model based checks), the simplicity of the code could be assumed to be free of design faults. The only relevant faults for these modules are real-time faults, which are monitored to ensure non-blocking execution of these modules.

Detection modules allow real-time evaluation of the robotic system identifying the operational and potentially defective functionalities. The detection module output is a set of Boolean data, called residuals, reflecting the correct or incorrect behavior of the monitored function. This information is used in the diagnosis process to identify faults affecting the robot and its control architecture.

4.3. Step 3: Fault diagnosis

We propose to base the diagnosis phase on detection residuals and fault signature analysis [48]. This choice enables us to use a simple, efficient, flexible and classical approach, which can be easily implemented whatever the modular control architecture. Furthermore, this diagnosis technique is a good tradeoff between efficiency and resource consumption, which is a decisive element in our embedded and real-time context.

4.3.1. Residuals and incidence matrix

The incidence matrix structure is given in Figure 2 for one functioning mode. The rows correspond to detection residuals, the columns represent the module's fault identifiers (those identified in the FMECA table, see Table 1). A given column corresponds to a fault signature, i.e. a specific set of residuals, which corresponds to the detection of a specific fault.

Since the detection modules are specifically designed to detect identified faults, most of the signatures are strongly isolating, i.e. they immediately isolate a fault. But detection is not always very easy and could be weakly isolating,

Sub-objective X, Functioning mode Y

Residual	Module 1		Module 2			...	Module N	
	IdF1	IdF2	IdF3	IdF4	IdF5		IdFk-1	IdFk
R1								
..								
Rm								

Table 2: Incidence matrix structure

i.e. the signature could correspond to several faults. However, in all cases, the incidence matrix is based on the FMECA table, and thus should cover all relevant faults that can affect the robot. Consequently the diagnosis process should always converge and isolate the original fault(s).

If this is not the case, i.e. if there is a detected problem for which the origin could not be identified, this could correspond to either an incomplete FMECA analysis, and the original fault has not been identified, or an insufficient detection process, and there are not enough nor relevant residuals. In both cases, a backup solution must be found to stop the mission and put the robot in a safe state.

4.3.2. Fault identification

Based on the detection module information, the residual values of a functioning mode are updated during each execution cycle. The diagnosis logic presented in Figure 6 allows determination of the faulty modules, using the incidence matrix and information on the logical relation between failures specified in the FMECA table. Depending on the isolating properties and logical relations, unique or multiple faults could be diagnosed as being the origin of one or several failures.

The identification process also defines the severity associated with the diagnosed fault. The severity is used to guide the recovery process. In case of multiple fault (or failure) configurations, this severity depends on the most relevant diagnosed fault and failure (the final severity is either the worst one, in case

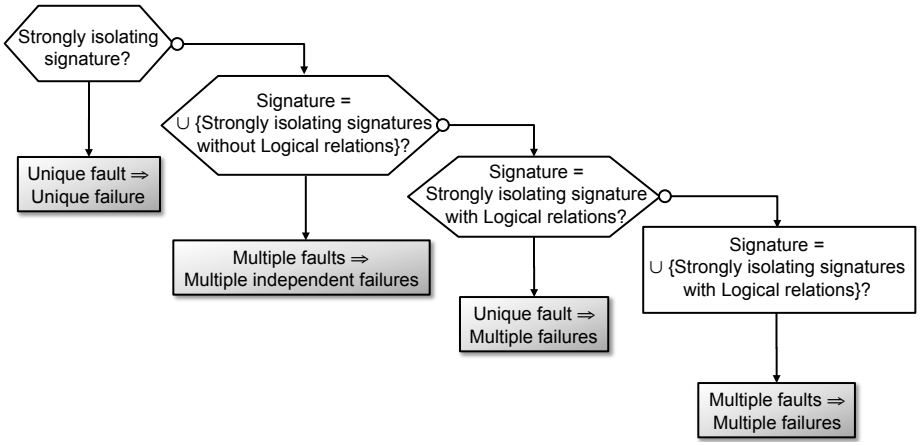

Figure 6: Diagnosis process logic

of independent failures, or one of the initial failures, in case of linked failures).

4.3.3. Updating the module state

At the beginning of the mission, all modules that can be employed in a functioning mode are considered to be in an **operational** state. When a module is diagnosed as being faulty, its corresponding state will be modified, depending on the duration of the fault (its type). If the failure mode is considered to be permanent, then the module state becomes **non-operational** and the module will never recover its associated functionality. If the failure mode is transient, the module state is considered to be **reversible** and the module may eventually recover its functionality.

The module state database is permanently updated according to the diagnosis process results. This information will be useful for making the recovery decisions.

4.4. Step 4: Fault recovery

The final step of fault tolerance is the fault recovery process. The most adapted recovery behavior must be selected from the identified fault and its associated severity information produced in the diagnosis phase. We implement

the modality switching principle, which is a reactive, flexible and efficient way to implement recovery.

