Faculty of Engineering of the University of Porto

Safety features for unmanned maritime vehicles

André Manuel Matos Leite

This Dissertation was developed for the
Integrated Master's in Electrical and Computer Engineering

Major: Automation

Supervisor: Aníbal Castilho Coimbra de Matos

Co-supervisor: Andry Maykol Gomes Pinto

June 25th 2018

Abstract

The continued development of mobile robots (MR) must be accompanied with an increase in robotics' safety measures. Not only must MR be capable of detecting and diagnosing their faults, they should be capable of understanding when the dangers of a mission, to themselves and the surrounding environment, warrant the abandon of their endeavors. Analysis of fault detection and diagnosis techniques helps shed light into the challenges of the robotic field, while also showing a lack of research into autonomous decision-making tools. This paper proposes a new skill-based architecture for mobile robots together with a novel risk assessment and decision-making model to overcome the current difficulties currently felt in autonomous robot design.

Acknowledgments

Although I always look to the past in search of guidance, I always lived in the present thinking of the future. I have no doubt that there are many in my past I should thank for being where I am, people I knew, but no longer do. Perhaps some of them, it would be unfair to forget in these words, but should I remember all, maybe another thesis should be written only on that. Therefore, my thanks will go to those in my Present:

To my parents, Manuel and Idaline Leite and my sister, Patrícia Leite, to my nephews, Duarte, Pedro and Tiago… The former three for all the support, the latter for always making me smile.

To my advisor, Aníbal Matos, and especially my co-advisor, Andry Pinto, for all the knowledge and instruction they gave me over the last years.

To three of my friends, Pedro Rocha, Guilherme Braz and Tiago Leite. The first and second, having known me since my times in communication science, having indulged my “madness” when needed and stopped it when necessary, for all we lived together through the times… The third, whom I’ve met in Engineering, and without whom I doubt I’d have “survived” until the end of this work.

Last, but no way least, to Samira Traub: When I had lost hope, you brought it back to me, when I saw no future you returned me to the path, when my strength failed me you gave me your own… Saving me from myself, my stressful nature, you have been the rock that as held me… I’ll return the favor soon… I love you…

I hope I did not forget none, there are truly too many… These are the first that came to mind, maybe because some are more recent, and others are always there, maybe because they were the most important in this last step… Too all, mentioned and not mentioned, my many thanks for helping me find my path again.

Table of Contents

Table of Figures

List of Tables

List of Acronyms

INESC TEC	Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência
UMV	Unmanned Maritime Vehicle
NASA	National Aeronautics and Space Administration
TMR	Triple Modular Redundancy
PB	Primary/Backup
PE	Primary/Exception
UAV	Unmanned Aerial Vehicle
AUV	Autonomous Underwater Vehicle
SSMC	Single Sided Matrix Converter
FTC	Fault Tolerance Capability
FPGA	Field Programmable Gate Array
FTS	Fault Tolerant System
CAN	Controller Area Network
MR	Mobile Robots
EU	European Union
ISO	International Organization for Standardization
IEC	International Electrotechnical Commission
CENELEC	European Committee for Electrotechnical Standardization
FDD	Fault Detection and Diagnosis
PS	Parity Space
HMM	Hidden Markov model
PF	Particle Filters
OOM	Observable Operator Model

Chapter 1

Introduction

1.1 - Scope

Robotics can be seen as a scientific art, a branch of engineering that aims to develop automated tools that can mimic human ability to manipulate the environment. Such systems, to achieve this goal, require various components, which can be divided in sensor, power source, actuators, controllers and a physical structure, a body, that can house all these parts [1], akin to how the human body houses various organs, bones and muscles to allow the perceiving and the manipulation of their surroundings. This path diverges into two distinct categories, however, as robots might be used as tools, operated by humans, or they might be made to be capable of autonomous operation, going as far as to replace humans in some jobs. Nevertheless, not only replacement is of concern, but also the potential for robots to completely surpass humanity. The European Union Parliament ordered a study, published in 2017, that comes to these conclusions [2]. In an age robots are becoming so important in our daily lives, it is important to find ways to assure they are reliable and safe.

One of the questions that arises with autonomous mobile robots, and therefore needs to be focused on, is how to keep robots, operating with little to no human interaction, from causing damage to people or the surrounding environment? One possible solution is the use of a safety monitoring system, capable of defining when, given a mission, the robot has the necessary conditions to follow it through or when it should cancel the operation and, for example, return to base for repairs. For this decision to be correct though, it can't be taken blindly. There must be knowledge of many variables, like the state of sensors, actuators and the decision layer. This is where the importance of fault diagnosis arises, since it provides a means of both understanding when a fault occurs and what subsystem is affected [3]. It is also important to keep in mind that most fault tolerance principles are mixed with the control architecture [4]. This lack of separation is not the best approach, since it implies more complicated Hardware and Software to not only perform the control and fault tolerance procedures, but also integrate them together without degrading both. One of the key challenges in the development of mobile robots that can self-recover from damage, continuing their mission, is the better understanding of the recovery strategies, failures and their interaction. Namely, it is important to understand, when many strategies are possible, which one is the best [5]. Nevertheless, autonomous mobile robots (MR) have clear advantages when compared with manned vessels. The lack of a human crew allows for more hazardous missions to be performed, reduces maintenance costs and increases human safety. Their lower weight and dimensions confer better maneuverability and, in the case of maritime MR it allows to navigate areas which are barred to bigger vessels, like shallow waters. Considering these advantages, it is natural that there is continuous pressure from various communities to develop fully autonomous MR, particularly maritime MR [6]. The current situation in what robotics is concerned can be resumed in the word safety. Robots need to be as safe as possible to be trusted with autonomous missions. This means they must not create unacceptable risks for humans or themselves, which implies maintaining their function for specific periods of time, without changing their behavior, significantly, due to noise and being able to maintain their abilities after partial damage. It is not a matter reserved for MR, or even for autonomous robots. Every field of engineering can benefit from these characteristics. Every kind of robot requires these characteristics. In the medical field, operated robots must be safe to be proper tools of healing. In the aeronautical field, systems must be safe in order for airplanes to maintain their status as a proper means of transportation of goods and people. In MR, its important is even greater because of the increasing demand and need for full autonomy in hazardous environments. An autonomous robot, moving and operating in a nuclear power plant, if it is not safe could even go as far as produce a meltdown with the consequences known from past events (like Chernobyl or Fukushima).

1.2 - Motivation

Civilization needs sets of rules to work properly, one of the reasons being that they are needed to try to guarantee people's safety. The same has to apply to our technology, as evidence by the mentioned European Union (EU) ordered study [2]. Although the typical response is to legislate after it is shown the societal or technological innovation require new laws, in the case of robotics and artificial intelligence the discussion is already starting. Although there appears to be some standstill over what to even legislate about robots, including considerations to if intellectual property can apply to robots without given them legal person status, there are more direct discussions, like those concerning liability over damage caused by the autonomous robots actions. But even on this topic there doesn't seem to be a final recommendation. The possibilities of either separating autonomous robots from traditional robots or keeping them ruled by the same laws is a consideration that itself has not been settled on.

Even with all the indecisions in the study above mentioned, one thing remains a certainty. "We need to build systems that will acknowledge the existence of faults as a fact of life" (KOREN and KRISHNA, 2007) [7], not because we should allow system faults to remain unchecked, but because only the acceptance of their existence, and therefore of the systems' weaknesses, will enable us to correctly investigate:

- What causes these faults: it is equally important to know which the most usual and hazardous increasing faults are, for the design and development stages, and which ones are occurring during a robot's operation, for better application of fault tolerance, monitoring or supervision techniques;
- What are their predictable consequences: will a fault in a motor immediately cause the robot to have steering difficulties? Do they immediately endanger the mission? Does the risk increase to the point that it surpasses the benefits? If we can predict the consequences, it is easier to avoid hazardous scenarios;

- What unpredictable consequences might exist: if it is possible that some extreme situation, not thought probable or likely could occur. Does a fault in a specific sensor increase the risk of malfunction that affects the overall performance of behavior, do some unlikely circumstances turn a minor fault into a failure and a fatal consequence?

In short, recognizing the existence of the problem allows us to try and design tools and architectures that will prevent harm to the system, to the environment or, more importantly, to the people that interact, directly or indirectly, with the robotic system. It is also an important issue that faults can happen both in Software or Hardware. In fact, according to [8], Software faults are more frequent. It can also be argued that, if a Hardware fault goes undetected, Software that is working properly might be cause of harm because of misinformation introduced in the decision process.

Among the various international standard organizations, ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission) are of particular interest for the development of robotics. Founded in 1946, ISO claims to “facilitate the international coordination and unification of industrial standards”. IEC, founded in 1906, emphasizes that “millions of devices that contain electronics, and use or produce electricity, rely on IEC International Standards and Conformity Assessment Systems to perform, fit and work safely together”. Both organizations have standing agreements with the EU own standard organization, CENELEC (European Committee for Electrotechnical Standardization), that give precedence to the work developed at ISO (Vienna Agreement [9]) and IEC (Frankfurt Agreement [10]). The IEC 61508 standard is a good basis for a set of rules to govern the field of robotics because, although thought as a generalization to “all industry sectors”, it was meant to work as a draft for future works aimed at specific sectors and it is already being accepted by “safety equipment suppliers and users” [11]. Despite this, there is still not a universal safety standard that can be applied to all robotic fields directly, creating a setback for service robot companies, since while developing their products they end up following robot development and certification procedures as if their products were the same as any common household appliance, like a refrigerator. There are, however, a few other standards for safety and a few for specific applications of service robotics. It might be, therefore, possible to extract some guidelines from these standards:

- ISO 12100:2010 - Although it does not cover risk to the environment, including property, it gives guidance on risk assessment and reduction in the design of machinery [12];
- ISO 13849-1:2015 – Despite having a few limitations concerning safety functions and performance levels, it covers the safety-related parts of control systems and it includes guidance for software [13];
- IEC 62061:2005 – It is a specific implementation of IEC 61508 targeting machinery. Approaches various stages of the life-cycle of safety-related electrical, electronic and programmable electronic control systems [14];
- ISO 10218:2011 – This standard applies to industrial robots only, but the safety principles can be used in service robots that share a few characteristics. Part 1 focuses “safe design, protective measures and information for use of industrial robots”, while part 2 is more concerned with the integration of robot systems [15,16];

- ISO/TS 15066:2016 – In the ISO website it shares a similar description with ISO 10218-1:2011, meaning it is also a standard that applies to industrial robotic applications, but the safety principles can apply to other systems. Focuses collaborative robots [17];
- ISO 13482:2014 – Completely dedicated to robotic applications for personal care, ISO 13482 is dedicated to the physical interactions between humans and robots and includes hazardous situations that involve property and animals [18].

