Data-driven Fault Detection for Component Based Robotic Systems

by

Raphael Golombek

Dissertation

Faculty of Technology
Bielefeld University

Bielefeld - April 2013

Abdruck der genehmigten Dissertation zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.). Der Technischen Fakultät der Universität Bielefeld am 05.04.2013 vorgelegt von Raphael Golombek. Verteidigt und genehmigt am 15.07.2013.

Gedruckt auf alterungsbeständigem Papier nach ISO 9706

Abstract

Advancements in the field of robotics enable the creation of systems with cognitive abilities which are capable of close interaction with humans in real world scenarios. These systems may take over jobs previously executed by humans like house cleaning and cooking or they can be supportive and act as a helper for elderly people. One consequence of this progress is the increased need for dependable and fault tolerant behavior of today's robotic systems because they share the same spaces with humans and operate in close proximity to them. Unreliable and faulty behavior may frustrate users or even endanger them resulting in poor acceptance of robotic systems.

The contribution of this thesis is a fault detection approach called AuCom. Fault detection is a basis element for fault tolerant system behavior which is the ability of a system to autonomously cope with occurring faults while it is engaged in interaction. The approach is designed to tackle the specific needs of cognitive robotic systems which feature a component based hardware and software structure and are characterized by frequent changes due to research and development efforts as well as uncertain and variant behavior resulting from the interaction in real world environments.

The solution presented in this thesis belongs to the class of data-driven fault detection approaches. This class of approaches assumes that fault relevant information can be directly derived from data gathered in the robotic system. The data exploited in this work for fault detection is the communication between the system's components. This communication is represented with features which are common to all elements of the communication (i.e., they are generic). Furthermore, the approach assumes that the current element of the communication can be estimated from the history of the system's communication and that a deviation from the expected estimate indicates a fault. This assumption is encoded in the model in terms of a novel representation of the communication as a time-series of temporal dynamic features.

A concrete integration of the approach into a real system is exemplified on our robotic platform BIRON. In addition, exemplary integration solutions for robotic frameworks currently prominent in literature are discussed in this thesis. The actual capability of the approach to report faults is evaluated for several artificial systems in simulation and on BIRON in an off-line and on-line manner. The performance is compared to a histogram-based baseline approach.

Acknowledgements

First of all, I want to thank my supervisors, Sebastian Wrede, Marc Hanheide and Martin Heckmann for their continuous support and encouragement throughout my journey as a researcher. Thank you for the many discussions we had and the support you gave me. It helped me a lot to finally find my way.

I want to thank all my colleagues from the Bielefeld University and especially from the Research Institute for Cognition and Robotics for making the past years working with them a great time in my life. A special thanks goes to Johannes Wienke, Arne Nordmann and Christoph Dreyer for their comments on this work. Another thanks goes to Frederic Siepmann for the countless helping hands he lent and for the moral support throughout the last month and of course for all the “Moccaklatsch” sessions. I also want to thank Heiko Lex who always had a good advice for me and time to drink a coffee (even though he doesn’t drink any).

I want to thank my whole family for their endless support and especially I want thank you Lena for all your patience and understanding of my peculiarities I developed during the writing of this thesis. I am a very happy to have you in my life! Finally, I want to say a few words in German to my parents.

Liebe Eltern, vielen Dank für all die Möglichkeiten, die Ihr mir eröffnet habt und für alles, was Ihr für mich getan habt ohne jemals etwas zu hinterfragen. Ohne Euch wäre ich niemals so weit gekommen.

Contents

List of Figures

List of Tables

1. Introduction

One of the grand challenges in the field of robotics is the development of artificial systems equipped with cognitive abilities which take over dull or dirty jobs previously performed by humans in real world scenarios. Imagine a service robot ready to support and serve 24 hours a day by taking care of repeating household tasks (see figure 1.1). The system could for example serve beverages and food to humans and subsequently bring the dirty dishes back into the kitchen and clean them. It could take over chore weekly cleaning tasks like vacuuming, dusting or ironing. It could provide a helping hand whenever needed for example by cooking your favorite meal just in time when you are back from work and take care of all emerging disorderliness [49]. It might help elderly people to retain their autonomy by supporting them during everyday tasks, helping remember important dates, or call for help in case of an emergency [72, 55].

A robot capable of autonomously performing these activities would be very helpful in many conditions of our daily life. The demand for supportive robotic systems is substantiated through the statistics published by the IFR Statistical Department in [38]. According to this, in 2011 alone, around 1.7 million service robot have been sold for the private (i.e., non-industrial) usage. Although, most of these were vacuum cleaners, lawn-mowing robots, research robots and a wide variety of entertainment systems with by far less sophisticated capabilities than necessary in order to cope with the aforementioned tasks, these numbers imply that a great desire for robotic systems exists.

Figure 1.1.: The idea of a supportive robot. The picture shows the Care-O-Bot 3 system acting as a butler [47].

Advances in miniaturization and the performance of hardware in combination with trends in machine learning and architectural concepts enable the research of systems whose capabilities slowly converge towards those necessary to accomplish complex cognitive tasks. Great progress in this domain can be particularly perceived through the RoboCup@Home competition [80, 134] which is part of the RoboCup initiative [43]. Here, the participating teams demonstrate their expertise in various home related scenarios requiring cognitive capabilities like object and speech recognition, manipulation of the environment, localization or scene awareness and many others. As considerable research progress can be attested to the cognitive robotics domain [77] and "*situations in which novice users come into contact with service* *robots that operate in close proximity to them and share the same spaces are becoming more and more common*" [82] dependability becomes a vital aspect of robotic systems. For example, imagine again a robotic system designed to support elderly people moving around in their home. What if the human needs urgent help but the system is in a standby mode and poor integration leads to multiple restarts before the system reaches full functionality? What if an unhandled exception or a memory leak leads to a crash of a component leaving the system unresponsive and requiring the intervention of a technician? Or what if the system gets stuck in its movement during a supportive walk with a human? These situations illustrate that faults occurring in field may frustrate the user and render the system useless or even dangerous.

Although crucial, dependability seems not to be first priority in today's robotic systems. For example, an analysis of fifteen mobile robots over three years has shown that these systems are rather unreliable having a mean time to failure of 24 hours [25, 24] implicating the visit of a technician each day if such a system were to be deployed in a home environment. While these studies have been concerned with mobile platforms they suggest that robots in close human interaction suffer from similar problems and thus require additional means to increase their reliability and dependability.

1.1. Contribution

There exist several ways to improve the dependability of a robotic system and they are applicable in different phases of its life-cycle (see Chapter 2). The work laid out within this thesis focuses on *fault detection* as a means of detecting faults while the system is in field. With respect to the example introduced in the previous section fault detection would allow the system to autonomously find out that some kind of fault has occurred (e.g., a component crash) and eventually trigger recovery routines. While this approach does not improve the system's reliability in terms of mean time to failure it has a positive impact on the system's operation time by reducing the number of faults which need to be handled by a human. The main contribution of this thesis is the answer to the following question:

How to design a data-driven fault detection approach for a Component Based Robotic System that does not modify the system's components?

The *AuCom*[1] fault detection approach developed in this thesis takes account of the specific aspects of cognitive robotic system. Compared to other fields (e.g., standard industrial robotics) cognitive robotics often involves close, bidirectional, human robot interaction which takes place in open-end dynamic environments and makes the system's behavior variable and difficult to predict. Furthermore, algorithms used to implement cognitive skills frequently employ machine learning techniques which increase variation and uncertainty of the system's behavior.

To tackle these issues the proposed approach is based on the data-driven fault detection paradigm. This class of fault detectors[2] relies on the assumption that in order to detect faults it is sufficient to gather fault-sensitive data in the target system, derive a model from it and apply the model to decide whether the system behaves normal or not. For complex systems this procedure is beneficial since the model can be learned instead of generating it manually which can be a tedious task and requires system expert knowledge. In case of system changes, adaptations of the detector often can be realized by re-training the model with new data.

In this work a system is regarded as a Component Based Robotic System (CBRS) whose components communicate with each other in order to fulfill a task. A novelty of the approach proposed here is that it exploits the communication between the system's components as fault-sensitive input. In the remainder of this thesis the communication is called the *inter-component communication* (ICC) of a system. The

[1] The work within this thesis was initially started within the context of Autonomic Computing [49]. Although the focus has shifted away from this concept, the author decided to keep the acronym AuCom as the name of the detection approach.

[2] In this work the implementation of a particular fault detection approach in software is called a detector and the implementation of the approach presented in this work is called the AuCom-detector.

advantage of relying on ICC as input is that it can be gathered in the system without the modification of single components.

The approach proposed in this work exploits solely the generic information of the communication. Considering ICC as a set of single communication *data-points* produced by different components and ordered by time this means that the features extracted to represent each single data-point have to be common to a wide range of components and systems. This way of encoding the ICC simplifies the application of the AuCom-detector to different systems and makes it more robust in the face of changes.

Fault detection is basically a classification task. This classification can be realized either with one or with several classes. In the former case the classifier maintains only one class which represents the system's normal behavior and outliers to this class are regarded as faults. In the latter case the classifier possesses additional classes representing faulty states and detection takes place by deciding to which class the current system behavior belongs to. While the latter may yield a more accurate detection, it is dependent on data from faulty behavior. In this work, I therefore choose to exploit a one class classifier solution.

Another contribution made within this thesis is the implementation of the AuCom fault detection approach in a modularized way which makes it easy to apply the approach to other robotic systems. Furthermore, an experimental framework was designed which facilitates the repetitive execution of experiments.

1.2. Outline of the Thesis

The remainder of this thesis is structured as follows. The upcoming chapter 2 consists of two parts. Part one is concerned with basic terms and concepts of dependability including definitions of the terms fault and fault detection. It provides the basis for the upcoming discussion on fault detection concepts and brings fault detection in line with the framework of dependability. The second part is concerned with the description of Component based Robotic systems (CBRS) which are the targeted systems in this work. The chapter concludes with an introduction of challenging aspects when applying fault detection to the targeted systems.

Chapter 3 provides an overview of the various fault detection approaches present in the literature. The discussion starts with an introduction of historical approaches summarized as legacy solutions. Subsequently, the two prominent groups of model-based and data-driven techniques are presented. In the last part of this chapter, the literature is discussed in the light of the challenges introduced at the end of chapter 2.

In chapter 4 the discussion turns towards the AuCom fault detection approach developed within this thesis. The benefits of ICC as the input for the AuCom-detector are discussed together with the notion of *Temporal Dynamic Features* (TDF) as a generic representation of ICC. Furthermore, the different processing steps of the algorithm are introduced covering both the training and the application phase.

Implementation and integration efforts of the approach are discussed in chapter 5. The modularized structure of the AuCom-detector is presented together with typical workflows in which the AuCom-detector was used during the research period. Furthermore, integration aspects of the detector into robotic systems are introduced. Concrete integration efforts are presented for the robotic system called the BIelefeld Robot Companion (BIRON) [59] together with hypothetical integration efforts for further communication frameworks currently prominent in robotic systems.

In chapter 6 the quantitative evaluation of the AuCom-detector executed in simulation and on a real robot is presented. The simulation experiments were conducted on different artificial systems implemented based on the CAST communication framework [63]. The evaluation on a real system was executed on the BIRON system in an off-line and on-line manner.

In Chapter 7 the findings of this thesis are summarized, conclusions are drawn and an outlook with implications for future work is given.

2. Dependability in Component Based Robotic Systems

The following chapter is concerned with the conceptual framework of dependable computing and the domain of Component based Robotic systems (CBRS). The intention is to i) define the concept of fault detection as a major foundation of fault tolerant systems and ii) give a comprehensive description of the targeted systems. In addition, the coherent terminology introduced in this chapter shall facilitate the discussion in the remainder of this thesis. The chapter begins with the concept of dependability in section 2.1 comprising attributes, threats and means of dependable robotics. Among others, this involves the concepts of fault and fault detection. In section 2.2 the discussion turns towards the description of the set of CBRS. Component based Software Engineering (CBSE) [64, 121] which is a fundamental concept of CBRS is introduced and it is argued that it is meaningful to focus on systems build this way because CBSE is a successful and promising concept for the development of robotic systems. The section concludes with the introduction of challenging aspects on fault detection in the context of CBRS.

2.1. Dependability: Attributes, Threats, Means

This section introduces the fundamental terms and concepts of dependability and dependable computing as an overarching concept of the actual topic of this work, namely fault detection. The main intention is to provide a set of crisp definitions in order to simplify the discussion of the fault detection approach from different points of view in the upcoming chapters. The concepts on dependability have their origin in the engineering domain and were subsequently adapted and extended for their use in the robotics domain leading to slightly differing terms used in these domains. In this section the definitions from both domains are introduced due to the fact that they are both frequently used in the literature on fault detection in robotic systems.

We begin with a definition of dependability given by Avizienis et al. [6]:

> *Dependability is the ability to avoid service failures that are more frequent and more severe than is acceptable.*

This definition takes into account that a system can (and usually does) fail whereas it is up to the user of the system to decide whether the system can be still regarded as dependable or not. In addition, this definition implies that the dependability of a system also depends on the actual task to be executed. Furthermore, Avizienis et al. describe dependability as an integrating concept which encompasses several different attributes[1]:

▷ Availability: The probability that a system will operate satisfactorily and effectively at any period of time.

▷ Reliability: The ability of a system to perform a required function under stated conditions, within a given scope and during a given period of time.

▷ Safety: The ability of a system not to cause damage to persons or the environment. It can be also described as the absence of catastrophic consequences on the user or the environment.

▷ Integrity: The absence of improper system alterations.

[1] It shoud be noted that enumeration of the attributes of dependability follow the work of Avizienis et al. [6]. However, the descriptions used in [6] were too short and thus have been augmented with the work of Iserman [70].

- ▷ Maintainability: The ability of the system to undergo modifications and repairs.
- ▷ Security: The ability of a system to prevent unauthorized access or handling of system information.

These attributes may vary in their importance in regard to a given robotic system and the intended application of the system. For example, in case of a robot that is licensed under open source, security might not be an attributes with top priority[2]. Instead, if the system is intended to work in populated spaces, safety is a crucial attribute. The extent to which a system incorporates the attributes of dependability should be measured in a relative and probabilistic sense because the unavoidable presence of occurrence of faults, a system will never be totally available, reliable, safe, or secure [6].

2.1.1. Threats to Dependability

The dependability of a system can be affected by threats throughout its entire life-cycle which consists of a development and an application phase [6]. These threats are usually known as *faults*, *failures*, *malfunctions* and *errors*. There exist two major definitions of threats widely used in the domain of engineering [70, 39] and computer science [6]. For the engineering domain Isermann et al. [70] define a fault as follows:

Definition 2.1.1.1
A fault is an unauthorized deviation of at least one characteristic property (feature) of the system from the acceptable, usual, standard condition.

This definition marks a fault as a state within the system whereas the transition to a faulty state may develop abruptly (step-wise) or incipiently (drift-wise). Based on this, the definition of a failure is given as follows:

[2]Assuming that in an open source system anyone is able to access any information related to the system and its application.

Definition 2.1.1.2
A failure is a permanent interruption of the system's ability to perform a required function under specified conditions.

A failure is an event which may result from one or more faults and may occur in different types (i.e., random, deterministic, systematic, causal etc.). Similar to a failure a malfunction is defined as:

Definition 2.1.1.3
A malfunction is an intermittent irregularity in the fulfillment of a system's desired function.

Thus, a malfunction can be seen as a discontinuous failure i.e., a temporary interruption of the system's function. The last threat, the error, remains undefined in the engineering domain[3].

In the context of dependable computing and robotics the definitions of the aforementioned threats are slightly different whereas the focus lies on the concepts of error, fault, and failure [6].

Definition 2.1.1.4
An error is a part of system state that may cause a subsequent failure.

This definition is similar to 2.1.1.1 since it addresses the fact that if something goes wrong in a system, it is most probably reflected in the system's overall state. However, here the system state is used to define an error. As a consequence, fault detection approaches which follow this definition are often also termed error detection approaches. Based on the definition of an error Avizienis et al. propose an alternative definition of a fault:

Definition 2.1.1.5
A fault is an adjudged or hypothesized cause of an error.

[3]Although there exist definitions for the term error in the engineering domain, this is not the case for the work of Isermann [70] referenced in this thesis.

This definition describes a fault as the reason why the state of a system may change resulting in an error. The authors in [6] define a classification of faults based on the following eight viewpoints:

- ▷ **Phase of creation:** Depending on the phase a fault can be a development or application fault.
- ▷ **System boundaries:** Here faults are differentiated according to whether they are internal to the system (internal faults), or come from its external environment (external faults).
- ▷ **Phenomenological cause:** The phenomenological nature of fault enables the classification into (a) natural faults, i.e. faults which are caused by natural phenomena and (b) human-made faults.
- ▷ **Dimension:** Faults can be either associated with hardware (hardware faults) or software (software faults).
- ▷ **Objective:** From the security point of view faults can be also classified into malicious and non-malicious.
- ▷ **Intent:** This category also affects the security standards of a system. One can differentiate deliberate faults carried out to cause harm or damage and non-deliberate faults usually introduced by a user or developer without being aware of them.
- ▷ **Capability:** This category targets the human factor when introducing faults into a system. One can differentiate accidental faults, which are similar in their semantics to non-deliberate faults and (b) incompetence faults, which are often caused by a lack of competence of the parties involved.
- ▷ **Persistence:** This category describes the temporal behavior of faults. Transient (sometimes called intermittent) faults are difficult to detect due to their temporary nature. Persistent faults are continuously present in a system which simplifies

the detection problem.

In the domain of dependable computing a failure has the following definition:

Definition 2.1.1.6
A failure is an event that occurs when the delivered service deviates from correct service.

Following this definition, a service delivered by a system is its behavior as it is perceived by its user(s) [6]. Failures can be categorized into content failure (deviation from expected content) or timing failure (deviation from the expected time of arrival or duration of delivery of a service).

As previously said in the literature on fault detection in robotic systems both definition (Isermann and Avizienis) are being frequently utilized which leads either to the use of the concept of *fault detection* (in case of the engineering point of view) or of *error detection* (for the dependable computing domain). Often the usage of one of these definitions is implicit i.e., it can be only derived from the exploited terminology and the content of the work. Both aspects hamper a coherent discussion of the literature. In this work I therefore decide to use fault detection a synonym for both fault and error detection. Furthermore, the terminology used for the description of the AuCom approach corresponds to the one defined in the engineering domain and follows the work of Isermann et al. [70]. As a consequence, fault detection in this work is defined by exploiting definition 2.1.1.1 as follows [70, p. 61]:

Definition 2.1.1.7
A fault detection approach is a method which uses the relations between several measured variables to extract information on possible changes caused by faults.

2.1.2. Means of Dependability

Throughout the years many ways to improve the dependability of artificial systems have been developed. Following [6] these means can be broadly divided into four complementary categories:

- **Fault prevention** techniques are part of the general engineering task and involve best practice methodologies for software (e.g. information hiding or modularization) and hardware (e.g. design rules) and for the development process as such (e.g., revision control of code or automatized regression tests).
- **Fault removal** techniques are targeting the reduction of the number and severity of faults by manually verifying conditions, diagnosing the system and removing faults. These three steps can be either executed during development or while the system is in use. In the latter case the user is involved in the fault removal process by reporting occurred faults which are then fixed during the maintenance of the system.
- Further techniques are concerned with **Fault forecasting** where the goal is to qualify failure indicators (failure modes) in the system and quantify them in terms of probabilities. By exploiting this information an estimate of the expected number of faults in a system can be given or the probability of future occurrences can be estimated.
- The fourth group of techniques is concerned with **Fault tolerance**. In general, fault tolerance comprises three phases. First, faults are detected in the system based on monitored fault-related measures. The next step comprises techniques to diagnose the fault and identify the reason for its occurrence. Finally, recovery routines are executed which bring the system back into a normal operation mode.

Figure 2.1 depicts a summary of the attributes, threats, and means of dependability in a tree-like structure.

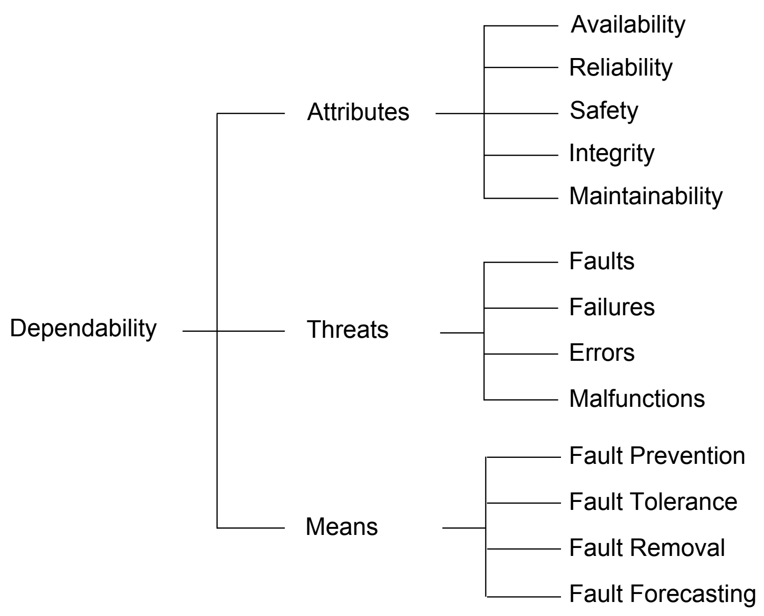


Figure 2.1.: A representation of the attributes, threats, and means of dependability in a tree-like structure [6].

The research carried out in this thesis is focused on fault detection as a basic ingredient for fault tolerance. Fault tolerance techniques play an important role in robotic systems which primarily operate in close interaction with humans in an autonomous way. They enable the automatic removal of faults in order to prevent failures from generating a harmful impact on humans, the environment, or the system itself.

2.2. Component Based Software Engineering for Robotic Systems

This section introduces CBRS which represents the set of robots build upon the concept of Component Based Software Engineering [20, 21]. Simultaneously, CBRS defines the set of systems eligible for the application of the fault detection approach developed in this work. CBSE is an approach that has arisen in the software engineering community in the last fifteen years and aims to move the focus when building a system from the long-established programming to the composition of systems as a mixture of pre-built and custom built components [18]. In CBSE a system is regarded as a set of components which communicate with each other to cooperate and solve a given task. This perspective on a system emphasizes the separation of concerns, facilitates re-use of components, and improves the scalability, maintainability, and robustness of the system. It proved to be beneficial in the development process of large and complex systems in various domains like factory automation, avionics, or automotive [20].

> *Modem robots are considered complex distributed systems consisting of a number of integrated hardware and software modules. The robot's modules cooperate together to achieve specific tasks. [86]*

The congruency of the CBSE definition and this perspective on today's robots suggest the utilization of CBSE for robotic systems. Despite this, in the robotic context where research plays an important role, the strong emphasis on software reuse accelerates the research progress, since researchers do not have to constantly reinvent the wheel but can rather build upon already present solutions implemented by others. Often, large scale projects are realized by a heterogeneous group of cooperating researchers potentially distributed all over the world. By considering the different research topics during system decomposition CBSE also enables to challenge the topics independently and in parallel, thereby facilitating the cooperative aspects of research.

The benefits of CBSE led to the development of a plethora of CBRS. Among others, exemplary systems are BIRON [131], Dora [62], cosero [119], PR2 [16], or Nao [54]. Besides complete systems, plenty of robotic frameworks and middleware solutions which follow the CBSE principle were developed with a particular emphasis on the needs in robotics. Here, the most popular ones are XCF [136], YARP [44], ORCOS [22], OpenRTM-aist [4], ROS [103], Cast [63], and RSB [132].

The arguments presented here show that CBRS already comprises a huge number of modern robots and suggests that Component Based Software Engineering is an important concept for the development of future systems. This renders the choice of these types of robots as the set of targeted systems for the fault detection approach developed in this work a meaningful option. To facilitate further discussions in the remainder of this thesis, basic terms and concepts of CBRS are introduced now based on the work of Brugali and Scandurra [20, 21]. The set of presented concepts is limited to aspects which are sufficient in order to discuss the fault detection aspects in this work. For an exhaustive description the interested reader is referred to [20, 21].

A **System** is an entity that interacts with other entities, i.e., other systems, including hardware, software, humans, and the physical world. These other systems are the **environment** of the given system. Each system is regarded as being composed of components which interact and cooperate with each other, and thus generate the behavior of the system. Each **component** represents a modular part of a system that implements a coherent set of robotic functions and whose manifestation is replaceable within its environment. In robotic systems a component may implement a skill of the system like the detection and recognition of faces and objects or self-localization and navigation. Other components act as connectors for the hardware of the robot and manage sensors (i.e., stereo cameras) or actuators (e.g., a gripper). The behavior of a component is completely determined by its implementation and its **interfaces**. Interfaces describe the interaction capabilities of a component and consist of operations and data elements. Two different types of interfaces can be distinguished: i) provided interfaces which expose services implemented by a component including data elements and ii) required interfaces which describe the necessary input of a component in terms of required services and data. Components in a system have their required interfaces connected to the provided interfaces of other components of the system. All infrastructural functionality necessary for example for the communication between the components has to be implemented in the **middleware** of the system. Thus, the middleware is a vital element in a CBRS. It provides means for component communication, execution, life-cycle management, and other required features [111]. In a middleware based system components are not connected directly with each other but through the middleware which acts as a mediator by providing appropriate implementations of various communication paradigms. In this work, communication plays the role of the primary source of information in order to detect faults in the targeted system. Thus, the fault detection approach developed here puts also particular demands on the system's middleware. However, the concrete discussion of these requirements makes the most sense when conducted in the light of the implementation aspects of the approach and is therefore postponed to chapter 5. Figure 2.2 summarizes the description of the various terms and concepts of CBRS based on an exemplary two-component system.

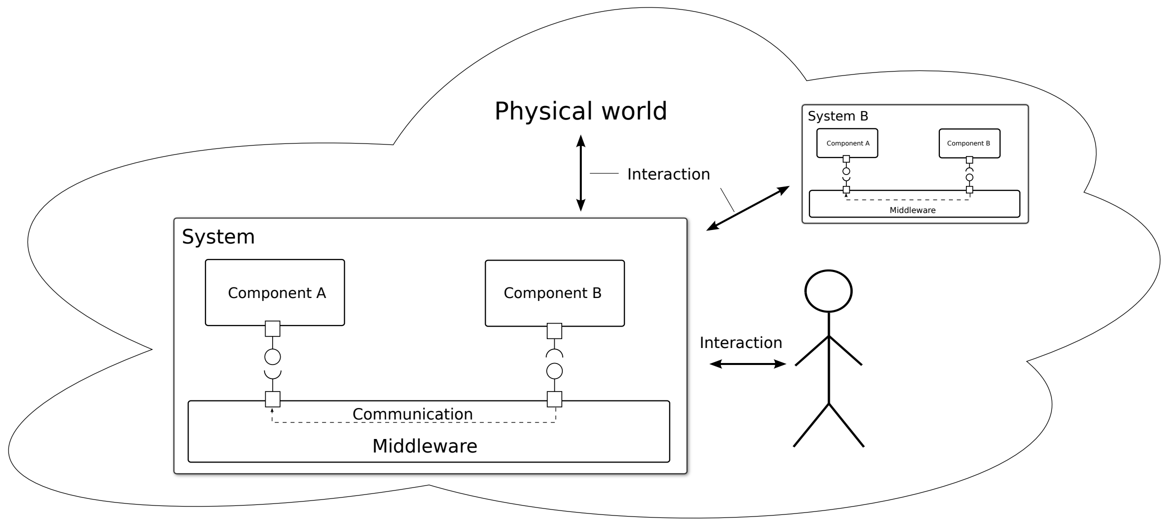


Figure 2.2.: A conceptual view on a Component Based Robotic System. The system interacts with other systems, humans, and its environment. The components in the system communicate with each other through their interfaces and with the help of a middleware as a mediator.