The fault recovery must take into account: the severity associated with the detected fault; the robot's current state, i.e. its autonomy level, sub-objective and functioning mode, and the available resources, i.e. the module state (operational or not, or reversible).

The state machine shown in Figure 7 illustrates, for a given autonomy level, the proposed recovery principles (supposing there are only two functioning modes). According to the severity associated to the fault, there may be five generic reactions:

- Reconfiguration (severity = weak): Troubleshooting just locally affects one module, which only requires adaptation of its parameter values. The current functioning mode, sub-objective and consequently autonomy level are preserved.

- Adaptation (severity = medium): A module is out of order and the current functioning mode cannot be maintained at all, but an alternative solution can be found while preserving the current autonomy level. This alternative solution generally leads to degradation of the robot performance, changing the functioning mode but keeping the same sub-objective (i.e. robotic task) and autonomy level.

- Autonomy Adjustment (severity = serious): The faulty functionality is essential for the mission success but cannot be recovered autonomously. Human interaction is needed to locally (detect, understand and solve the problem) or definitively (an operator provides the faulty service) resolve the problem and continue the mission. This reaction requires operational communication.

- Safety Waiting (severity = serious): The fault is serious but the communication functionality is unavailable. No alternative sub-objective functioning mode can be immediately considered, but some essential functional

modules are in a reversible state. The mission is interrupted and the robot is safety parked, awaiting hypothetical recovery of its key functionalities (including communication) to continue its mission.

- Definitive Stop (severity = fatal): No alternative solution can be planned and some of the necessary functionalities are permanently non-operational. The mission must be aborted and the robot safety parked.

At any time, if some modules again become operational via the reversibility principle, the best and most efficient autonomy level and functioning mode will be engaged. The reversibility possibilities are shown with dashed lines in Figure 7.

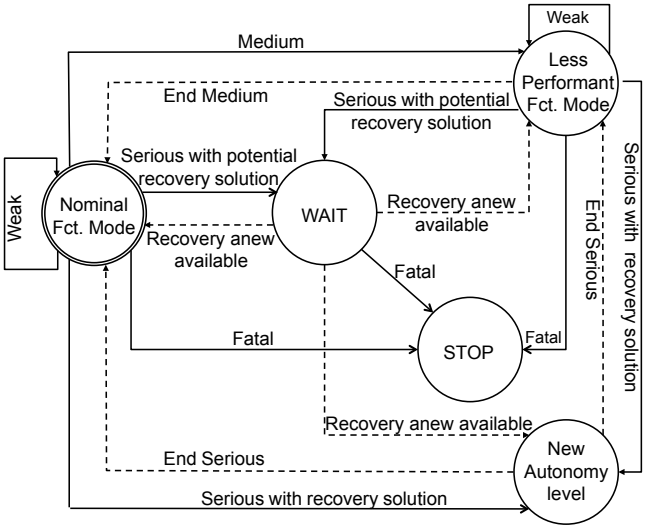

Figure 7: Recovery principle management

Some implementation and experimental results of our fault tolerance approach are summarized hereafter.

5. Fault Tolerant Control Architecture Implementation

The proposed approach was implemented in the COTAMA (COntextual TAsk MAnagement) control architecture [49].

Basically, this modular and component oriented architecture supports middleware which manages the real-time constraints and module interactions. Each module has specific ports used to convey the data and control flows.

COTAMA involves two main layers (white boxes in Figure 8). The executive layer pools the low-level modules implementing robotic and communication functionalities (sensing, control, actuating, etc.). Moreover, it also integrates the scheduler module, which controls the module's execution according to the current sub-objective functioning mode and the real-time constraints. The decisional layer manages the execution of the mission progress using global and local supervisors. The global supervisor decomposes the mission into a set of objectives. A local supervisor splits an objective into sub-objectives that are controlled by the scheduler.

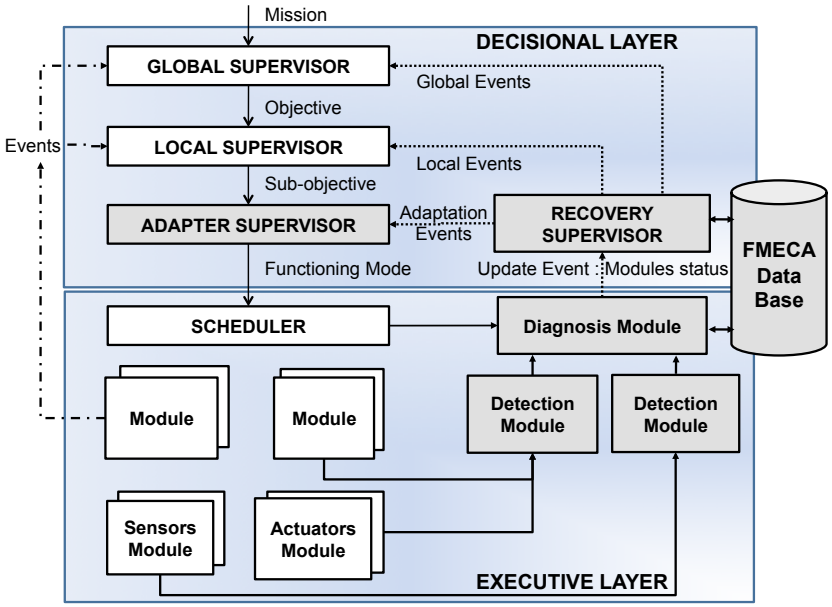

Figure 8: Fault tolerant COTAMA architecture

Based on the previous elements, the COTAMA architectural organization was adapted to integrate fault tolerant principles (grey boxes in Figure 8). In the executive layer, detection modules are added to implement fault detection.