The previously mentioned standards, although they can provide guidelines, are still far from providing a unified answer to robotics' challenges, as even in the case of IEC 61508, although a good general principle, does not provide rules that can be accepted blindly in every single application. IEC 61511, for example, is a standard that implements IEC 61508 in a process industry context [19], and the already mentioned IEC 62061 is also a specification of IEC 61508. It is not farfetched to believe that other standards will continue to appear to specifically

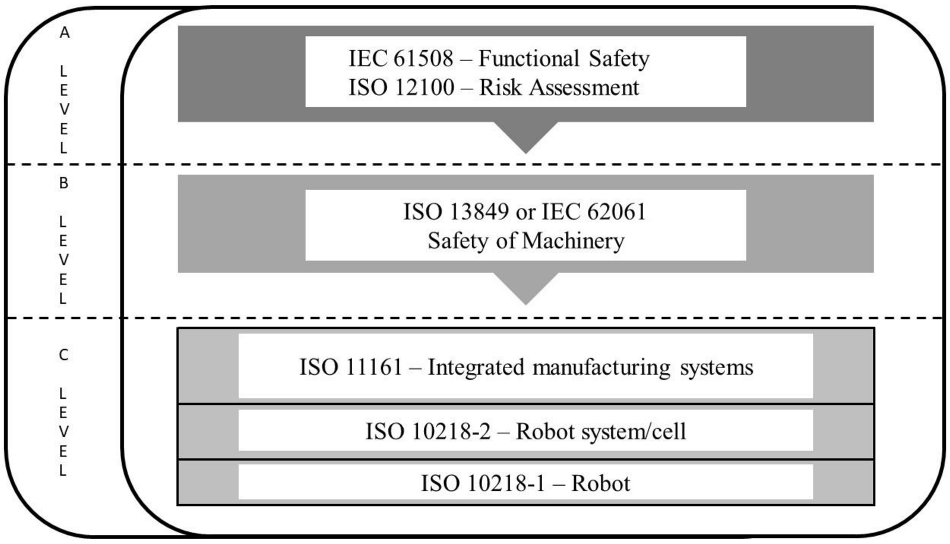
A LEVEL
IEC 61508 – Functional Safety
ISO 12100 – Risk Assessment
B LEVEL
ISO 13849 or IEC 62061
Safety of Machinery
C LEVEL
ISO 11161 – Integrated manufacturing systems
ISO 10218-2 – Robot system/cell
ISO 10218-1 – Robot

Figure 1: Safety standards perceivable hierarchy

address the various fields of engineering. Especially considering that, even though development of fault detection and fault diagnosis has been traced back to the early 1970's [20], there are still accidents with industrial robots or, in more recent projects o autonomous mobile robots, a fatal accident has occurred involving a self-driving car test [8] and Despite the importance of the IEC 61508 standard, literature based on it seems to indicate a much higher concern with the average probability failure of components or with the probability that a failure will result in a safe state [11,21].

1.3 - Objectives

In this work, the main concern is what to do when a fault occurs, which means that the questions that need answering are not "how likely a fault is to occur", but rather "what the decision should be now that a fault has occurred or has been detected". This decision cannot be taken lightly, because there are many aspects to consider: How potentially dangerous is the mission? How much of that risk is directed towards the robot, the environment or people? Is the mission critical enough that the cost of a lost robot is acceptable? What role does the faulty component play? Is the fault permanent or transient? All these questions might need to be considered when an operator makes such a decision, but when a system is either designed or adapted to decide on its own, the same questions may need to be considered. Although it isn't an easy task to fully develop an universally applicable solution, this research aims to be a general guide for the creation of a safety oriented decision-making model. To achieve this final goal, it is important that this work achieves the following objectives:

- Research current safety trends – To understand be problem and the impact that the proposed work, the decision-making model, can have in the industry;
- Investigate the state of fault analysis – namely, what centralized and decentralized approaches exist to fault handling, what types of tools are used (detection, diagnosis, tolerance, monitoring, supervision, etc.);
- Propose a Skills Architecture - capable of building highly complex operations by a systematic aggregation of low complexity blocks, using the concept of primitives, skills and tasks, therefore allowing separation between operation and safety procedures;
- Propose a Safety Monitoring Model - using a mathematical function to determine the risk using the safety state, which is defined by the fault processes affecting the skill;
- Propose a Decision-making model - which will receive the risk evaluation as an input and classify the situation, and therefore decide the action to take, as an output;
- Foster national scientific investigation – translated in the production of an article to be published in a journal of the field of robotics [22].

1.4 - Organization

The rest of the manuscript is organized as follows:

- Section 2 provides an overview of current advances in safety measures, especially fault detection and diagnosis;
- Section 3 presents the system architecture, the risk assessment and the decision-making models;
- Section 4 exemplifies how to apply the presented model on a real system, including description of a first integration with SkiROS, a skill-based control platform [23];
- Section 5 concludes this manuscript.

Chapter 2

State of the Art

Research on fault diagnosis and fault tolerance directed at mobile robots is a recent development. Most literature on this subject is found in the 2010s and, although some research and mentions to mobile robots can be found in earlier years, not earlier than the year 2000. Prior research exists, however, on safety and reliability of engineering systems. Despite the lack of literature concerning strategies aiming fault tolerance, monitoring and supervision of the decision layer, the research into the hardware strategies can be used as a source of reliable introduction to safety features in robotics.

2.1 – Overview

Both in [20] and [24] we can find compilations of some developments in fault diagnosis. Through analysis of these works, it becomes evident that some techniques are not feasible for integration in mobile robots or that their full application can become impractical, either because of high amounts of processing power or because they are valid during design only. Tools like FMECA (Failure Mode, Effects and Criticality Analysis) and Fault Trees involve the creation of, respectively, tables and graphics to list faults, their causes and consequences and both description mention the designer of the system. Detection methods were mostly model-based, which may not be easy to create for already existing systems. The first set of methods approached in [20] are, indeed, the model-based fault detection which are described as being based on the generation of residuals for whichever type of model type is being used (State and output observers, Parity equations or Identification and parameter estimation). As it might be expected from model-based approaches, the way of detecting faults in these methods ends up being the comparison between the behaviors of the robots and their models in search for deviations. However, when the second set of methods are presented, which are already in the realm of fault diagnosis, the author immediately jumps into probabilistic methods, artificial neural network or fuzzy clustering. It is important to note that, analyzing the rest of the document, in particular the appendix, it seems that in [20] mobile robots were either only shortly considered, or not considered at all. This assumption gains more strength when reading in combination with [24], which is separated by two years only, as this one clearly emphasizes industrial robots, although the authors also state that the techniques covered are not limited to only one type of robotic system. Concerns about model-based methods and residuals are also expanded. The high dependence on sensor information is presented as a problem, since sensors are not perfect, producing many false positives if the methods are used in their purest form, i.e. without additional techniques that can compensate or adjust for the imperfections. Detection, in this paper, is coupled with Tolerance in the sense that both require, at least, some redundancy. Although redundancy is mainly based on hardware redundancy, both by the usage of component multiplication or through the increase of the needed degrees-of-freedom, the authors do address the problem with robotic redundancy, which is the limits each application presents, as it is mentioned the size and power consumption limits on medical application robots, for example. Despite these obvious potential problems presented by these techniques, the authors did present some of the possible answers to improve them. Among them, applying thresholds to limit false detection caused by sensor and modeling imperfections is one of the most interesting. First, because even though the simple operating principle of this technique (giving a tolerance to a variable's maximum and minimum values to compensate for the noise a real scenario introduces to the modeled expectation) doesn't properly solve the issue by itself, it is possible to create dynamic versions that can be applied to the ever changing environment faced by MR. The second reason is that limitation by thresholds is such an important technique that it is mentioned by Crestani et al. (2015) [4] as one of the basic mechanisms that is used to achieve FDD (fault detection and diagnosis) in autonomous robots. In fact, Isermann (2008), when approaching mechatronic systems [25], identifies thresholds (named limit value checking and plausibility checks) as a simple and classic approach to fault diagnosis. The main difference that [4,25] and [24] offer in the use of this tool, in that the former identify it as a valid FDD technique, while the latter only refers to it as a measure of control for the model-based FDD methods. Indeed, this does not seem to be even considered by Isermann (2008), as when headdresses the importance of model-based methods he sees them as alternatives (to threshold usage) which do not possess the limitations of the methods they replace. Moving into reliability and fault tolerance, he refers FMEA (Failure Mode and Effect Analyses) and fault-tree analysis as essential tools that must be used on the design stage so to compensate the lower reliability of electronics, but that the fault-tolerance itself will usually require the usage of hardware redundancy.

An area where safety is essential, like aeronautics, also references redundancy, both hardware and software, as techniques essential for their flight control systems. The heavy redundancy of this systems, however making them extremely safe, imposes high costs, not only financial, but also in the dimensions, power consumption and wiring, prompting Sghairi et al. (2008) to develop a new architecture[26]. In describing fault tolerance applied to flight control systems, besides the Hardware and Software redundancies already mentioned, it is also mentioned the concept of segregation, i.e. the isolation of the redundant architecture elements so that there is always at least one active element (actuator controlling a surface and computer controlling an actuator) always exists.

Generic solutions towards improvements to fault tolerance can be seen that also end up adding to the fault detection field, as the former usually needs the latter to happen. An approach based on sensor-actuation channel switching and dwell time is a good example of this since, even though it is a detailed explanation of the mathematical model that provides the tolerant scheme, it first presents the mechanism to detect [27]. Another example, albeit less obvious, can be seen on an approach to state synchronization after a partial reconfiguration of fault tolerant systems using Field Programmable Gate Array (FPGA) circuits. Even when discussing the fault tolerance method of Triple Modular Redundancy (TMR), it is mentioned that the reconfiguration only occurs after the fault is detected by the failure of one of the redundant circuits [28].

More recent compilation works, specifically directed at mobile robots, continue to cite model-based methods of fault detection as being the mostly used in robotics. Nevertheless, there is mention to the lack of effort of incorporating fault detection and diagnosis, or even to progress in the field of fault tolerance concerning Unmanned Surface Vehicles (USVs) actuators or communication systems [6]. Even in the case of Unmanned Ground Vehicles (UGVs), both

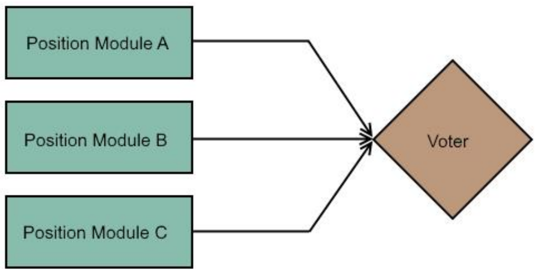

Figure 2: Triple Modular Redundancy for a position system.

military and rescue, an analysis showed a problem with mobile robots reliability [4]. Also, there is mention to the lack of studies focusing the impact that a decision layer on mobile robots has on engineering system's usual safety measures [8]. And yet, even though these critics exist, most consulted authors maintain the importance of the model-based models and of the use redundancy on the software level. There is, however, the introduction of data-based methods as well. Model-based methods are once again described as mathematical models of the system, with faults being detected by the real system deviating from the model system behavior. There is also a mention the extension of these methods with the inclusion of mechanisms to estimate the probability that conclusions are correct. Data-based are described as models that don't require prior knowledge of the system, but rather large amounts of historical data that allow the creation of a knowledge base [4,8].