2.2.1. Challenges

Having introduced the class of robotic systems targeted in this work (i.e., CBRS), this section is concerned with challenging aspects for fault detection when applying it to these systems. In this context, challenges arise from i) specific properties of the targeted systems, ii) the development process and iii) the application of the system in real-world scenarios. In particular, the following challenges were discovered:

- ▷ A central aspect of the development and application of today's robotic system is research and education. Often many students and researchers work on the same robot either in context of a common projects or on separate ones thereby sharing the robotic environment. The developers add, exchange or modify the functionality of the commonly used system. This renders **frequent changes** an integral part of the life-cycle of today's robots. Changes to the system's behavior can also emerge on-line during its interaction with the environment for example when the system learns new skills [127] or improves older ones. All in all, frequent system changes need to be considered in the design of a fault detection approach.
- ▷ Applying a fault detector to a concrete robot always requires integration efforts. In general, it involves modifications of the targeted system in order to access data which is expected to be fault sensitive and adaptations of the detector in order to cope with new input data. One (rigorous) option is to modify each single component of the system and gather component internal features as fault detection input. This forces tight coupling between the target system and the detector and reduces the portability of the approach to other systems. In addition, for large systems it may result in a high amount of integration efforts. This proceeding is also incompatible with the previous challenge since component modifications may render the monitoring and detector code to be subject to modifications, too. Thus, the second challenging aspect identified in this work is **minimal invasive integration** of the detector into a target system.

▷ Another challenge which needs to be tackled is the usage of expert knowledge during the application of a fault detection approach to a particular system. In general, such knowledge is convenient as it enables the design of precise and fine-tuned models with improved detection capabilities. For example, knowing that a component A produces an output 100 milliseconds after receiving input from component B could be used to define an appropriate rule and check for it. However, as CBRS are regarded to be complex systems with an increasing amount of functionality often realized based on machine learning techniques, incorporating expert knowledge becomes a challenging and tedious task. The rules to be defined become more complex due to a larger number of components and complex interaction among them. In addition, relying upon expert knowledge requires a system expert at hand each time changes have to be made to the detector. As a consequence, the **minimization of expert knowledge** exploited for fault detection is considered to be another challenge which needs to be tackled.

▷ A cognitive robotic system is exposed to a broad range of environmental conditions during real-world interaction like changing light and sound conditions, different rooms and open spaces and varying human interaction partners. This diversity in the environment leads to variations in the systems behavior even if no explicit changes are applied to it. In addition, variance results from the implementation of behavior with the help of machine learning in order to realize capabilities like object recognition [51], face detection [128], speech recognition [40] or simultaneous localization and mapping methods [123]. Further uncertainty is introduced into the system through the system's sensors which produce inherently noisy readings. Varying input data may for example lead to classification errors of objects, faces, or spoken language. These findings suggest that **uncertain and variant behavior** should be considered throughout the designing process of a fault detection approach, too.

▷ Robotic systems consist of components realized either in hardware or software.

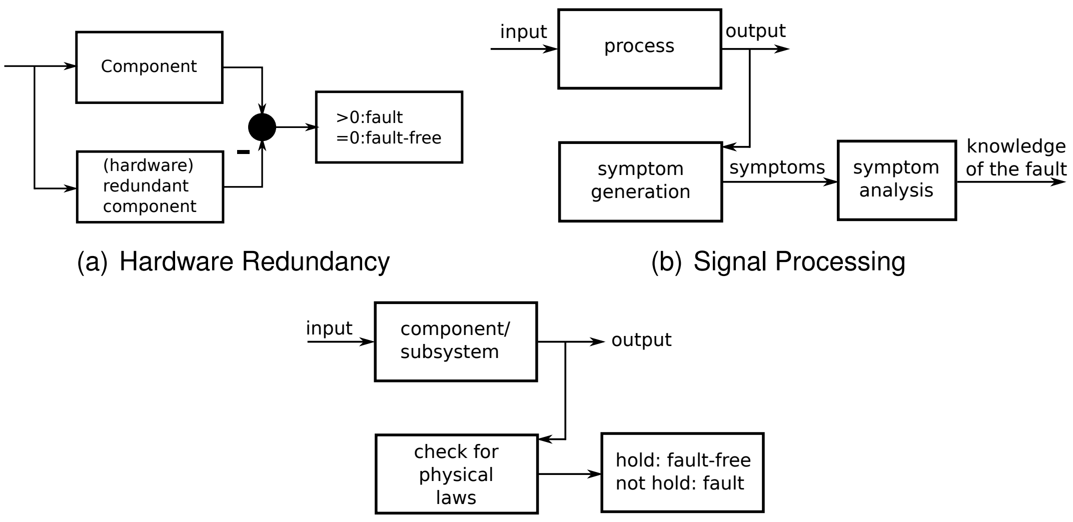
Clearly, both types of components are subject to faults. In case of hardware components this might happen due to e.g., wear or overheating while software components often fail because of programming failures (e.g., leading to memory leaks), unconsidered inputs and system states or temporarily not available resources (e.g., out of memory situations). Consequently, an appropriate approach should provide means for fault detection in **hardware & software** components.

2.3. Summary

This chapter has introduced two vital aspects which together provide a conceptual frame for the discussion of the fault detection approach presented in this work. In section 2.1 the notion of dependability and dependable computing has been introduced. The intention was to provide the reader with an overview of the theoretical framework of dependability in robotics, to show how fault detection fits into this framework and to introduce a coherent terminology which will be used throughout the remainder of this thesis. In particular, the concept fault detection, which is central in this work, has been introduced and identified as a fundamental element of fault tolerant systems. The second part of this chapter was dedicated to Component Based Robotic Systems as the set of systems the fault detection approach developed in this work can be applied to. The underlying concept of Component based Software Engineering has been introduce and arguments have been provided to support the decision of the author to focus on this type of systems. Finally, challenges on fault detection in the context of CBRS were discussed.

3. Fault Detection Techniques

Fault detection has a long tradition in the field of artificial systems dating back to the early 1970s [68]. First approaches were based upon *hardware redundancy*, *signal processing* and *plausibility checks* and their application field were mainly industrial plants. The idea of *hardware redundancy* [120, 139] is depicted in figure 3.1(a) where two or more functional identical components are fed with the same input and are therefore expected to produce the same output. The detection of faults is then based on deviations of the outputs. Hardware redundancy offers precise, dependable, and fast detection of faults sourced in the system. Simultaneously, it comprises the diagnosis aspect by indicating which component delivers unexpected output. A preferred variant of the scheme is to implement the redundant components in different ways which renders the approach more robust to externally induced faults because it is expected that different implementations are not influenced in the same way thus reducing the probability of a dead loss. However, this technique is expensive in terms of money and space. As a consequence, this technique is only recommendable for a limited number of extraordinary critical components in a system and thus can be regarded as a complementary solution. *Signal processing* approaches assume that the output of a component contains valuable information which can be used to detect faults. Detection is done by applying mathematical or statistical operations on the output in order to detect unexpected changes. The signal processing scheme is depicted in figure 3.1(b). Due to the fact that only the output is considered, these techniques have its limits for service robotics where many components have high dimensional state spaces and ambiguous input output mappings. Another class of approaches



(a) Hardware Redundancy

(b) Signal Processing

(c) Plausibility Checks

Figure 3.1.: Schematic description of the processing schemes for hardware redundancy, plausibility check and signal processing [39].

is called *Plausibility checks*. These approaches assume that fault detection can be done by checking whether the values of the monitored system output variables are located within plausible ranges. By checking state variables plausibility checks can be customized for specific operating conditions representing rough process models. However, in complex systems a consistent coverage of the process requires a huge amount of rules and a lot of expert knowledge drawing these approaches less interesting for this work. The scheme for plausibility checks is depicted in figure 3.1(c).

Based on the experience with these three concepts more advanced and improved fault detection approaches have been developed. They can be broadly divided into model-based and data-driven solutions [39]. Model-based approaches can be further partitioned into analytical and knowledge-based solutions. Data-driven fault detection can be further differentiated based on the exploited data processing technique. Figure 3.2 summarized all the just mentioned concepts in a taxonomy tree whereas hardware redundancy, signal processing, plausibility checks have been summarized as legacy approaches.

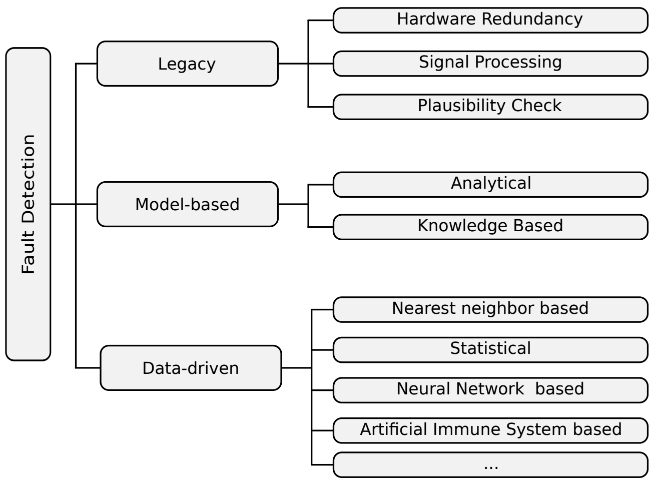


Figure 3.2.: A taxonomy of fault detection approaches divided into three main groups: i) legacy, ii) model-based, and iii) data-driven. The dots in the data-driven listing indicate that the list is not exhaustive.

The remainder of this chapter is structured as follows: Section 3.1 introduces model-based fault detection approaches and gives an overview of the literature on analytical and knowledge-based solutions. Subsequently, in section 3.2 data-driven techniques are presented. The chapter concludes in section 3.3 with a discussion of the presented concepts in terms of applicability to component based robotic system thereby considering the challenges introduced previously in section 2.2.1.

3.1. Model-based Approaches

Model-based approaches continue the idea of hardware redundancy yet the redundant component is a software model of the system (i.e., instead of a hardware component). The models are built upon the experiences with signal processing and plausibility check approaches [39]. Model-based approaches all follow a common basic structure depicted in figure 3.3. They all utilize a system model of quantitative or quali-

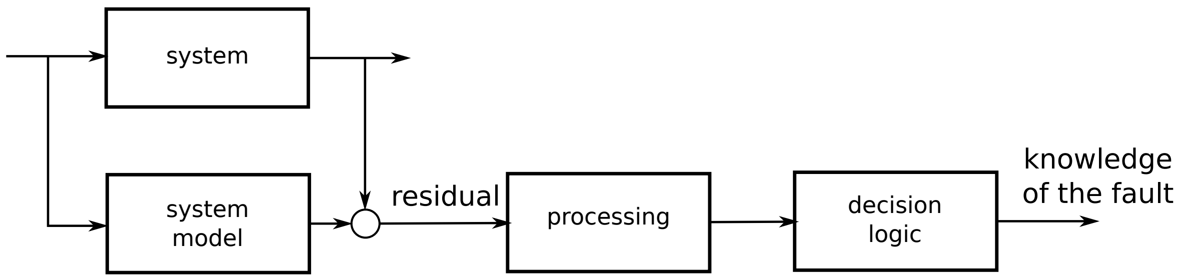


Figure 3.3.: Schematic description of the processing schemes of the model-based fault detection paradigm [39].

tative nature which is a description of the system's dynamic and static behavior. While the real system is in field the model runs in parallel and is driven by the same inputs. The assumption is that that while processing the same inputs, the outputs computed by the model will be identical or similar to the output of the real system in a fault-free situation. Calculating the difference between the real and the reconstructed values yields valuable information about the current health state of the system. This process is also called residual generation. Unknown disturbances and model uncertainties negatively influence the residual generation and need to be treated adequately to reduce false alarms [39]. Hence, additional processing in terms of filtering and extracting can be optionally applied to the residuals. Subsequently, some decision logic is applied in order to decide whether the residuals indicate a fault or not.

3.1.1. Analytical Approaches

Analytical approaches constitute the first type of model-based fault detection techniques dating back to the early 1970s [73]. The model exploited in analytical approaches is a mathematical description of the system grounded on first principles (e.g., physical laws) [15]. These approaches are therefore well suited for solutions which operate on information very closely related to the hardware of the targeted system. Commonly applied techniques are parameter estimation [67, 46], adaptive filtering [57, 113], (variable) threshold logic [65], or statistical decision making [113]. Next, the work of Freyermuth [46] will be presented in more detail in order to facilitate the basic idea of an analytical fault detection approach.

The task in the work of Freyermuth is to monitor and detect faults in an industrial robot with n many rotational axes. Common faults in such systems arise from deficient maintenance, mechanical collisions, wear of the mechanics or heating problems. The proposed approach is a parameter estimation based technique which means that the evidence of a fault is estimated from the parameters of a mathematical system model rather than from the model's output. Furthermore, the parameters in the proposed model represent physical process coefficients like friction or moments of inertia which is why the approach is said to be based on first principles.

In order to exploit the physical process coefficients for fault detection it is necessary to either measure them directly or estimate them based on other variables which can be accessed more easily. Freyermuth chose the latter solution which is less costly as no modifications of the system and no new hardware is required. He proposed a mathematical model which links the physical process coefficients to sensor measurements for the currents of the electro motors of the robot, angular velocities of the joints, and angular position of the robot's parts. The model is defined as a system of non-linear differential equations of the static and dynamic, behavior of the robot which is a common representation for such systems [78]. The model comprises one equation for each of the n axes which is defined as follows:

$$M_A(t) = J(\varphi_0, m_L)\dot{\omega}(t) + M_{D0} sign(\omega(t)) \tag{3.1}$$

$$+ M_{D1}\omega(t) + M_{D3}\omega^3(t) + M_G cos(\varphi(t)) + e(t) \tag{3.2}$$

Whereas the elements denote the following:

$$
\begin{aligned}
M_A(t) &: \text{Time dependent actuating drive torque} \\
J &: \text{Position and load dependent moment of inertia}
\end{aligned}
$$

$$
\begin{aligned}
M_{D0}, M_{D1}, M_{D2}, M_{D3} &: \text{Torque friction coefficients} \\
M_G &: \text{Gravitational torque} \\
m_L &: \text{Mass of load at end effector} \\
\varphi(t), \omega(t), \dot{\omega}(t) &: \text{Position, Velocity, Acceleration} \\
e(t) &: \text{Stochastic disturbances and slight model inaccuracies}
\end{aligned}
$$

This formula combined with the proportional relation $M_A(t) = \mathit{\Psi} I_A(t)$ connects the measurable parameters $\mathit{\Psi} = [J, M_{D0}, M_{D1}, M_{D2}, M_{D3}, M_G]$ to the latent parameters $\Theta = [I_A(t), \dot{\omega}(t), \omega(t), \varphi(t)]$. Each single latent parameter Θ_j represents a physical process coefficient except for some proportional sensor-specific factors. In the remainder of this example single latent variables Θ_j will be denoted by p_j. Monitoring and fault detection takes place in two steps. Firstly, the values for Θ are estimated based on samples taken from the sensors. Secondly, each estimated $p_j \in \Theta$ is classified in order to decide whether it deviates significantly from its expected value thereby indicating a fault.

Estimation of the latent parameters Θ is done by first taking $n+1$ consecutive samples for the measurable values $\mathit{\Psi}$ and inserting each of the samples into formula 3.1 which results in an equation only dependent on Θ. Overall, the equations represent an over-determined equation system which is transformed into a regression equation and is solved yielding estimates for Θ [69]. This estimation technique requires the number of samples n to be greater than the number of model parameters in equation 3.1.

In the classification step the estimates are examined for unexpected deviations which eventually indicate a fault. First, for each p_j a smoothed value μ_j is computed using a moving average filter (MA) thereby reducing the influence of noise and other uncertainties. Subsequently, each μ_j is transformed relative to an expected mean $\mu_0(p_j) \equiv \mu_j$ and variance $\sigma_0^2(p_j) \equiv \sigma_0^2$. The expected values are assumed to be known from experiments with the system or a dedicated training phase. The result is fed into membership function according to the concept of fuzzy logic [52] which defines the membership of the value μ_j to the linguistic expressions "numerical value of the respective coefficient changed" or "numerical value of the respective coefficient decreased". In case a μ_j is classified as being member of one of these expressions it is said to deviate significantly which again is interpreted as a system fault. In case where μ_j does not belong to any of the two expressions, no fault in the system is reported.

Another approach has been proposed by Fathi et al. with the goal to monitor a feedwater subsystem of a coal-fired power plant [42]. Regarding this system, common issues are problems with the deaerator, its controller and the gas transportation lines. The proposed method is based on structural decomposition of the system and an additional modeling of each of the resulting sub-systems with an adaptive Kalman filter [75]. In [113] Selkäinaho and A. Halme demonstrated their approach for fault detection in a dynamic positioning system of a ship. The model is realized as an adaptive non-linear filter [60] which describes the ship dynamics in terms of: a) externally generated movements forced by wind and waves and b) intendedly introduced forces by the ship's thrusters. Simulated faulty situations comprise a loss of a thruster and a gyro-compass fault. Another analytical approach was proposed by Saif and Guan in [108]. The system to be monitored is a vertical takeoff and landing (VTOL) aircraft. In this work the authors assume that a linear, time invariant, dynamical model is sufficient in order to detect faults in the sensors and actuators of the aircraft. The model is based on the unknown input observer (UOI) theory which enables the consideration of unknown external disturbances in the modeled dynamics [129]. The applicability of the approach is evaluated in simulation.

Clark et al. applied fault detection to a hydrofoil boat in order to detect incipient faults of the accelerometers, roll gyro and yaw rate gyro-compass which might happen due to wear [31]. The used model is based on the dedicated observer scheme [32] where each instrument has a dedicated observer linked to a model of the boat state. In order to detect which one of the instruments is actually faulty additional logic is applied on the common output of all observers. Fourteen different single-faults were induced in a system with four different instruments and could be all detect even if some system parameters vary (e.g., mass of the boat).

These examples are representative for the category of analytical fault detection approaches and demonstrate the range and type of systems where solutions tightly coupled to first principles have been successfully applied. Limitations for the application of analytical models arise from the complexity of the system in terms of the number of components and their interaction. In this case mathematical descriptions become impracticable [42]. Furthermore, analytical models are not suitable for detection of faults in components with functionality represented on a higher abstraction level than physical laws (e.g., software components).

3.1.2. Knowledge Based Approaches

Knowledge based approaches also exploit a system model and feed it with identical real system inputs in order to detect faults by comparing outputs. Yet, the model used here is defined in a more qualitative and abstract way rather than as an exact and physically correct description. By this means, complex systems can be modeled easier by abstracting from irrelevant and focusing on relevant details [39]. Furthermore, components of a system which are not based on first principles can be modeled, too.

Again, the discussion of the knowledge based fault detection literature begins with a detailed example which will highlight vital characteristics of this approach and the differences to analytical approaches. The detailed example is the work of Steinbauer et al. on a fault detection, diagnosis and repair approach which is applicable to the control software of mobile robots [117]. Although the approach is capable of identification of the source of the fault and can trigger recovery routines the description given here is concentrated on the monitor part of the approach to keep this example focused on the topic of this thesis. The targeted system in Steinbauer's work is a RoboCup MSL robot interacting within the robot soccer scenario [43]. The control software of the robot comprises separated modules called services. Each service is an independent process and implements a specific task e.g., image processing, world modeling or planning. The overall functionality of the system is given by the single tasks of the services and their interaction. The control system is organized in three levels with increasing abstraction. At the lowest abstraction level services like the laser abstraction service or the CAN-Bus directly communicate with the hardware. The next level contains services which perform computation of sensor inputs including image processing, or sensor fusion tasks. The planner is located on top of this hierarchy implementing an abstract symbolic representation of the knowledge of the robot together with reasoning capabilities. The various services can communicate with each other through two different techniques: 1) remote method calls and 2) the so called event channels. While the former follows a client/server paradigm the latter implements a publish subscriber relation between the participants and allows for a loose coupling of the participating components. In order to monitor the control software Steinbauer et al. apply the concept of dedicated observers. An observer monitors either the behavior of a single service or the communication between different services. By this means, a fault in the control system is detected if one observer determines deviation from the expected behavior. In order to represent different dynamics of components and the interaction between them, several types of observers have been defined:

▷ Periodic event production: This observer checks whether a specific event e is at least produced every n milliseconds. An example for this observer is the event *MotionDelta* containing odometry data which is produced every $50\,\mathrm{ms}$ by the Motion service.

▷ Conditional event production: This observer checks whether an event e_1 is produced within n milliseconds after an event e_2 occurred. An example for this observer is the event *WorldState* which is produced by the *WorldModel* service after an event *ObjectMeasurement* occurs.

▷ Periodic method calls: This observer checks whether a service calls a remote method m at least every n milliseconds. An example for this observer is the *RangeSensor* interface of the Sonar service which is regularly called by the *BehaviorEngine* service.

▷ Spawn processes: This observer checks whether a service spawns at least n threads.

The types of different observers were defined based on experiences made during the development of the mobile robot platform and its components. For the evaluation of the proposed diagnosis system and its implementation several experiments has been conducted on a team of soccer playing robots. Two types of faults were induced into the control system: a deadlock fault and a service crash deadlock whereas both could be detected successfully.

The example above demonstrates the benefits of utilizing a knowledge-based description of a system for fault detection thereby spanning the fault detection capabilities over several abstraction levels of the system. This allows to a) focus on important features, b) reduce the complexity of the created detection model and c) model software components of a system. These facts have been exploited in many other fault detection approaches for various systems. A model can be completely hand crafted as in the example above but it can also contain parameters which can be adjusted based on training input as in the work of Freitas et al. [35]. The authors proposed an approach for fault detection of locomotion faults of a waiter robot and a Mars-rover. A robot is assumed to be a complex non-linear process and is modeled as a mixture of linear processes. The different states of the system (i.e., normal and faulty) are explicitly represented by a discrete variable in the model. The distributions over the parameters of the model are estimated with particle filters, whereby the generation of new particles is directed with the help of a Kalman-filter resulting in a more directed sampling. The approach has been applied to two different mobile robot platforms. The first one is a waiter-robot and its purpose is to facilitate research on general purpose skills e.g., skills necessary for fetch & carry tasks [43]. The other system is a Mars-rover developed for extraterrestrial missions. In both cases only sensors and actuators already present in the system were used in order to detect faults in the locomotion of the robot. Here, the examined faults were inspired by experience with the particular platform e.g., a "rock under a wheel" fault for the Mars-rover.

Another knowledge-based approach was proposed by Bajawa and Sweet [8]. They applied a fault detection and diagnosis system called *Livingstone 2* to the main propulsion system of a spacecraft. Livingstone 2 is a consistency-based reasoning technique which operates on a discrete model of the monitored system thereby maintaining a belief state over time of the system's current mode (i.e., normal or faulty). The model consists of discrete state variables which represent states of the components of the system, constraints and transitions between this variables, as well as input commands to the system. The belief state of the system is updated over time based on an initial state and consecutive observations taken from the actual system. In each step the most consistent belief state is found by first finding the set of the most consistent belief states and then choosing the one with the highest a-priori probability. Bajawa and Sweet successfully demonstrated the performance of Livingstone 2 applied to a main propulsion system by inducing and detecting ten different faults.

Narasimhan and Bronston proposed in [93] a framework for fault detection and diagnosis called HyDE which offers the possibility to exploit different techniques for fault detection. HyDE supports stochastic modeling, allows for discrete and continuous variables in models as well as combinations of both which leads to the concept of hybrid models. The applicability of the framework was demonstrated in different NASA projects. For example, in the *Drilling Automation for Mars Environment* project HyDE was applied to detect known faults[1] related to the drill, the bit, and the auger of the system. The detection system based on HyDE was able to successfully identify faults like chocking of binding in about 85% of the cases with a false positive rate of about 5%. Another application took place in the *Autonomous Lander Demonstrator* project where this approach was applied to the propulsion system in order to detect common faults such as stuck valves and regulator failures.

Another knowledge based fault detection approach was proposed by Mikaelian et. al. [124]. They argue that models reflecting solely hardware components cannot capture occurred faults correctly in any case. Therefore, they proposed a novel approach which enables the mutual modeling of software and hardware components based on a probabilistic, hierarchical, constraint-based automata [133] which is a compact, hierarchical form of a Hidden Markov Model [104]. The application of this method was demonstrated on a vision-based rover navigation-system and a formation flying test-bed called SPHERES [96]. This test-bed was used to demonstrate novel control and autonomy technologies in the International Space Station.

Knowledge based approaches benefit from qualitative modeling and can therefore be applied to complex systems in order to detect faults in hardware and software. Yet, knowledge and analytical approaches share some common drawbacks. Firstly, in order to successfully apply them to real systems, a certain amount of expertise is always necessary. This includes knowledge about the interesting components of the system and their interplay. Furthermore, expertise needs to be gathered in order to select parameter values which represent normal system behavior. Secondly, depending on the complexity of the system changes to the models can be time intensive. Consider for example an upgrade of one of the central processing units (CPUs) in the system. In this case software components need less time to process and can produce outputs with a higher frequency. This consequently influences parameterization of the fault detection model which requires human intervention.

[1]A known fault is a fault which was experienced in the field at least once.

3.2. Data-driven Approaches

Data-driven fault detection techniques also employ a model of the system in order to detect faults. Yet, in contrast to model-based techniques the model is not hand-crafted but is learned from data gathered in the system. This is advantageous particularly when dealing with complex systems because hand crafting a model in this case may be a tedious task. The scheme for data-driven fault detection is depicted in figure 3.4. It can be represented as an extension to the model-based paradigm where an additional module (trainer) exploits gathered input and output data of the system in order to learn a system model. Another advantage results from the fact that characteristics

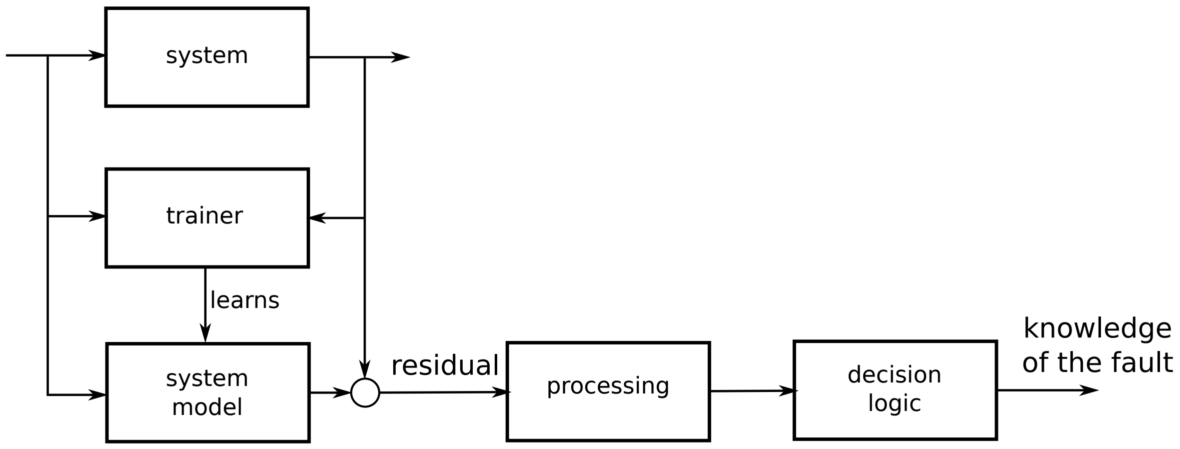


Figure 3.4.: Schematic description of the processing schemes of the data-driven fault detection paradigm.

about the system's health status are extracted from the training data which reduces the amount of necessary system expert knowledge in order to build an expressive fault detection model. In case of changes to the system, adaptation of the model can often be solved by gathering new data and re-training. The drawback of data-driven approaches is that they rely upon the number and the quality of the gathered training data.

Data-driven techniques have been applied to a number of different technical systems like aircraft engines [109], automotive [83, 98], autonomous underwater vehicles [5], space shuttle engines [101, 112, 10] or helicopters [100]. In robotics the most common targeted applications are the detection of faults in the locomotion of mobile robots [36, 37], faults in actuators [102] and sensors [30].

In their work *Fault detection in autonomous robots based on fault injection and learning* Christensen et al. [30] present an approach to detect faults occurring in the locomotion of a swarm-bot robotic platform [87]. The work is based upon the following hypothesis:

> *A hardware fault changes the flow of sensory data and the actions performed by the control program of the robot. By detecting these changes, the presence of a fault can be inferred*.

To derive a model which captures the sensor/action relationship the authors exploit a time-delay neural network (TDNN) [33]. TDNNs are feed-forward networks that allow reasoning based on time-varying inputs without the use of recurrent connections. The input layer of the network represents groups of historical sensor data and actuator control signals. Available sensors in this scenario are: a light sensor, proximity sensors, a camera and infrared (IR) ground sensors. While some readings from sensors like IR and ground sensors can be feed into the neural network in a straightforward manner, other require specific preprocessing based on system expert knowledge in order to be useful. This is the case for the camera sensor where the image is not directly fed into the neural network because it is too high dimensional. Instead, it is preprocessed in order to compute distances to specific objects in the robots environment i.e., it is used as a range sensor. These distances are then fed into the network. The input layer of the network is fully connected to a hidden layer which again is connected to a single output neuron. This output neuron indicates whether the system behaves normal (ideally 0) or abnormal (ideally 1). A threshold based classification is applied to derive the actual system state. The authors examine two different faults in three setups. The first fault causes one or both wheel to stop moving (stuck-at-zero) while the second fault leaves one or both motor(s) at constant speed (stuck-at-constant).