Then the diagnosis module, using the detection information, diagnoses the original fault(s) and the corresponding faulty modules. According to the FMECA database, it also identifies the associated severity of the most relevant fault. Finally, at the decisional layer, the recovery supervisor handles the recovery reaction. According to the robot's state, operational functionalities and severity of the failure, it determines the supervisor to engage depending on the chosen recovery action:

- Reconfiguration is done directly by setting the parameters of one module.
- Adaptation is done by the adapter supervisor, a new supervisor introduced in the control architecture for the functioning mode management.
- Autonomy adjustment corresponds to a change of the current sub-objective by the local supervisor.
- Safety waiting is done using a specific objective; it is managed by the global supervisor.
- Definitive stop is also done by the global supervisor, stopping the current objective and/or aborting the mission.

6. Experimental results

6.1. Experimental context

A conventional office delivery mission using a mobile robot was used to validate the proposed approach. The mission was carried out in a known environment: a map of the experiment area was available. However, the environment was dynamic and unforeseen obstacles could be encountered during the robot's movements.

Experiments were performed with a Pioneer-3DX from MobileRobots with two driving wheels using reversible DC motors. The robot has the following embedded sensors to perceive the environment: sonar and bumpers arrays, and a camera. An embedded laptop hosts the control architecture COTAMA, under

a Linux RTAI real-time operating system, and communicates with the robot microcontroller through a serial connection. It also communicates through a WiFi network with a remote PC, which manages the overall mission, including delivery ordering, path generation and human-robot interactions.

The experimental tests for fault tolerance were conducted using HIL (Hardware In the Loop). This allowed us to control, in a useful and flexible way, the fault injection process, and then to validate all the fault tolerant mechanisms of our architecture under a broad range of fault types, severities and date of occurrence. Indeed, some faults could be difficult to obtain, and it was almost impossible to obtain all of the desired faults in a limited-duration mission. For example, a DC motor problem will be hard to trigger during a mission because its occurrence probability is very low in a protected environment (indoor). Thus the robot actuators were really controlled, but some of the sensors were simulated (e.g. the odometric values were the real ones but the sonar values were simulated).

Figure 9 presents a simplified Petri net (human-robot interaction not specified) describing the autonomous mission progress. It is implemented within the global supervisor through rules using a dedicated logico-temporal vocabulary. A typical sequence of objectives cycle is as follows. Once a delivery mission has been received from the remote PC, the robot moves to the reception point, receives the object to deliver, moves to the delivery point and delivers the object. Throughout the mission, the robot must adapt its behavior according to the encountered problems. Each objective can be carried out in different ways according to the available functionalities. If no alternatives can be found while some modules remain in a reversible state, a wait objective is engaged until an operational state can be recovered (e.g. waiting for the WiFi communication recovery). This behavior was not considered for the reception and delivery objectives due to the associated simplicity. Regardless of the current objective, when a fatal error occurs, the mission must be definitively aborted.

When developing the mission, we principally focused on the moving objective. Three autonomy levels were defined. The first one is the autonomous level,