2.2 – Non-robotic Fields

In the engineering fields outside the robotic realm, there are a few interesting approaches to fault tolerance, which despite not being the purpose in this case, were nonetheless consulted in search of ideas that could be found to the fault monitoring and supervision issues. Considering the safety that aircraft requires, there was research into work on an Adaptive Fault Tolerance solution for dynamic systems, a solution that was evaluated through AWACS early warning aircraft. González et al. (1997) [29], in their endeavour in the aviation field, present a few notions and fault tolerance techniques that can be of use in the study of safety in robotics, beginning with their premise that static fault tolerance (expressed in the form of software redundancy in this instance) is not ideal in critical applications that must operate in real-time. Since it is introduced in the design stage in a very specific way and number, it can't be assured to function properly in certain circumstances, namely when a higher load of dynamic inputs than expected floods into the system, the static techniques might not be able to deal with all information causing highly critical tasks to be dropped, while less or not critical tasks are executed, perhaps even without fully using their capacity. The authors therefore propose an approach that can introduce real-time scheduling to allocate more redundancy (i.e. techniques with more resources) as needed, especially to the more critical tasks. A basic few techniques were presented, including TMR, and the Primary/Backup and the Primary/Exception. These latter techniques involve using a primary computation and, if a fault is detected, respectively, a backup copy or an exception handler must act as redundancy. However, it is never explained how this fault detection is done [29]. Another work, directed at fault tolerance, through multiprocessor architecture, in real-time microcontrollers, does hint at the same that was detected in the previously mentioned work: TMR, or redundancy in general, does offer an effective means of fault detection, since if two modules, software or hardware, perform the exact same function, with different, but equivalent processes, receiving the same inputs and come up with the same conclusions it is safe to assume that both modules are working properly [30].

2.3 – Industrial Robots

Concerning industrial robots, specifically robot arms or manipulators, the earlier mentioned survey of fault tolerance techniques defines them as “mechanical structure consisting of links connected by joints” [24]. As already noted, it follows the hardware redundancy trend of formerly mentioned fields, like avionics, while hinting at problems with using model-based approaches, since techniques like Kalman filtering are required to apply these methods. The downside being that such techniques do not respond well to the modelling inaccuracies that are arouse from the difficulty in modelling the robot precisely, the noise or the dynamic environment. Moreover, it is noted that the special characteristics encountered on the robotics field, like the limited available redundancy, increase the hardship in designing new methods for fault detection. It is also mentioned that, because of the imperfection of sensors, thresholds should be used, preferably dynamic, in order to avoid false positive detections. Despite the problems concerning model-based methods, in recent years it remained an important base for works in the field, including the development of a technique that aimed to overcome a deficiency found in most approaches, as most only consider the occurrence of isolated faults (one fault in one component) and fail to address simultaneous faults or, in alternative, require sensor redundancy to detect these situations [31]. The possible solution presented by Ma and Yang (2016) involved a second-order sliding mode law.

Even though Crestani (2015) [4] remarked the lack of separation of faults handling techniques from the rest of the control system in mobile robots, expanding this deficiency to industrial robots is not an extreme leap. Therefore, despite not being directly related to fault detection, diagnosis or tolerance techniques, one of the most notable works that can be found is a development concerning robot skills, by Pedersen et al. (2016) [32]. The main objective is the increase of the flexibility of industrial settings, attempting to mimic the capabilities of human manual labor, but nevertheless this architecture can be used as a basis to split the fault handling from the control system.

2.4 – Mobile Robots

On literature focusing mobile robots, the model-based fault detection and diagnosis is once again mentioned, with the same weaknesses being noted, mainly the prior knowledge needed, the single fault assumption, the thresholds or the determination of the robot model [3,33-35]. Kuestenmacher and Plöger (2016) compared different types of fault diagnoses techniques, although they found disadvantages in most models derived from characteristics inherent to mobile robots, namely the difficulty in modelling a robot precisely and the amount of computational power (and therefore time) needed for some of the models [3]. The authors compared Parity Space (PS), Hidden Markov model (HMM), Particle Filters (PF) and Observable Operator Model (OOM), remarking that the algorithms would have to deal with problems, described in other works as usual in robotics, like the noise and imprecisions of sensors and actuators, the limited computational power, the inability to observe some important information or the influence that many different operating conditions have on the system states. Description of each technique includes some advantages and disadvantages, so even before the experiment's description, it is possible to understand which method is expected to perform better. PS is identified as being highly sensitive to small amounts of noise and assuming a deterministic nature that robotic systems do not have; HMM is noted for being insufficient if not used in a hybrid version, since its usual form requires discrete states which is not adequate for robots; PF, although hailed for its ability in dealing correctly with the systems, encounters problems with the high computational power demands. Only OOM sees no reference to disadvantages, being complimented for addressing the need to break down action in small parts, as well as its ability to operate with non-stationary processes, which the authors consider to be necessary in dynamic environments for external fault diagnosis. The conclusion presented is that OOM should be the focus of diagnosis algorithms, although it was still needed to solve a problem with the possibility of negative probabilities.

On a fault tolerance only work, Ranjbaran and Khorasani (2010) [33] indicate that one of the main reasons why fault diagnosis and tolerance is so important is the increase of reliability and safety in the presence of actuators faults. Their solution is the usage of a parameter estimation algorithm to readjust the operational model of the system when an actuator is faulty, applying it to a quadrotor UAV. As interesting as the approach itself is the authors' assumption of the existence of fault detection modules for the system to work. This gives emphasis to the need to first figure a way to detect even multiple faults, to which there is also a proposed solution, by Zhang et al. (2011) [34], combining model-based method with qualitative trend analysis, which includes the usage of neural-networks. A similar approach is used by Saied et al. (2015) [36], with the authors presenting again a model-based method, this time combined with a nonlinear sliding mode observer, for proposed fault detection, diagnosis and tolerance architecture to address rotor failure in an UAV with eight rotors.

Even though most methods found were, at least to some degree, either limited to or build upon model-based techniques, an approach that is based on particle filters is proposed by Zajac (2014) [35]. Like in the previously mentioned method comparison [3], it is mentioned the usefulness of PF for robotics, due to the ability to estimate nonlinear systems, even disturbed by noise, while it is also highlighted the computational problem presented by these filters. Therefore, the author proposes a parallel processing solution, with thresholds being used to reduce the number of false positives in the fault detection and diagnosis technique. Despite using constant thresholds, this author also warns for the threats of not using dynamic techniques, underlined by his application of different chosen values to present the influence they have on fault detection.

Model-based methods once again are indicated as the norm when working towards multiple sensor fusion. Improving these techniques, while applying them to the estimation of an UAV attitude, Gu et al. (2016) [37] propose a three-step sensorial information fusion which starts by combining the mathematical models of the different Hardware through a Bayesian filter. It continues, on a second step, by using the redundant information to calibrate the sensor on an online process. Another data fusion approach, proposed by Bader et al. (2017) [38], underlines the usefulness of these techniques, as the problems of data received from complementary sensors can be crosschecked to reduce or eliminate problems like noise or lack of precision. But nevertheless, it warns to the increased risk of faults, both hardware and software, due to the higher number of sensors. It also points out a validation problem, not only because of the dynamic environment of robots creating constant unforeseeable scenarios, but also because the programming is usually done with a declarative paradigm that is harder to validate and not recommend by a European standard in what concerns critical systems. Despite the warnings, this technique does show that redundancy is continuously used, as even the fusion consists of two different blocks, each fusion different sensor information, with fault detection being achieved by comparing both block outputs. The authors also present a notion of dependability which is divided in attributes, the concepts which a system must have (and which include safety and reliability), threats (which are shown in a hierarchy as faults, which can cause errors, which can produce a failures) and means (the fault handling tools that can counter faults and their consequences).

One last noteworthy technique shows consideration for one variable that was rarely, if ever, saw reference: the center mass. This approach does not consider it fixed, but rather considers the situations, like the operation of a mechanical arm, in which the center shifts [39]. These techniques, however, are presented with mathematical heavy or highly complex explanations that make it hard to summarize them, or even adapt them to the different context that is presented here. Still, this work does remind that even the smallest or more overlooked details can cause important changes.

2.5 – Conclusions

It is interesting to note that, although, as was described, there is a relatively long temporal extension through which fault detection, diagnosis and tolerance have been under study, there has been an acceleration on the treatment of the subject in recent years, with many works being found in the 21st Century, especially when the focus is mobile robots in particular. Moreover, there is a clear trend over the general techniques being used or proposed. Redundancy and model-based methods are repeatedly considered, steaming from fields of engineering in which safety has been an important concern for many years, like avionics. Having migrated into the world of robots, industrial manipulators at first, but now also mobile robots, these trail proven methods have been finding their way into the top of the hierarchy of these fields, but there have been detected some shortcomings that are related to the nature of robotic systems themselves, such as the dynamic environment they operate in, which have prompted some to find ways to adjust these methods, other to attempt to develop completely different techniques that can, at the very least, be used in parallel to triumph over the challenges of safety.

Also noteworthy is the lack of autonomous decision-making studies, pertaining to what the system should decide in case of faults. Although fault detection, diagnosis and tolerance have been revised many times, even under different names (underlining some resistance to the use of the unified language that has been proposed in recent years), fault monitoring and supervision have not been granted much attention, especially not in enough detail. This can be seen as an important deficiency as it seems that most authors are either assuming that fault tolerance can be developed until it is capable of solving any and all problems a system might face during its operation cycle, which is not realistic, or they are yet not aware that these absence of investigation exists, which could reveal that fault handling is a very underdeveloped field even today, a deduction that seems far more realistic.

Chapter 3

Skill Architecture, Safety Monitoring and Decision-Making

One of the problems with robotics is the absence of a unified architecture that can be adapted for most applications. This causes severe difficulties in applying safety features universally [8] and, for that reason, this research proposes a safety monitoring model by extending the conceptual formulation of “skills” [32]. This new architecture aims to provide meaningful information about the safety and the integrity for the robot, the environment or the users. Moreover, a new process for decision-making is also introduced which analyses the safety-level of the robot and determines a good functional behavior to avoid hazardous situations.

The concept of safety has been evolving in recent years, so it is important to clarify it. In the context of mobile robots, it makes sense to use the definition presented in [40]. Therefore, the concept of safety considered in this paper includes an artifact (mobile robot), the possibility of a human operator (although it is not expectable that such will be common in an autonomous mobile robot context), a human effector (human that receives the effects of the system) and the interactions between the three. Furthermore, the used definition assumes the "3R-safety":

- Reliability - the probability that there will be no interruptions to the system's ability to operate during a pre-defined time interval;
- Robustness - the amount of noise or perturbations, both internal and external, that the system can sustain without losing functions;
- Resilience - the ability of the system to sustain damage, but still be able to recover its function.