The two faults were successfully detected in all three setups.

In [116] the authors train multiple local model neural networks to capture the input-output relationship for the components in a robot for which faults should be detected. The authors focus on detecting faults in the wheels of a robot. The input corresponds to the different voltages of the motors which accelerate the robot's wheels and the output consist of the speeds of the wheels, respectively. Supervised learning is used to train neural networks as models for the single wheels of the robot (i.e., local models). During operation, the speeds predicted by the local models are compared to the actual speeds and the residuals are computed. For evaluation purposes, a blocked wheel fault has been simulated and successfully detected.

Another interesting approach for data-driven fault detection was proposed by Canham et al. in [23]. It is biologically inspired and follows the idea of artificial immune systems (AIS) [45]. The immune system which can be found in higher living beings is a multi-layered, distributed system that can identify numerous disease-causing foreign organisms and other harmful effects [23]. The basic idea of an AIS fault detection approach is to distinguish between self and non-self [23]. In fault detection, self corresponds to fault-free operation while non-self refers to observations resulting from a faulty system behavior. Canham et al. applied an AIS based fault detector to a mobile robot in order to detect faults in its locomotion. In their work the authors assume that the system can be described as a function which maps input values to a single output in a deterministic manner. This function (self) is approximated with a set of detectors which represent different parts of the system state (i.e., different parts of the mapping). The approach has been evaluated on a Khepera [74] robot and a control module of a BAE Systems Rascal^{TM} [34] robot. The two systems are first trained during fault-free operation and their capacity to detect faults in their motion controller is then tested.

The just presented approaches cover only parts of a complete robotic system i.e., sensors or actuators whereas data-driven solutions which monitor a complete system are not known to the author. However, recently a set of approaches has been proposed which target the efficient monitoring of data in the system and offer pre-processing capabilities which are well suited in order to develop holistic data-driven fault detection approaches. In [11] the authors introduce a learning algorithm to automatically discover a state-transition model of the system's behavior. The algorithm monitors the communication between architectural components, in the form of function calls, and finds the frequencies at which various functions are polled. Based on this, it determines the states according to what polling frequencies are active at any time. The approach is unsupervised and the authors call it *task-agnostic* which means that it can be applied to any new task or architecture with minimal effort. The authors also discussed the possibility to exploit the model for fault detection.

Niemueller et al. in [95] proposed a generic robot data base which taps into common robot middleware to record data produced at run-time. The system features the ability to record and store any and all data produced on the robot in real-time. It integrates with typical robot middleware like ROS [103] or Fawkes [94]. Furthermore, it requires only minimal configuration efforts and is easy adaptable to evolving data structures. Niemueller et al. demonstrate how to exploit the generic robot data base for post-mortem fault detection.

3.3. Discussion

The related work presented in this chapter shows the huge spectrum of fault detection solutions for various application domains and systems present in the literature. We now want to discuss them in the context of the challenging aspects for fault detection in CBRS system.

Recalling the discussion from section 2.2.1 an adequate detection approach should i) cope with frequent system changes, ii) rely only on a minimal amount of system expert knowledge, iii) cope with uncertainty and variation in the system's behavior, iv) consider faults residing within software and hardware components of the system.

Frequent changes and minimal system expert knowledge render data-driven fault detection approaches more suitable solutions for CBRS systems in contrast to hand crafted model-based solutions due to their ability to derive a model from exemplary data. Variation and uncertainty in the target system behavior is frequently tackled in the literature with statistical modeling techniques [113, 58, 105, 26, 138], fuzzy logic [46] or artificial neural networks [30]. This work lines up with these ideas and exploit statistical modeling techniques as discussed in the upcoming chapter. The detection of faults in the software & hardware constitutes the last challenge when dealing with fault detection in CBRS. While hardware faults are well covered, only a few approaches exist in literature that directly address the problem of the detection of software faults in general [117, 124] and fault detection in Component Based Robotic System [117], in particular. Though, these solutions belong to the class of model-based approaches which already have been rendered less suitable for fault detection in CBRS. Further presented solutions for robotic systems target the detection of faults in parts of the overall system like locomotion, actuators or sensors and thus have the potential to be applicable for these particular tasks in CBRS. Yet, even if the aforementioned partial detectors could be combined to a single one, still the resulting solution would not be capable to detect faults residing within the software components of a CBRS. In contrast, the detection approach presented in this thesis makes no assumption whether faults occur in software or in hardware and thus provides means to detect both types.

This chapter introduces the general concept of fault detection, gives an overview over the various solutions proposed in literature and illuminates them in relation to the challenges in this work. The upcoming chapter is devoted to the data-driven fault detection approach developed in the context of this thesis.

4. The AuComFault Detection Approach

The discussion at the end of the previous chapter suggests that state-of-the-art fault detection approaches do not entirely tackle the challenges discussed in section 2.2.1 which arise in the context of Component Based Robotic systems. In this chapter I now introduce the AuCom fault detection approach which is designed to cope with these challenges. The approach belongs to the class of data-driven techniques and exploits the communication between the system's components as the only source of information for fault detection. The features exploited to build the model are based on generic attributes of the communication only. To create a model of the system's normal behavior I utilize statistical techniques and exploit durations between the communicated data in the system as features thereby assuming that a deviation from the statistics over the durations indicate a fault. By this means, my approach is robust in the face of minor changes as well as uncertain and variant behavior and can be easily adapted to changes if necessary. In addition, it can be integrated with minimal changes to the targeted system, and is largely independent from the presence of system expert knowledge.

This chapter is structured as follows: Section 4.1 describes the data selected for fault detection in this work. It also contains a discussion of alternative data sources and provides the arguments why I chose the communication between the components as input. Subsequently, section 4.2 introduces the notion of a complex robotic system as a stochastic process. Using this definition the features selected to represent the input data in the algorithm are introduced and discussed in relation to alternative modeling techniques for such processes. In section 4.3 the model is introduced while section 4.4 is concerned with the actual estimation of the system state in terms of normal or faulty behavior. This chapter concludes with a summary of the findings in section 4.5.

4.1. Input Data Selection

An initial step in the development of a fault detection approach is the selection of an adequate subset of system data for further processing. The main concern when selecting data is to find the balance between the reduction of the input size to speed up the detection process while simultaneously preserve enough fault sensitive information to enable successful fault detection. Besides this, in literature other criteria have been used to select the appropriate subset. For example, Freyermuth [46] decide to estimate fault-indicating attributes in an n-axes robotic arm like friction and moments of inertia from data already present in the system rather than measure them directly. In particular, he derives these attributes from dc-motor currents, motor angular velocities and axis positions. By this means, no additional sensors are needed which simplifies data recording and reduces costs but may also render the approach less accurate. Another example is the work of Steinbauer et. al. [117]. Here, the goal is to detect different faults in the control software of a soccer playing robot. To do so, the authors exploit the communication between different services of the system and the number of threads each service usually spawns during operation. This data can be retrieved uniformly for all services which reduces the implementation efforts and enables easy adaptation in case new services are introduced in the system.

In this work the decision which data to choose is partially driven by the challenges for fault detection discussed in section 2.2.1. In particular, it is influenced by i) the need to cope with frequent system changes, ii) the challenge to integrate the detector minimally invasive into a system, and iii) the request for minimal usage of system expert knowledge for fault detection. To tackle these challenges I follow a similar idea as in the work of Steinbauer et. al. [117] and exploit the inter-component communication of a system as input. In contrast to Steinbauer I omit the usage of information like the number of threads per component which additionally reduces the coupling of the approach to a specific robot or its underlying operating system. I also omit any explicit information about the structure of the target system (e.g., the inter-connection between the components). If I would like to exploit this information, adaptations to the learned detection model would be necessary each time the system is subject to changes. As changes were identified to be frequent in the type of systems regarded in this work (see section 2.2.1), adaptations of the model would be frequent, too. As a consequence, means to automatically retrieve structural information would be necessary in order to still be able to match the challenges defined in the context of this work. Such means already exist in some software architecture frameworks like in the work of Nordman and Wrede [97]. Here, access to structural information is enabled because the authors provide an explicit description of the system with the help of a domain specific language. This explicit description can be also exploited to enhance a fault detection model. However, an automatically processable description cannot be assumed for robotic systems in general and thus relying on it would eventually restrict the set of targeted systems.

A crucial aspect of data exploited for fault detection is that it needs to contain information which characterize different system states and thus can be used to distinguish them. I assume this characteristic for the inter-component communication of a system (which is the input of the AuCom-detector) based on the following considerations: *The overall behavior of a Component Based Robotic System (CBRS) emerges from the interaction and communication of its components. This communication contains patterns which are characteristic for a specific system state (e.g., normal or faulty). Based on this, I assume that if I have a representation of these patterns recorded during the system's normal behavior I can exploit it for fault detection by interpreting significant deviations from it as system faults.*

In order to substantiate this statement let us consider an example derived from the robotic system BIRON [59]. Research on BIRON is dedicated to the development of a robot companion capable of social like interaction [135]. This requires many different hardware and software components which realize cognitive capabilities like object, face and obstacle detection/recognition, speech understanding, object manipulation or navigation. However, to keep the example simple I only regard a subsystem of BIRON. It consists of the following three elements: a laser, an odometry and a Simultaneously Localization and Mapping (SLAM) component. All three components are depicted in figure 4.1 together with their communication paths indicated by black arrows. In case where all three components perform as intended (which equals nor-

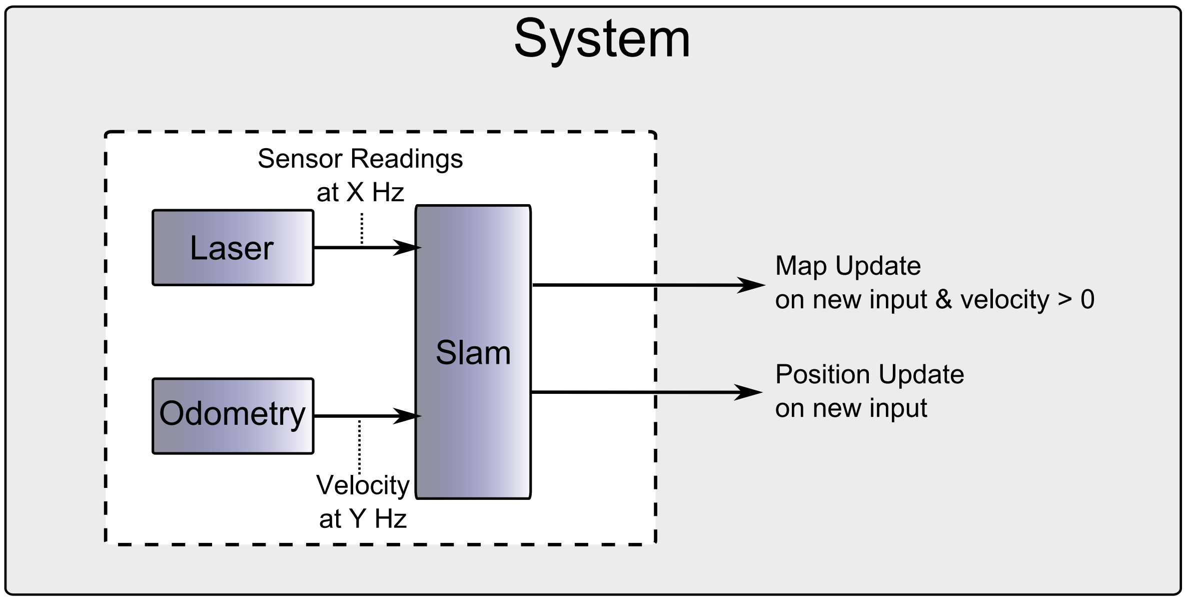


Figure 4.1.: Exemplary communication in a subsystem of a mobile robot. The subsystem consists of a laser, an odometry and a SLAM component. The laser and the odometry produce outputs at fixed rates but independently of each other. The SLAM component depends on the input of both laser and odometry and produces two outputs which are dependent on its input.

mal system behavior), the communication can be described as follows: The laser component provides readings from the laser scanner to the system at a fixed rate independently from other components. The odometry component estimates the robot's driven distance over time and supplies this information also at a fixed frequency. The SLAM component depends on the input of both laser and odometry and produces two different outputs. Each time new input data is available SLAM estimates the robot's current position and provides it to other components in the system. Thus, the generation of the position is directly correlated to the outputs of the laser and odometry components. Additionally, while the robot is moving, SLAM updates the system's internal environment representation. The output of this information is also correlated to the odometry and laser output but in addition it is also bound to the condition that the robot is moving thus rendering the resulting pattern more complex. Each of these input/output descriptions can be seen as a simple pattern present in the ICC and based on the previously stated considerations I expect that significant deviations from such patterns indicate a system fault.

An additional benefit gained when exploiting component communication for fault detection is that recording it can often be realized upon the build-in communication functionality of the system (i.e., the functionality provided by the system's middleware). If this is the case, the recorder acts as another system component and recording takes place without modifying the system's components thus contributing to the request for minimal invasive integration.

In the remainder of this work the communication between the system's components is regarded as a time-series D of communication elements. Each element of D is called a *data-point* and is denoted as d. Each d consists of two parts: a payload and metadata. The payload represents the specific information a component wants to communicate to other components in the system while the metadata encompasses general information like the timestamp of occurrence or additional information about the sender of a data-point.

4.1.1. Attributes

After discussing the choice of the input data for the AuCom-detector, this section is concerned with the selection of attributes used for feature generation in further processing steps. The goal of this step is to further reduce the input for the algorithm while preserving fault-sensitive information and transforming it into a representation which can be processed automatically. Continuing the subsystem example from the previous section (Figure 4.1) one could use existent expert knowledge about each single component of the system and extract specific information from its data-points. An example on how to exploit expert knowledge about a component for fault detection is given by Christensen et. al. in its work on fault detection in mobile robots [30]. Christensen relates specific transformations of the sensory information of the robot to commands of the control program responsible for the locomotion of the robot. In particular, each image gathered by the robot's camera sensor is segmented into sixteen slices. Each slice corresponds to a single input value of the algorithm and represents the distance to the closest object perceived in this slice. Used this way, the camera sensor effectively becomes a range sensor for objects while the input for the algorithm is significantly reduced. Christensen also argues that this idea helps to optimize the amount of fault-sensitive information available in the subsequent processing steps. However, this procedure conflicts with the requirements resulting from the challenges defined in section 2.2.1 for the following reasons:

- ▷ It requires expert knowledge for each of the components in order to select the attributes.
- ▷ It introduces strong coupling between the fault detection approach and the actual components of the system leading to adaptations of the detector in case of system changes.
- ▷ It hampers the portability of this approach to other systems.

Consequently, specific attributes are unsuitable for the approach in this work. Therefore, I focus on generic attributes of the data-points which do not suffer from the aforementioned issues. An attribute is considered to be generic if it is common for all data-points in a system and if it can be extracted in an analogous manner from all data-points. This definition also includes data-points which might be present in the system in future e.g., due to a new component added to the system. It implies that generic attributes are independent of the payload's content and thus from the component which sends it. In addition, further processing for fault detection is decoupled from a concrete system. By this means, generic attributes increase the robustness of the approach to changes of the targeted system. On the component level changes to the content of a data-point do not influence the retrieval of generic attributes. On the system level, new components can be added or old can be removed without changes to the set of used attributes or their retrieval mechanism.

The selected attributes for feature generation in this work partially result from commonalities of the communication paradigms exploited in CBRS. Therefore, they are now shortly introduced in order to facilitate the description of the attributes. These communication paradigms are *Caller/Provider* and *Broadcaster/Listener* [21]. Caller/-Provider is also called remote procedure call (RPC) [13] or in the context of object oriented languages remote method invocation (RMI) [85]. It enables the calling of individual functions exposed through the component's interface across network boundaries. It features a point-to-point communication style on a per-function level. In the Broadcaster/Listener communication paradigm components are equipped with the ability to broadcast[1] and listen to messages [21]. The messages sent between the components are self-contained and are not restricted to a given signature of an operation which differs from the former Caller/Provider communication style. Extensions of this basic idea have been developed like Publish/Subscribe where components subscribe to *topics*. Afterwards, components will receive only messages which correspond to a given topic description [137]. This selective message delivery helps to reduce load on both, the overall communication of the system and the processing of

[1]Broadcasting refers to a method of transferring a message to all recipients simultaneously.

a component. Broadcaster/Listener has also been adopted in event-driven architectures (EDAs) [41]. Here, a message is called an event and it represents a significant state change of a component. Listener-components are notified about an event only if they match a subscription previously defined by this component [91].

In the light of these communication paradigms the first commonality exploited as a generic attribute is the fact that each data-point sent in the system has a sender component which I call its *Source*. Secondly, each data-point is also received by a set of recipients. This set may consists of only one recipient as it is the case of point-to-point connections or a selected set of recipients e.g., the set of subscribers to a specific topic in a Publish/Subscribe implementation. The set of components, which receives the data-point, is called the *Scope* of that data-point. The third and last generic attribute is derived from a semantic interpretation of data processed in robotic systems. In such systems consecutive data-points sent by the same component can be semantically connected. This connection indicates whether a data-point represents semantically *new* information in the system, an *update* of already present information, or whether the data-point represents a *deprecated* piece of information. This generic attribute is called the *Type* of the data-point. To illustrate the idea of this attribute consider the following example: A user enters the field of view of a robot in order to interact with it. The robot's face recognizer detects and identifies the user and sends a message into the system which indicates that a new face has been recognized. During the interaction a user stays in the field of view of the robot and the face recognizer keeps recognizing him and sends data-points which represent the same face but with possibly different coordinates of the face relative to the robot's position. These data-points represent updates of information already existing in the system. Finally, the user finishes the interaction and walks away from the robot and its face is no longer detected. After a while the corresponding information gets deprecated and the face recognition component sends another data-point which indicates the invalidity of the face. This semantic information may not be present for all data-points send in a system either because the data does not have this semantics or because it has not been considered during the development of the component. However, it provides valuable information for fault detection for data-points where the semantic is valid. In the case of the face recognizer for example few frequently updated interaction partners may indicate normal behavior while many new recognized faces in a row could be an indication of a fault. Data which does not correspond to this semantic has its type attribute set to the “new” value. Table 4.1 provides a consolidated view on the introduced generic attributes. In table 4.2 the different values of the type attribute together with the corresponding semantic interpretations are summarized.

Attribute	**Short description**
Source	The sending component of a data-point. Possible values are single components.
Scope	Representation of the receiving components of a data-point. Possible values are sets of components.
Type	Semantic interpretation of a data-point in relation to previous data-points sent by the same component. Possible values are listed in table 4.2.

Table 4.1.: Summary of the different generic attributes used to represent data-points in the inter-component communication of a CBRS.

Type attribute	
Value	**Semantic interpretation of the value**
new:	Represents new information in the system. It is also the default value for data which does not correspond to the type attribute semantic.
update:	Represents an update of already present information in the system.
deprecated:	Represents the deprecated status of an information in the system.

Table 4.2.: Summary of the type attribute values and the corresponding semantic interpretation.

4.2. Feature Extraction

In this section I introduce the novel concept of a Temporal Dynamic Feature (TDF) as meaningful representation of a data-point d based upon the generic attributes introduced in the previous section. This representation is the input for the further processing steps of my algorithm i.e., Model Training (Section 4.3) and System State Estimation (Section 4.4). Thus, it has to encapsulate information about communication patterns in the system's ICC which are assumed to be sensitive to faults. This section starts with an introduction of the basic idea of a TDF and explains its benefits as a representation of the system's communication in comparison with state-of-the-art approaches suitable for modeling such data. Subsequently, a TDF is defined in mathematical terms and the procedure to generate TDFs from input data is presented.

A complex robotic system can be thought of as a complex process that is not completely visible or accessible. From this point of view a meaningful way to deal with the ICC of a system is to regard it as the output of a *stochastic process* which is monitored at discrete points in time (here the points in time correspond to the timestamps of sending data-points) and has a discrete state-space consisting of the different data-points. The result is a time-series ordered based on the timestamp when data-points d were sent as depicted in figure 4.2. A prominent approach to model a state-space and time discrete process is to assume that the process satisfies the *Markov Property* and thus can be expressed as a Markov Chain [122]. The assumed *Markov Property* has the following definition: Given a history of values at points in time $t-n$ to t for a variable X. In order to predict the value of X at $t+1$ it is sufficient to know the value of X from time t [14]. A Process which fulfills this property is also called *memoryless* because the history of the time-series is discarded. This model is often used because of its efficient implementation and often good approximation capabilities [118]. In cases where the data is more complex, the incorporation of the history becomes necessary leading to models like the Markov Chain of order n [14]. *Hidden Markov Models* (HMM) constitute another extension of the Markov Chain. Here,

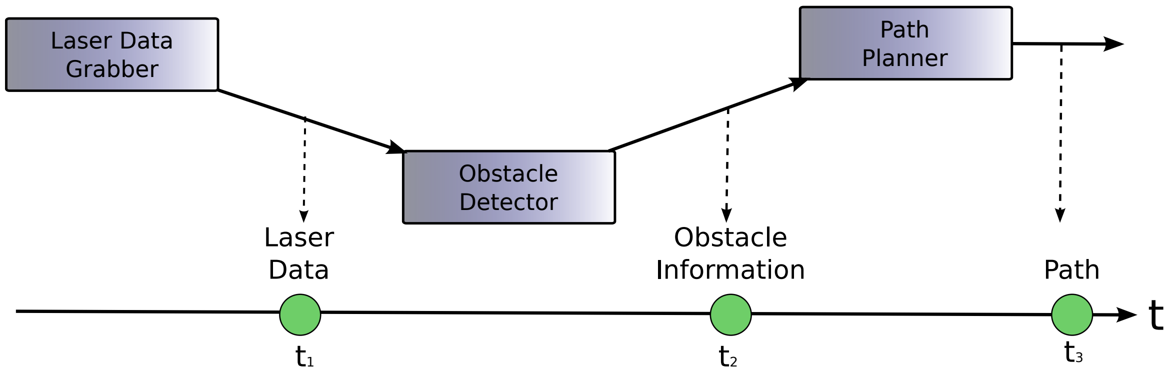


Figure 4.2.: Representation of the inter-component communication as a time-series ordered based on the timestamps when the data was sent.

the assumption is that the actual state of the process is hidden and the observed time-series values represent process outputs which can be related to the states of the process [106]. Hidden Markov models are known for their application in temporal pattern recognition such as speech [40], handwriting [89], or gesture recognition [88].

Regarding the ICC of a system modeling the Markov assumption between consecutive data-points proves to be challenging. In order to explain this let us continue the three-component example from section 4.1 (Figure 4.1) and extend it with a camera and object detection component. Both components act independently from the three components introduced in the initial example. The communication between these new components can be described as follows: The camera component outputs images grabbed from the camera sensor at a fixed frequency while the object detection component analyses each image for detectable objects in the field of view of the robot and generates an output if it finds any. In order to model this communication and the one described for the initial example, two separate Markov chains could be used. Both chains together would then represent the system's normal behavior. However, as mentioned in section 2.2.1 in this work explicit information about the system's structure is omitted and thus the approach is not aware of this two independent system parts. Consequently, the communication in this example is regarded as a whole where the two communication chains are interleaved leading to a more challenging modeling situation in case the model is based upon the Markov assumption. One could try to cope with this issue by considering a history of k data-points which would enable to relate consecutive elements of one communication chain to each other even if it is interleaved with non-related data-points (i.e., in this case, the ones coming from the second communication chain). This can be done in two different ways: Firstly, the current data-point could be related to its k consecutive predecessors which corresponds to the idea of a Markov chain of order k. Secondly, k consecutive data-points could be grouped to one *composite* data-point of higher dimensionality and the resulting time-series of composite data-points could be treated as a common first order Markov chain. The two solutions are visualized in figure 4.3. The problem in both

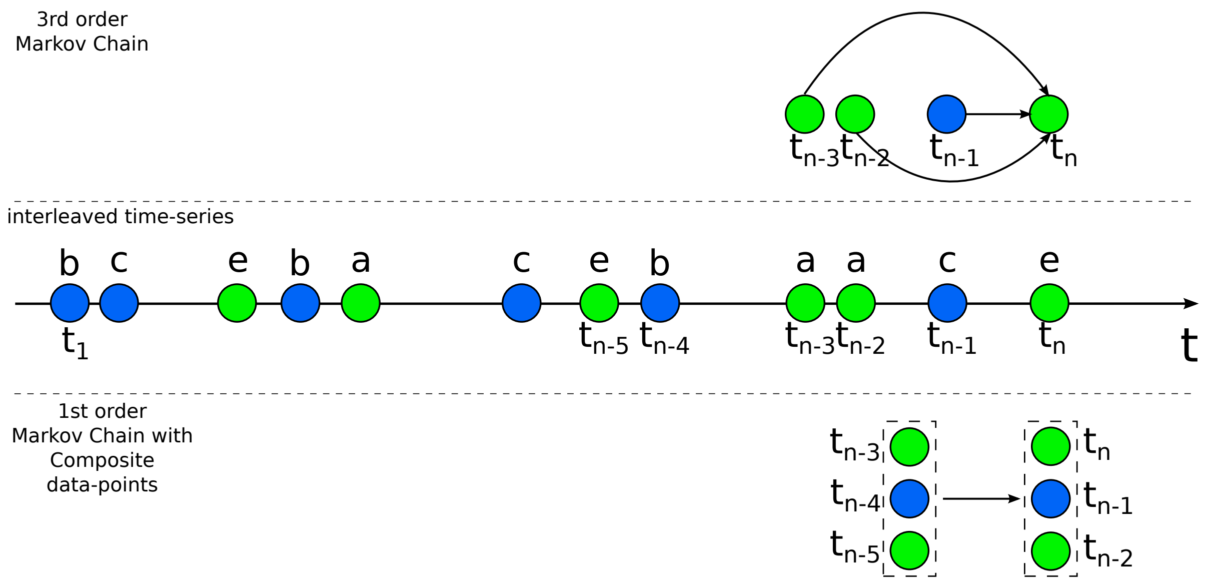

3rd order Markov Chain
t_{n-3} t_{n-2} t_{n-1} t_n
interleaved time-series
b c e b a c e b a a c e
t_1 t_{n-5} t_{n-4} t_{n-3} t_{n-2} t_{n-1} t_n
t
1st order Markov Chain with Composite data-points
t_{n-3} t_{n-4} t_{n-5}
t_n t_{n-1} t_{n-2}

Figure 4.3.: Two possible means to cope with an interleaved communication chain by exploiting the Markov property. The first solution (upper part of the figure) is based on a Markov chain of order k (here $k = 3$). In the second solution consecutive data-points are grouped first. The resulting composite data-points are modeled with a first order Markov chain. Different data-points are labeled with letters. Color coding indicates the different communication chains.

cases is that the model complexity increases exponentially with k either in the number of the needed probability distributions for the first solution [14] or in the size of the state-space for the second one. Furthermore, each non-related data-point of the k data-points in the regarded history projects the feature derived from this history into another sector of the state space. By this means, the actual distribution is artificially and unnecessarily spread across several areas (thinning-out), which increases its variance. The ICC communication could also be modeled by exploiting a Hidden Markov Model. In this case, data-points are interpreted as observations emitted in either a normal or faulty (hidden) system state. Observations in a state can be single data-points or (similar to the second case of the Markov chain approach explained above) can be grouped to form composite data-points. Such a model could be then used to estimate whether the system is either in a normal or faulty state given an observation (i.e., a data-point or a composite data-point). Unfortunately, to learn this model training data from faulty situations would be necessary in order to estimate the emission probabilities of the observations in the faulty state. Due to the considerations at the beginning of this chapter this data is not available which prevents the application of such a model. Alternatively, one could remove the faulty state from the model and learn only emissions for the normal state resulting in some kind of a degenerated HMM. This model would correspond to the previously described Markov Chain model but without the transition probability matrix between consecutive data-points or composite data-points since there is no description of transitions between consecutive observations in a HMM. This renders the one-state HMM less expressive than the Markov Chain solution and therefore also a less suited solution.