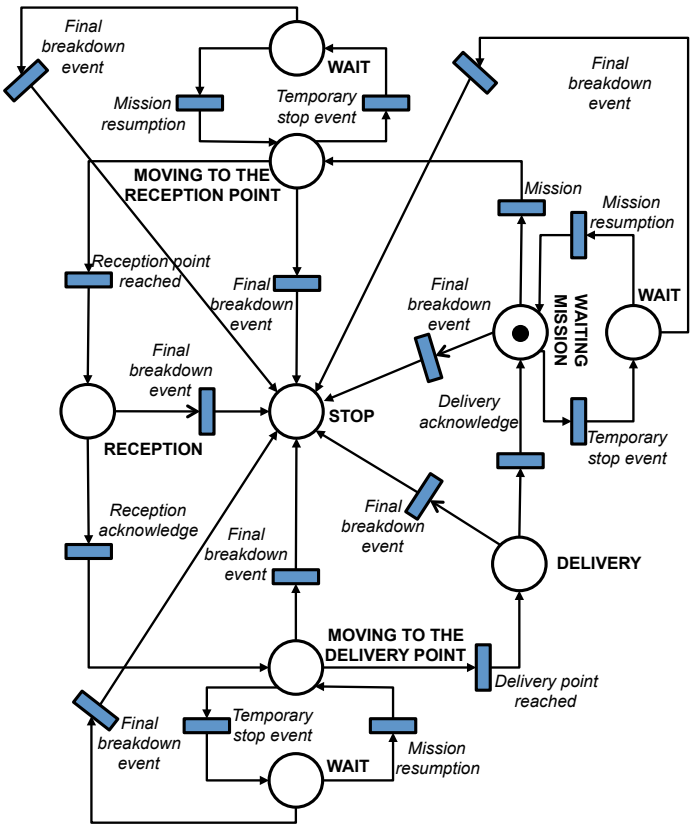

Figure 9: Simplified Petri net of the global supervisor for autonomous behavior

when the robot is able to reach its objective itself and overcome the encountered faults. The two others are teleprogrammed and teleoperated, when the robot needs some human assistance to overcome a fault or compensate for a non-operational functionality. In teleprogrammed mode, human intervention is sporadic, giving the robot a solution, whereas in teleoperated mode it replaces the missing functionality itself.

6.2. Mission implementation: focus on the moving objective

The moving objective consisted of two sub-objectives: path generation, and path following. We focused on this last one. For its implementation, a set of control modules (Table 3) had to be developed for: sensor and actuator management, navigation, guidance (obstacle avoidance) and control; and a set of detection modules for fault detection. The executive layer of the control architecture then consisted, for this sub-objective, of 11 modules implementing the robotic control algorithms (e.g. probabilistic approach for localization with the Monte-Carlo approach or obstacle avoidance using a Deformable Virtual Zone (DVZ) approach or some embedded functionalities (e.g. communication management). One specific module was also integrated to simulate sonar sensors for the HIL experiments, and another one for remote PC communication to allow supervision and the human-robot interaction (HRI). Moreover, 9 detection modules were developed for the detection of one of the 30 different faults identified during the FMCEA analysis. Real-time error detection using the timing check principle predominated (17 of the 30 identified faults were detected with this mechanism). Reasonableness check was also largely developed for sensor, environment perception or energy problems detection (9/30). These two check methods were sometimes used in association with an historical analysis of the data values. With this simple robot, monitoring for diagnosis principle using a model based approach was only developed for DC motor wheel malfunctioning with MMKF (Multiple Model Kalman Filter) (2/30) and particle dispersion detection (1/30). No safety bag checking safety properties were implemented. The last fault was associated with a classical transmission error detection method.

Control module	**Role**	**Detection module**	**Role**
P3D	Robot / control architecture communication interface	BKF	Wheel fault monitoring: Kalman filter [26]
UST	Sonar management	LOC	Localization quality monitoring
ODO	Robot's odometry management	RTC	Real-time fault monitoring
MCL	Monte-Carlo localization (adapted from [50])	OWF	LAN quality monitoring
NAV	Robot's state computing	OSM	Quality monitoring of the SMZ path following
GUI	Path following	ODV	Quality monitoring of the DVZ path following
CTL	Asymptotic control with actuator velocity saturation	ACT	Actuator information monitoring
DVZ	Obstacle avoidance using DVZ [51]	SNS	Sonar information monitoring
SMZ	Obstacle avoidance using SMZ [52]	OP3	Bumpers, robot energy and internal communication monitoring
UDP	Robot / remote PC communication interface		
SIM	Sonar simulator for HIL		

Table 3: Modules for the autonomous path following sub-objective

According to the existing functional redundancy, many functioning modes can be defined for the autonomous path following. The robot's movement can be safe or not if an obstacle avoidance approach is employed. Moreover, the localization scheme can be based on the probabilistic approach or on odometer information.

6.3. Example of a fault tolerant mission

Figure 10 shows an example of an HIL scenario where a robot initially waits for a mission at point A, must recover an object at this point, and must deliver it to office B. All along the recorded trajectory, several points (numbered from 1 to 9) identify relevant events.

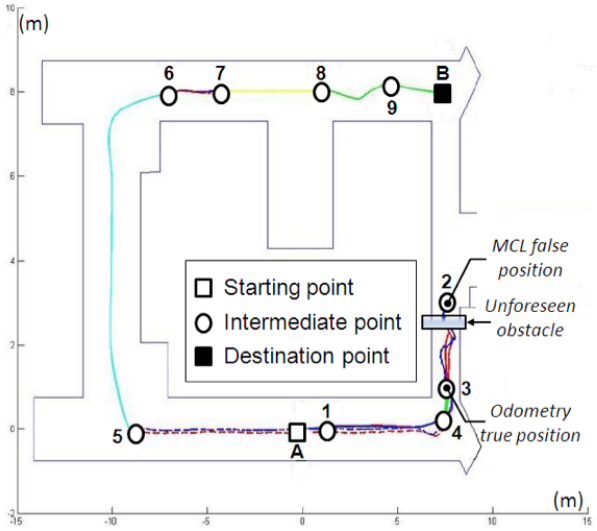

Figure 10: Delivery mission scenario

The control loop execution constraint is set at 0.1 s. When moving, the robot speed is 0.3 m/s in autonomous configuration. The mission duration is 4.68 min and the length of movement is approximately 75 m.