In [5], a model called FCBPSS (Function-context-behavior-principle-state-structure) and its implications with resiliency expands on the understanding of the concept itself. The main reflection is that the resilience could be achieved by different strategies, with reconfiguration, when needed, being produced by changes not only of structure, but of state, behavior, context or principle. Of the three strategies considered by the authors, however, only the first seems to be fully applicable on the present context, as the fault monitoring and supervision that is conceived in this work targets mobile robots that have a, mostly, rigid physical configuration. Although the architecture considered in this paper is similar to the FCBPSS model, there is a clear distinction in the focus that leads to both ways of design. The FCBPSS seems to be design from top to bottom, meaning that the inspiration seems to go from the higher function that the system needs to perform down to the hardware primitives [41]. The skill architecture here considered has a bottom to top approach, considering the primitives first, and building up bigger blocks (skills) until it has the necessary tools to perform different tasks. Another important difference is that the FCBPSS model seems to give a greater emphasis to even the hardware design, as its approach seems is applicable to a variety of systems that even extends further than mobile robots alone. [42]. The skill architecture, as conceived in this paper, does not attempt to focus any other area that not mobile robots. Both methods are, therefore, not directly or easy to compare, but might be found to be complementary to each other, an option that might be explorable in a context not considered in this paper.

3.1 - Skills architecture

Although the conceptual model of skills [32,43] is being developed for industrial robots (mainly, for human-robot collaboration) it can be extended for mobile robots by incorporating a safety monitoring model. The control architecture can be formulated by a set of building blocks that define high-level capabilities that are installed in the robot. These building blocks define a program or a specific task that can be conducted in a controlled manner (with parametrization attributes, feedback and self-adjusting routines) and that allow the robot to complete missions (a set of tasks). Thus, this conceptual model defines (see Figure 3):

- **primitives** as the hardware-specific program blocks. The primitives define the lowest layer of the conceptual model which is characterized by sensing and actuating capabilities (e.g., data acquisition from sensors and real-time control loops). Each primitive performs a unique operation in the system.
- **skills** as intuitive object-centered robot abilities that are self-sustained and can be used for task-level programming. Each skill has input parameters, executing instructions (combinations of primitives: sensing and acting operation sequences) and it is able to verify if the execution of the program was successful (or not).
- **tasks** as a set of skills planned to overcome a specific goal. Each task defines the input parameters of skills, accesses state variables of the robot/world and changes these variables by controlling the execution of a predefined sequence of skills.
- **missions** as a sequence of tasks that are executed whenever necessary to achieve specific goals. Each mission captures the high-level behavior that should be obtained after the execution of a set of predefined tasks. It defines the input parameters and controls the workflow of tasks.

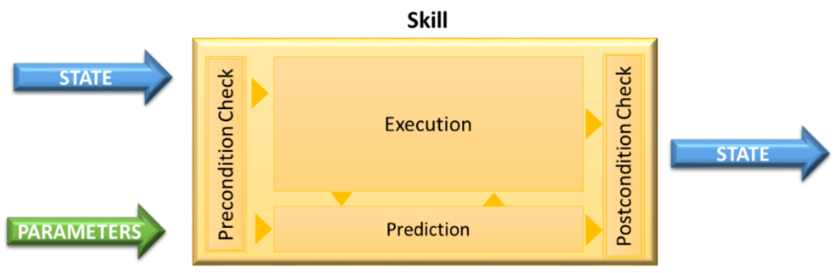

Figure 3: Concept of skills without safety monitoring

This model provides task-level programming based on lower level entities, called skills, that instantiate actions or sensing procedures. In this way, complex behaviors can be described in a more systematic and flexible manner, as well as, the effort to program new mission types is lower and the usage of primitives creates a hardware abstraction which turns the task specification/program completely reusable; however, it is not clear how safety monitoring modules, fault supervision or even fault tolerance strategies can be incorporated.

3.2 – Safety Monitoring Model for Skills Architecture

This section presents a safety monitoring model that extends the skill architecture. Generally, a robot architecture usually comprises three types of components:

- sensors, which allow to receive input from the scenario;
- actuators, which allow to interact with the scenario;
- decisional layer (primitives, skills, tasks and missions), corresponding to the software module that defines or manages specific behaviors for the mobile robot.

Although the literature about fault tolerance, monitoring and supervision of hardware systems is rich with many strategies focused on functional redundancy, the same cannot be said for decisional layers for mobile robots since it is quite rare to encounter a real application where the developers did risk analysis to their software modules.

The safety monitoring model proposed by this research is based on a functional decomposition of the decisional layer. This functional decomposition makes it possible to establish a hierarchical and distributed safety monitoring strategy which extends the conceptual model of skill. A fault monitoring process is incorporated to each skill, see Figure 3, that aims to define a safety state of the skill by verifying deviations in the execution, and, considering perception uncertainties, potential hazardous situations (using causality assumptions), and probabilities of unwanted events related to software failures or human errors (events that are difficult to estimate). By incorporating fault monitoring techniques for error detection (such as temporal control by a watchdog, model-based diagnosis monitoring), therefore, the concept of skill of Figure 3 can be extended by incorporating fault monitoring procedures, see Figure 4.

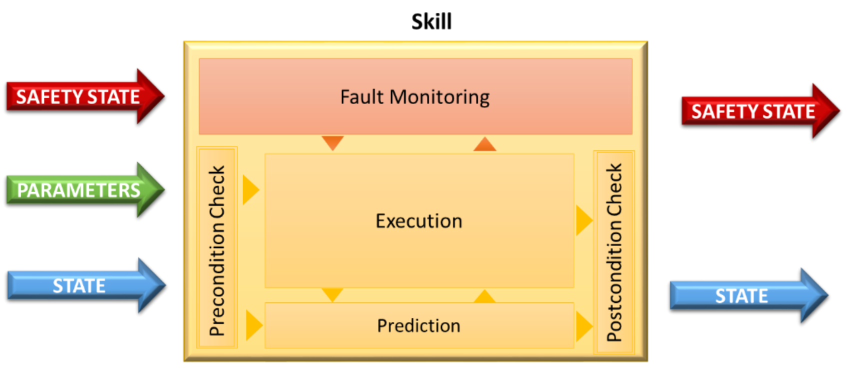

Figure 4: Concept of skills with safety monitoring.

The fault monitoring of Figure 3 resorts to a different set of primitives, e.g., fault detection, fault location, skill reconfiguration or verification of system's results. Safety primitives are different in nature and so, they usually provide a qualitative or quantitative characterization of the safety status of the hardware and software components. For this reason, the specificity of these primitives is inherent and dependent on the abilities defined by the skill (through the execution process), as well as, the harm that potentially rises from their errors - a causality analysis should be determined by considering hazardous situations that could happen in the context of that skill. The fault monitoring process aims to reduce the risk expected for that specific skill and, therefore, it provides safety features by running the monitoring process continuously in parallel with the remaining execution of the operational primitives. This fault monitoring estimates the safety state of the skill, which is defined by the availability and quality (which defines the dependability) and the potential failure severity of the service delivered by a component module (skill).

The safety state can be formulated using 5 levels that characterize a component module with different system integrity:

- **High (0-level)**. When no failures were detected and the service can be delivered.
- **Medium (1-level)**. When failures were detected. The service can be delivered using the same function mode while adapting some sub-objective parameters.
- **Weak (2-level)**. When failures were detected. The service can be delivered using an alternative (potentially degraded) sub-objective while remaining at the current autonomy level.
- **Serious (3-level)**. When failures were detected. The service cannot be delivered but other component modules can continue the mission with the current autonomy level (there is a reduced risk to hazardous situations).
- **Fatal (4-level)**. When failures were detected. The service cannot be delivered and the mission must stop since there is a high risk to hazardous situations.

The safety state of a skill can be determined, defined by a function (1) of the fault processes affecting the skill:

$$Safety\ State = max_{\,i}\, \Upsilon\,(i)\ \times [\chi(\mathrm{i})\ + \psi(\mathrm{i})],\ \ \forall\, i \in A\ , \tag{1}$$

where A is the set of primitives being used by the skill, the $\boldsymbol{\Upsilon}\,(\boldsymbol{i})$, $\boldsymbol{\chi}(\mathbf{i})$ and $\boldsymbol{\psi}(\mathbf{i})$ specifies, respectively, the severity of harm, the influence on the availability and quality of the service provided by the skill which is caused by a failure on the primitive “i”.

The **quality assessment $\boldsymbol{\psi}(\mathbf{i})$** is defined by the persistence and the frequency of occurrence of a failure, see equation (2).

$$\psi(\mathrm{i}) = Persistence(i) \times Occurence(i) \tag{2}$$

The **persistence metric** measures the temporal nature of a failure: permanent (when the service quality is definitely affected), transient (when a failure is no-recurrent for a finite length of time) and intermittent (when a service oscillates between faulty and fault-free operation). While a permanent failure is more likely to be identified correctly, a transient or intermittent failure have a higher chance of being a false positive, a glitch that occurs because the environmental input led the system to a false conclusion, for example. This difference between permanent and intermittent was already established by the IEC 60050. Permanent fault is one that won’t disappear unless there is intervention. Intermittent fault, is a transient fault, meaning it does not require intervention to disappear, that continues to occur [44]. In this way, the persistence metric is defined by:

- **Persistence(i) = 1**, then the failure “i” is Intermittent (or transient);
- **Persistence(i) = 2**, then the failure “i” is Permanent.

The **occurrence metric** defines the confidence that a skill will provide the service (at a random time) within the expected functional limits. The cost function of the frequency of occurrence can defined by qualitative four levels:

- **Very low (= 1).** There will not be service interruptions;
- **Low (= 2).** Service interruptions will happen at a low-frequency or high-confidence in delivering the expected result;
- **High (= 3).** Service interruptions will happen at a medium-frequency basis or low-confidence in delivering the expected result;
- **Relatively high (= 4).** Service interruptions will happen at a high-frequency basis.

The occurrence metric might be hard to calculate without a certain level of abstraction to the frequency of occurrence related to an unexpected event or a primitive that fails to deliver a pre-defined service. The frequency of occurrence can be estimated by preliminary tests (eg., using a simulator or empirical experiences) to obtain the level of confidence about the integrity level of executing a set of primitives that form the operative skill.

The availability assessment $\boldsymbol{\chi}(\mathbf{i})$ does not measure the frequency of occurrence of failures but instead, it pertains to how much the fault affects the ability of the system to perform a certain service. The availability metric measures the readiness for the correct service in the skill and can be defined by:

- **Redundant (= 0)**, when the service remains available without loss of quality;
- **Eminent (= 1)**, when the service remains available with loss of quality;
- **Singular (= 2)**, when the fault removes the system's ability (the service can no longer be delivered).

The availability assessment can be understood by a simple example: on a four-wheel Ackerman robot let us consider the existence of four motors, one on each wheel is controlled by a primitive (PID-controller). If one primitive has a fault it does not stop the ability of the robot to move, it only reduces the number of viable options. If another primitive fails later on, that condition will put the availability of the skill responsible for moving the robot in "Eminent" state, which means that no other fail is tolerable. However, if a fault occurred on the power source that feed all the four motors, that fault will remove the ability of the robot to move.