In this work I follow an alternative way to model the inter-component communication. At first data-points d of the communication are represented as *composite features* e by exploiting the three generic attributes (Source, Scope, Type) introduced in section 4.1.1. Next, the composite features e are used to generate Temporal Dynamic Features tdf which incorporate temporal dependencies to historical composite features $\hat{e}$ in the communication of the system. In other words, rather than trying to cope with temporal dependencies of the communication in the structure of the model, in this work this information is encoded in the features used by the AuCom-detector. A tdf for a given composite feature e consists of the durations between the time-stamp of e and the last occurrences of all other composite features $\hat{e}$ present in the system. To explain this idea in more detail, consider figure 4.4 which shows an extract of an exemplary interleaved communication in the system recorded between two points in time t_0 and t_n. The single circles on the time-line represent data-points, the related letters show the corresponding composite features and the color coding indicates that the communication consists of different communication chains which interleave each other. The upper part of the figure shows a Temporal Dynamic Feature tdf for

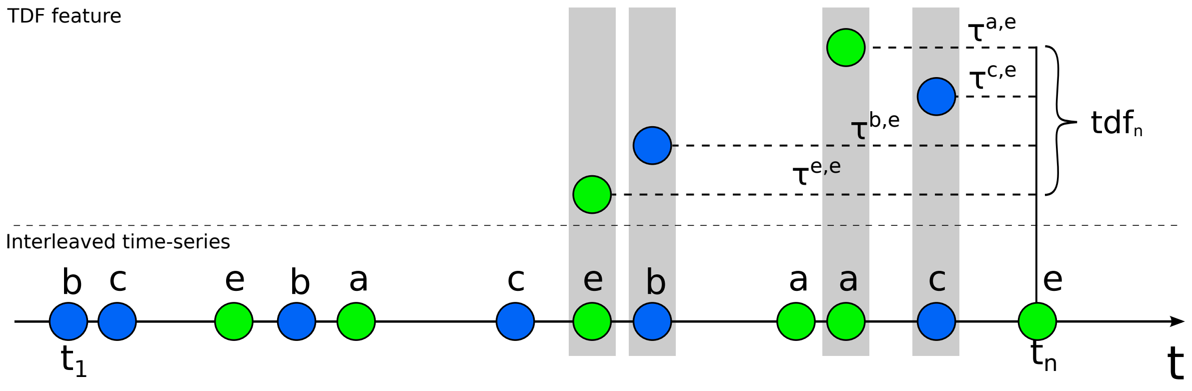


Figure 4.4.: Visualization of the TDF idea for a data-point at time t_n based on the interleaved communication example. The tdf_n consists of the set of durations between the last occurrences of all different data-points in the history of the time-series. Data-points are labeled with letters representing different composite features. Color coding indicates the different communication chains.

the data-point d at time t_n with the related composite feature e. The upper part also depicts the durations $\tau^{a,e}$ between the last occurrences of the different composite features present in the system (i.e., e, a , b, and c) and the composite feature e (belonging to d) at time t_n. It can be seen that a tdf does not consider all consecutive data-points in a communication but may skip some of them because it only encodes the last occurrences. In general, the TDF representation follows the same basic assumption as the previously described (Markov-based) approaches namely that current values of a time-series (here data-points represented by composite features) are related to past ones. However, the difference is that instead of assuming that the prediction can be done based on a fixed number of consecutive data-points, the hypothesis in this work is that most information can be extracted when regarding the contribution from the last occurrences of each of the different data-points d (i.e., data-points mapped to different composite-features e) in the system independently of their position in the history of the time-series. These last occurrences are highlighted in figure 4.4 with grey bars. From another perspective this means that while the previous models (e.g., the n-th order HMM) assume a fixed time horizon (e.g., the last n data-points) our features cover varying time horizons for the different data-points. In addition, while Markov-based models suffer from an increased complexity when the history growth in the case of TDFs this merely leads to larger values which need to be modeled but has no impact on the model's complexity. Furthermore, as TDFs do not explicitly represent the order of the data-points in the time-series they do not suffer from the thinning-out argument previously described for the Markov Chain based models.

Next, the generation of a time-series R of TDFs tdf from data-points d in the input time-series D is described in mathematical terms. As mentioned above, each $d \in D$ needs to be transformed into a composite feature e, first. The transformation of the complete time-series D results in a new time-series E of composite features. Afterwards, the durations between all $e \in E$ are computed and the corresponding tdf features are created. Let $\mathcal{D}$ be the range of different data-points d which are sent between the components of a given system. The mapping function which transforms d into a composite feature e is written as follows:

$$f(d) : \mathcal{D} \mapsto \mathcal{E} := e \tag{4.1}$$

where $\mathcal{E}$ is the domain of composite features e. In combination with the findings from section 4.1.1 the concrete mapping function in this work is defined as:

$$f(d) : \mathcal{D} \mapsto SOURCE \times SCOPE \times TYPE \tag{4.2}$$

Thus, each data-point d is represented as the Cartesian product of the generic attributes Source, Scope and Type. Applying this function to the whole time-series D yields the intermediate time-series E of composite features e.

$$f(D) = (f(d_1), \ldots, f(d_n)) := E \tag{4.3}$$

In order to define a Temporal Duration Feature tdf^b for a data-point d represented by a composite feature b, first the notion of the last occurrence $last(a, b) = \hat{a}$ of a composite feature a relative to b in the time-series E needs to be defined. Let t^b be the timestamp of b. Then, the last occurrence of a before b is defined as follows:

$$last(a, b) := \hat{a} = a : t^a < t^b \cap \ \nexists\, \tilde{a} \in E : \tilde{a} = a \cap t^a < t^{\tilde{a}} < t^b \tag{4.4}$$

Equation 4.4 states that $\hat{a}$ has occurred at time $t^{\hat{a}}$ which is smaller than t^b and no other composite feature with the value a has occurred between $t^{\hat{a}}$ and t^b in E. Based on this, the duration between b and the last occurrence $\hat{a}$ of a can be computed as:

$$\tau^{a,b} = t^b - t^{l(a,b)} = t^b - t^{\hat{a}} \tag{4.5}$$

Using the formulas above, a temporal duration feature tdf_b for $b \in \mathcal{E}$ can be defined as follows:

$$tdf(b) := tdf^b = \{\tau^{a,b} \mid \forall\, a \in \mathcal{E}\} \tag{4.6}$$

It is worth noting that $\mathcal{E}$ is a fixed set which means that all TDFs in a time-series have the same dimensionality. In order to relate equation 4.6 to E we write:

$$tdf(E(i)) = tdf_i := tdf^b : b = E(i) \tag{4.7}$$

and finally

$$tdf(E) = (tdf_1, \ldots, tdf_n) := R. \tag{4.8}$$

The subsequent pseudo code 4.1 illustrates how the extraction of the TDFs from a time-series E is computed. At first the necessary structures R and *lastTimestamps*

Algorithm 4.1 $R =$ tdf_generation($E, \mathcal{E}$)

```
1: initialize $R$
2: initialize lastTimestamps
3: **for all** $a$ in $\mathcal{E}$ **do**
4:     mean$_a$ $=$ calculateMeanDuration$(E, a)$
5:     lastTimestamps$(a) \leftarrow E(0)$.timestamp $-$ mean$_a$
6: **end for**
7: **for** $(i = 0; i < $ size$(E); i++)$ **do**
8:     **for all** $a$ in lastTimestamps.keys **do**
9:         $b = E(i)$
10:        $\tau^{a,b} = b$.timestamp $-$ lastTimestamps$(a)$
11:        $tdf^b(a) = \tau^{a,b}$
12:    **end for**
13:    lastTimestamps$(b) = b$.timestamp
14:    $R(i) = tdf^b$
15: **end for**
16:
17: **return** $R$
```

are initialized. R is the output time-series and lastTimestamps is a hash map used to track the timestamps of the last occurrences during the processing of an input time-series. Initially lastTimestamps is set to the timestamp of the first element of E minus the mean duration between two occurrences of the same composite feature e. This initialization is done for all $c \in \mathcal{E}$. The rationale here is that when starting to generate tdfs based on E there is no information present about the time before E. Thus, a reasonable guess is to assume that a particular composite feature e occurs at a point in time marked by the mean duration of self-occurrence before the beginning of E. By this means, there always exist a timestamp for all $e \in \mathcal{E}$ which prevents the generation of an incomplete tdf for composite features e at the beginning of E where some of the composite features from the set $\mathcal{E}$ didn't occur yet. Next, the loop which starts in line seven runs over the whole input time-series E and for each composite feature b

at position i the tdf^b is computed.

In the next section the discussion is focused on how to generate a model of the system's normal behavior based on the just introduced notion of Temporal Dynamic Features.

4.3. Model Training

Model training describes the procedure executed in order to create a representation of the system's states from collected data. Following definition 2.1.1.7 from section 2.1.1 which states that *fault detection uses the relations between several measured features to extract information on possible changes caused by faults* the model exploited in this work represents the normal behavior of the targeted system. Data-driven fault detection approaches in the literature often exploit additional models which represent specific faults in order improve their discriminative capabilities. For example, in the work of Christensen [30] the neural network based classifier is trained with data gathered during normal behavior of the system and while the wheels of the robots were stuck at zero or while they were rotating with a constant velocity which represents the two induced faults in Christensen's work. Such an approach provides additional information valuable for the diagnosis of the detected fault. Yet, it requires that exemplary fault data is gathered which might be challenging in some cases. In addition, relying on fault models may have the effect that if there is no model for a particular fault, it might remain undetected. In this work, therefore a model is employed which consists of data gathered during normal system behavior only.

In this thesis, the definition of the system's normal state is done in a qualitative manner based on the execution performance of the system in relation to the given task. In particular, this means that a human who knows the intended behavior of the system during a task decides whether the system behaves normal and thus whether the data gathered during this interaction can be used for training of the normal behavior model. Gathering normal data in this way requires only task specific knowledge to be present rather than detailed expert knowledge about the behavior of single components which constitutes another characteristic attuned towards the challenges stated in this work. It also enables the acquisition of training data by non-developers which reduces efforts when the model needs to be retrained at short notice. On the other hand, this way of defining normal state introduces additional variance in the training data in case the data is gathered by different users. The additional variance results from the fact that each user may have its own subjective view on normal behavior of a system in relation to a given task. In addition, some internal faults may remain unrecognized in case they do not alter the system's perceivable behavior.

The concrete model in this work is based upon statistical modeling techniques according to the following assumption: *Normal data instances occur in high probability regions of a stochastic model, while anomalies occur in the low probability regions of the stochastic model. [28]* By this means, we gain a statistically justifiable solution for fault detection. Furthermore, a statistical model naturally provides means to describe the uncertainty and variation present in the training data which results from the system's interaction (see section 2.2.1) and the previously mentioned way to define the normal state of the system. In the remainder of this section the mathematical description of the model is introduced together with a discussion of the selected statistical representations in relation to alternative ones.

4.3.1. Mathematical Formulation

Let $\mathcal{E}$ be the set of the different composite features a, b present in a system then the fault detection model M is described as a set of probabilities $P(\tau^{a,b})$ for single durations $\tau^{a,b}$ as follows:

$$M = \{P(\tau^{a,b}) |\ \forall\ a, b \in \mathcal{E}\} \tag{4.9}$$

Equation 4.9 defines M as the set of probabilities over the durations for all possible combinations of two composite features of a given system. From this definition it can be concluded that I assume the durations of a temporal duration feature tdf^b to be independent from each other which yields the following probability $P(tdf^b)$:

$$P(tdf^b) = P(\tau^{a^1,b}, \tau^{a^2,b}. \cdots, \tau^{a^n,b}) = P(\tau^{a^1,b}) \cdot P(\tau^{a^2,b}) \cdot \cdots \cdot P(\tau^{a^n,b}) \qquad (4.10)$$

The equation 4.10 serves as the basis for system fitness estimation and fault detection introduced in the subsequent sections.

An important aspect for the generation of the model M is the choice of a concrete probability distribution $P := P(\tau^{a,b})$. When selecting a distribution for P two things need to be considered. Firstly, the domain space of all $\tau^{a,b}$ is the set of positive natural values. Secondly, variation in the system's behavior and different system states during interaction are potentially reflected in the ICC thus leading to training sets for each $\tau^{a,b}$ which contain different regimes. Regarding P this means that an appropriate distribution needs to be able to cover positively valued multi-modal data. Probability distributions which are well suited to model durations have been developed in domains which are concerned with waiting-times between events [48]. Here, exemplary events are telephone calls, durability of atoms in a decay process, or the life-span of machines. The distributions used in this context are the exponential distribution, gamma distribution, erlang distribution or the poisson distribution. However, all these distributions are unimodal which makes them unsuitable due to the multi modality of the data. An alternative which covers both aspects is the histogram-based distribution [14]. This non-parametric approach can adapt to multi-modal data and allows to restrict its density to natural numbers (i.e., $\mathbb{N}$). However, a problem of this distribution is that the estimated density has discontinuities and that it requires a lot of training examples because no assumptions are made about the shape of the modeled probability. In extreme cases adjacent duration values may have widely differing probabilities which makes the approach unsuitable from the generalization point of view. The problem of discontinuities can be tackled with the use of Gaussian Mixture Models (GMMs) [19] instead of histograms. A GMM is a linear superposition of a constant number k of Gaussian distributions which provides a good estimate for multi-modal data even for a small number of samples (i.e., small training data-set). A drawback of GMMs is that the number of distributions k needs to be predefined before learning the distribution. Regarding the TDFs which have to be modeled in this work, it might be challenging to estimate k, because the number of different regimes is unknown. The distribution chosen to represent durations in this work is therefore the Kernel Density Estimator (KDE) [107]. Given a training set $\Gamma^{a,b}$, the KDE places a kernel $k(u)$ centered around each element $\tau_i^{a,b} \in \Gamma^{a,b}$ of the training set. In order to estimate the probability of a data-point $\tau^{a,b}$ the following formula is used:

$$f(\tau^{a,b}) = \frac{1}{nh} \sum_{i=1}^{n} K(\frac{\tau^{a,b} - \tau_i^{a,b}}{h}) \tag{4.11}$$

where n is the number of elements in the training set and $h > 0$ is a smoothing parameter called the bandwidth. The bandwidth h controls how well the KDE adapts to the data. Very small values of h lead to a noisy estimate with many artifacts while too big values would wash out any multi-modal structure in the data. As $k(u)$ any function can be chosen which is subject to the conditions $k(u) \geqslant 0$ and $\int k(u)du = 1$. Here I use the Gaussian kernel which has convenient mathematical properties and thus enables efficient bandwidth selection [17]. The resulting equation looks as follows:

$$f(\tau^{a,b}) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{\sqrt{(2\pi h^2)}} e^{\frac{(\tau^{a,b} - \tau_i^{a,b})^2}{2h^2}}. \tag{4.12}$$

Based on the model M defined in this section the system state can now be assessed for an input time-series. The details on the assessment represent the topic of the next section.

4.4. System State Estimation

This step of the algorithm is concerned with the actual decision whether a system shows normal or abnormal behavior. Similar to the model training step, the state estimation algorithm gets a time-series R of temporal duration features as input. Together with the model M developed in the previous section, the system state is classified into normal or abnormal. We introduce this procedure in two steps. Firstly, the computation of a *score* value is explained which expresses the fitness of the regarded system as a single real-valued variable. Secondly, the mapping of the score value to one of the two possible states normal or abnormal is explained. This mapping is realized with the help of a binary classifier whereas a particular difficulty stems from the facts that there exist no counter examples in order to train the classifier. Here, ideas from the field of *one class classifiers* [76] play an important role.

Score Estimation: Let us assume that we have already trained a model M of normal behavior for a robotic system. Furthermore, let us assume that we have a test time-series D of data-points d which were recorded while the system was in field. We now want to assess how well the recorded time-series D fits the model M, that is, how well did the system perform during the recording of D in relation to the model M. We first apply the findings from sections 4.1 and 4.2, and transform D to a time-series R of temporal duration features tdf. Then, the performance for each tdf is expressed as a single fitness value s called *score*. The equation used to compute s for a tdf^b based on the model M looks as follows:

$$\text{score}(tdf^b) := s^b = \text{log}(P(tdf^b)). \tag{4.13}$$

Using equation 4.10 we write

$$\begin{aligned}\text{score}(tdf^b) &= \text{log}(P(\tau^{a_1,b}) \cdot P(\tau^{a_2,b}) \cdot \dots \cdot P(\tau^{a_n,b})) &(4.14)\\ &= \sum_{i=1}^{n} log(P(\tau^{a_i,b})). &(4.15)\end{aligned}$$

Additionally, we apply a weighting scheme to each contributing duration $\tau^{a_n,b}$ leading to the final equation:

$$\text{score}(tdf^b) = \sum_{i=1}^{n} log(P(\tau^{a_i,b})) \cdot w^{a_i,b} \tag{4.16}$$

$$\tag{4.17}$$

whereas

$$w^{a_i,b} = 1 - \frac{h^{a_i,b}}{H^b} \tag{4.18}$$

and

$$H^b = \sum_{j=1}^{n} h^{a_j,b}. \tag{4.19}$$

The value $h^{a_i,b}$ is the entropy of the distribution $P(\tau^{a_i,b})$ as defined in the information theoretical context [66]. It is a measure of the uncertainty in a random variable whereas higher values indicate greater uncertainty. Equation 4.16 states that the score of a tdf^b is the sum of weighted log-likelihoods of the corresponding durations $\tau^{a_i,b}$. The rationale behind this equation is that each $\tau^{a_i,b}$ shall contribute to the score s^b but to different proportions. The contribution is driven by the strength of the correlation between each two composite features a_i, b whereas highly correlated features should contribute more than weakly correlated ones. It is implicitly represented in equation 4.16 through the probabilities $P(\tau^{a_i,b})$ and explicitly enhanced by the entropy based weight $w^{a_i,b}$. In order to illustrate this, consider two composite features a_j and b which have a low correlation. Low correlation means that the corresponding data-point of a_j and b do not form a strong pattern in the communication data but are rather sent at arbitrary points in time in relation to each other. This again results in a wide range of durations $\tau^{a_j,b}$ which is then used to train $P(\tau^{a_j,b})$. This distribution features high variance and at average will provide lower probabilities for queried values. This implicitly lowers the impact of a_j, b on the score s^b of tdf^b. Furthermore, $P(\tau^{a_j,b})$ has also a high $h^{a_j,b}$ entropy value. $h^{a_j,b}$ is explicitly used in the weight whereas high values have a negative impact on the contribution of a_j, b to the overall score.

Classification: Based on the score value s for tdf it can be now decided whether the current system state is normal or faulty in regard to the learned model M. This is done by a binary classifier:

$$\text{faulty}(tdf) = \begin{cases} \text{True} & : s < s^* \\ \text{False} & : \text{else} \end{cases} \tag{4.20}$$

If s is smaller than the threshold s^*, then tdf is declared as being in a faulty system state. Otherwise tdf is assumed to represent a normal system state. The threshold s^* is calculated based on the following equation:

$$s^* = w \cdot s^*_{val} + (1 - w) \cdot s^*_{val} \cdot \frac{S_{var}}{s^*_{var}} \tag{4.21}$$

where $0 \leq w \leq 1$. Equation 4.21 expresses the assumption that the classifier can be improved by incorporating the variance S_{var} of a history of length k of scores $S = (s_{j-k}, \cdots, s_{j-2}, s_{j-1})$ into the computation of the threshold. The rationale is that we expect S_{var} to be lower for data-points recorded during normal system behavior represented by the *normal* model M than for data-points which do not belong to M and thus represent a faulty system state. If the variance of the history values is low s^* is lowered and thus can be easier exceeded. Otherwise s^* increases which makes it harder to surpass it by a score value. To realize this, additional parameters s^*_{var}, $k = |S_{var}|$ and $w \in [0, 1]$ are added to the classifier. The weight w is used to balance the impact of the constant fraction of the threshold and the variance based one. In total, the parameters of the threshold s^* are represented as $\alpha = (s^*_{val}, s^*_{var}, w, k)$.

Recalling that the approach in this thesis exploits data of the system's normal behavior only, means that the estimation of the classifier's parameter needs to be done without counter examples (i.e., without data which represents faulty situations). The absence of counter examples renders the definition of the class border (here the definition of the threshold s^*) challenging because usually it is not feasible to gather enough training data-set in order to cover the whole input space and find the true border between normal and faulty behavior. Instead of the true border, I use scores $s_i \in S$ computed for a training data-set and choose the parameters α of s^* such that it is as close as possible to the scores of the training data-set by minimizing the squared sum of errors between the scores s_i and the corresponding thresholds s_i^* [14, p.3]:

$$E(\alpha) = \frac{1}{2} \sum_{i=1}^{|S|} ||s_i^*(\alpha) - s_i||^2 \tag{4.22}$$

Determining s^* in this way may result in a threshold which is frequently higher than scores s_i because for the minimal value of $E(\alpha)$ it only matters that $s_i^*(\alpha)$ and s_i as as similar as possible but it is irrelevant which one is bigger. The consequence of s^* being higher than the scores s_i is a high rate of false alarms for the training set and most likely also for data gathered while the system is in field. To resolve this issue, the set of valid thresholds is restricted to the ones which do not exceed a certain false alarm rate level $sfar_l$ following the idea of Japkowicz in [71]. In general, false alarms are not desirable because they may lead to frequent recovery action and one could be inclined to set $sfar_l = 0$. However, setting $sfar_l > 0$ enables to account for additional variance in the data which has not been covered by the model M(e.g., due to the incompleteness of the training data-set). In order to estimate the parameters of s^* which minimize equation 4.22 a grid search [79] (constrained by the false alarm rate) is performed in the parameter space spawned by α.

4.5. Summary

In this chapter the fault detection approach developed in this work has been introduced. The presented solution is designed to cope with the challenges defined in section 2.2.1. In particular, the approach is data-driven which enables fast and easy adaptation to changes in the system and reduces the amount of expert knowledge in order to apply it. The training input used to learn the model is the recorded inter-component communication (ICC). Exploiting ICC for fault detection reduces the coupling to specific system components and the amount of component specific knowledge necessary in order to integrate the detection approach into a system. In addition, it enables fault detection without modifying its components in case the system's build-in communication functionalities can be used to gather fault-sensitive data. The approach exploits generic attributes from the data-points in the ICC for feature generation only. By this means, the need for expert knowledge is additionally reduced and makes the approach robust against changes to the system. We consider the ICC to be the output of a time and state discrete stochastic process and model relations between current and past data-points (represented by generic attributes) as temporal dynamic features (TDFs). TDFs differ from other techniques for state and time discrete time-series by providing efficient means to represent relations between non-consecutive elements. The actual model is based on statistical modeling techniques and represents only the normal state of the system without any representation of possible faults. By this means, the approach does not depend upon the presence of exemplary data of fault situations in order to detect them. The normal state of a system is estimated by the user based on the system's performance for a given task. This definition is based on task related knowledge without further information on the normal behavior of single components which enables training data acquisition even to non-system-experts.

Having introduced the AuCom fault detection approach, the upcoming chapter is concerned with important implementation details of the presented approach, the integration efforts into robotic systems and its empirical evaluation.

5. Implementation & Integration

This chapter is concerned with implementation and the integration efforts conducted within these thesis. It starts in section 5.1 with the description of the modularized implementation of the algorithm introduced in chapter 4. In section 5.2 typical workflows in which the AuCom-detector was used during the research period are presented. At last, in section 5.3 integration aspects of the detector into a robotic system are discussed. Section 5.3 starts with a discussion of the requirements on a target system in order to apply the detection approach. Subsequently, the concrete integration on BIRON [59] is presented followed by hypothetical efforts for currently unsupported but prominent robotic frameworks.

5.1. Implementation of the Algorithm

The description of the implementation efforts starts with an introduction of the data representation used throughout the code as a basis for further discussions. Subsequently, the actual implementation of the algorithm is discussed.

The implemented data types are depicted in a class diagram in figure 5.1. Among others, they represent the outputs of the different intermediate processing steps of the algorithm (see chapter 4). In particular, this means that composite features, temporal dynamic features, temporal dynamic probabilities, scores and classifications are represented with an own data type. All these types inherit from a common abstract class called *AbstractData*. AbstractData enables uniform handling of intermediate results on a higher abstraction level and provides facilities to add and delete attributes to a data object. For example, this is useful in order to mark an element of the recorded

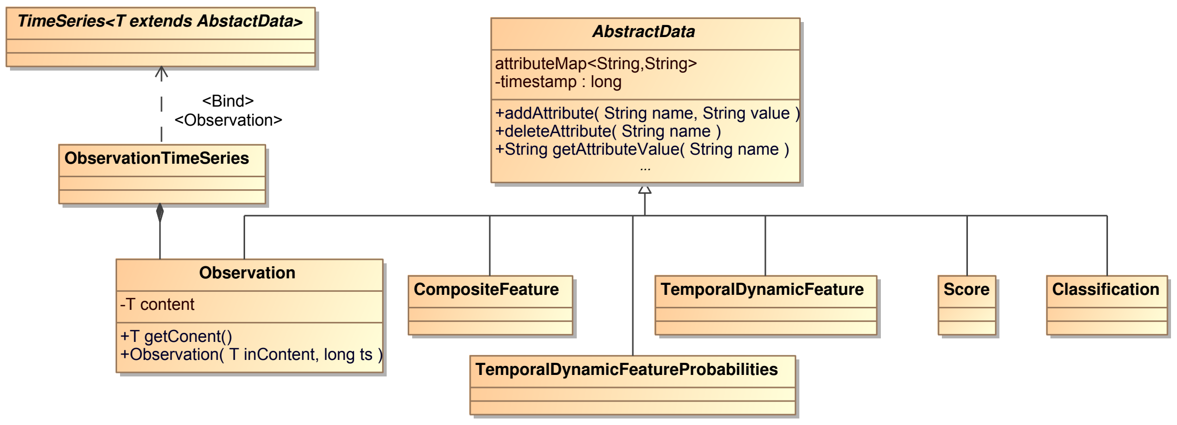

TimeSeries<T extends AbstactData>
<Bind>
<Observation>
ObservationTimeSeries
AbstractData
attributeMap<String,String>
-timestamp : long
+addAttribute(String name, String value)
+deleteAttribute(String name)
+String getAttributeValue(String name)
...
Observation
-T content
+T getConent()
+Observation(T inContent, long ts)
CompositeFeature
TemporalDynamicFeature
Score
Classification
TemporalDynamicFeatureProbabilities

Figure 5.1.: Class structure of the data types used in the fts-graphs. Subclasses of AbstractData represent processing results of the AuCom-detector except for the Observation class. It is an optional container format for a more sophisticated communication representation prior to algorithmic processing providing useful input for off-line analysis. The TimeSeries class is a composition of subclasses of AbstractData ordered in time.

inter-component communication as the first recorded element after a fault has been induced[1]. Besides the representation of intermediate results, an optional container data format called *Observation<T>* was implemented as a generic Java type [92]. This container provides a possibility to represent the input of the AuCom-detector in a customized format prior to the algorithmic processing i.e., before it is reduced to composite features as the first intermediate processing output (see chapter 4). As the detector is directly connected to the robotic system, this can be exploited in order to save a more sophisticated view on the inter-component communication for later evaluation or repetition of the experiments. Regarding the subsystem example depicted in figure 4.1 in section 4.1 an observation could consist of the complete laser

[1]This information is of particular interest when evaluating a detection approach under controlled conditions.

or velocity readings of the laser or odometry component. Furthermore, the set of data types used in this work also encompasses a representation of a time-series called *TimeSeries<T extends AbstractData>*. This class maintains temporal aspects of a time-series and manages the membership of single data objects to a concrete data-set (e.g., a training data-set or a test data-set).

The discussion now moves forward to the description of the actual AuCom-detector implementation which is aimed at two major goals: On the one hand, it shall provide a separation of the algorithmic part of the implementation from the part which encapsulates the target-system specific adaptations necessary in order to apply the approach to a concrete system. On the other hand, it shall enable flexible modifications of the different algorithmic steps. For example, this is beneficial in order to experiment with different variants of the algorithm like different classifiers.

To realize this, the AuCom-detector is implemented in Java [99] and realized within the *Filtering, Transformation and Selection* (FTS) framework proposed by Luetkebohle et.al. [84]. The aim of FTS is to foster re-use at the component level by providing a general pattern that supports the structuring of components into sub-parts of lower granularity. Components based on FTS are represented as transformation graphs consisting of inner-nodes as well as sinks and sources which act as inputs and outputs of a graph. Edges in the graph represent possible data-flow between fts-nodes. A general policy in a FTS-graph is that data is processed from sources to sinks.