Table 4 summarizes the most relevant information describing the mission from the reception to the destination point. The left part explains, for each point of the mission the robot's current sub-objective and functioning mode and the corresponding autonomy level. The right part describes the normal mission events (normal type) or the fault tolerant events (italic type) leading to the current robot task. In the fault tolerant event cases, the table also specifies the associated characteristics of the detected faults and failures (C: class, T: type, S: severity) and the corresponding recovery action.

Phase : Point	Sub-objective *(functioning mode)*	Aut. level	Relevant event				Recovery and *reversibility*
			Event	F/F char			
				C	T	S	
1 : A	Reception	AUT	Reception acknowledge				
2 : A-1	Moving *(using SMZ and MCL)*	AUT	*Real-time error*	P	T	Weak	MCL Reconfiguration
3 : 1-2	Moving *(using SMZ and MCL)*	AUT	*Localization error*	P	T	Medium	Local adaptation
				P	T	Ignored	
4 : 2-3	Moving *(using SMZ without MCL)*	AUT	*Important backward step*	C	T	Serious	Autonomy adjustment
5 : 3	Human decision	HRI	Teleoperation choice				
6 : 3-4	Teleop. *(using DVZ)*	TOP	Teleoperation				
7 : 4	Human decision	HRI	Teleprogrammation choice				
8 : 4	Human localization	TPG	Teleprogrammation				
9 : 4	Human decision	HRI	Autonomous choice				
10 : 4-5	Moving *(using SMZ and MCL)*	AUT	*Sonar blind*	P	T	Medium	Local adaptation
11 : 5-6	Moving *(using SMZ without MCL)*	AUT	*Sonar default recovery*				*End of medium fault: Local adaptation*
12 : 6-7	Moving *(using SMZ and MCL)*	AUT	*Sonar out of order*	C	P	Medium	Local adaptation
13 : 7-8	Moving (without obstacle avoidance)	AUT	*Bumper crash detection*	P	T	Serious	Autonomy adjustment
14 : 8	Human decision	HRI	Tele-operation choice				
15 : 8-9	Teleop (without obstacle avoidance)	TOP	*Loss of WiFi link*	P	T	Serious	Autonomy adjustment
16 : 9	Waiting	AUT	*WiFi link recovery*				*End of serious fault: Autonomy adjustment*
17 : 9-B	Teleop (without obstacle avoidance)	TOP	Destination point reach (Human decision)				

Table 4: Mission description: relevant information

The mission run is also explained below:

- Phase 1: reception of the object to deliver at point A.
- Phase 2: the robot starts to move autonomously to the destination point. All its actuators, sensors and software modules are operational, thus the

most efficient functioning mode can be employed (using SMZ-type obstacle avoidance and particle filter stochastic localization (MCL)). However, at point 1, a real-time fault of weak severity is encountered due to an excessively high particle number in the localization algorithm. The MCL module needs to be reconfigured, while reducing the number of particles.

- Phase 3: The mission continues until point 2, where a marked difference between the MCL localization and the odometric measurements is observed. This deviation is due to an unforeseen obstacle, which completely blocks the robot's course. Due to its obstacle avoidance functionality, the robot avoids the obstacle and the MCL localization finally loses the real localization. In fact, the diagnostic module identifies two potential origins, i.e. either MCL malfunctioning due to a weak environment disturbance (mismatch between the map and the actual sensor information) or false odometric information induced, for example, by sliding wheels. Since this last fault is supposed to be neglected for short distances within the laboratory, MCL malfunctioning is the more severe fault. Then a sub-objective adaptation is engaged where localization is now based on odometric data.

- Phase 4: Unfortunately, this new position (point 3) is very different from the previous one and an important and unwanted backward step in the path following functionality is detected. The robot is lost and cannot handle the situation itself. The detected fault is thus considered to be serious and a human analysis (human-robot interaction - HRI) is required.

- Phases 5-9: To analyze the situation, the operator chooses to change the autonomy level, teleoperating (TOP) the robot, to observe the environment using the embedded camera. Once the unforeseen obstacle is observed, the operator decides, at point 4, using teleprogrammed (TPG) sequences, to give new information to the robot: the correct localization and a new path to follow. The human also updates the module status database (MCL is operational again). Finally, the operator again starts the nominal moving sub-objective in autonomous mode.

- Phases 10-11: At point 5, a new problem is encountered: due to a too wide corridor configuration, the sonar's range is not sufficient to correctly model the robot environment. So the sonars are considered to be transiently blind for distant objects, but still useful for close obstacles. This transient fault with medium severity makes it necessary to use an adapted moving sub-objective based only on odometric information. Once point 6 is reached, the new corridor configuration makes it possible to detect that the sonars are operational. Thanks to this reversibility, the optimal moving sub-objective can be retried.