Finally, the **severity of harm $\boldsymbol{\Upsilon}(\boldsymbol{i})$** is defined in two components namely, severity and extent of the functional failure. It is related to the severity of damage to the health of people, to the integrity of the robot or to the environment (including human built structures) and, is defined by equation (3).

$$\Upsilon(i) = Severity(i) \times Extent(i) \tag{3}$$

The IEC 61508 defines risk as the "combination of the probability of occurrence of harm and the severity of that harm", however harm is considered to only pertain to people [11]. This research extends the definition of harm to include people, environment and the robot itself. Thus, the extent of a functional failure is related to the fault propagation to other decisional modules. It tries to determine the danger of the fault: is it a self-contained fault or could it spread to other components? The propagation of the failure depends on how contagious the fault is, for instance, a failure in the skill of "localization" will jeopardize with relatively easy other navigation modules. It can be defined by:

- **Isolated disturbances (=1).** Faults affect only the current component.

- **External disturbances (=2)**. Faults propagate to other components and have a global impact.

The functional severity depends on the specificities of the robot, the mission and the environment. It specifies the acceptable consequences of the failures:

- **Absence of failures (= 0)**, when the normal functioning of the decisional model does not causes harmful consequences;
- **Minor failures (= 2)**, when the harmful consequences lead to similar costs to the benefits provided by correct service delivery;
- **Catastrophic failures (= 6)**, when the harmful consequences lead to costs with orders of magnitude higher than the benefit provided by correct service delivery.

These severity levels help to design and implement recovery strategies for skills and defines the level of harm that could be expected for a particular failure, i.e., the physical injury or damage to the health of people and damage to property or the environment either by directly or indirectly consequence of the mobile robot.

The severity of harm is an important consideration and, in this analysis, is tied to the level of extent and the functional severity of a system (skill) failure. The estimation of the severity of the harm is essential for a fault supervision and tolerance efficiency since it can be used to guide the recovery or reconfiguration of decisional modules. It can also be argued that the severity of each parameter over the others is hard, if not impossible, to generalize. The functional nature of the fault might seem more important because of the potential to remove an ability from the mobile robot, but the propagation might cause multiple sub-systems to present problems, while the persistence might be the difference between a real fault and a false positive. Analysis of the importance of each of these parameters might even result on an infinity of situations in which their importance highly varies. This formula, on the other hand, allows a standard to emerge. And if deemed necessary, allows an easy process of adaptation to different contexts by adding coefficients, to be decided according to each case, to each parameter.

Equation (1) formula, although it makes it hard to discriminate between different integrity levels, it offers a simple way to evaluate the potential of the fault to endanger the mission (causing a hazardous situation). Considering Table 2, which is achieved with Table 1 as a first step, the safety state can be defined by assigning thresholds for each level namely, "*High*" up to 5, "*Medium*" is between 6 and 20, "*Weak*" is between 21 to 42, "*Serious*" is between 43 to 60 and "*Fatal*" is between 61 to 120. This classification can be adapted as more details are known. In cases in which the mission does not have the potential to endanger people and it is vital that it is completed, for example, "*Serious*" and "*Fatal*" might be adapted and the entire classification can be shifted. However, it should always be noted that the first consideration must be risk. The safety of people must always be placed above the mission unless an extreme scenario is presented. To which skills to apply this model is also an important consideration. The standard form is defined for operational skills that are essential for the mission. For skills that are not crucial, this model can help to assess if there is any danger of faults spreading.

Table 1: Determination of the Quality and Availability Assessments.

	χ(i)			ψ(i)	Occurrence	Persistence
	Redundant	*Eminent*	*Singular*			
χ(i) +ψ(i)	1	2	3	1	*Very Low*	*Intermittent*
	2	3	4	2	*Low*	*Intermittent*
	3	4	5	3	*High*	*Intermittent*
	4	5	6	4	*Relatively high*	*Intermittent*
	2	3	4	2	*Very Low*	*Permanent*
	4	5	6	4	*Low*	*Permanent*
	5	6	7	5	*High*	*Permanent*
	8	9	10	8	*Relatively high*	*Permanent*

Table 2: Determination of the Safety State as a function of the severity of harm, quality and availability assessments.

	χ(i) +ψ(i)										Y(i)	Extent	F. Severity
	1	2	3	4	5	6	7	8	9	10			
Safety State	0	0	0	0	0	0	0	0	0	0	0	*Isolated*	*Absence*
	2	4	6	8	10	12	14	16	18	20	2	*Isolated*	*Minor*
	6	12	18	24	30	36	42	48	54	60	6	*Isolated*	*Catastrophic*
	0	0	0	0	0	0	0	0	0	0	0	*External*	*Absence*
	4	8	12	16	20	24	28	32	36	40	4	*External*	*Minor*
	12	24	36	48	60	72	84	96	108	120	12	*External*	*Catastrophic*

The fault monitoring must resort to a set of safety primitives and user parametrizations to characterize the level of persistence, occurrence, availability, extent, functional severity, required to estipulate the safety sate from equation (1). These primitives must follow strategies for fault-detection and be implemented according to specificities of the robotic application, operational environment and potential hazardous situations. They could be implemented by a set of safety rules is done with a generic method: limit checking, trend checking, signal-model based, process-model based and multivariate data analysis [45].

3.3 – Decision-Making Model

Highly complex software is more likely to have faults [8]. If that is a problem for the design of tasks and skills, or the control for that matter, it is even worse when concerning a sub-system, or module, which objective is to handle the autonomous decision-making, i.e., the ability of the system to decide, without human intervention, when in the presence of detected abnormal working condition (such as detected and diagnosed faults), if it should cancel the mission, proceed as normal or proceed with different parameters. Therefore, it should be designed with as much complexity as needed and as much simplicity as possible. To that end the terms must first be better defined. Thus, the decisional architecture of the mobile robot can be represented by Figure 5:

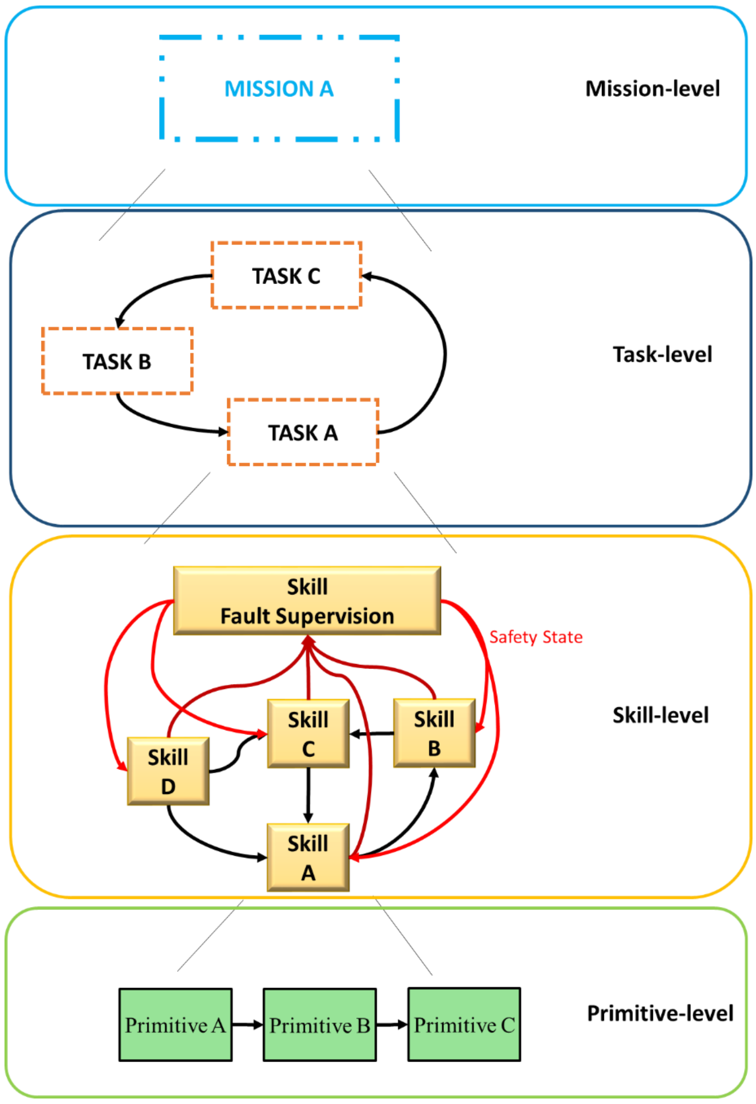

Figure 5: Representation of the Tasks and Skills Hierarchy.

This new hierarchical representation of the conceptual model adds a safety skill “Fault Supervision” to each task. This special skill assesses the overall safety level of the entire task by considering the safety state obtained from other operational skills. This means that the “Fault Supervision” provides a general view of several risk analysis made by each skill so it is possible to conduct a functional system analysis. The system is forced to a safe state (recovery) should some hazardous behavior be detected (error detection) by an external and independent layer.

To make the decision to either continue or abort the mission, the safety state should be the main concern. How to proceed in case of the mission being cancelled should be evaluated to minimize the severity of harm for each application.

It isn’t a matter that can always be judged in a simple binary solution, however. The only situation in which the decision to procced or abort the mission is straight forward is when a skill reaches the Fatal level, as in that moment it is immediately understood that the risk of a hazardous situation is too high. On the opposite side of the spectrum, when all skills’ safety state is High, it is the obvious decision that the mission should procced, as there is no foreseeable risk of a hazardous event. At all other levels, the situation will be harder to assess.

What combination of Medium, Weak and Serious safety states warrants caution? When does it become impossible for the system to deliver what the mission requires? As with so many things in life, many variables might need to be considered. How big is the vessel? What is the mission? Is it worth the potential of elevating the risk? Can some skills recover, thus improving their safety state?

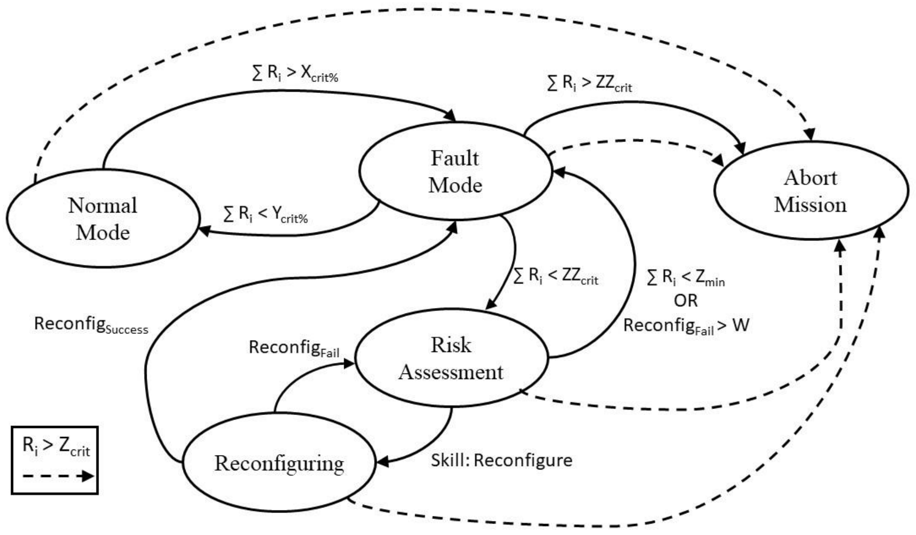

Figure 6: Decision-making State Machine.