Following the FTS idea the functionality of the AuCom-detector was decomposed and encapsulated in fts-nodes. The granularity of the nodes reflects the various processing steps discussed in the algorithmic part of this work in chapter 4. In addition, the chosen decomposition is fine grained enough in order to facilitate reuse among graphs and it supports the decoupling of system-specific and system-independent parts of the algorithm. In particular, the following nodes have been implemented:

▷ *SourceAdapter*: Implements the connection to the monitored system consisting of functions to connect, disconnect and reconnect to the system and routines necessary to discover and gather the system communication. This node may have a significant impact on the monitored system. Consequently, computational efforts should be minimal which is why its output is expected to contain the data-points d still encoded in the systems native communication format.

- *ObservationCreator*: This node provides the possibility to pre-process a data-point d before actually applying the fault detection processing steps on it. For example, it enables to save the ICC of a system in a more complete manner i.e., other than solely the required input for the algorithm. The output of the node is of type Observation<T> whereas T may be any type of data. T could also be the native communication format of a system. In this case the node merely passes through all incoming input.

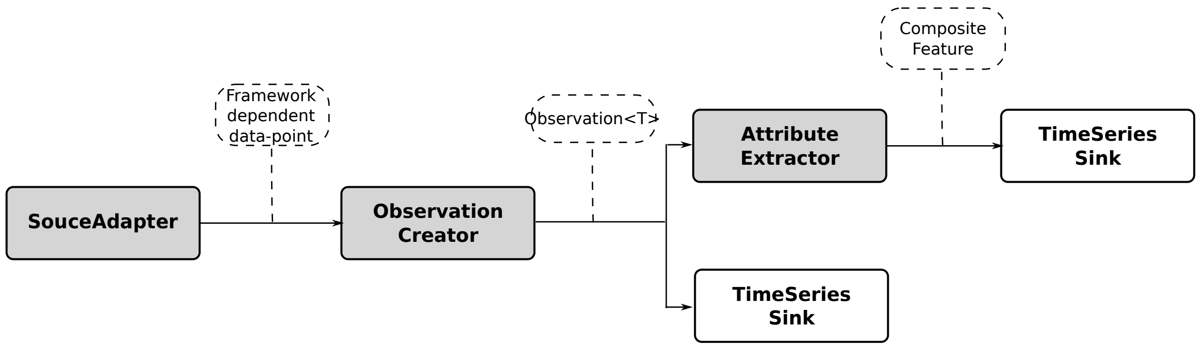


Figure 5.2.: The system connection fts-graph. It manages the access to the robotic system and provides input to the AuCom-detector as observations or as composite features.

- *AttributeExtractor*: Extracts the generic attributes Source, Scope, and Type from the content T of an observation object and creates a composite feature e. In other words, it implements the mapping function defined by equation 4.2.

- *TimeSeriesSink*: A sink which adds incoming data elements to a time-series. Input data is required to be a subtype of the AbstractData class.

- *TemporalDynamicFeatureCreator*: Maintains a history of the last occurrences of each composite feature $e \in \mathcal{E}$ present in the system. The node uses this history to compute a Temporal Dynamic Feature tdf^e for an input composite feature e. To do so, it implements the algorithm defined in listing 4.1.

- ▷ *Trainer*: Implements the training of the model M of the system's normal behavior. Each incoming tdf^e is exploited to train corresponding probability distributions for each $\tau^{a,e} \in tdf^e$ of M whereas a is a predecessor composite feature of e as defined in section 4.2.
- ▷ *ModelQueryNode*: Maintains an instance of the model M. For each incoming tdf^e computes single probabilities $P(\tau^{a,e}) = p^{a,e}$ for the durations $\tau^{a,e} \in tdf^e$. The output is the set $Pr^e = \{p^{a,e} | \; \tau^{a,e} \in tdf^e\}$ of probabilities.
- ▷ *ScoreCalculator*: Calculates a score s by applying equation 4.16 to the set of probabilities Pr^e.
- ▷ *Classifier*: Classifies the system's behavior into normal or faulty state based on an input score value s. This node implements the equation 4.20.
- ▷ *TimeSeriesSource*: A source that reads single data elements from a given time-series and pushes them into the connected graph. Input data is required to be a subtype of the *AbstractData* class.

The presented nodes have been assembled to graphs in order to implement the functionality of the AuCom-detector. To illustrate this, three exemplary graphs are discussed now which realize the detector's capability to i) connect to a robotic system, ii) detect faults and iii) record data to a file for later analysis.

The system connection graph is shown in figure 5.2. The purpose of this graph is to manage the connection to the robot through the SourceAdapter node and provide input for the AuCom-detector. Input is provided in two ways: i) as a time-series of composite features which is the first processing step of the algorithm (i.e., the generic representation of the communication between the system's components) and ii) as a time-series of Observations. The time-series of observations can be used for example to save the ICC in a more complete format (see the introduction of the Observation data type previously discussed in this section). The composite feature time-series is used as input for further algorithmic processing of the AuCom-detector. It is a system independent format which means that graphs which use it as input are also system independent. Integration efforts of the AuCom-detector into a new system concentrate on the grey shaded nodes of the system connection graph[2].

The next graph to be discussed is the fault detection graph depicted in figure 5.3. It realizes the processing chain of the detection algorithm proposed in section 4. The graph requires a composite feature time-series as input and can be directly linked to the system connection graph. In this configuration (Figure 5.3) the AuCom-detector is ready to be applied in an on-line fault detection scenario. Alternatively, the time-series can be read from the file system in case the AuCom-detector is applied off-line. Single elements of the time-series are pushed into the graph by the TimeSeriesSource node. After the input is processed by the TemporalDynamicFeatureCreator, ModelQueryNode, ScoreCalculator and the Classifier the result is saved to a classification time-series in the TimeSeriesSink node. The time-series can be accessed by other

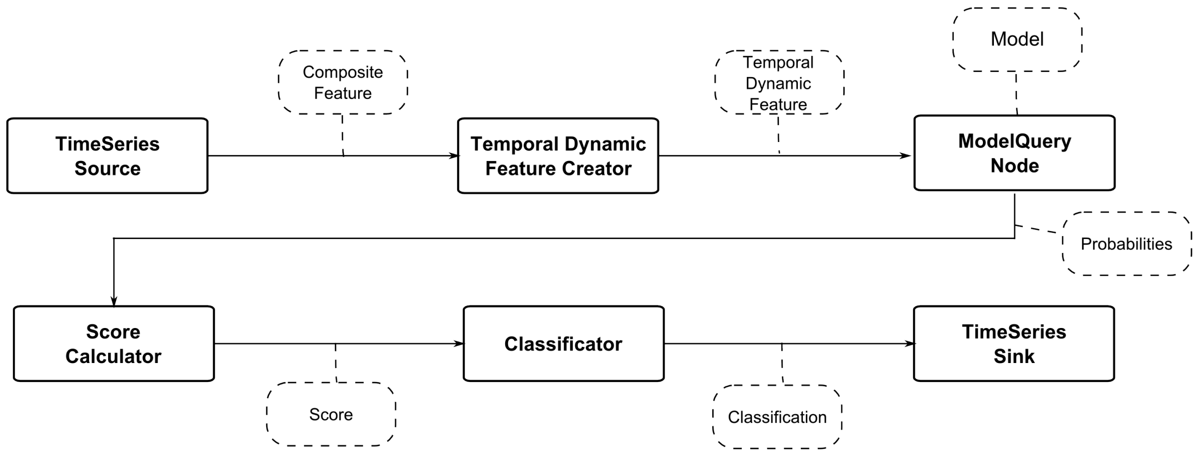


Figure 5.3.: The AuCom-detector graph realizes the processing chain of the detection algorithm proposed in chapter 4. Its output is a classification time-series.

[2]For detection purposes only, robot specific implementations of these three nodes are actually sufficient. However, if Observation data has to be saved to and loaded from a file additional converter classes need to be implemented (See the data recording graph explanation further below).

sub-components of the AuCom-detector, too. An exemplary component which uses the above mentioned time-series is the visualizer. This sub-component plots the classifications over time using the *JFreeChart* plotting library [50]. In order to transform the classification results into a representation suitable for the plotting library the visualizer again exploits a fts-graph.

The third graph is depicted in figure 5.4 and is called the data recording graph. It gets its input in terms of Observations from a time-series source node and its purpose is to save data for a later in-depth analysis and result reproduction. The second node in the graph provides the possibility to mark a passing Observation as being the last one in the current record by providing an appropriate public function *setMarkNextAsLast()*. This mark is used to decide when to close the output file in the *OutputStreamSink* node. Since this function is public, it can be called externally for example in a function related to a button press in a graphical user interface (GUI) or by a timer in order to record a specific duration of the communication. The sink node takes care of the file writing procedure. The payload (i.e., the object with the generic type T) of each incoming Observation is streamed into a file. The formatting of the output is delegated to a specific converter. By this means, data can be saved in different formats like the Extensible Markup Language (XML) [130] or the comma separated values (CSV) format. Specific converters need to be provided by the developer for each generic type T which is why the OutputStreamSink node is shaded grey indicating target system dependency.

The decomposition of the functionality of the AuCom-detector into fts-nodes yields high flexibility during the application and experiments phase whereas the predefined graphs provide larger building blocks which can be combined in order to easily adapt the detector to different application cases. One example is the case of on-line fault detection. Here, the system-connection graph is linked to the AuCom-detector graph providing its input. This setup is sufficient for the task of on-line fault detection. However, in order to reproduce the results at a later point in time, it is necessary to save the ICC to the file system. A simple solution is to connect the recorder graph to the

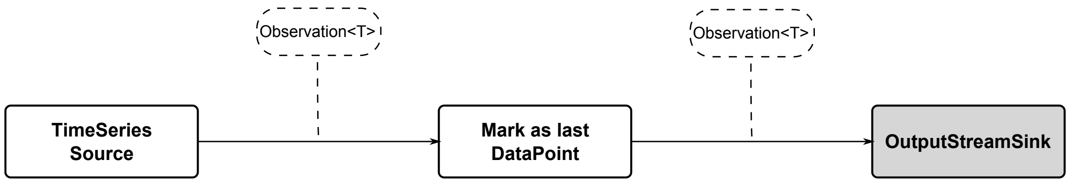


Figure 5.4.: The data recording graph used to save recorded inter-component communication to a file for later analysis.

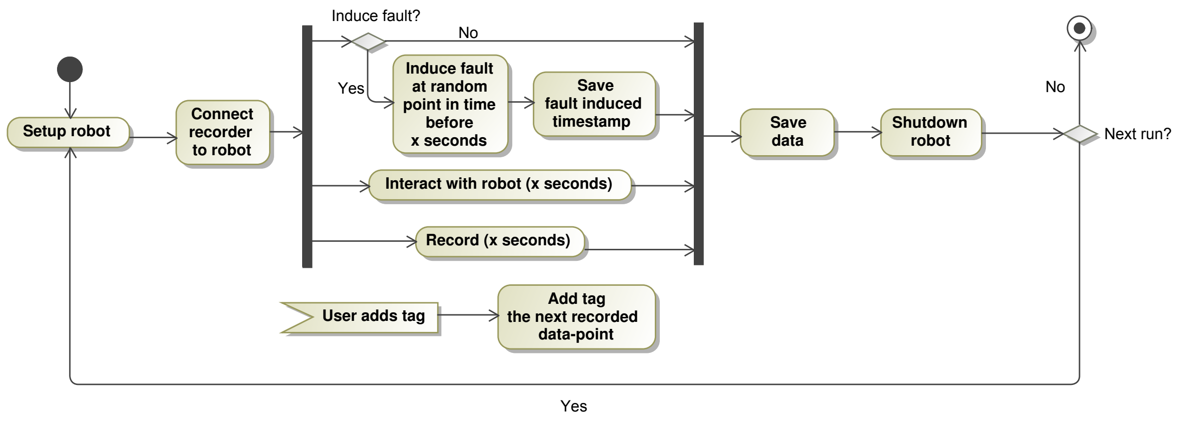
system-connection graph and record the data in parallel. Graphs can be also extended with additional nodes. An example of an actually realized extension is the addition of a SlidingWindow node between the ScoreCalculator node and Classifier node of the AuCom-detector graph. This node computes a mean over score values in a defined time span and replaces the input for the classifier node with that computed value. By this means, unwanted noise in the score can be diminished which eventually leads to a reduction of false alarms. Particular nodes can be also exchanged in order to evaluate different implementations of the same functionality or they can be run in parallel and compared on the fly.

5.2. Common Workflows with the AuCom-detector

Throughout the research project the AuCom-detector has been used in different application scenarios. The intention of this section is to present the most prominent workflows executed during this time and document the flexibility of the chosen implementation. The presented workflows are: i) data acquisition, i) off-line experiments, iii) and on-line fault detection.

5.2.1. Data Acquisition

The workflow used to record data without fault detection is depicted in figure 5.5 as an activity diagram. It begins with the setup of the robot and the connection of the



Induce fault?
No
Yes
Induce fault at random point in time before x seconds
Save fault induced timestamp
Setup robot
Connect recorder to robot
Interact with robot (x seconds)
Record (x seconds)
Save data
Shutdown robot
No
Next run?
User adds tag
Add tag the next recorded data-point
Yes

Figure 5.5.: Activity diagram of the data recording workflow.

recorder component to its communication framework. Next, several actions take place in parallel. The recording component gathers the inter-component communication in the system while the system is engaged in interaction with a user. If the intention of the data acquisition run is to record a test data-set, another task which is executed in parallel is the induction of a fault. Inducing a fault is executed by a dedicated component and takes place at a random point in time following a probability distribution P. After a time period of X seconds the recorded data is saved and the robot is turned off. The complete run can be repeated in order to acquire several sets of training and test data for a specific fault in a controlled way.

In addition, during the interaction the operator of the recording component is provided with a means to add short comments (tags) to the recorded communication data indicated by the *User adds tag* event and the *Add tag to next data-point* action in figure 5.5. Tagging provides an immediate means for the developer to annotate interesting observations relative to the point in time when they were perceived. An example is the annotation of experienced *real* faults which were not induced on purpose i.e., while the system was expected to behave normally.

5.2.2. Off-line Experiments

Off-line experiments conducted in this work comprise the comparison of alternative implementations and parameterizations of the different algorithmic steps, the replication of results acquired on-line and the comparison of the AuCom-detector to the performance of another approach. These experiments are characterized by numerous iterations on previously recorded data with slightly varying conditions (e.g., parameters). Furthermore, they do not require a real robot or a human interaction partner which makes them suitable for automatization. Off-line experiments have been conducted within an experiment execution framework which is briefly introduced now.

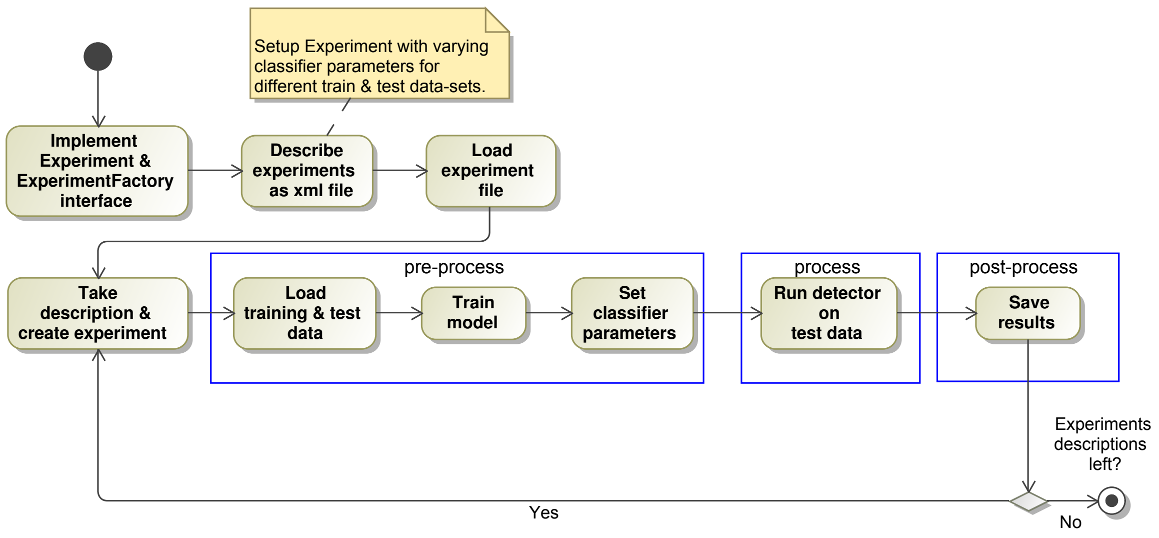

Setup Experiment with varying classifier parameters for different train & test data-sets.
Implement Experiment & ExperimentFactory interface
Describe experiments as xml file
Load experiment file
Take description & create experiment
pre-process
Load training & test data
Train model
Set classifier parameters
process
Run detector on test data
post-process
Save results
Experiments descriptions left?
Yes
No

Figure 5.6.: Activity diagram of the off-line analysis case.

The Experiment Execution Framework The intention of this framework is to provide a simple means to facilitate recurrent execution of experiments. The structure of this framework is shown in figure 5.7. The framework provides an interface called

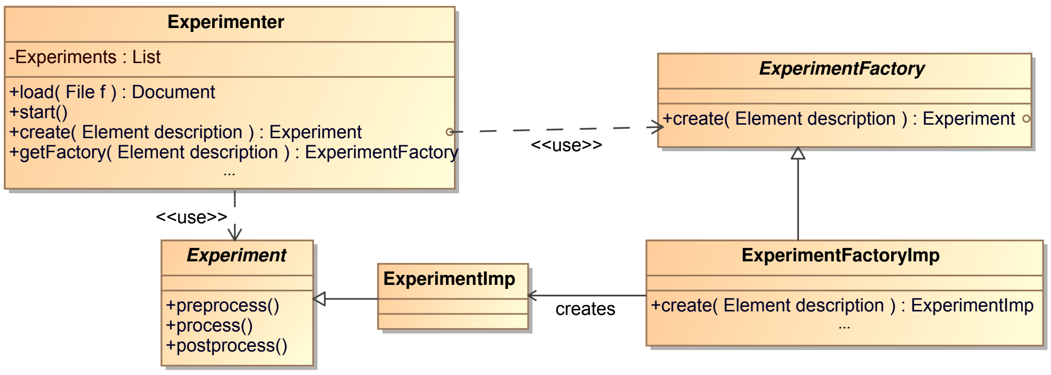

Experimenter
-Experiments : List
+load(File f) : Document
+start()
+create(Element description) : Experiment
+getFactory(Element description) : ExperimentFactory
...
<<use>>
ExperimentFactory
+create(Element description) : Experiment
<<use>>
Experiment
+preprocess()
+process()
+postprocess()
ExperimentImp
creates
ExperimentFactoryImp
+create(Element description) : ExperimentImp
...

Figure 5.7.: Class diagram of the experiment framework.

Experiment which structures the execution of an experiment into three phases: *pre-process*, *process* and *post-process*. While there is no strict rule on how to implement these phases, a general policy is that the pre-process step should comprise all actions necessary in order to prepare the experiment like loading all necessary training and test data-sets or set the parameters of the AuCom-detector. The processing phase represents the core of the experiment i.e., all code which produces the outcome of the experiment. The post-processing step is the right place to put code which e.g., saves results and cleans up the environment. The execution of experiments is managed by the *Experimenter* class which reads the description of experiments from an XML configuration file, instantiates, and runs them in consecutive order. Instantiation of an experiment is delegated to a factory class which has to be provided by the experiment designer. The *ExperimentFactory* interface defines a single function called *Experiment create(Element description)* which gets an XML element as input and outputs an Experiment object.

An exemplary workflow is depicted in figure 5.6. It shows the different steps in a scenario where different parameter settings of the classifier described in section 4.4

are evaluated on several training and test data-sets. The first step is the implementation of the Experiment and ExperimentFactory interfaces. The implementation of the Experiment interface comprises the definition of the aforementioned three phases of pre-processing, processing and post-processing. Here, these phases correspond to the steps five to nine of the workflow as highlighted in figure 5.6. The implementation of the ExperimentFactory is used in the fourth activity of the workflow in order to correctly instantiate and parameterize an experiment from the XML-description. Afterwards, the different variants of the experiments are described in an XML file. Subsequently, experiments are loaded from the file by the Experimenter, instantiated and executed one after the other. This includes model training, classifier parameterization, fault detection on test data, and saving the results.

5.2.3. On-line fault Detection

The primary application scenario for a fault detector is the on-line application which is why the present approach has also been evaluated in such a scenario. In the on-line fault detection case the detector runs in parallel to the monitored system and is expected to detect faults during interaction. The workflow for this type of experiments

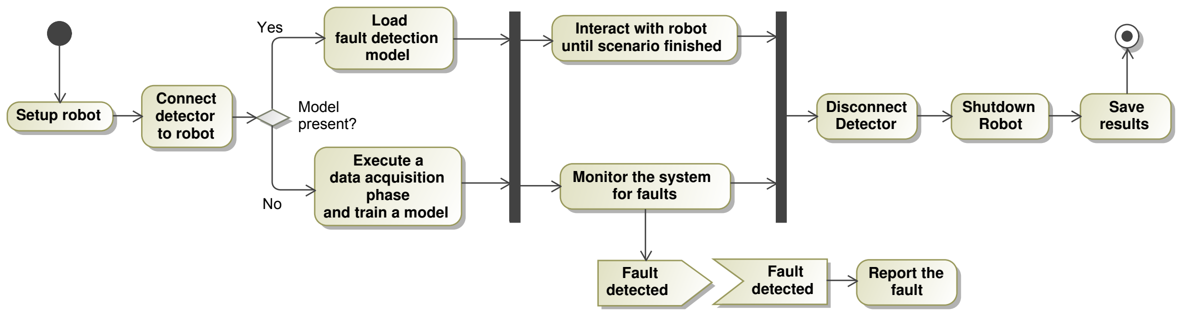

Setup robot
Connect detector to robot
Model present?
Yes
No
Load fault detection model
Execute a data acquisition phase and train a model
Interact with robot until scenario finished
Monitor the system for faults
Fault detected
Fault detected
Report the fault
Disconnect Detector
Shutdown Robot
Save results

Figure 5.8.: Activity diagram for the on-line fault detection case. The system is in interaction with a human and the detector monitors the system in parallel. In case of a fault detection the user is informed.

is depicted in figure 5.8. In the initial step the robot needs to be setup for the specific interaction scenario. Next, the detector needs to be configured for the targeted system. This can be done in two different ways. An appropriate model can be loaded from the file system if present[3]. If not, data has to be gathered and a model needs to be trained. Subsequently, the detector runs in parallel to the interaction of the robot within the given scenario. If a fault is encountered, the detector gathers fault relevant information like the timestamp of occurrence and a history of the inter-component communication shortly before the fault has been detected and reports it to the user. After finishing the scenario the detector is disconnected, the robot is shutdown and the results are saved.

[3]A model can be present if it was learned during a previous interaction and the system setup didn't change.

5.3. Integration

This part of the thesis is concerned with integration aspects of the AuCom-detector. Section 5.3.1 introduces the requirements imposed on a robotic system in order to apply the detector to a new system. Subsequently, in section 5.3.2 a concrete example on how the approach was incorporated into our robotic system BIRON [59] is given. At last, in section 5.3.3 integration case studies are discussed for prominent robotic frameworks.

5.3.1. Requirements

Recalling chapter 4 we know that a fundamental design decision of the fault detector in this work is that it is solely grounded on the inter-component communication data of the targeted system as input. As a consequence, integration of the approach into a new system is concerned with the communication framework and the architectural layout of the system only thus omitting any constrains on individual components. Still, there exist several requirements which have to be fulfilled by a system in order to enable the application of the AuCom-detector. These requirements result from the challenges stated in section 2.2.1 and from the design decisions taken in section 4.1 and are discussed next.

Granularity The partitioning of a system has a significant impact on the set of faults detectable by the approach. It predetermines the granularity of the communication between the components of a system which constitutes the input for the fault detection approach. If the data gathered this way turns out to contain insufficient fault-sensitive information, a possible solution is to incorporate additional monitoring probes into the system's components. This enables the recording of the communication between sub-components, single routines or functions.

Scalability A system is said to be scalable if it can handle the addition of users and resources without suffering from a noticeable loss of performance or increase in administrative complexity [115]. Regarding the communication framework this means that additional components incorporated into the system should have no significant impact on the communication latency, reactivity and thus performance of the overall system. This requirement is often considered in communication frameworks used in robotic systems [137]. Consequently, it seems that simply incorporating the detector as an additional component into the robotic system should have minimal impact and thus should be unproblematic. However, the detector challenges the scalability of a communication framework in a different way. It records the *complete* ICC in the system, which is a case that often is not explicitly considered by the developers of a particular framework and thus may actually have a great impact on the system's overall performance. If it turns out to be the case, an alternative way of data recording is necessary. A possible solution is to intercept the communication on a lower layer of the communication stack within the framework in order to gather the data more efficiently. For example, in the CoSy Architecture Schema Toolkit (CAST) [63] components subscribe to specific information based on a filtering system in order to implement the communication efficiently. For the AuCom-detector no filters would be used thus not taking advantage of this optimization possibility. If this way turns out to be not efficient enough, CAST also allows to establish a TCP/IP connection to the core module in order to grab the complete unfiltered communication more efficiently.

Architecture Level Introspection Robust and autonomous application of the AuCom-detector requires means to automatically discover & connect to communication channels in the system. This requirement also helps to tackle the challenge for minimal system expert knowledge because the connection has to be implemented only once for a communication framework. Afterwards, it can be reused for different system configurations without modification. Holistic communication recording can be realized by discovering all active communication channels and exploiting ordinary communication mechanisms of the framework. Communication channel discovery is often a built-in functionality of a framework. It is used by the system's components to dynamically connect to other components during run-time. An example is the trading-service in CORBA [114]. Alternatively, if no such service is present, the to be monitored communication channels can be predetermined and are static during run-time which renders the approach less flexible in case new communication channels are opened during runtime. In cases where no framework support is present the same alternative solution can be applied as in the case of scalability i.e., the recording mechanism can be implemented into the communication framework manually either on the system, network, or programming language level.

Communication Level Introspection The next requirement with functional implications for the detector is the need to obtain required attributes of the communicated data. In particular, this means access to the timestamp of occurrence of a data-point as well as to the generic attributes Source, Scope, and Type introduced in section 4.2. The timestamp of occurrence of a data-point is often added by the respective communication framework to the metadata of that data-point. This is for example true for XCF [137], ROS [103], RSB [132], and CAST [63] which altogether cover a huge range of robotic systems. The attributes Source and Scope, if not provided in the same way as the timestamp, require additional adaptations of the framework. However, here only the communication framework is subject to modification while single components of the system remain untouched. The Type attribute has been described in section 4.2 to be a semantic interpretation of the data-point regarding its novelty in the system which means a data-point may represent *new*, *present* or *deprecated* information regarding a certain aspect of the robot's internal or external state (e.g., information about a human interaction partner). This kind of information is valid for many cognitive tasks a robotic system has to perform like visual tracking of objects [7], learning new environments [62] or executing tasks which involve the interaction and coordination of several objects [61]. As such, this attribute is strongly related to the semantics of the component which generates the data-point. If this kind of semantic information is not derivable from the communicated data-points a subsequent implementation has to be done for each component individually by the component developer. In this case it might be more efficient to omit this attribute and represent data-points by Source and Scope solely.

5.3.2. Integration into BIRON

This section describes the integration of the detector into BIRON [59] realized for the empirical evaluation of the approach which is presented discussed in the subsequent chapter 6. The BIRON efforts are aiming at developing a robot companion exhibiting social interaction capabilities [23]. The robot is i.e. able to focus and recognize its current interaction partner from a number of persons, which allows the robot to interact in a more efficient and natural way while carrying out useful tasks. One of the scenarios the research is aiming at is the so-called home-tour scenario which is driven by the vision of future household robots being introduced to the new home for the first time after purchase. The robot is guided through its new interaction environment and is supposed to learn important places and objects. Similar interaction scenarios are defined in the context of the RoboCup@Home competition [80]. Here, robots compete with each other in solving household relevant tasks like identifying and fetching objects or safely navigating through a realistic apartment thereby following a human. As depicted in figure 5.9

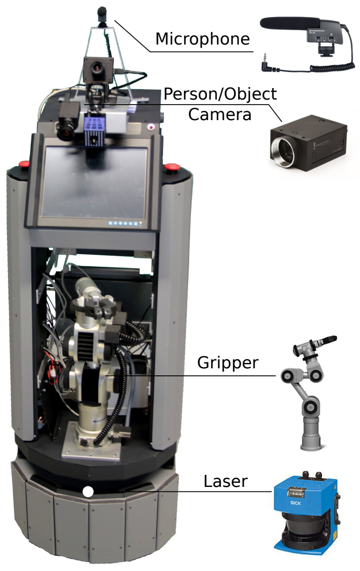


Figure 5.9.: The BIelefeld Robot Companion (BIRON) for which the concrete integration of the AuCom-detector was executed.