- Phase 12: A permanent sonar breakdown is simulated at point 7. Faced with this serious and permanent fault, a local adaptation is engaged where an unsafe displacement without obstacle avoidance is proposed.

- Phases 13-14: The risk is, like at point 8, that the robot bumps into an obstacle. This is a serious fault: no autonomous solution exists so far for the moving sub-objective. An operator decision is needed and an HRI is begun. The operator decides to complete the mission in teleoperation mode without obstacle avoidance.

- Phases 15-17: During teleoperation, a loss of WiFi link is simulated for a few seconds (at point 9). This fault is serious but with a potential recovery solution. Then the architecture switches to the waiting sub-objective, awaiting signal recovery. Once the link is back, the robot's teleoperation is reinitiated until the destination point is reached at point B.

These results demonstrate the efficiency, flexibility and adaptability of the developed fault tolerant control architecture. Throughout the mission, all failures severities and then all recovery mechanisms were investigated (except the hard severity leading to definitive mission abortion), included the treatment of multiple failure occurrences and functionality reversibility. three autonomy levels, two specific modes (HRI and waiting) and five functioning modes (three autonomous and two teleoperated) were successively used to successfully com-

plete the mission.

7. Conclusion

This article investigates a central and critical issue encountered in the development reliable robotic systems - the fault tolerance paradigm. Fault tolerance is conventionally based on three main principles: fault detection, fault diagnosis and recovery. An analysis of existing studies demonstrates that fault tolerance is seldom considered in robotic control architectures. Furthermore, in spite of numerous existing results, the basic fault tolerance principles are usually embedded into the architecture and generally empirically considered. Finally, the different steps (forecasting, detection, diagnosis and recovery) are seldom considered together, while a common design enhances the efficiency of each.

A structured approach to develop a fault tolerant architecture is presented to overcome these limitations. Based on a fault forecasting phase obtained with a FMECA analysis, each fault tolerant principle is reified within the control architecture: dedicated detection modules are added for failure detection, signature analysis is developed for fault diagnosis, and finally fault recovery mechanisms are implemented based on modality switching, while integrating autonomy level adaptation.

Systematization of the FMECA analysis is essential for fault tolerance efficiency. Even if it could be considered as a heavy phase, the advantage of our solution is that the FMECA tables are developed at the module level. The module codes and module FMECA tables could be kept on the shelf to be readily available. Our detection concept allows adaptation of the observation load, while executing or not the detection modules. A tradeoff must be obtained depending on the embedded resources and the reliability needed. The diagnosis phase is also well adapted to the embedded and real-time context: the signature method is simple and efficient. Furthermore, we complete it with specific mechanisms to manage the multiple fault and failure configurations. Finally, we proposed and implement a wide range of recovery solutions with execution

control and human interaction, while also integrating automatic recovery of the nominal behavior in case of transient faults.

This approach was successfully validated on a mobile robot for a delivery mission. The executive layer includes twelve control modules and nine detection modules, and five high level modules are dedicated to mission and fault tolerance management. The experimental mission allows validation of all fault tolerance mechanisms detecting and diagnosing all possible types of faults and engaging all possible recovery solutions. But the validation work is not finished: it is now necessary to test our methodology and architecture in more complex and realistic missions, and in a more unstructured environment, as encountered outdoor.

Indeed, fault tolerance makes it possible to identify and react to relevant faults that can affect autonomous systems. Most usual sensing, software or hardware faults can be reasonability anticipated. However, it is more difficult to consider the environment non-determinism through robot action or interaction. Using robots in the field means being able to cope with harsh experimental conditions. The development of more effective detection techniques is thus crucial. Moreover, to deal with fault uncertainty, statistical identification methods should certainly be considered. Such an analysis of occurrence probability could also help to refine the detection load accuracy. Furthermore, to extend the recovery efficiency and try to overcome the modality principle limits of the execution control, it would be interesting to integrate re-planning capacities in the fault tolerant approach. Multi-robot approaches could also be considered, using different robots as functional redundancy for fault tolerance.

Finally, fault tolerance needs could be dynamically adapted, depending on the embedded resources and reliability needs. Our fault tolerant architecture disposes of mechanisms that allow this adaptation. It is however necessary to develop an adaptation strategy, with resource estimation techniques, and with specific metrics to evaluate the exact reliability of a fault tolerant system within a given environment.

References

References

[1] N. Tomatis, G. Terrien, R. Piguet, D. Burnier, S. Bouabdallah, R. Siegwart, Design and system integration for the expo.02 robot, in: proc. of the Workshop on Robots in Exhibitions, IEEE/RSJ International Conference on Intelligent Robots and Systems, Lausanne, Switzerland, 2002.

[2] G. Griffiths, N. Millard, S. McPhail, P. Stevenson, P. Challenor, On the reliability of the autosub autonomous underwater vehicle, International Journal of the Society for Underwater Technology 25 (4) (2003) 175–184.