Figure 6 illustrates the decision-making process through a state machine. It depends on various parameters:

- Ri – Safety state of skill i;
- ∑ Ri – Cumulative safety states of every skill;
- Zcrit – Fatal level threshold for a single skill Ri;
- ZZcrit – Mission abort threshold for all skills ∑ Ri;
- Zmin – Threshold for all skills. If ∑ Ri is lower, the system reevaluates if the Fault Mode is required;
- Xcrit% – Threshold for all skills. If ∑ Ri is higher the system enters Fault Mode;
- Ycrit% – Threshold for all skills. If ∑ Ri is lower the system returns to Normal Mode;
- ReconfigSuccess – Flag that signals that the system/skill reconfiguration was successful;
- ReconfigFail – Flag that signals that the system/skill reconfiguration was unsuccessful;
- W – Control of the number of failed reconfigurations. Depends on which skills the Fault Supervision attempts to reconfigure.

As can be seen, should any skill, at any time, reach the Fatal level the mission must be immediately terminated. In Normal Mode the cumulative safety states present no or negligible risk and so no changes are needed. Fault Mode is applied when the risk is high enough that it requires evaluation. Should the cumulative safety states create an intolerable risk, the mission is canceled. If, however, the risk is still not high enough to cause an immediate abandonment of the mission, then the Assessment state must be called. In this, the Fault Supervision must consider all skills safety states to prioritize the calling of the reconfigure skill. The

Reconfiguring state attempts to reconfigure each skill, according to the parameters defined in the previous state. It returns the functional risk so that the reconfiguring of each skill can be made in a more targeted way or, in extreme situation, that other preventive measures might be taken. These measures, should there be a need for them, are to be decided in the Fault Mode state.

The actions to be taken when the Abort Mission or Fault Mode states are activated will also vary with the mission type and the specific characteristics of the mobile robot, as well as with the highest risk faults detected. They cannot be universally defined because, without prior knowledge of all these variables, it is difficult to predict which decisions will cause more harm.

As can be seen in Figure 7, each skill requires inputs and, during its execution, generates outputs. Both inputs and outputs of skills are used in conjunction with the functional risk returned by the Reconfigure state of the "Fault Supervision" skill on the fault monitoring. This is an essential characteristic as it allows to reevaluate the outputs before they reach the process.

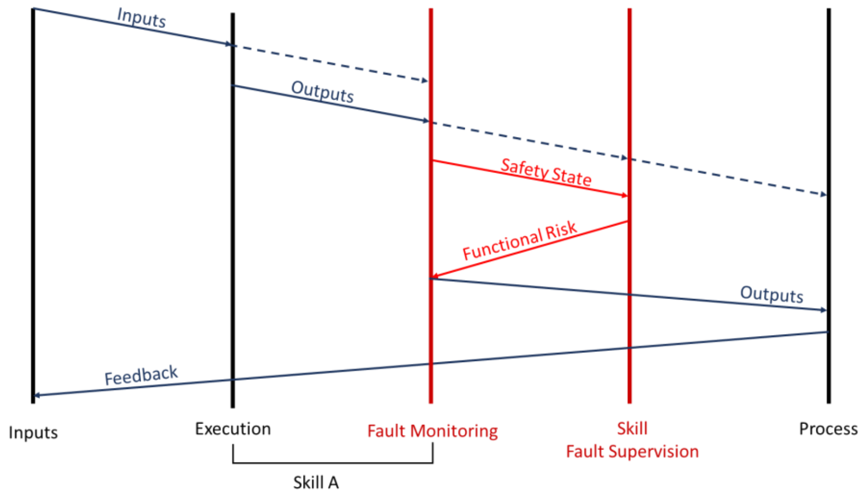

Figure 7: Temporal representation of the system.

The fault supervision takes into consideration the safety state of each operational skill in the same task which defines the nature of the failure: the severity associated with the detected fault, the available resources, the robot´s current state and the autonomy level. The process for decision-making that describes the fault supervision must comprise the following major primitives:

- Reconfiguration (general safety state = medium)
- Adaptation (general safety state = weak)
- Autonomy adjustment (general safety state = serious)
- Safety Waiting (general safety state = serious)
- Definitive Stop (general safety state = fatal)

The final step of the decision-making, the decision itself, can also be standardized, but as previously stated, must allow customization to different situations, if needed. Systems vary, missions vary, even the moment in which the decision, inside the mission, is taken might vary. These considerations are important so that even aborting a mission does not turn a high-risk situation into the hazardous event it tried to prevent.

Chapter 4

Practical Application

To exemplify how to apply this model, it will be considered that it is being used on an Autonomous Surface Vehicles (ASV), specifically a Zarco, illustrated in Figure 8, with a payload that gives it the ability to inspect bridges. It will be assumed that the system is fully capable of fault detection and diagnosis. The system is a catamaran developed by INESC TEC (Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência).

The Zarco is equipped with motion actuators, i. e., actuators that allow it to move, in the form of two electrical thrusters in a differential configuration. Next, an energy module composed of lead-acid batteries is the power source. Moving to sensors, Zarco only has value sensors, i.e., sensors that return a value or a set of values and which concern variables like positioning, acceleration or velocity. These include a pair of GPS receivers (L1+L2 RTK), an Inertial Measurement Unit (IMU) and a magnetic compass. The navigation and communication, in terms of hardware, are based on an onboard computer and a Wi-Fi antenna and, in terms of software, into the functions that allow for the autonomous mission completion and communication with the Ground Control Station (GCS) [46].

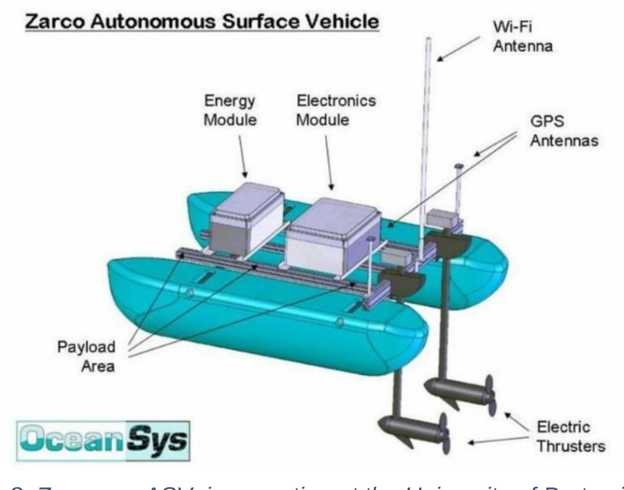

Figure 8: Zarco, an ASV, in operation at the University of Porto since 2005.

With the hardware and software that this system possesses, there are primitives that can be drawn. To exemplify the application of the proposed decision-making model, only a single movement-based skill will be considered.

One can easily define Rotation and Line as, respectively, a primitive that manages the angular movement and a primitive that manages the linear movement. Both are dependent on the motion actuators and the odometry (obtained by GPS and IMU), but the functions that control them are expressed by different mathematical equations. Furthermore, let us consider that there is a third primitive, called AccelerationControl, to evaluate the velocity of the ASV (avoiding dangerous accelerations/deaccelerations that may cause overcurrent) and therefore is dependent on all value sensors (GPS, IMU and magnetic compass). With these three primitives it is possible to build a GoTo skill. This skill will be composed by a sequence Rotation-Line-Rotation, with Acceleration running in parallel.

As Figure 9 implies, the skill (and its primitives) require a set of parameters:

- Final Position – given by (x,y, θ), which translates into coordinates x and y, in meters, and by orientation θ, in degrees;
- Tolerable Position Error – given by (e, ε), which are small values for the tolerance of coordinate and orientation, respectively, error values that are within an acceptable margin;
- Nominal Velocity – named vn, expressed in meters per second, establishes the maximum trajectory speed;
- Elapsed Time – (P1, P2) seconds. Used to determine Persistence (used to calculate the quality assessment ψ(i)). Will be further explained bellow.

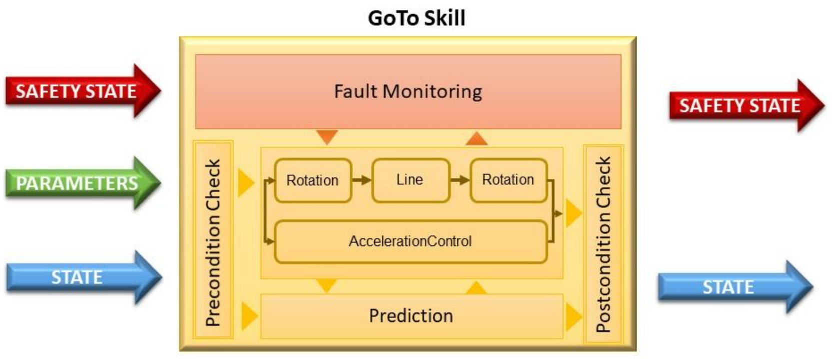

Figure 9: Illustration of the GoTo skill.

The first three parameter sets are straightforward. They establish basic information that the system must know to perform its movement tasks. Although the error information and the nominal velocity could be constant, there is no harm in passing them as parameters to control the precision of the task. The Elapsed Time, however, only makes sense when used in the context of measuring the persistence of faults, as to assess it there is the need to control how long a task is taking. Specifically, it requires to consider the following:

- If task time exceeds a certain measure (/timeout) there is a permanent fault in the skill;
- If the distance error does not reduce in less than P1 seconds, there is an intermittent fault in the primitive Line;
- If the orientation error does not reduce in less than P1 seconds, there is an intermittent fault in the primitive Rotation
- If the acceleration is greater than the maximum allowed for more than P2 seconds, there is an intermittent fault in the primitive AccelerationControl.

These measures of time will enable the fault monitoring to extrapolate if a fault is intermittent or permanent, therefore solving one of the Safety State variables. A fault causing the /timeout to be exceeded is automatically a permanent fault (Persistence(i) = 2), while a fault causing elapsed times bigger than P1 or P2 is automatically an intermittent fault (Persistence(i) = 1).