BIRON possesses a microphone, a camera, a laser scanner and a 3D-sensor. BIRON is also equipped with a differential drive to move along and a gripper in order to manipulate objects. Its cognitive capabilities are implemented in a set of software components which will be introduced in section 6.3.1.

The communication between BIRON's components is based upon the XML enabled Communication Framework (XCF) [137]. XCF provides several means of interaction, the most important one being the active memory [9] and the publish/subscriber communication styles. In regard to the active memory style, components can insert, update, delete and retrieve contents in memories. The content is represented as XML documents with optional binary attachments. Components can also subscribe to events resulting from the modifications of memory contents by using XPath statements [12].

Communication channel discovery in XCF is provided through a unified interface called the *XcfManager*. Monitoring a system is realized by i) registering to all publishers, ii) subscribing to all changes to the complete data in all memories. Listing 5.1 depicts a small but complete example written in Java on how to implement monitoring. This code is extracted from the XCFSourceAdapter class which is the XCF implementation of the SourceAdapter node of the fts-graph introduced in section 5.1. For the sake of simplicity the listing omits the code which handles the removing and adding of publishers after the initial connection phase. At first a XCF specific *SynchronizedQueue* is instantiated which is used to collect XcfEvents from one or more event sources. To do so, the function *convertFromXcfEvent* needs to be defined. It is used to convert each inserted XcfEvent into another data type before saving it in the queue. Keeping processing efforts minimal in the *XCFSourceAdapter* node each XcfEvent is directly inserted into the queue. Next, the XcfManager is used to add listeners to all present publishers in the system. The listeners are represented by QueueAdapter objects which reference the previously defined queue. In a similar way, listeners are added to all memories known to the xcfManager. After this code was executed the *XCFSourceAdapter* is ready to forward BIRON's communication to any connected fts-nodes which can process XcfEvents.

Listing 5.1: Connect to Xcf

```
void connectToXcf(){
  SynchronizedQueue<XcfEvent> queue = new SynchronizedQueue<XcfEvent>() {
    @Override
    public XcfEvent convertFromXcfEvent(XcfEvent e) {
      return e;
    }
  };

  for(String publisher:getXcfManager().getRegistry().publisherList()){
    Subscriber sub = getXcfManager().createSubscriber(publisher);
      sub.addListener(new QueueAdapter(queue));
  }

  for(String memory:getXcfManager().getRegistry().serverList()){
    if(isActiveMemory(memory){
      ActiveMemory m = getXcfManager().createActiveMemory(memory);
        m.addListener(
          new QueueAdapter(queue,
            new Condition(MemoryAction.ALL, new XPath("/"))));
    }
  }
}
```

One such node is the AttributeExtractor presented in the implementation section. It converts XcfEvent to an internal representation called Composite Feature which consists of the generic attributes Source, Scope, Type and the timestamp of occurrence of the XcfEvent. Listing 5.2 depicts the function *getGenericFeatures* used to extract the generic attributes. Each XcfEvent is casted to a specific sub-class representing either a publisher or a memory event thus enabling the use of specific functions of each sub-class. In case of the publisher event the Source attribute is the name of the publisher. It is extracted from the XML content of the event. This information was inserted into the XML document by the communication framework and is thus consistent for all publisher events. The Scope and Type attributes are set to default values representing a publisher.

Listing 5.2: Extract generic features

```
Map<String,String> getGenericFeatures(XcfEvent event){
  Map<String,String> features = new HashMap<String,String>();

   if(isPublisherEvent(event)){
     PublishEvent p_event = (PublishEvent)event;
     features.put("Source, getGenerator(p_event));
     features.put("Scope","publisher");
     features.put("Type","INSERT");
  }else{
     MemoryEvent m_event = (MemoryEvent)event;
     features.put("Source",m_event.getName());
     features.put("Scope",m_event.getMemory());
     features.put("Type",m_event.getType().toString());
  }
     return features;
}
```

In case of a memory event all three attributes are extracted using MemoryEvent member-functions thereby representing the component which led to the emission of the event (Source), the memory the event was generated in (Scope) and the change made in the memory (Type). The timestamp of occurrence of a particular event is obtained with the extractTimestamp function shown in listing 5.3. This function takes advantage of the convention that each event generated in the XCF framework has coherent time information about the instant of time where it was created and about the time when it was modified lastly. These two timestamps are equal for publisher events. In case of events resulting from changes in a memory these two timestamps may differ for example when the event represents an update of data already present in that particular memory.

Listing 5.3: Extract timestamp of event occurrence

```
Long extractTimestamp(XcfEvent event){
  String xpath = ".//ts[@key='xcf:pub']";
  XPathContext context = new XPathContext("xcf", "http://xcf.sf.net");
  Document data = event.getData().getDocument();
  Element timing = (Element)data.query(xpath, context).get(0);
  Long timestamp = Long.valueof(timing).getAttributeValue("ms"));
  return timestamp;
}
```

Listings 5.3, 5.2 and 5.1 represent all modifications necessary to adapt the AuCom-detector to an XCF-based system like BIRON. Furthermore, these changed only affect the SourceAdapter and the AttributeExtractor fts-nodes defined as system specific in section 5.1 while all other processing nodes of the detector remain unmodified.

5.3.3. Integration into Common Communication Frameworks

In this section the integration of the AuCom-detector into communication frameworks prominent in robotics is discussed. These solutions have not been implemented yet, instead they shall demonstrate how the specific communication capabilities of each framework can be theoretically exploited in order to integrate the detector. As such, they act as sanity checks.

Robot Operating System (ROS) ROS is a structured communications framework designed for the development of large-scale robotic systems [103]. In ROS, modules or components are presented as nodes which exchange messages via so called topics in a publish/subscriber way. A node sends a message by publishing it to a given topic, which is for example a string such as *odometry* or *map*. Nodes that are interested in a certain kind of data can subscribe to the specific topic. In recent years, ROS became popular and is now used in various robotic systems such as mobile manipulators, mobile robots, humanoid robots, unmanned aerial vehicles, or autonomous cars (see [1] for an exhaustive list of systems).

Communication recording in ROS encompasses the subscription to all topics. One way to do this is to exploit the Master_API provided by the roscore: a collection of nodes (including the master node) and programs that are pre-requisites of a ROS-based system [2]. The retrieval of all topics is realized by calling the *getSystem-State* function on the master node which provides an overview of all topics that are available for subscription as well as the publishers and subscribers of these topics. Subsequently, listening to topics is realized by registering subscribers using the *reg-isterSubscriber* function on the master node with the respective topic. The extraction of generic attributes can be done by reusing the information about the publishers to represent the Source attribute and subscribers for the Scope attribute. ROS does not provide explicit support for the semantic classification of the exchanged messages into new, update and deprecated like it is the case for the active memory concept [9] or the memory concept in CAST [63] and thus the Type attribute is subject to the particular component. If it is not present, communication data can be represented by the Source and Scope attributes solely.

CORBA based Communication Frameworks The Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group which provides a framework for building distributed systems, regardless of the hardware or software platform [90]. There exist various communication frameworks for robotic systems which are based on the CORBA standard like Miro [125], Orocos [22], Orca [18] or Smartsoft [110].

The CORBA standard defines the *ORB-Core* to be the central point of transparent communication handling between components in a distributed system. The access to components regardless of their physical location is realized based on *Interoperable Object References* (IORs). Components can acquire the IOR of a component they want to interact with through a *Naming-Service* given they know its name. In order to find components available for communication the *Trading-Service* can be used. This service and the possibility to intercept client calls and server answers via *Portable Interceptors* [56] can be exploited for ICC monitoring in CORBA based communication frameworks. The interceptor enables access to the complete parameters of the call and thus provides the information about the Source of the call (i.e., the client) as well as the target which can be interpreted as the Scope attribute of the communicated information. Thus, it can be assumed that CORBA based systems should provide the capabilities to gather ICC data in order to apply the AuCom-detector.

CoSy Architecture Schema Toolkit (CAST) CAST [63] is robotic framework which was developed at the University of Birmingham. In CAST components of a system exchange information by adding, overwriting or deleting objects in so called *working memories*. These operations are broadcasted in the system's architecture as events. Components can subscribe to specific events based on the executed operation, the source of the event, and the object and data-type the operation was performed on. When receiving an event, a component can use the information it contains to access the referenced object in the memory. A robotic system build upon CAST typically consists of several sub-architectures which represent coherent concepts. An example is the sub-architecture structure of *Dora the Explorer* which is a mobile robot with a sense of curiosity and a drive to explore its world [62]. In the DORA system sub-architectures represent different modalities like vision, language or mapping. The components belonging to a sub-architecture share information through a memory. Sub-architectures are again connected in order to be able to fuse information from different modalities. Integration of the fault detection approach developed here into a CAST based system involves the registration on all events independently of the memory the event is inserted into. This can be done by registering a callback function using the *addChangeFilter* function with a *WorkingMemoryChangeFilter* customized to accept all occurring events in all working memories of the system [63]. An initial integration into CAST was done by Jeremiah Via in its bachelor thesis [126]. He uses an alternative way to puck off the communication between the components by opening a socket connection to the CAST server instance. This solution is fast but may be vulnerable to changes because it operates on a string representation of the messages.

5.4. Summary

In this chapter the modularized realization of the detector's functionality based of the fts-toolkit [84] was presented. The modularization is aligned to the AuCom algorithm introduced in chapter 4. It simplifies the exploration of alternatives, the visualization of intermediate processing results and the integration of the detector into new robotic systems. In addition, common workflows have been introduced in which the detector was used. The second part of this chapter was dedicated towards integration aspects of the detector. This involves the presentation of requirements in order to apply the detector to a robotic system. Major results of this discussion are: i) all integration requirements only affect the communication framework of the targeted robotic system and ii) the approach can often be integrated by exploiting built-in functionalities of a particular communication framework of a system. To support these statements, the concrete integration into the XCF framework used in the BIRON system has been presented in section 5.3.2. In addition, possible means to integrate the detector into common frameworks in robotics were discussed in section 5.3.3 showing the potential of the approach to be applicable to other robotic systems.

6. Evaluation

In this chapter the discussion turns over to the quantitative evaluation of the AuCom-detector which was executed in simulation and on a real robot. Simulation experiments were conducted on different artificial systems implemented based on the CAST communication framework. They are discussed in section 6.2. The evaluation on a real robot was executed on the BIRON system in an off-line and on-line manner and is presented in section 6.3. Yet, before presenting any results the measures used to evaluate the algorithm are introduced next.

6.1. Performance Measures

Measures to assess the performance of a fault detector should provide information about two major aspects: i) the speed of detection i.e., the latency $\Delta t = t_d - t_o$ between the occurrence of the fault (t_o) and the actual detection time (t_d)[1] and, ii) the reliability of the detection [29]. By interpreting fault detection as a binary classification task the latter aspect can be evaluated based on measures derived from a confusion matrix [3]. A confusion matrix represents the *false positives* (FP), *false negatives* (FN), *true positives* (TP), and *true negatives* (TN) values of a (not necessarily binary) classifier applied to a classification data-set. The matrix for binary classifica-

[1]In this work the detection time corresponds to the timestamp of the first data-point d after fault occurrence which has been classified as being faulty. It is worth noting that this is not always the case e.g., for fault detection approaches which compute a system health estimate periodically rather than for each gathered data-point.

tion is depicted in table 6.1. The terminology is adapted to the fault detection application case. Given a time-series C of classifications computed for a time-series D of

		System Behavior	
		faulty	normal
Detector Out-come	faulty	True Positive (TP)	False Positive (FP)
	normal	False Negative (FN)	True Negative (TN)

Table 6.1.: A confusion Matrix used to visualize the performance of a classification algorithm. The wording is adapted to the terminology of a fault detector.

data-points d which was recorded during a system run[2] where a fault has occurred at timestamp t_o. C can be re-written as:

$$C := (C^+, C^-)$$

where C^+ contains all classifications c with $t(c) < t_o$ (i.e., before the fault was induced) and C^- all c with $t(c) \geq t_o$. Here, $c = 1$ indicates a fault while $c = 0$ stands for no fault. Now, computing the confusion matrix on a set $\mathcal{C}$ of such classification time-series C_i can be done in two different ways: In the first case, the resulting values are called the *intra-run* measures and the word *intra* is added as a subscript to their notations in order to differentiate them from the second case. In this case, the FP value represents the number of fault detections in C_i^+ and TN the number of c which indicate no fault detection. Accordingly, TP stands for the number of detections in C_i^- and FN stands for all c in C_i^- representing no fault detection. Based on this, for a single classification time-series C_i the following equations can be defined:

$$TP_i^{intra} = \sum_{c \in C_i^-} c, \qquad FP_i^{intra} = \sum_{c \in C_i^+} c$$

[2]Here, a run is defined as a period of time where the system interacts with its environment.

and

$$TN_i^{intra} = \sum_{c \in C_i^+ \wedge c=0} 1, \qquad FN_i^{intra} = \sum_{c \in C_i^- \wedge c=0} 1.$$

In the second interpretation case, the values are called *inter-run*. Here, each single fault detection in C_i^n renders the complete run as a false positive run (FP) while a true positive (TN) requires C_i^n to contain no detections. Similarly, each single fault detection in C^- renders the whole run as TP and if C^- contains no single detection the run is marked as FN. The equations for the inter-run measures are given as follows:

$$TP_i^{inter} = min(1, \sum_{c \in C_i^-} c), \qquad FP_i^{inter} = min(1, \sum_{c \in C_i^+} c)$$

and

$$TN_i^{inter} = \lfloor \frac{1}{n} \sum_{c \in C_i^+ \wedge c=0} 1 \rfloor, \qquad FN_i^{inter} = \lfloor \frac{1}{n} \sum_{c \in C_i^- \wedge c=0} 1 \rfloor$$

where $\frac{1}{n}$ is a normalization factor and n is the number of runs considered for the computation of the measure. Figure 6.1 depicts the discussed measures on an exemplary time-series C_i together with the latency. It also shows the corresponding score time-series S_i and the threshold used to decide upon faulty and normal system state.

Computing these measures for the set $\mathcal{C}$ is realized by summing up over all $C_i \in \mathcal{C}$ (e.g., $TP^{inter} = \sum_{C_i \in \mathcal{C}}$). On the basis of the inter-run measures for $\mathcal{C}$ two more expressive measures can be computed which are the false alarm rate (FAR) and the fault detection rate (FDR):

$$FDR = \frac{TP^{inter}}{TP^{inter} + FN^{inter}}, \qquad FAR = \frac{FP^{inter}}{FP^{inter} + TN^{inter}}$$

FDR (also called *sensitivity*) indicates the likelihood that the fault detector will correctly detect a faulty state. FAR indicates the likelihood that the system in normal state will be wrongly classified as being faulty. FAR is closely related to *specificity* (1-FAR)

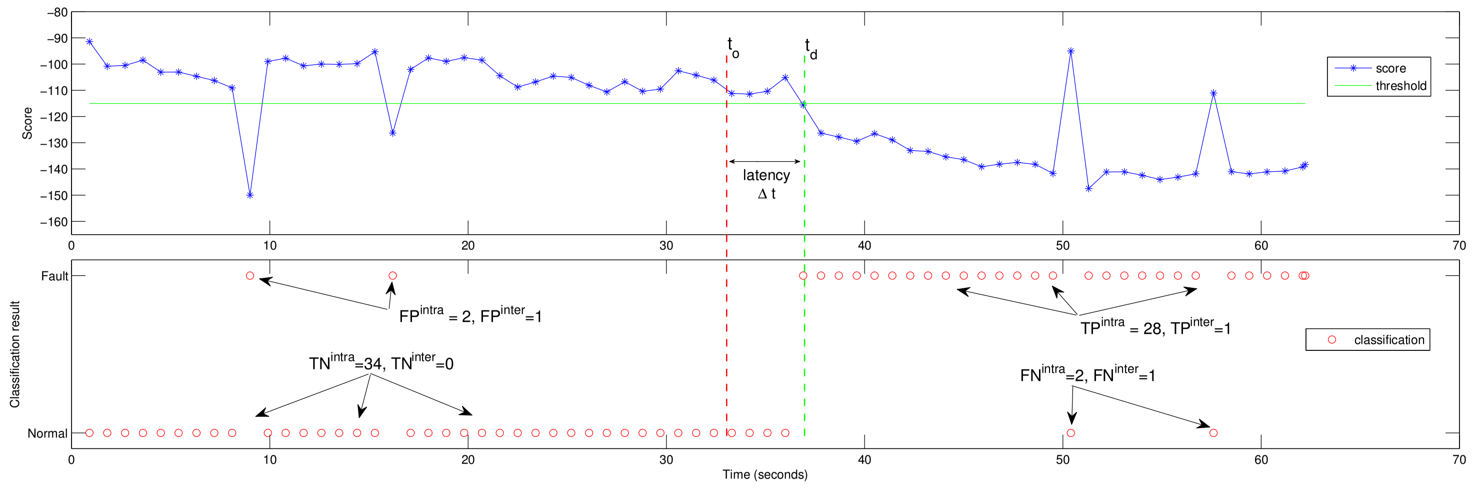


Figure 6.1.: Visualization of the FP, FN, TP, TN inter and intra measures along an exemplary score (top) and a classification (bottom) time-series. The timestamps of fault occurrence and fault detection are marked in both plots together with the latency and the values of the measures calculated for this concrete example.

which describes the likelihood that a normal state is detected correctly. While both measures express similar information, here FAR is preferred for the discussion of misclassification issues.

Applying the same formulas to the intra-run measures yields another two measures:

$$FTR = \frac{TP^{intra}}{(TP^{intra} + FN^{intra})}, \qquad SFAR = \frac{FP^{intra}}{FP^{intra} + TN^{intra}}$$

The first measure yields an estimate of the percentage of data-points in one run which were correctly classified as faulty. It is called the *fault tracking rate* (FTR). The second one provides an estimate of the average percentage of false alarms within one run and is called the *seriousness of false alarm rate* (SFAR).

The intuition behind the inter measures (FDR and FAR) is that they shall provide an estimate on how unreliable one single application of the detector to a system is i.e., how often does a given approach yield a fault detector which misses faults or reports non-existent ones. On the other hand, the intra measures (SFAR and FTR) enable the discussion of reliability within one run of an experiment which has practical implications in case further processing of the results of the detector takes place. For example, a detector may provide a poor FAR value over several trials, which makes it not useful in practice because the high number of wrongly reported faults would frequently result in (automatic) repair actions. However, if the detector features a low SFAR value additional temporal post processing may be applied to significantly reduce false alarm rate. Likewise, an approach may have a very high detection rate thus indicating good application performance. However, having a low FTR value (i.e., only few consecutive detections) may diminish its practical value because in this case it might be hard to distinguish real faults from false positives.

6.2. Simulation Experiments

Simulation experiments were conducted on artificially designed component systems using the CAST communication framework. This work was done in corporation with Jeremiah Via in the context of his bachelor thesis [126]. The primary goal was to apply the AuCom-detector to another robotic system providing evidence for its portability. Beyond this, Jeremiah's work yields insights into the performance of the fault detection approach applied to artificial patterns under controlled circumstances. This can be considered as a proof of concept for this approach.

In order to evaluate the system, Jeremiah created three different artificial CAST systems, each with properties to challenge the algorithm in some way. In each of the experiments, the data published by each component is mapped to a single composite feature i.e., the number of components corresponds to the number of composite features. The induced fault in the different system is a simulated crash of a component.

The artificial systems designed to evaluate the AuCom-detector on CAST are depicted in figure 6.2 and can be described as follows:

Linear Chain System: This is the simplest artificial system used for evaluation. It consists of three components which are connected in pairs resulting in a communication chain. Each component produces its output following a normal distribution with a mean of $100\ \mathrm{ms}$ and a standard deviation of $10\ \mathrm{ms}$ conditioned on the input of the predecessor component. The fault in this system affects the second component in the chain (Comp$_2$ in figure 6.2) . This system implements the Markov property which means that the probability of the data-point at time t to be generated by component $Comp_i$ is only dependent on the output generated at time $t-1$ by the component $Comp_j$. It shall provide evidence that the temporal dynamic features defined in section 4.2 are able to preserve enough information in order to model the Markov property.

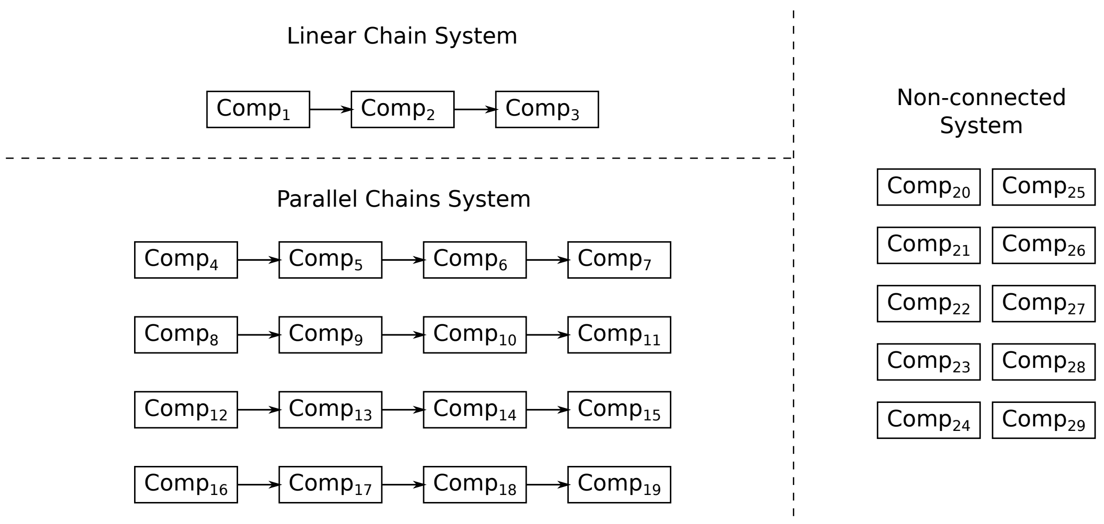


Figure 6.2.: The three different systems used in simulation in order to evaluate the AuCom-detector.

Parallel Chains System: This system simulates the problem of interleaved communication chains discussed in section 4.2. It consists of four independent linear chain systems running in parallel whereas the fault was induced in the first component of the third linear chain (Comp$_{12}$ in figure 6.2) . Regarding the complete inter-component communication the Markov property cannot be assumed like for the linear chain system. The goal here is to evaluate whether the approach can cope with this issue.

Non-connected System: In the previous two systems each component produced an output upon the input from another component within a strictly defined time frame describable by a unimodal distribution. However, in a realistic system interaction between components often feature more complex patterns. For example, components may decide to postpone the generation of an output based on their internal state leading to an increased variation of the patterns. In the extreme case the variation may reach a level where the components seem to interact arbitrarily. Although this corner case is not very realistic, it is simulated here with a set of ten unconnected components in order to evaluate how the AuCom-detector copes with this situation.

The fault induced in this system affects the first component (Comp$_{20}$ in figure 6.2).

6.2.1. Methodology

For each system Jeremiah executed ten experimental runs whereas each run consists of two phases. During the first phase, 4000 data-points are collected from a normal run of each system. This data is used to train the model. In the second phase, each system is run until another 4000 data-point were collected with a fault being induced at the 2000 message data-point mark. Jeremiah analyzed the performance using the latency, the FTR and the SFAR measures introduced in section 6.1.

6.2.2. Results

System	FTR	SFAR	Latency
Linear	1.00	0.00	$0.37\,\mathrm{s}$
Parallel	1.00	0.10	$0.15\,\mathrm{s}$
Non-connected	0.99	0.06	$0.50\,\mathrm{s}$

Table 6.2.: Results of the application of the AuCom-detector in simulation. They depict the performance of the approach for three different artificial systems.

Table 6.2 summarizes the results of the experiments. Jeremiah found out that the approach applied to the artificial patterns had a nearly perfect fault tracking rate in all experiments meaning correct classification occurred during almost the complete time where the system was in a faulty state. For the linear system case the algorithm has a SFAR value $0.0\,\%$ meaning that no false alarms were produced within the runs whereas in case of the parallel and non-connected systems the higher complexity of the systems results in several false alarms. All faults could be detected in less than half of a second. The approach was least performant on the non-correlated component system. This result is reasonable because this case does not correspond to the assumption made for the design of temporal dynamic features in section 4.2 where correlation between current and past data-points is presumed.

From the findings of the simulation experiments two conclusions can be drawn. First, the AuCom-detector is generally capable of detecting faults based on temporal dynamic features drawn from a sequence of data-points which were generated in simulation. The detector also shows a good performance in case where the correlation has not been introduced into a system on purpose (i.e., in the non-connected system case). Second, the developed approach could be successfully integrated into CAST which is a different communication framework than the initial one used to develop it. In addition, this integration was accomplished by Jeremiah Via, an external research student who has to initially get familiar with the topic of fault detection in general and my approach in particular. This emphasizes the plainness of applicability of this approach.

6.3. Robotics Experiments

After the evaluation of the AuCom-detector on artificial data this section is dedicated towards experiments with a real robot. It begins with a description of the interaction scenario used for the experiments in section 6.3.1. Next, section 6.3.2 is dedicated to experiments executed on previously recorded data in an off-line manner. Afterwards, in section 6.3.3 evaluation results of the approach in a scenario which could be executed on-line are presented.

6.3.1. The Interaction Scenario

Robotic experiments were conducted within the “Follow Me” interaction scenario which belongs to the set of standard scenarios designed for the RoboCup@Home competition 2010 [81]. In this competition the performance of robotic systems is compared under different interaction conditions. In the Follow Me scenario the robot has to execute navigation tasks commanded by a human. In particular, the human interaction partner uses the commands *follow me*, *stop*, *turn right* and *turn left* to indicate to the robot what to do. In order to make the interaction more robust the robot is allowed to ask questions (e.g. do you want me to follow you?) if it is unsure which action to perform and the user can correct the command if necessary.