[3] J. Carlson, R. Murphy, How UGVs physically fail in the field, IEEE Transactions on Robotics 21 (3) (2005) 423 – 437.

[4] D. T. Phuoc-Nguyen Nguyen-Huu, Joshua Titus, G. Ulsoy, Reliability and failure in Unmanned Ground Vehicle (ugv), Tech. Rep. GRRC Technical Report 2009-01, Ground Robotics Research Center, University of Michigan (2009).

[5] Coordination action for robotics in europe (CARE) and european robotics technology platform (EUROP), Robotic visions to 2020 and beyond, the strategic research agenda for robotics Europe (2009).

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

[7] V. Verma, R. Simmons, Scalable robot fault detection and identification, Robotics and Autonomous Systems 54 (2) (2006) 184 – 191.

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

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

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

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

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

[13] R. J. Patton, Fault-tolerant control: the 1997 situation, in: proc. of the IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes, Vol. 2, Kingston Upon Hull, UK, 1997.

[14] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, H. Das, The claraty architecture for robotic autonomy, in: proc. of the 2001 IEEE Aerospace Conference, Big Sky Montana, USA, 2001.

[15] J. Borrelly, E. Coste-Manière, B. Espiau, K. Kapellos, R. Pissard-Gibollet, D. Simon, N. Turro, The ORCCAD architecture, International Journal of Robotics Research 17 (4) (1998) 338–359.

[16] R. Alami, R. Chatila, S. Fleury, M. Ghallab, F. Ingrand, An architecture for autonomy, International Journal of Robotics Research 17 (1998) 315–337.

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

[18] G. Steinbauer, M. Mörth, F. Wotawa, Real-time diagnosis and repair of faults of robot control software, in: RoboCup 2005, Springer-Verlag, Berlin,

Heidelberg, 2005, Ch. Real-Time Diagnosis and Repair of Faults of Robot Control Software, pp. 13–23.

[19] 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, in: 18th European Conference on Artificial Intelligence, ECAI, Patras, Greece, 2008.

[20] F. Py, F. Ingrand, Real-time execution control for autonomous systems., in: proc. of the 2nd European Congress ERTS, Embedded Real Time Software, Toulouse, France, 2004.

[21] Z. Duan, Z. Cai, 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, ICRA, Barcelona, Spain, 2005.

[22] M. Brandstötter, M. W. Hofbaur, G. Steinbauer, F. Wotawa, Model-based fault diagnosis and reconfiguration of robot drives, in: proc. of the IEEE/RSJ International Conference on Intelligent RObots and Systems, IROS, San Diego, USA, 2007.

[23] J. Mendoza, M. Veloso, R. Simmons, Mobile robot fault detection based on redundant information statistics, in: IROS Workshop on Safety in Human-Robot Coexistence and Interaction, Vilamoura, Portugal, 2012.

[24] J. Mendoza, M. Veloso, R. Simmons, Motion interference detection in mobile robots, in: proc. of International Conference on Intelligent Robots and Systems, IROS, Vilamoura, Portugal, 2012.

[25] A. Aguiar, Multiple-model adaptive estimators : Open problems and future directions, in: proc. of the European Control Conference, ECC, Kos, Greece, 2007.

[26] S. I. Roumeliotis, G. S. Sukhatme, G. A. Bekey, Fault detection and identifcation in a mobile robot using multiple model estimation, in: proc. of

the IEEE International Conference on Robotics and Automation, Vol. 3 of ICRA, Leuven, Belgium, 1998.

[27] R. Volpe, I. A. D. Nesnas, T. Estlin, D. Mutz, R. Petras, H. Das, Claraty: Coupled layer architecture for robotic autonomy, Tech. rep., NASA - Jet Propulsion Laboratory (2000).

[28] A. Ranganathan, S. Koenig, A reactive robot architecture with planning on demand, in: 2003 IEEE/RSJ International Conference on Intelligent RObots and Systems, IROS, Las Vegas, Nevada, USA, 2003.

[29] B. Morisset, G. Infantes, M. Ghallab, F. Ingrand, Robel: Synthesizing and controlling complex robust robot behaviors, in: proc. of the 16th European Conference on Artificial Intelligence, ECAI, Valencia, Spain, 2004.

[30] S. Lemai, F. Ingrand, Interleaving temporal planning and execution: IXTET-EXEC, in: Workshop on Plan Execution, ICAPS, Trente, Italie, 2003.

[31] R. P. Goldman, D. J. Musliner, M. S. Boddy, K. D. Krebsbach, The circa model of planning and execution, in: Working Notes of the AAAI Workshop on Robots, Softbots, Immobots : Theories of Action, Planning and Control, Providence, Rhode Island, USA, 1997.

[32] S. Lemai, F. Ingrand, Interleaving temporal planning and execution in robotics domains, in: proc. of the 19th National Conference on Artifical Intelligence, AAAI'04, San Jose, California, USA, 2004.