To evaluate the remaining variables, and since they are mostly dependent on the affected primitive, we need to consider how they interact with the rest of the system or the implementation of the primitives themselves. The Occurrence, for example, only depends on how much confidence we have that something will do its job when it is required. Therefore, to calculate the Safety State of this GoTo skill, we must list some assumptions:

- Because there is not sufficient hardware redundancy in the actuators, if they are the source of the fault the Availability assessment $\chi(\mathrm{i})$ is Singular (= 2);
- Line has been deemed such a perfect algorithm that to reduce complexity only one implementation exists, although it never fails, Occurrence(Line) = 1;
- Rotation has been found to fail to perform in very rare occasions, but there are two different algorithms that can be called at any time, Occurrence(Rotation) = 2;
- AccelerationControl is a new feature, i.e. there is a low degree of confidence, and therefore has a medium-frequency of interruption, but to counter there are multiple algorithms that perform this task being compare against each other, Occurrence(AccelerationControl) = 3;
- Considering the above descriptions of the primitives' implementations, if a fault is caused on the software side, the Availability assessment $\chi(\mathrm{i})$ of the primitives is different: Availability assessment $\chi(\mathrm{Line}) = 2$, Availability assessment $\chi(\mathrm{Rotation}) = 1$, Availability assessment $\chi(\mathrm{AccelerationControl}) = 0$;
- Field tests and expert assessment determined that a fault in the Rotation has a real, but small chance of causing harmful consequences, Severity(Rotation) = 2, but Line and AccelerationControl would have a high risk of catastrophic consequences, Severity(Line) = Severity(AccelerationControl) = 6;
- Because it is the only skill installed so far, but also because any mission could be adversely affected by a fault causing a position or orientation error beyond the tolerable margins, any fault to these primitives is considered to propagate, Extent(i) = 2.

Under these assumptions, all the variables are easily solved. We now know exactly the value of each component of the Safety State in most, conditions. Persistence is to be decided dynamically according to time measures, Availability is also dynamic dependent on the redundancy left, Occurrence and Severity are hard coded, as well as the Extent due to the nature of this skill. A table can be built to aid in quickly calculating any Safety State for any situation, considering the current example:

Table 3: Possible Safety State values after applying known values.

	$Safety\ State = max_i\, \Upsilon(i) \times [\chi(i) + \psi(i)]$ (Severity x Extent) x [Availability + (Persistence x Occurrence)]	Safety State (w)
Line	$(6 \times 2) \times$ [Availability + (Persistence $\times$ 1)]	$12 \leq w \leq 48$
Rotation	$(2 \times 2) \times$ [Availability + (Persistence $\times$ 2)]	$8 \leq w \leq 24$
AccCtrl	$(6 \times 2) \times$ [Availability + (Persistence $\times$ 3)]	$36 \leq w \leq 96$

We are now aware of the possible values for the Safety State of the GoTo skill, but we still need to consider the Fault Supervision. In this case, due to the small dimension of the system, including the single skill, the supervision is limited to a small Decision-making process, as there are no cumulative values to consider. This means that Ri = ∑ Ri and that Zcrit = ZZcrit, since it would not make sense for GoTo's Safety State to cause the mission to abort with two different values:

- R – Safety state of GoTo skill;
- Zcrit – Fatal level threshold. Considering the basic level definition previously presented, 61 is the Fatal level;
- Zmin – Since, although there is only one skill, we still want the system to operate with some degree of confidence in its safety, unless reconfiguration is successful, reevaluation of the Fault Mode requires the Safety level to be at least Weak (w < 42);
- Xcrit% –The system enters Fault Mode if the Safety level is Weak (w > 20);
- Ycrit% – The system returns to Normal Mode if the Safety level is well within Medium level margins (w < 11);
- W – Although some reconfiguration might be possible, hardware redundancy is so low that it is considered that it is only possible for the system to reconfigure itself successfully using Software changes, so this value is 1.

Considering, for example, an intermittent fault in the thrusters, all primitives will have Persistence(i) = 1 and Availability (i) = 2. That would mean that the Safety State of Line, Rotation and AccelerationControl would be, respectively, 24, 8 and 60. As a skill's Safety State mirrors that of its worse primitive, GoTo would be 60. So the Decision-making would not deem it sufficient reason to abort the mission, but would enter Fault Mode and then proceed to Risk Assessment. As the level is Serious, it is above Zmin (Serious starts at Safety State = 43 and Z min is 42). The system does not have enough redundancy to reconfigure Hardware, so it may only try to reconfigure Software. In this case, it could try to change the Nominal Velocity parameter, or even the internal parameters of the AccelerationControl primitive. Let us assume it tried to change the Nominal Velocity, cutting it down by half. This does not change, in this case, the Safety State of the GoTo skill. However, it is a protection measure that allows the system to continue operating in the Fault Mode. But should the /timeout for the mission be reached, then the Safety State would change. Since AccelerationControl was already at the upper Serious level limit, we can easily start by calculating that and we will reach a Safety State of 96. This means its level is now Fatal. Since the skill assumes the Safety State of the primitive with the highest value, we now know that every possible value for GoTo skill places it at Fatal level, and therefore the mission is immediately aborted (such as would be the need in this case considering it is highly possible that the system cannot move at all).

This is a mere example of application, with some assumptions that are, admittedly, farfetched for simplicity of explanation. It is, nonetheless, capable of being expanded to real systems, where the assumptions are usually more factual. This example also exposes the need for a process that helps standardize the values of the parameters used in Safety State calculation, which should be based on a probability distribution function, as exemplified in [47] as soon as it would be possible to collect enough expert opinions to create such a model.

30 - 3.3 – Decision-Making Model

Chapter 5

Conclusions and Future Work

5.1 - Conclusions

The creation of a universal decision scheme is not an easy task. In the field of robotics, especially considering the diversity of mobile robots that can already be seen, there are many variables that cannot be easily quantified. The number of components, the variety of skills, the amount of redundancy or the degree to which the system should be autonomous or “phone home” on limit scenarios are just some of the possible variables that one must grapple with in defining every single detail of a decision-making. If the purpose is to give mobile robots the tools to become more autonomous, then their decision ability must be closer to the human ability to make decisions. One does not need to look past personal experience to understand that the risks people are willing to take depend on the sense of urgency, sometimes much more than on the knowledge of their own abilities or the danger of the situation. It is essential to know what components the system has, but also the purpose it has. The literature concerning fault detection, diagnosis and tolerance, however, seems to identify a trend for optimal solutions to be limited to the system design timeframe. The decision-making module here presented can’t completely escape that paradigm but attempts to move the limit to the timeframe of its own development. This is done because a truly correct decision cannot be taken without all physical and non-physical aspects of the system being known.

This thesis approached the various challenges by splitting the problem in three components: the skill-based architecture, the safety monitoring model and the decision-making model. With the proposed architecture, skills no longer have to be deduced since they are explicitly defined. It is known what each does and what primitives make it up. The components and the redundancy information become “stored” in the most basic units of the architecture, the primitives, and therefore, although it is still needed to know these variables, at least this knowledge is organized in a way that makes it easily accessible to the fault supervision, although not a direct way. What this means is that the proposed formulation is of such a modular character that not the information processed on the primitives’ fault monitoring, but rather the results of the data treatment is visible to the monitoring done at the skill level. This layer in turn sends only the relevant data to the fault supervision. Therefore, information is only directly accessible where it is needed and indirectly accessible only where its importance is reduced. In application, this means that the primitives (representing the first layer of fault monitoring) see the components and which are redundant or not. This information is used in calculating the Safety State and then sent to the skills (representing the second layer of fault monitoring). The skills do not require the information of Hardware or Software they use, because that data was already used. They only need to know which primitives they are composed of to compare the various Safety States, so they find the maximum value, therefore knowing their own Safety State. This way, the fault supervision is shielded from over information, which would increase the needed computational power, but instead works only with the Safety States of each skill. The fault monitoring effectively acts as a buffer to the fault supervision, allowing it to reduce the complexity, making faster decisions with more reliability. The model proposed, therefore, attempts to deal with the variables by creating a basis that is simple and effective enough that it can be applied as presented on systems, with the complexity of each model (Safety State and Decision-making) being reduced by using principles extracted from the black-box philosophy. There are still two things to address. The wished degree of autonomy and the time-frame in which to introduce the fault handling tools. However, even this can be solved by the modular nature of the formulation proposed. The decision-making model uses thresholds to access which is the safety level. These can be made to be adjustable according to the mission type, or a priority measure. Although a small change, it was already considered in Chapter 3 and by itself would greatly increase the customization of the model, allowing it to fit the same system in many different scenarios. The actions to take at each safety level can be, likewise, programmed so that the robot decided the actions to take in any situation, so it calls for the GCS to give input when the situation is more problematic or an hybrid solution could be found in which the system took decisions by itself, but was overwritten by commands given from the GCS.

This approach has the added benefit of trying to contain the propagation of errors to sequential skills or others decisional layers. As seen in Figure 4, the robot architecture proposed as the basis for the decision-making model isolates, as much as possible, the different software components of the system. Missions are broke-down into tasks to be completed in a pre-determined order. Tasks are composed of different skills which are under permanent evaluation so that faults do not cause an abnormal behavior that leads to a hazardous event. Each skill is composed of simple primitives, rudimentary hardware-specific program blocks.

As importantly, the pursuit of the objectives of this thesis feel achieved. The research of current safety trends showed a lack of studies into fault monitoring and supervision, and the investigation of the state of fault analysis allowed to see that the need for decentralized methods was already detected. The success of the proposal of the skill architecture, the safety monitoring model and the decision-making model also feel validated by the success of the last objective, the fostering of national scientific investigation, expressed by the achieved publication of an article, based on this thesis, on an international scientific journal, MDPI's Robotics [22].

5.2 – Future Work

While this thesis is aimed at conceptualizing an unified architecture for mobile robots, with fault monitoring and supervision included, the thinking process started from the ground up, with small to medium size mobile robots in mind. It is also a work grounded in the theory realm for the most part. Therefore, it can't be considered the end of the challenge, but the first stepping stone that any endeavor requires, a first chapter of a bigger book yet to be written.

Future work, in this case, should not restrict itself to the only the implementation of the model, but also, although perhaps to a lesser extent, to the continuous development of the theoretical work. Starting by the latter, even though this thesis is heavily focused on the theory side of the matter, science has taught us that no exact science is ever completely closed. In the same way, no subject in engineering should stop being studied, but rather further investigated. There is more to develop, namely better ways to dynamically assess the redundancy and integrate multiple component knowledge. It is also necessary to develop a better set of tools and input methods to improve the customization of the model, not only during installation in the system, but also when the system is performing a mission, but something causes parameters to shift, like encountering an object that changes priority for example. All these are important aspects that should not be overlooked in the future. However, the most urgent aspects of any future developments are related to integration of this model with real systems. As previously described, the model was developed with small to medium size systems in mind. Therefore, it is an absolute priority to implement the model into such a system (such as the Zarco considered in Chapter 4), to better assess the strongest and weakest aspects of the current formulation. After that is established, the implementation should advance to progressively bigger system, mainly due to the possibility of using this model in, for example, a merchant ASV. It will also be necessary to better define the decision sets, per safety level, that the model should allow. Although this could be considered a theory-based work, it is doubtful that it can be fully separated from a most practical approach as substantial testing might be required.