The experiments with a real robot were executed on the robotic platform BIRON which was already introduced in section 5.3. BIRON was configured with components for spoken language understanding, utterances generation, interaction partner detection & tracking, and navigation. Figure 6.3 provides an overview of the components used in this scenario together with the exploited hardware. The system also contains an active-memory component called *ShortTerm* for communication purposes. Arrows between components represent direct communication while arrows to the memory represent insert, replace, or remove actions on the ShortTerm instance. No arrows were drawn for possible subscriptions of components on the memory as this information is not exploited in the model anyway and would clutter the figure. The numbers near the arrows correspond to the composite features listed on the right. They represent the communicated data in the BIRON system based on equation 4.2 from section 4.2. Note that the same composite feature may label the communication between different partners if the source is equal because the receiving component is not encoded in the feature. All in all, the system consists of ten different components which exchange data represented as fourteen different composite features. The inter-component communication in the system has an overall frequency of $163\ \mathrm{Hz}$ whereas single components produce outputs with frequencies ranging from $0.5\ \mathrm{Hz}$ to $86\ \mathrm{Hz}$. All experiments were executed in our laboratory and in the hallway adjacent to the labo-

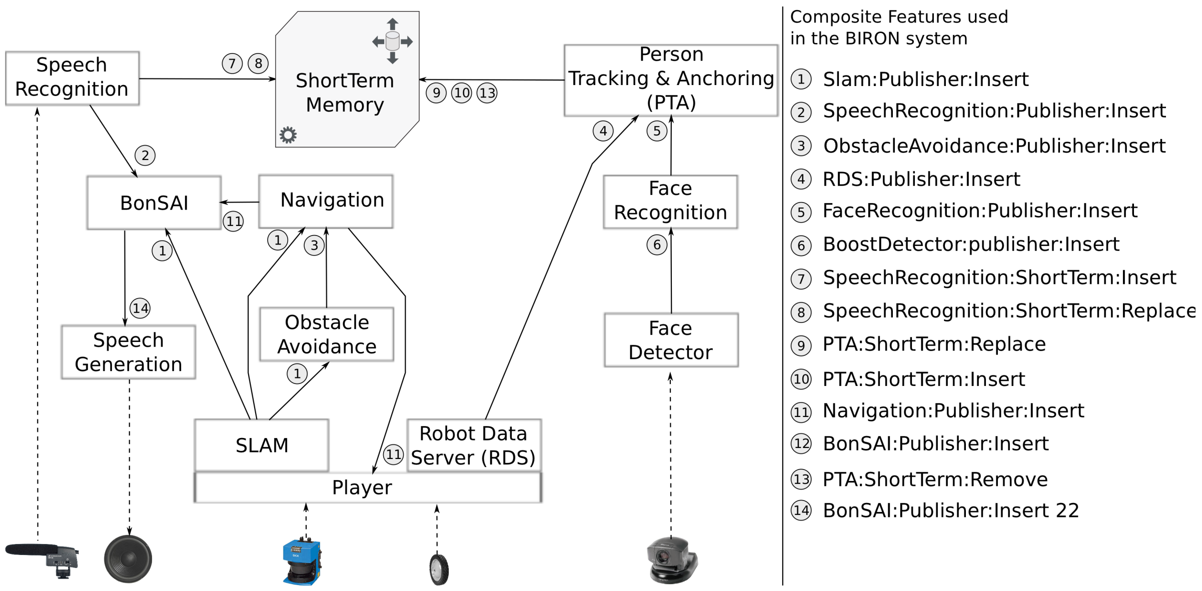

Speech Recognition
ShortTerm Memory
Person Tracking & Anchoring (PTA)
BonSAI
Navigation
Face Recognition
Speech Generation
Obstacle Avoidance
Face Detector
SLAM
Robot Data Server (RDS)
Player
Composite Features used in the BIRON system
1 Slam:Publisher:Insert
2 SpeechRecognition:Publisher:Insert
3 ObstacleAvoidance:Publisher:Insert
4 RDS:Publisher:Insert
5 FaceRecognition:Publisher:Insert
6 BoostDetector:publisher:Insert
7 SpeechRecognition:ShortTerm:Insert
8 SpeechRecognition:ShortTerm:Replace
9 PTA:ShortTerm:Replace
10 PTA:ShortTerm:Insert
11 Navigation:Publisher:Insert
12 BonSAI:Publisher:Insert
13 PTA:ShortTerm:Remove
14 BonSAI:Publisher:Insert 22

Figure 6.3.: A component view on the BIRON system used for evaluation. Arrows between components indicate direct communication while arrows to the memory represent insert, replace, or remove actions. The numbers represent composite features listed on the right.

ratory by the author of this thesis. The aforementioned commands were used in an arbitrary order and with arbitrary time intervals between them. The details on the concrete procedure of each experimental-run differ between the off-line and the on-line application case and are thus explained later on in the corresponding sections.

6.3.2. Off-line Evaluation

The first evaluation of the approach on a real robot was conducted in an off-line manner. This interim step on the way to an on-line application enables us to focus on the evaluation of the algorithm's performance thereby omitting any possible difficulties resulting from an on-line application like performance impact on the target system or an insufficient amount of data present for training. The results of this evaluation were published at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2010 [53].

Methodology Ten data-sets during the system's normal behavior were recorded and used as training input. Furthermore, for each different induced fault in the system additional ten data-sets were recorded. All sets represent 100 seconds of inter-component communication. In case of the faulty-runs the fault was randomly induced between second twenty and sixty. A leave-one-out cross-validation approach [14] was applied during the training phase which means that nine of the ten data-sets of normal behavior were used to estimate the model parameters and one data-set was used to estimate the parameters of the classifier. This procedure results in 100 runs for each fault. The measures used to assess the off-line performance are the same as in the simulation experiments. Four different types of faults were induced. The first two are crashes of components which are denoted with CC_1 and CC_2. For the CC_1 fault a crash of the component called player which publishes laser data into the system was triggered. This crash results in the immobility of the robot. The second fault affects the Simultaneous Localization and Mapping (SLAM) component which generates hypothesis about the robot's position in the environment. Without the information from the SLAM component, the robot cannot navigate anymore. The intention of these two cases was to evaluate if the approach is capable of detecting faults on different levels of data processing i.e., barely processed sensory data which occurs at a high frequency and hypothesis of the robot's position which is computed with lower frequency. As third fault a resource starvation (RS) problem was triggered by inducing a high CPU-load via a dedicated *busy worker* component. Resource starvation faults are of interest as they occur fairly often when integrating independently developed components into a robotic system. The last induced fault occurs in the context of distributed systems and results from asynchronous clocks on physically distinct machines. This fault affects all pairs of components which rely on synchronization based on the system clock if they run on different hardware. This case is called the asynchronous communication fault (AC).

Results The results for the detection performance are summarized in table 6.3. Compared to the simulation experiments the performance of the detector drops slightly for all used measures. This can be assigned to the increased variance in the data. While in the simulation case all components have a data-production-rate guided by a normal distribution with fixed parameters the real system does not explicitly encode such rules. Furthermore, additional variance and uncertainty results from the interaction of the robot with a human partner. The additional variance slightly increases the SFAR value which ranges from 2% to 12% with a mean of 4%.

Fault	**FTR**	**SFAR**	**Latency**
CC_1	$0.95\ (0.07)$	$0.12\ (0.04)$	$1.40\ (0.9)\ \mathrm{s}$
CC_2	$0.99\ (0.01)$	$0.02\ (0.01)$	$1.65\ (0.6)\ \mathrm{s}$
RS	$0.98\ (0.02)$	$0.04\ (0.03)$	$1.88\ (0.8)\ \mathrm{s}$
AC	$0.25\ (0.09)$	$0.05\ (0.05)$	$1.01\ (1.7)\ \mathrm{s}$

Table 6.3.: Fault detection results for the off-line application of the AuCom-detector on the BIRON system. The table shows the fault detection rate, seriousness of false alarm rate and the detection latency in regard to the four induced faults. The values are averaged over the trails of the experiments. The numbers in the parentheses represent the corresponding standard deviations δ of the averaged values.

Table 6.3 illustrates that good performance was achieved for the CC_1 fault which is the crash of the player component. Player's output is used by other components like SLAM and RDS and its absence influences the output of these components, too. Altogether, the CC_1 fault has a significant and rapid impact on the score manifesting in a tracking rate of 95% and a low detection latency averaging to $1.40\ \mathrm{s}$. For the CC_2 fault, which affects the SLAM component, the detector shows a nearly perfect tracking rate of 99%, although it features a slightly increased detection latency. This can be accounted to the lower data producing frequency of the SLAM component and reduced impact on the remaining system i.e., only one other component is affected by the CC_2 fault which is the RDS component.

Regarding the resource-starvation RS fault the detector shows a similar tracking rate as for the previous two faults ($98\,\%$). In contrast to the aforementioned faults which have a direct impact on a specific component known beforehand, the impact of the RS fault is undefined a priori. It rather affects all components which are currently running by “stealing” their resources. The detection latency increases slightly in comparison to the two previous faults featuring an average of $1.88\,\mathrm{s}$. The worst performance was achieved for the asynchronous communication fault where only $25\,\%$ of its occurrences could be detected. This can be accounted to the fact that the AC fault only affects components that are dependent on an accurately timed input from components which reside on the remote machine affected by the asynchronous fault. An example of such a component in the present system is the Person Tracking and Anchoring (PTA) component. PTA fuses information from different modalities in order to detect and track interaction partners. These modality data comes from the RDS and from the face recognition component, where RDS runs on the machine which is affected by the asynchronous fault. The usual behavior of this component during interaction is to initially detect a person at the beginning of the interaction and track it in subsequent processing steps. Detection of a new interaction partner triggers an Insert action on the ShortTerm memory while tracking a person is executed with Replace actions. Consequently, the communication data generated by the PTA during fault-free interaction is dominated by a pattern of consecutive Replace actions. After inducing the AC fault the communication also features this pattern almost consistently. Yet, there exist points in time where the PTA loses the current interaction partner indicated by a Remove action in the ShortTerm memory and followed by a period of missing Replace actions where the PTA could not establish a tracking mode for a few seconds. This pattern is a significant change to the modeled system behavior and the detector produces a lower score for this period of time eventually leading to the detection of a fault. After a while the PTA component manages to track the interaction partner again and the detector stops reporting a fault. Figure 6.4 depicts the described situation along an exemplary asynchronous fault experiment. It is divided into two sub-figures where the top one shows the computed score as a green line and the detector's fault detection output in red. The bottom figure shows the occurrences of the different composites features in the BIRON system with the discussed Replace actions highlighted in black. The timestamp of fault occurrence is marked in the figure at around second $25\,\mathrm{s}$. It is immediately recognized by the detector because the introduction of the asynchronous fault means a shift in time for all components which run on the machine affected by that fault. In subsequent timestamps the fault can be reported only infrequently. The most significant fault tracking period is marked in the figure in correspondence to the missing Replace action. This period correlates with the missing PTA Replace actions in the ShortTerm memory.

The AC fault case has shown that the data-driven AuCom-detector is not able to reliably track faults which manifest itself only temporarily in the inter-component communication of the system. In general, this flaw could be compensated by adding an additional decision mechanism on top of the current approach which exploits the output of the classifier and learns signatures of volatile faults. However, this extension is not in the focus of this work and thus in the remainder of the evaluation chapter the AC fault is omitted.

Results for Reduced Training Sets The presented results for the detection of faults on a real robot show a good performance of the detector. However, the experiments were executed in an off-line mode which is not the primary application case for a fault detection approach. The next step of the evaluation in section 6.3.3 shall therefore concentrate on the evaluation of the on-line performance. Before moving on to this scenario the performance of the detector in relation to the amount of data provided for training shall be examined here. The rationale is that while gathering training data in the off-line evaluation lasts approximately seventeen minutes, a shorter duration is desirable for the on-line application of the detector.

For that purpose, the experiments were performed on ten different portions of the original training data-set ranging from $90\,\%$ to $10\,\%$. The size of the test data remains

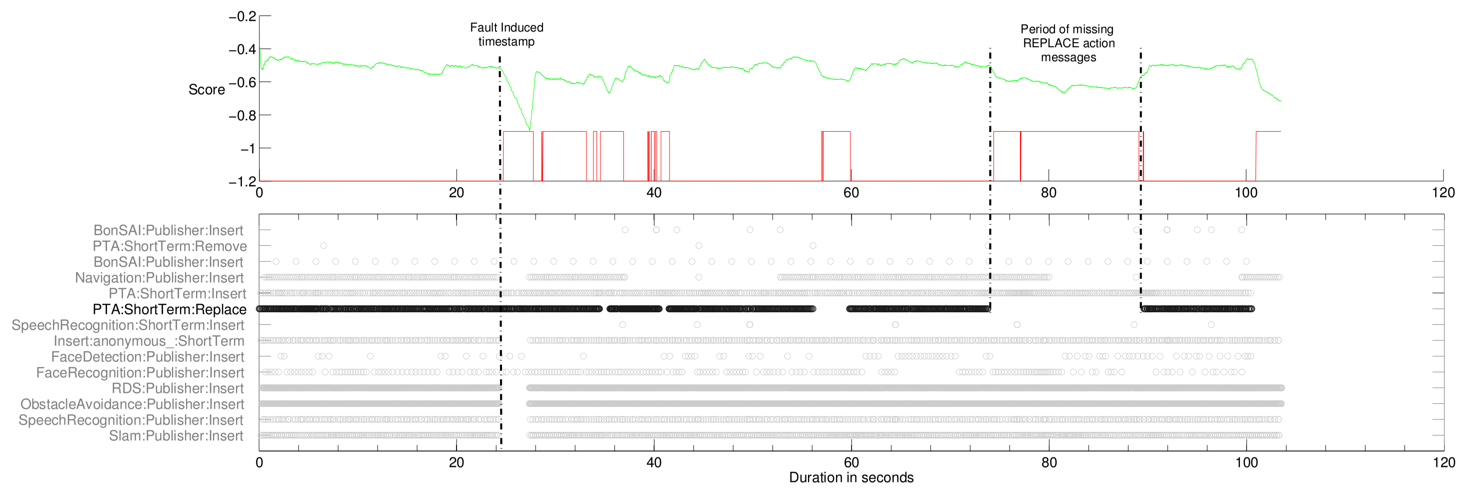
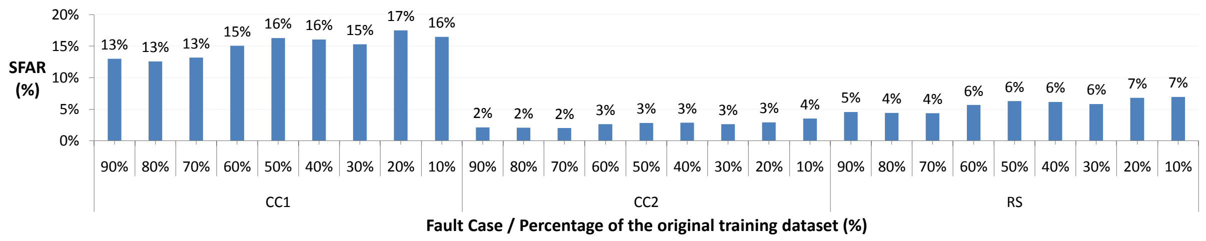
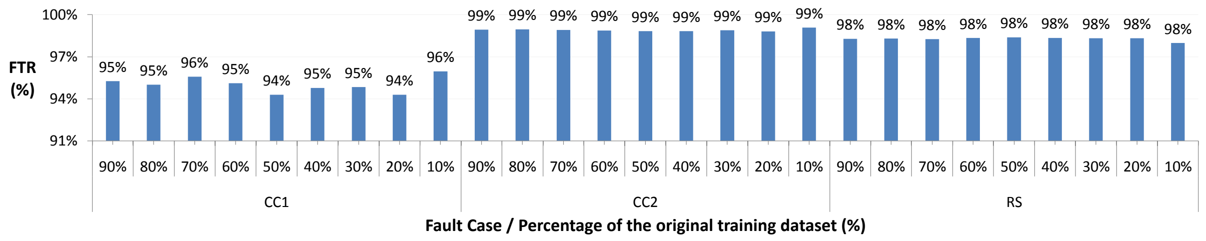


Figure 6.4.: Asynchronous communication fault example. The upper part of the figure shows the score (green) and the classification result (red). The bottom part shows the occurrences of the different composite features in the BIRON system. The figure also contains markers for the fault-induced timestamp and for the most significant fault tracking period.



(a) Seriousness of False Alarm Rate progression for differently sized training data-sets.



(b) False Tracking Rate progression for differently sized training data-sets.

Figure 6.5.: SFAR and FTR results for different percentages of the original training data-set.

the same. Figures 6.5 and 6.6 summarize the impact of the reduced amount of training data on the three applied measures SFAR, FTR, and latency. The results suggest that in general the reduced amount of training data has minor negative impact on the performance of the detector. Regarding the SFAR value the performance drops only slightly even with only $10\,\%$ of the original training size. Regarding FTR and latency the results reveal a minor improvement of the detector's performance. Collectively, the results show that for smaller training data-sets the detector tends to report faults more often which either lead to more false positives or to a higher fault tracking rate and a lower latency. Based on these results the amount of data for the evaluation in an on-line scenario was significantly reduced by roughly $70\,\%$ which corresponds to about $210\ \mathrm{s}$ of recorded ICC.

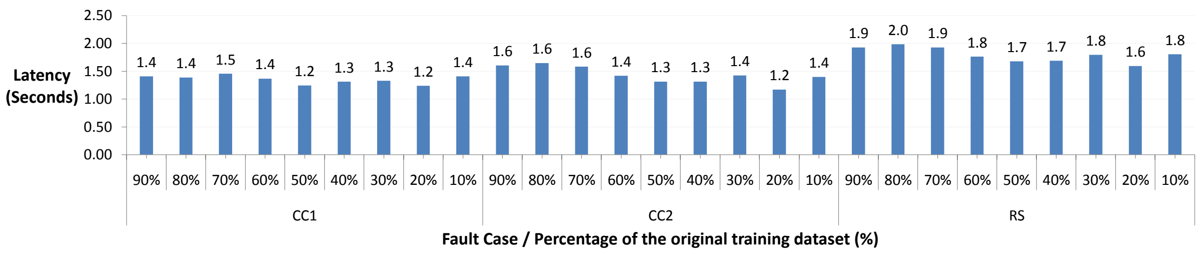


Figure 6.6.: Latency results for different percentages of the original training data-set in regard to the three faults CC_1, CC_2, RS.

6.3.3. On-line Evaluation

We now move on to the evaluation of the AuCom-detector in an on-line scenario. In this work, on-line fault detection means that data recording, model training and detection is executed in parallel to the running system. However, this section also contains evaluation results computed off-line (but still on the data recorded during the on-line experiments). This off-line experiments were conducted in order to evaluate an optimization to the original approach and to compare the performance of the AuCom-detector to a baseline fault detection approach.

This part of evaluation begins with the introduction of a baseline approach. The remainder of this section is structured as follows: Firstly, the methodology used in the on-line case is presented. Secondly, the results of the on-line application and the outcomes of the comparison to the baseline approach are discussed. Finally, an adaptation of the AuCom-detector is evaluated which enables the reduction of false positives generated by the detector.

The Baseline Fault Detector The chosen baseline approach is a data-driven fault detector. Similarly to the AuCom-detector, it incorporates temporal aspects of the input data into the model. The input for the approach is a time-series of composite features e representing communicated data-points d (See. section 4.2 for details on composite features). The baseline approach also benefits from the generic representation of the communication but does not exploit the temporal dynamic features (TDFs) introduced in this work. Consequently, comparing the AuCom-detector to the baseline provides also a rough estimate on how fault sensitive the TDF features are in contrast to the generic representation of the communication.

The fault detection idea for the baseline approach is to count the number of occurrences of the different $e \in \mathcal{E}$ (i.e., all known composite features in a given system) in a time interval Δt and compare it to the expected number of occurrences. If the number of occurrences meets the expectation the system is assumed to behave normally, otherwise it is assumed to be in a faulty state. To implement this idea histograms are exploited as follows:

Let E be a time-series of composite features e representing recorded inter-component communication. E is segmented into intervals E_i of length Δt. Each interval is represented with a histogram M_i consisting of one bin m_i^e for each $e \in \mathcal{E}$. A bin m_i^e is a function which counts the number of e in E_i:

$$m_i^e = m^e(E_i) = \sum_{\hat{e} \in T_i : \hat{e} = e} 1$$

A model $\overline{M} = \{\overline{m}^e | e \in \mathcal{E}\}$ of normal system behavior is a histogram which represents the mean number of occurrences of $e \in \mathcal{E}$ in the segments E_i of a training time-series E. The bins $\overline{m}^e$ are computed on E as follows:

$$\overline{m}^e(E) = \frac{1}{n} \sum_{E_i \in E} m^e(E_i)$$

where n is the number of segments E_i of length Δt generated from E.

Now, let E be a time-series recorded during an unknown system state. Assessing whether E belongs to a normally behaving system consists of again segmenting E into intervals E_i of length Δt, computing histograms M_i and measuring the histogram distance $\check{d}$ between M_i and $\overline{M}$. The distance $\check{d}$ is given as:

$$\check{d} = \sum_{g=1}^{k} |m_i^k - \overline{m}^k| \tag{6.1}$$

which is the Manhattan distance between two histograms [27]. The decision if M_i represents an interval E_i recorded during a faulty system state is made by applying the following equation:

$$\text{faulty}(M_i) = \begin{cases} \text{True} & : \check{d}(M_i, \overline{M}) > \check{d}^* \\ \text{False} & : \text{else} \end{cases} \tag{6.2}$$

whereas $\check{d}^*$ is a constant threshold which separates normal behavior from faulty one. In contrast to equation 4.20 normal behavior is indicated by small $\check{d}$ i.e., a small distance between the histograms. If the distance is too large, the a fault is assumed. The estimation of $\check{d}^*$ is done in a similar way as for the AuCom-detector approach (See the classification paragraph in sec. 4.4). Since the number of bins in each histogram is predefined by the number of different composite features $c \in \mathcal{E}$, the parameters which need to be estimated in order to apply this approach are the interval Δt and the threshold $\check{d}^*$.

Methodology Having described the baseline approach I now introduce the methodology for the on-line fault evaluation experiments: Ten evaluation runs were conducted for each induced fault. Each run consists of three phases, all performed on-line during the *Follow Me* task. In phase one data from 210 seconds of interaction is recorded and used to train the fault detection model. Next, the last thirty seconds of the training data are used to optimize the parameters of the classifier. After training the model and assessing the parameters of the classifier the detector is ready to monitor the system. The subsequent evaluation phase is defined as follows: First, 30 seconds of normal interaction are monitored. Second, the fault is induced at a random point in time but no later than after additional 20 seconds. After that, the faulty state is then monitored for 30 seconds. If the detector reports a fault within this time, the fault is considered to be detected. Otherwise, the detector fails to detect the induced fault.

The analysis in the on-line case is done based on all measures mentioned in section 6.1 i.e., FDR, FPR, detection latency, the SFAR, and FTR. The induced faults are the same as in the off-line evaluation case except for the asynchronous communication (AC) fault which was omitted due to the reasons discussed in the off-line evaluation section. It was replaced with a new fault case which is the crash of the speech recognition (CC_3) component of the robot resulting in the inability of the system to understand spoken commands. This fault widely influences the human robot interaction because the robot cannot receive further commands after fault occurrence.

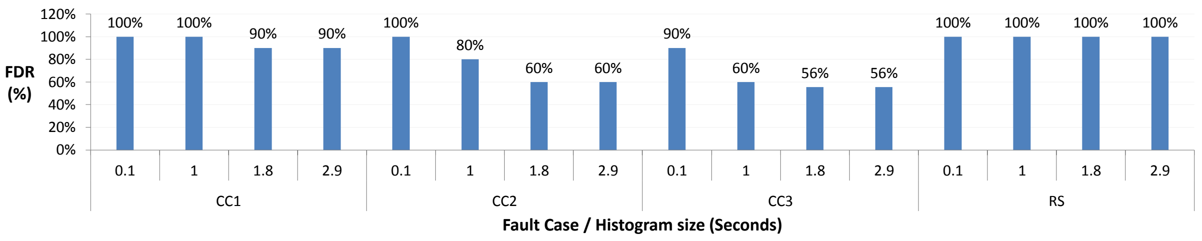
Results The experimental results of the application of the AuCom-detector to BIRON in an on-line manner are shown in table 6.4. In general, the results suggest

Fault	FDR	FTR	FAR	SFAR	Latency
CC_1	1.00	$1.00\ (0.00)$	1.00	$0.19\ (0.15)$	$0.84\ (0.49)$ s
CC_2	1.00	$0.72\ (0.23)$	0.80	$0.17\ (0.06)$	$2.83\ (3.70)$ s
CC_3	0.90	$0.81\ (0.23)$	0.70	$0.22\ (0.28)$	$9.18\ (6.15)$ s
RS	1.00	$0.84\ (0.18)$	0.80	$0.19\ (0.28)$	$0.67\ (2.63)$ s

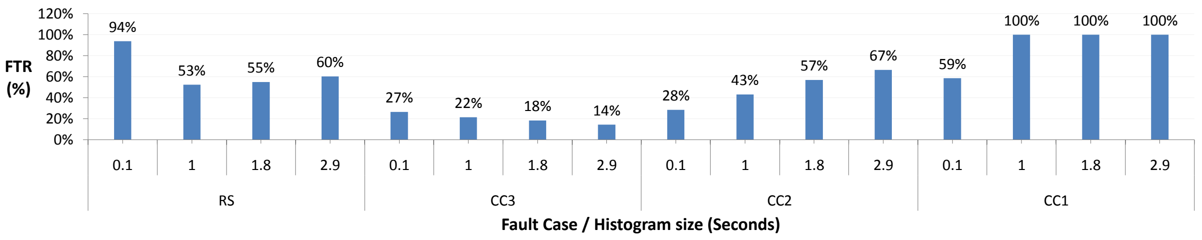
Table 6.4.: Fault detection results for the on-line application of the AuCom-detector on the BIRON system in regard to the four induced faults. The FTR, SFAR and Latency values are averaged over the trails of the experiments. The numbers in the parentheses represent the corresponding standard deviations δ of the averaged values. δ for the FDR and FAR values is not useful because these two measures represent binary decisions for each experiment run (i.e., a fault could be detected (FDR) or a false alarm occurred during the run (FAR)).

that the AuCom-detector applied in the on-line scenario shows a weaker performance compared to the previous scenarios which can be accounted to the reduced amount of training data. The fault tracking rate is still high ranging from $72\,\%$ to $100\,\%$ resulting in a fault detection rate between $90\,\%$ and $100\,\%$ which means that in worst case the fault couldn't be detected in two of ten runs. Regarding unnecessary alarms of the detector the performance degrades noticeably in relation to the previous experiments. The intra-run measure SFAR shows that about $20\,\%$ of the assessed data before fault induction was classified wrongly and that false alarms exist in $70\,\%$ of the runs in best case. The detection latency covers a range between $0,84\ \mathrm{s}$ for the CC_1 fault which is the crash of the player component up to $9.18\ \mathrm{s}$ for the newly introduced fault i.e., the crash of the speech recognition. The overall performance drop is also reflected in an increased variance for the measures depicted in table 6.4. Next, these results are discussed in comparison to the performance of the introduced histogram based baseline detector.

Figures 6.7, 6.8, and 6.9 depict the outcome of the histogram based detector for different time intervals Δt. The complete evaluation was executed on four intervals from the range of $0.1\ \mathrm{s}$ to $2.9\ \mathrm{s}$. The baseline approach shows a detection rate between 90% and 100% for the smallest interval i.e., $\Delta t = 0.1$ seconds which is comparable to the performance of the AuCom-detector. With an increase of Δt the detection performance drops significantly for the CC_2 and CC_3 faults. Regarding the FTR measure it can be seen that the AuCom-detector outperforms the baseline approach for every faulty state independently of the chosen Δt except for the RS fault with the parameter $\Delta t = 0.1\ \mathrm{s}$ where both perform well while the baseline solution is slightly better. Furthermore, the plots reveal that an increase of the interval for histogram generation almost always reduces the tracking abilities of the detector except for the CC_2 fault case. In case of the latency figure 6.9 suggests that the base-line approach is faster than the AuCom-detector with detection latencies ranging between $0.13\ \mathrm{s}$ and $2.23\ \mathrm{s}$ for the best interval value which is again $\Delta t = 0.1\ \mathrm{s}$. However, if the SFAR value of the detector for $\Delta t = 0.1\ \mathrm{s}$ is taken into account the fast detection can be at least partially accounted to the presence of a high number of false positives (see figure 6.8). This



(a) Fault Detection Rate

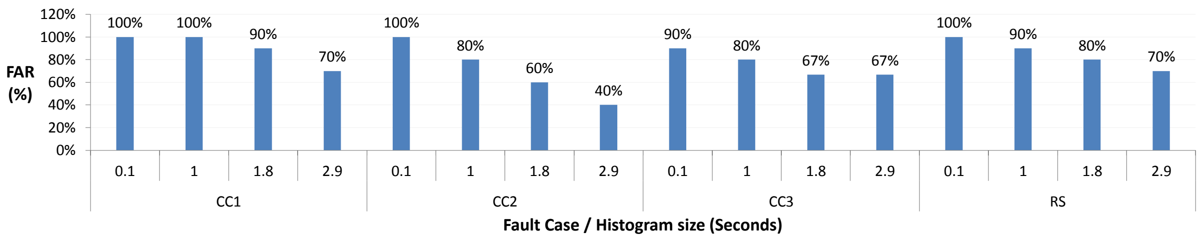


(b) Fault Tracking Rate

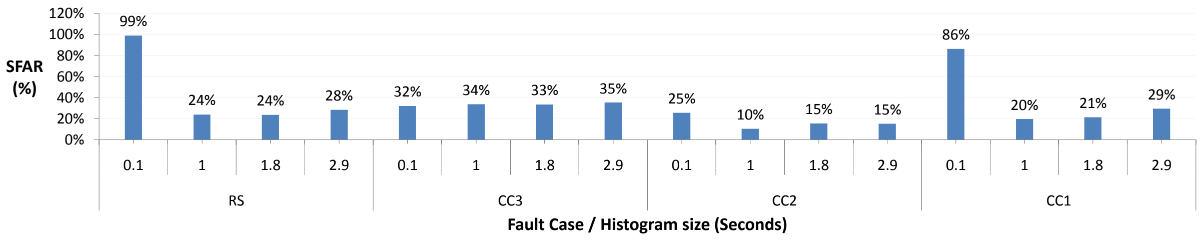
Figure 6.7.: FDR and FTR results for the baseline approach in regard to the four faults CC_1, CC_2, CC_3, RS and different time intervals Δt.

assumption is also supported by the progress of the latency in case Δt is increased. In all cases it either doubles as in case of CC_3 or is increased by at least one order of magnitude for the remaining faults. The results of the baseline approach for the interval $\Delta t = 0.1\,\mathrm{s}$ show a high false alarm rate of at least $90\,\%$ and a SFAR value ranging between $99\,\%$ and $25,\%$. The increase of Δt leads to the reduction of the FAR values ranging between $70\,\%$ and $40\,\%$ depending on the specific fault. The SFAR measure also improves reaching values between $32\,\%$ and $10\,\%$. In general it can be concluded that the increase of Δt improves the performance of the baseline approach in terms of the FAR and SFAR measures. It negatively influences the performance in terms of the FDR, FTR values and the detection latency. The simultaneous performance reduction in terms of the SFAR and FTR (i.e., more fault detections altogether) measures suggests that the approach gets insensitive to faults in general when Δt is increased.