[33] T. Estlin, R. Volpe, I. Nesnas, D. Mutz, F. Fisher, B. Engelhardt, S. Chien, Decision making in a robotic architecture for autonomy, in: proc. of the International Symposium on Artificial Intelligence, Robotics and Automation for Space, i-SAIRAS, Montreal, CA, 2001.

[34] A. Avizienis, Toward systematic design of fault-tolerant systems, Computer 30 (4) (1997) 51–58.

[35] S. Mercier, F. Dehais, C. Tessier, Adjustable autonomy without levels, in: NATO SCI-202 Symposium on Intelligent uninhabited vehicle guidance systems, Neubiberg, Germany, 2009.

[36] C. Miller, H. Funk, P.Wu, R. Goldman, J. Meisner, M. Chapman, The playbook ^{TM} approach to adaptive automation, in: proc. of the Human Factors and Ergonomics Society's 49^{th} Annual Meeting, Orlando, FL, USA, 2005.

[37] D. J. Bruemmer, D. D. Dudenhoeffer, J. L. Marble, Dynamic autonomy for urban search and rescue, in: AAAI Mobile Robot Competition and Exhibition (technical report), no. WS-02-18, 2002.

[38] B. C. Zimmel, M. T. Long, J. Carlson, R. R. Murphy, Distributed error handling and hri, in: proc. of the IEEE International Conference on Robotics and Automation, ICRA, 2004.

[39] R. R. Murphy, D. Hershberger, Handling sensing failures in autonomous mobile robots, International Journal of Robotics Research 18 (4) (1999) 382–400.

[40] Guide to failure modes, effects and criticality analysis (FMEA and FMECA), British Standards Institution Group - BS 5760-5, 1991.

[41] V. Selvaraj, Failure mode analysis of an autonomous guided robot using JDBC, Master of science thesis, College of Engineering, University of Cincinnati (2000).

[42] J. Guiochet, C. Baron, UML based risk analysis - application to a medical robot, in: proc. of the 5th International Conference on Quality Reliability and Maintenance, Oxford, UK, 2004.

[43] Y. Maddahi, A. Maddahi, S. M. H. Monsef, Design improvement of wheeled mobile robots: Theory and experiment, World Applied Sciences Journal 16 (2) (2012) 263–274.

[44] Integration definition for function modeling (IDEF0) (December 1993).

[45] K. Ishikawa, What is Total Quality Control? the Japanese Way, Prentice-Hall, 1985.

[46] M. BARBIER, J.-F. GABARD, D. VIZCAINO, O. BONNET-TORRS, ProCoSA: a software package for autonomous system supervision, in: proc. of the 1st National Workshop on Control Architectures of Robots, Montpellier, France, 2006.

[47] X. Olive, FDI(R) for satellites: How to deal with high availability and robustness in the space domain?, International Journal of Applied Mathematics and Computer Science 22 (1) (2012) 99–107.

[48] J. Gertler, Fault Detection and Diagnosis in Engineering Systems, CRC Press, 1998.

[49] A. E. Jalaoui, D. Andreu, B. Jouvencel, Auv control architecture with management of embedded instrumentation, in: proc. of the 4th IFAC Symposium on Mechatronic Systems, Heidelberg, Germany, 2006.

[50] L. Zhang, R. Zapata, P. Lépinay, Self-adaptive Monte Carlo localization for mobile robots using range finders, Robotica 30 (2) (2012) 229–244.

[51] L. Lapierre, R. Zapata, P. Lépinay, Combined path following and obstacle avoidance control of a wheeled robot, The International Journal of Robotics Research 26 (4) (2007) 361–375.

[52] L. Lapierre, R. Zapata, A guaranteed obstacle avoidance guidance system: The safe maneuvering zone, Autonomous Robots 32 (3) (2012) 177–187.

Crestani Didier is Professor in the French University of Montpellier 2. He belongs to the Explore team of the robotic Departement of the LIRMM laboratory. He obtained his PhD in computer science in 1991. His areas of interest are software real-time control architecture and fault tolerance for mobile robotics and environment exploration.

Lionel Lapierre received his Ph.D. degree in Robotics, from the University of Montpellier 2, Montpellier, France, in 1999. Then, he joined the team of Professor A. Pascoal within the European project FreeSub for three years. Since 2003, he has been with the Underwater Robotics Division of the EXPLORE team, Laboratoire d'Informatique, de Robotique et de Microelectronique de Montpellier (LIRMM), Montpellier, France.

Karen Godary-Dejean received her PhD degree in Computer science from INSA of Lyon, France, in 2004. This thesis was an industrial collaboration with the Renault Technocentre research center (Guyancourt, France). She joined the LIRMM (UMR 5506) in 2005 to be an Assistant Professor at the University of Montpellier 2. She currently belongs to the EXPLORE team of the Robotics Department and collaborates with the INRIA DEMAR team. Her research interests are related to dependability for embedded systems, including fault tolerance for robotics control architecture and formal modeling and validation for critical discrete event systems.

Photo_DC

Photo_LL

Photo_KGD