To summarize, future works should look deeper into the decisions that should be taken by the “Fault Supervision” skill, particularly on big systems and less theoretical, or academic, scenarios. Retaking the example of a merchant ASV, on a system of such scale it might be important to reduce the nominal velocity even when one skill reaches a weak safety state, or special signals might be required to use to warn the other ships in transit that the ASV might pose a certain level of risk. Even using two different models, each with its own safety state thresholds, and cross checking with a fuzzy model might be useful to better evaluate when actions must be taken.

References

1. Zhang T, Zhang W, Gupta MM. An underactuated self-reconfigurable robot and the reconfiguration evolution. Mech Mach Theory. 2018;124:248–58.
2. Nevejans N. European Civil Law Rules in Robotics. Dir Intern Policies Policy Dep C Citizens' Rights Const Aff. 2017;34.
3. Kuestenmacher A, Plöger PG. Model-Based Fault Diagnosis Techniques for Mobile Robots. IFAC-PapersOnLine. 2016;49(15):50–6.
4. Crestani D, Godary-Dejean K, Lapierre L. Enhancing fault tolerance of autonomous mobile robots. Rob Auton Syst [Internet]. 2015;68:140–55. Available from: http://dx.doi.org/10.1016/j.robot.2014.12.015
5. Zhang T, Zhang W, Gupta M. Resilient Robots: Concept, Review, and Future Directions. Robotics [Internet]. 2017 [cited 29 Apr 2018];6(4):22. Available from: http://www.mdpi.com/2218-6581/6/4/22
6. Liu Z, Zhang Y, Yu X, Yuan C. Unmanned surface vehicles: An overview of developments and challenges. Annu Rev Control [Internet]. 2016;41:71–93. Available from: http://dx.doi.org/10.1016/j.arcontrol.2016.04.018
7. Koren I, Krishna CM. Fault-tolerant systems [Internet]. San Francisco: Elsevier Science; 2007 [cited 30 Jan 2018]. Available from: https://ebookcentral.proquest.com/lib/feup-ebooks/reader.action?docID=294597&query=
8. Guiochet J, Machin M, Waeselynck H. Safety-critical advanced robots: A survey. Rob Auton Syst [Internet]. 2017;94:43–52. Available from: http://dx.doi.org/10.1016/j.robot.2017.04.004
9. European Committee for Standardization. Agreement on Technical Co-operation Between ISO and CEN (Vienna Agreement) [Internet]. Brussels: CENELEC; 2001. p. 1–3. Available from: https://www.cencenelec.eu/intcoop/StandardizationOrg/Pages/default.aspx
10. European Committee for Electrotechnical Standardization. Cenelec guide 13 - IEC-CENELEC Agreement on Common planning of new work and parallel voting [Internet]. Brussels: CENELEC; 2016. Available from: https://www.cencenelec.eu/intcoop/StandardizationOrg/Pages/default.aspx
11. MTL Instruments Group. An introduction to Functional Safety and IEC 61508 [Internet]. MTL Instruments Group. 2002 [cited 28 Mar 2018]. Available from: https://www.mtl-inst.com/images/uploads/datasheets/App_Notes/AN9025.pdf
12. ISO 12100:2010 - Safety of machinery -- General principles for design -- Risk assessment and risk reduction [Internet]. [cited 21 Jun 2018]. Available from: https://www.iso.org/standard/51528.html
13. ISO 13849-1:2015 - Safety of machinery -- Safety-related parts of control systems -- Part 1: General principles for design [Internet]. [cited 21 Jun 2018]. Available from: https://www.iso.org/standard/69883.html
14. IEC 62061:2005 - Safety of machinery - Functional safety of safety-related electrical, electronic and programmable electronic control systems [Internet]. [cited 21 Jun 2018]. Available from: https://webstore.iec.ch/publication/6426
15. ISO 10218-1:2011 - Robots and robotic devices -- Safety requirements for industrial

robots -- Part 1: Robots [Internet]. [cited 21 Jun 2018]. Available from: https://www.iso.org/standard/51330.html
16. ISO 10218-2:2011 - Robots and robotic devices -- Safety requirements for industrial robots -- Part 2: Robot systems and integration [Internet]. [cited 21 Jun 2018]. Available from: https://www.iso.org/standard/41571.html
17. ISO/TS 15066:2016 - Robots and robotic devices -- Collaborative robots [Internet]. [cited 21 Jun 2018]. Available from: https://www.iso.org/standard/62996.html
18. ISO 13482:2014 - Robots and robotic devices -- Safety requirements for personal care robots [Internet]. [cited 21 Jun 2018]. Available from: https://www.iso.org/standard/53820.html
19. Kolek L, Ibrahim MY, Gunawan I, Laribi MA, Zegloul S. Evaluation of control system reliability using combined dynamic fault trees and Markov models. 2015 IEEE 13th Int Conf Ind Informatics [Internet]. 2015 [cited 28 Mar 2018];536–43. Available from: http://ieeexplore.ieee.org/document/7281791/
20. Isermann R, Ballé P. Trends in the application of model based fault detection and diagnosis of technical processes. Control Eng Pract. 1997;5(5):709–19.
21. Yoshimura I, Sato Y. Safety achieved by the safe failure fraction (SFF) in IEC 61508. IEEE Trans Reliab. 2008;57(4):662–9.
22. Leite A, Pinto A, Matos A. A Safety Monitoring Model for a Faulty Mobile Robot. Robotics. 2018;
23. Rovida F, Crosby M, Holz D, Polydoros A, Großmann B, P. A. Petrick R, et al. SkiROS—A skill-based robot control platform on top of ROS. Studies in Computational Intelligence. 2017. 121-160 p.
24. Visinsky ML, Cavallaro JR, Walker ID. Robotic fault detection and fault tolerance: A survey. Reliab Eng Syst Saf. 1994;46(2):139–58.
25. Isermann R. Mechatronic systems-Innovative products with embedded control. Control Eng Pract. 2008;16(1):14–29.
26. Sghairi M, De Bonneval A, Crouzet Y, Aubert JJ, Brot P. Architecture optimization based on incremental approach for airplane digital distributed flight control system. Proc - Adv Electr Electron Eng - IAENG Spec Ed World Congr Eng Comput Sci 2008, WCECS 2008. 2008;13–20.
27. Stoican F, Olaru S, Seron MM, Doná A De. A fault tolerant control scheme based on sensor-actuation channel switching and dwell time. Int J Robust Nonlinear Control [Internet]. 2014 [cited 28 Mar 2018];24(Day 4):775–92. Available from: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5718146
28. Szurman K, Miculka L, Kotasek Z. Towards a state synchronization methodology for recovery process after partial reconfiguration of fault tolerant systems. Proc 2014 9th IEEE Int Conf Comput Eng Syst ICCES 2014. 2014;231–6.
29. González O, Shrikumar H, Stankovic JA, Ramamritham K. Adaptive fault tolerance and graceful degradation under dynamic\nhard real-time scheduling. Proc Real-Time Syst Symp. 1997;
30. Strollo E, Trifiletti A. A fault-tolerant real-time microcontroller with multiprocessor architecture. In: Proceedings of the 23rd International Conference Mixed Design of Integrated Circuits and Systems, MIXDES 2016. 2016. p. 431–6.
31. Ma HJ, Yang GH. Simultaneous fault diagnosis for robot manipulators with actuator and sensor faults. Inf Sci (Ny) [Internet]. 2016;366:12–30. Available from: http://dx.doi.org/10.1016/j.ins.2016.05.016
32. Pedersen MR, Nalpantidis L, Andersen RS, Schou C, Bøgh S, Krüger V, et al. Robot skills for manufacturing: From concept to industrial deployment. Robot Comput Integr Manuf [Internet]. 2016;37:282–91. Available from: http://dx.doi.org/10.1016/j.rcim.2015.04.002
33. Ranjbaran M, Khorasani K. Fault recovery of an under-actuated quadrotor aerial vehicle. Proc IEEE Conf Decis Control. 2010;4385–92.
34. Zhang M, Wu J, Wang Y. Simultaneous faults detection and location of thrusters and sensors for autonomous underwater vehicle. Proc - 4th Int Conf Intell Comput Technol Autom ICICTA 2011. 2011;1(October):504–7.
35. Zajac M. Online fault detection of a mobile robot with a parallelized particle filter. Neurocomputing. 2014;126:151–65.

36. Saied M, Lussier B, Fantoni I, Francis C, Shraim H, Sanahuja G. Fault diagnosis and fault-tolerant control strategy for rotor failure in an octorotor. Proc - IEEE Int Conf Robot Autom. 2015;2015–June(June):5266–71.
37. Gu Y, Gross JN, Rhudy MB, Lassak K. A Fault-Tolerant Multiple Sensor Fusion Approach Applied to UAV Attitude Estimation. 2016;2016(April):0–13.
38. Bader K, Lussier B, Schön W. A fault tolerant architecture for data fusion: A real application of Kalman filters for mobile robot localization. Rob Auton Syst [Internet]. 2017;88:11–23. Available from: http://dx.doi.org/10.1016/j.robot.2016.11.015
39. Shen Z, Ma Y, Song Y. Robust Adaptive Fault-tolerant Control of Mobile Robots with Varying Center of Mass. IEEE Trans Ind Electron [Internet]. 2017 [cited 28 Mar 2018];65(3):1–1. Available from: http://ieeexplore.ieee.org/document/8012422/
40. Cai MY, Liu CJ, Wang JW, Lin Y, Van Luttervelt CA, Zhang WJ. A new safety theory: Concept, methodology, and application. Proc 2016 IEEE 11th Conf Ind Electron Appl ICIEA 2016. 2016;1323–7.
41. W.J. Zhang, Y. Lin NS. On the Function-Behavior-Structure Model for Design. Can Des Eng Netw Conf 2005. 2005;(August 2011).
42. Zhang WJ, Wang JW. Design theory and methodology for enterprise systems. Enterp Inf Syst [Internet]. 2016;10(3):245–8. Available from: http://dx.doi.org/10.1080/17517575.2015.1080860
43. Schou C, Andersen RS, Chrysostomou D, Bøgh S, Madsen O. Skill-based instruction of collaborative robots in industrial settings. Robot Comput Integr Manuf [Internet]. 2018;53(June 2016):72–80. Available from: https://doi.org/10.1016/j.rcim.2018.03.008
44. IEC 60050-192 International electrotechnical vocabulary - Part 192: Dependability. 2015.
45. Muenchhof M, Beck M, Isermann R. Fault tolerant actuators and drives - Structures, fault detection principles and applications [Internet]. Vol. 42, IFAC Proceedings Volumes (IFAC-PapersOnline). IFAC; 2009. 1294-1305 p. Available from: http://dx.doi.org/10.3182/20090630-4-ES-2003.00211
46. Cruz N, Matos A, Cunha S, Silva S. Zarco – an Autonomous Craft for Underwater Surveys. In: Proceedings of the 7th Geomatic Week. 2007.
47. Cai M, Lin Y, Han B, Liu C, Zhang W. On a simple and efficient approach to probability distribution function aggregation. IEEE Trans Syst Man Cybern Part A. 2017;47(9):2444–53.