The just discussed findings reveal that the AuCom-detector is generally superior in terms of fault detection and tracking. Furthermore, depending on the chosen parameter for the baseline approach the AuCom-detector either outperforms it in terms of



(a) False Alarm Rate



(b) Seriousness of False Alarm Rate

Figure 6.8.: FAR and SFAR results for the baseline approach in regard to the four faults CC_1, CC_2, CC_3, RS and different time intervals Δt.

SFAR or shows equivalent performance. Regarding the number of experiment-runs with false alarms (i.e., FAR) the baseline approach can be parameterized to perform better than the AuCom-detector yet at the costs of higher detection latencies. The detection latency results initially suggest that the baseline shows a better performance. Yet, this only happens at the cost of a very high SFAR value which usually cannot be tolerable in an on-line scenario. When omitting this single configuration (i.e., $\Delta t = 0.1\,\mathrm{s}$) the AuCom-detector performs better in terms of latency except for the RS fault.

Adjusting the False Alarm Rate The general application case of a fault detector is to run in parallel with the monitored system and report detected faults. Usually, this information is then used to execute analysis tasks in order to localize and identify the fault and trigger recovery actions. Recovery actions may be as simple as restarting or reconfiguring a component or may lead to the restart of the complete system. In any case recovery takes time. Bearing this in mind, the role of false positives

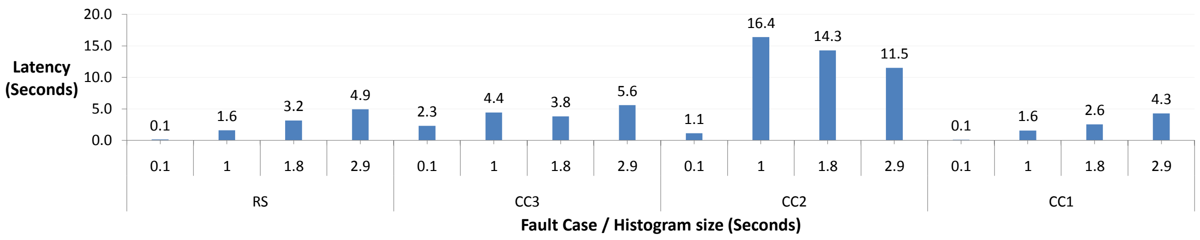


Figure 6.9.: Latency results for the baseline approach in regard to the four faults CC_1, CC_2, CC_3, RS and different time intervals Δt.

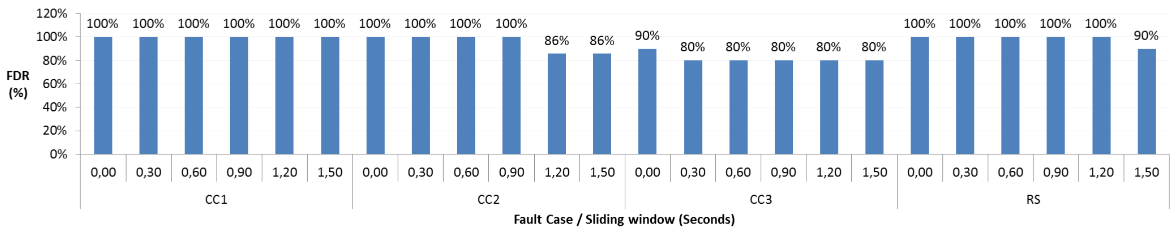
becomes critical for on-line fault detection which is why this paragraph evaluates an extension of the previous approach which enables the active control of false positives. The extension is a sliding window function *sw* applied to a short history of scores s_i relative to a score value s computed for a temporal dynamic feature tdf. The function is defined as follows:

$$\text{sw(s,}\Delta t) := \bar{s} = \frac{1}{|W|}\sum_{\hat{s}\in W}\hat{s} \tag{6.3}$$

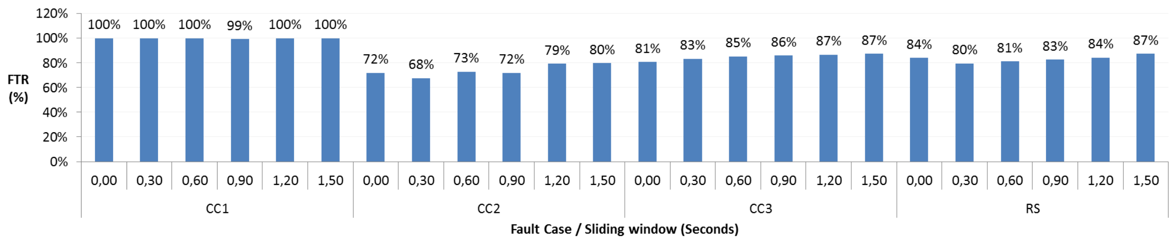
where W is defined as:

$$W = \{s_i | t^{s_i} \leq t^s \ \wedge \ t^{s_i} > t^{s_i} - \Delta t\}. \tag{6.4}$$

The sw function was applied to the data of the on-line scenario in a post-mortem evaluation step. The results of the experiments conducted on intervals ranging from $0.3\ \mathrm{s}$ to $1.5\ \mathrm{s}$ seconds with a step size of $0.3\ \mathrm{s}$ can be seen in figures 6.10 and 6.11. For comparison, the figures also contain the original results (i.e., without smoothing) indicated by the $0.0\ \mathrm{s}$ interval. In general, the sliding window has the expected impact on the results. With an increasing Δt the SFAR can be reduced by $7\,\%$ to $15\,\%$ depending on the type of fault. Similarly, the percentage of runs having false alarms (FAR) is also reduced although it remains at a relative high level ranging between $40\,\%$ and $57\,\%$. The consequence of these results is that in order to take action based on the outcome of the detector further processing steps need to take care of false alarms e.g., by incorporating additional fault decision logic on top of the detector's results. The capability to detect faults (FDR) in a run is slightly diminished whereas the track-



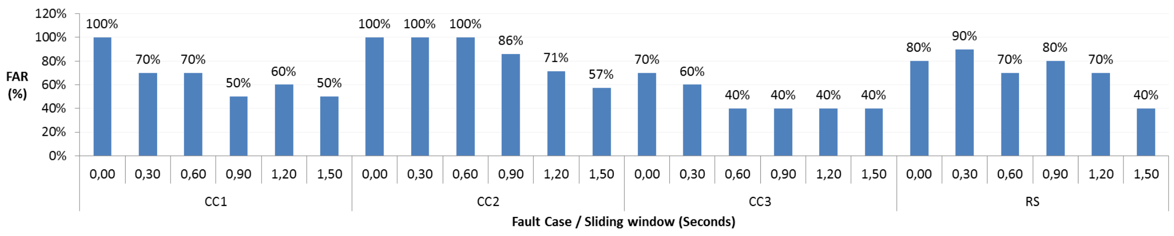
(a) Fault Detection Rate



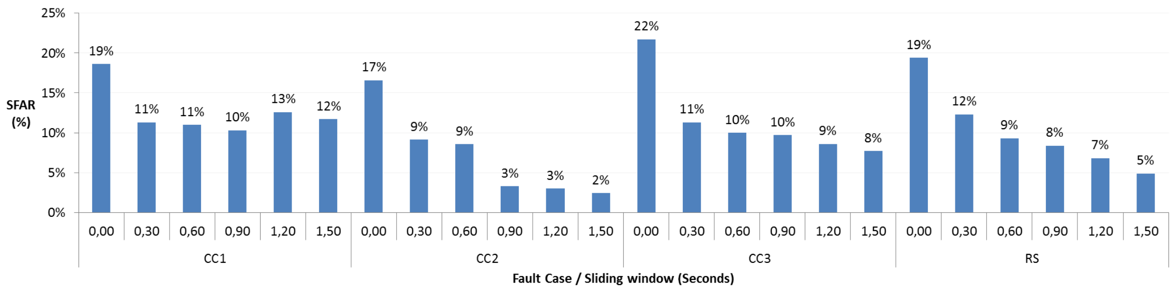
(b) Fault Tracking Rate

Figure 6.10.: FDR and FTR results of the AuCom-detector after applying the sliding window function as defined by equation 6.3. The interval Δt ranges between $0.3\,\mathrm{s}$ and $1.5\,\mathrm{s}$. The values for $0.0\,\mathrm{s}$ are the original results without sliding window usage.

ing rate (FTR) increases by $3\,\%$ to $8\,\%$. The FDR values indicate that with a smoothed score a single application of the approach consisting of training and parameter optimization has a marginally lower probability to provide a fault sensitive detector. The FTR results however, indicate an increase in the detector's capability to track faults. In combination with the reduced SFAR measure the results imply that the sliding window smoothing provides an improved decision-making basis for further processing steps i.e., diagnosis and recovery. The smoothing of the score also impacts the detection latency in the expected way. Depending on the induced fault the latency increases between $0,84\,\mathrm{s}$ and $2,43\,\mathrm{s}$ as can be seen in figure 6.12. The severity of this increase in detection latency strongly depends on the influenced components and the current state of the system which makes it difficult to make a general statement. For example, the occurrence of the RS fault during interaction in the "Follow Me" scenario would result in the system's inability to react upon new commands. This may lead to an



(a) False Alarm Rate



(b) Seriousness of False Alarm Rate

Figure 6.11.: FAR and SFAR results of the detector after applying the sliding window function as defined by equation 6.3. The interval Δt ranges between $0.3\ \mathrm{s}$ and $1.5\ \mathrm{s}$. The values for $0.0\ \mathrm{s}$ are the original results without sliding window usage.

unsatisfactory interaction experience but most probably does not result in harm to a human, the system or its environment. On the other hand, if the system experiences a fault in the navigation components (i.e. fault CC_2) while it is moving, a low detection latency is recommended in order to prevent the robot to hit anything. Altogether, the results show that by applying the sliding window function to the score the detector's reliability can be significantly improved at the cost of the detection latency.

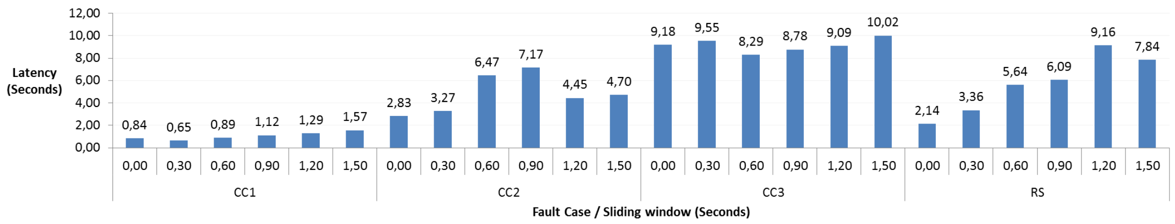


Figure 6.12.: Latency results of the detector after applying the sliding window function as defined by equation 6.3. The interval Δt ranges between $0.3\ \mathrm{s}$ and $1.5\ \mathrm{s}$. The values for $0.0\ \mathrm{s}$ are the original results without sliding window usage.

6.4. Discussion

This chapter summarizes experimental results conducted within the scope of this thesis. At first, fault detection in three artificial systems was evaluated in simulation. Here, the primary goal was to demonstrate that the approach can be successfully applied to systems with a different underlying communication framework than the one used in BIRON which is the robotic system the detector has been initially developed for. The experiments were conducted on artificial systems designed within the CAST communication framework representing three probable communication situations in a real robotic system which are: i) a linear communication chain based on the Markov property, ii) several linear communication chains executed in parallel resulting in an interleaved communication as discussed in section 4.1, and iii) non-connected communication chain which simulates a system of independently acting components. In all three cases the AuCom-detector shows good performance.

Next, the results of experiments executed on data gathered during interaction with the robotic system BIRON have been discussed. In the off-line case four different faults have been examined targeting different processing aspects in the system. While for three of them the results suggest good performance of the AuCom-detector, the asynchronous communication fault case reveals the detector's weakness in case where a fault manifests itself only sporadically in the inter-component communication. This can be accounted to the fact that my detection approach assumes that a fault has a continuous impact on the system's inter-component communication. Furthermore, the approach does not utilize dedicated fault models to capture specific fault patterns. Both aspects represent reasonable research topics for future extensions of the AuCom-detector. Another finding resulting from the off-line evaluation is that the training input for the data-driven fault detector can be reduced significantly with only minimal impact on the detector's performance. This reduces the time to set up the detector which is particularly beneficial when the detector has to be trained and used on-line.

The findings from the off-line experiments influenced the evaluation of the detector in an on-line scenario in two ways: i) the training data has been reduced to 30% of the size of off-line training sample and, ii) the asynchronous fault case has been replaced by a fault impacting speech recognition of the robot. Results from the on-line case have also been compared to the performance of a base-line fault detection approach and proved to outperform it. A relevant aspect in the on-line case is the presence of false positives if the results of the detector are used to trigger additional processing steps. Frequent false alarms may result in unnecessary execution of recovery strategies which is undesirable as it has a negative impact on the overall interaction experience with the system. As a consequence, in the last evaluation case the AuCom-detector has been extended with a sliding window approach which turns out to be an effective way to reduce the number of false positives. The downside of the extension is that with an increase of the sliding window size the latency of the detector increases, too.

The major results of the evaluation chapter are the following: Data-driven fault detection based on temporal dynamic features extracted from the inter-component communication of a robotic system is feasible. In general, the approach shows a high detection rate of the induced faults and performs with an acceptable false alarms rate which can be further reduced at the costs of detection latency. The question whether the achieved detection latency is acceptable or not cannot be answered in general.

The concrete answer depends on the interaction state of the robot and the type of experienced fault. For example, experiencing resource starvation during interaction in the “Follow Me” scenario may delay the system’s reaction upon new orders leading to an unsatisfactory interaction experience. Yet, most probably it does not result in harm to a human, the system or its environment. In this case, several seconds of delay until fault detection and subsequent triggering of recovery routines may be acceptable. In another situation for example when a fault disables the navigation component of the robot (i.e. fault CC_2) while it is moving a low detection latency is required in order to prevent the robot from hitting anything on its way. Hence, from a conservative point of view the latency performance of the AuCom-detector is the prime area for further improvements.

7. Conclusion

Fault detection is a key element for fault tolerant behavior and a mean to increase the system's dependability. Dependability is considered to be of particular importance for robotic systems which act closely to human users and share the same spaces. In this thesis I proposed a fault detection approach as a means to autonomously detect faults in cognitive robotic systems build upon the concepts of Component Based Software Engineering also known as Component Based Robotic Systems (CBRS). In particular, I answered the question on how to apply a data-driven fault detection approach to a CBRS without modifying its components. This section summarizes the findings of this thesis and provides suggestions on what questions need to be addressed in future research.

7.1. Summary of Contributions

Successful fault detection in an artificial system depends on the ability of the approach to cope with challenges arising in the context of that specific system. Regarding cognitive robotics challenges considered in this work arise from i) specific properties of the targeted system, ii) its development process and iii) the application of the system in real-world scenarios. The challenges identified for the targeted systems in this work were discussed in section 2.2.1. In particular, these are: minimal invasive integration, minimization of expert knowledge required for application, robustness in face of frequent changes as well as uncertainty and variant behavior, and detection of hardware

& software faults. These challenges were used to assess the applicability of fault detection approaches in literature in chapter 3. They also guided the development of the fault detection algorithm in chapter 4 and its implementation in chapter 5.

The literature review conducted in chapter 3 showed the broad range of different fault detection approaches but also revealed a lack of solutions which tackle the introduced challenging aspects and thus would be suitable for fault detection in CBRS. As a consequence, in chapter 4 I presented a novel fault detection approach which belongs to the set of data-driven approaches. Learning a model from data provides a convenient way to adapt to changes in the systems. The training input for the presented approach consists of the system's inter-component communication. Furthermore, the approach exploits only generic attributes extracted from this communication. By this means, the components of the system remain untouched which reduces the coupling of the detector to a system and the amount of component specific knowledge necessary in order to apply the approach to a new robot. The generic attributes are used to represent each communicated information element as a temporal dynamic feature which encodes temporal relations between past elements in the communication. The fault detection model is founded on statistical modeling techniques and represents only the normal state of the system without any representation of possible faults. By this means, the approach does not depend upon the presence of exemplary data of fault situations in order to detect them and can cope with unknown faults.

The approach was implemented in a modular way by utilizing a graph like processing structure. The decomposition into sub-modules is aligned to the algorithmic processing of the detector. This simplifies reconfiguration and reduces adaptations efforts when applying the approach to a new system. The discussion of integration costs in section 5.3 led to the following results: i) Integration requirements affect the communication framework of the targeted robotic system only and ii) the approach can often be integrated by exploiting build-in functionalities of a particular communication framework of a system. Concrete integration efforts were shown for the robotic platform BIRON where in fact build-in functionality could be exploited. In addition, possible means on how to apply the approach to communication frameworks commonly used in cognitive robotics were presented.

Evaluation of the algorithm was conducted in simulation as well as on the robotic platform BIRON. Simulation results yield an initial proof of concept for the detector and demonstrate its integration into a communication framework other than the one the approach was initially developed for. Robotic experiments were executed off-line and on-line.

Major evaluation results are: The data-driven fault detection based on temporal dynamic features extracted from the inter-component communication of a robotic system is appropriate for the detection of the faults induced in this thesis. The approach showed a high detection rate of the induced faults and performs with an acceptable false alarms rate which can be further reduced at the costs of detection latency. The question whether the achieved detection latency is acceptable couldn't be answered in general. The answer depends on the interaction state of the robot and the type of experienced fault. The latency performance of the AuCom-detector was identified as a prime area for improvement.

7.2. Discussion and Future Perspectives

This thesis has shone light on the topic of fault detection for CBRS with cognitive capabilities. The findings and experiments executed in the context of this work suggest that a data-driven approach based on inter-component communication is a reasonable solution in order to detect faults in these type of systems. Being able to detect faults provides the option to complete the fault tolerant loop consisting of additional means like fault identification, fault localization, and fault recovery. Consequently, one next step which needs to be taken in future research is to extend the current approach and add capabilities to backtrack the source of the fault. The main question is how the fault can be located if it has been propagated in the system and is evident in the communication output of several components which makes the pinpointing of the real source challenging.

The evaluation presented in this work was focused on the robotic platform BIRON featuring the XCF communication framework and on the CAST communication framework. In addition, suggestions on how to apply the detector to further systems were given. In order to confirm the findings in this thesis an aim in future research should be the evaluation of the approach on additional robotic systems. A reasonable next step would be the evaluation of the approach on a robotic system based on ROS as a popular and widely accepted communication framework (see [1] for a list of systems). A similar objective can be stated for the number and type of evaluated faults in the system. In particular, the evaluation of hardware faults should be tackled in future experiments in order to confirm the performance of the proposed solution.

Robotic systems in this work feature close interaction with humans. As such, delays or unexpected behavior due to faults may have a negative impact on the user's experience. In such situations feedback to the user about the current health status of the system would be beneficial. The system may tell the user to wait while it fixes the problem or it may ask him to call for help after several recovery trials have failed. Therefore, another research direction which is worth to follow up is the question on how and how much feedback about faults should be provided to the user.

One of the challenges tackled in this work is variation in the system's behavior. Here, this issue was resolved by exploiting kernel density estimators as the basis for the statistical model. By this means, different system states were modeled implicitly. A possible future enhancement to the model may involve an explicit representation of the system's different behavior states. Besides potential detection performance improvements explicit states may be beneficial when debugging the system. The source of a fault can be tracked down faster because the fault can be related to a part of the behavior space which again represents a subset of components. In addition, this information can be also evaluated in regard to the intended behavior of the system in order to detect unintended or unexpected states or transitions between states. Explicit state representation could be done by re-interpreting the current model as a dynamic Bayesian network and adding an additional discrete variable which represents different behavior states. An estimate for the number of states and corresponding training data could be acquired by segmenting the original training data.

Bibliography

[1] Robots using ROS, 2013. http://www.ros.org/wiki/Robots#Mobile_robots.

[2] ROS Tutorials, 2013. http://www.ros.org/wiki/ROS/Tutorials.

[3] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2004.

[4] Noriaki Ando, Takashi Suehiro, and Tetsuo Kotoku. A Software Platform for Component Based RT-System Development: OpenRTM-Aist. volume 5325 of *Lecture Notes in Computer Science*, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[5] Gianluca Antonelli. A Survey of Fault Detection / Tolerance Strategies for AUVs and ROVs. *Fault Diagnosis and Fault Tolerance for Mechatronic Systems Recent Advances*, 1:109–127, 2003.

[6] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1:11–33, 2004.

[7] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. Visual tracking with online multiple instance learning. *IEEE transactions on pattern analysis and machine intelligence*, 33(8):1619–1632, December 2010.

[8] A Bajwa and A Sweet. The Livingstone model of a main propulsion system. In *IEEE Aerospace Conference*, volume 2, pages 869–876, 2003.

[9] Christian Bauckhage, Sven Wachsmuth, Marc Hanheide, Sebastian Wrede, Gerhard Sagerer, Gunther Heidemann, and Helge Ritter. The Visual Active Memory Perspective on Integrated Recognition Systems. *Image and Vision Computing*, 26:5–14, 2006.

[10] Stephen D. Bay and Mark Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, page 29, New York, New York, USA, August 2003. ACM Press.

[11] Constantin Berzan and Matthias Scheutz. What am I doing? Automatic construction of an agent's state-transition diagram through introspection. In *11th International Conference on Autonomous Agents and Multiagent Systems*, pages 189–196, June 2012.

[12] M Birbeck, J Duckett, and O G Gudmundsson. *Professional XML*. Wrox Press Inc., 2nd edition, 2001.

[13] Andrew D. Birrell and Bruce Jay Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1):39–59, February 1984.

[14] Christopher M Bishop. *Pattern Recognition and Machine Learning*, volume 4 of *Information science and statistics*. Springer, 2006.

[15] Mogens Blanke. *Diagnosis and fault-tolerant control*. Springer Verlag, 2003.

[16] Jonathan Bohren, Radu Bogdan Rusu, E. Gil Jones, Eitan Marder-Eppstein, Caroline Pantofaru, Melonee Wise, Lorenz Mosenlechner, Wim Meeussen, and Stefan Holzer. Towards autonomous robotic butlers: Lessons learned with the PR2. In *IEEE International Conference on Robotics and Automation*, pages 5568–5575. IEEE, May 2011.

[17] Z. I. Botev, J. F. Grotowski, and D. P. Kroese. Kernel density estimation via diffusion. *The Annals of Statistics*, 38(5):2916–2957, October 2010.

[18] A Brooks, T Kaupp, A Makarenko, S Williams, and A Oreback. Towards component-based robotics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 163–168, 2005.

[19] Lindsay Bruce. Mixture models: Theory, geometry, and applications. In *NSF-CBMS regional conference series in probability and statistics*, 1995.

[20] Davide Brugali and Patrizia Scandurra. Component-based robotic engineering (Part I). *IEEE Robotics & Automation Magazine*, 16(4):84–96, December 2009.

[21] Davide Brugali and Azamat Shakhimardanov. Component-Based Robotic Engineering (Part II). *IEEE Robotics & Automation Magazine*, 17(1):100–112, March 2010.

[22] Herman Bruyninckx. Open Robot Control Software, 2013.

[23] R. Canham, A.H. Jackson, and A. Tyrrell. Robot error detection using an artificial immune system. In *NASA/DoD Conference on Evolvable Hardware*, pages 199–207. IEEE Comput. Soc, 2003.

[24] J. Carlson and R.R. Murphy. Reliability analysis of mobile robots. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 274–281. IEEE, 2003.

[25] J. Carlson, R.R. Murphy, and A. Nelson. Follow-up analysis of mobile robot failures. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, volume 5, pages 4987–4994 Vol.5. IEEE, 2004.

[26] Marta Casar and José A.˜R. Fonollosa. Overcoming HMM time independence assumption using N-gram based modelling for continuous speech recognition. In *European Signal Processing Conference. EUSIPCO*, pages 3–7. EURASIP, August 2008.

[27] Sung-Hyuk Cha. Comprehensive Survey on Distance/Similarity Measures between Probability Density Functions. *International Journal of Mathematical Models and Methods in Applied Sciences*, 1:300–307, 2007.

[28] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly Detection: A Survey. *ACM Computing Surveys*, 4:1–58, 2009.

[29] Anders Lyhne Christensen. *Fault Detection in Autonomous Robots*. PhD thesis, Universit libre de Bruxelles, 2008.

[30] Anders Lyhne Christensen, Rehan O'Grady, Mauro Birattari, and Marco Dorigo. Fault detection in autonomous robots based on fault injection and learning. *Autonomous Robots*, 24(1):49–67, November 2007.

[31] R. Clark. Instrument Fault Detection. *IEEE Transactions on Aerospace and Electronic Systems*, 14(3):456–465, May 1978.

[32] R.N. Clark, D. Fosth, and V. Walton. Detecting Instrument Malfunctions in Control Systems. *IEEE Transactions on Aerospace and Electronic Systems*, AES-11(4):465–473, July 1975.

[33] D S Clouse, C L Giles, B G Horne, and G W Cottrell. Time-delay neural networks: representation and induction of finite-state machines. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 8(5):1065–70, January 1997.

[34] R. Dawkins, O. Holland, A. Winfield, P. Greenway, and A. Stephens. An interacting multi-robot system and smart environment for studying collective behaviours. In *8th International Conference on Advanced Robotics*, pages 537–542. IEEE, 1997.

[35] Nando de Freitas, Richard Dearden, Frank Hutter, Rube n Morales-Menendez, Jim Mutch, David Poole, N. Defreitas, and R. Morales-Menendez. Diagnosis by a Waiter and a Mars Explorer. *Proceedings of the IEEE*, 92(3):455–468, March 2004.

[36] Richard Dearden and Dan Clancy. Particle Filters for Real-Time Diagnosis of Planetary Rovers. In *Proceedings of the Thirteenth International Workshop on Principles of Diagnosis*, pages 1–8, Semmering, Austria, 2002.

[37] Richard Dearden, T. Willeke, F. Hunter, R. Simmons, V. Verma, and S. Thrun. Real-time fault detection and situational awareness for rovers: report on the mars technology program task. In *Proceedings of the IEEE Aerospace Conference*, volume 2, pages 826–840. IEEE, 2004.

[38] IFR Statistical Department. Professional service robots, 2012.

[39] Steven X Ding. *Model-based fault diagnosis techniques: Design Schemes, Algorithms, and Tools*. Springer, 2008.

[40] Xavier Domont, Martin Heckmann, Frank Joublin, and Christian Goerick. Hierarchical spectro-temporal features for robust speech recognition. In *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4417–4420. IEEE, March 2008.

[41] Ted Faison. *Event-Based Programming: Taking Events to the Limit*. Apress, Berkeley, CA, May 2006.

[42] Zohreh Fathi, W Fred Ramirez, and Jozef Korbicz. Analytical and knowledge-based redundancy for fault diagnosis in process plants. *AIChE Journal*, 39(1):42–56, 1993.

[43] RoboCup Federation. RoboCup, 2013.

[44] Paul Fitzpatrick, Giorgio Metta, and Lorenzo Natale. Towards long-lived robot genes. *Robotics and Autonomous Systems*, 56(1):29–45, 2008.

[45] S. Forrest, A.S. Perelson, L. Allen, and R. Cherukuri. Self-nonself discrimination in a computer. In *Proceedings of 1994 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 202–212. IEEE Comput. Soc. Press, 1994.

[46] B. Freyermuth. An approach to model based fault diagnosis of industrial robots. In *IEEE International Conference on Robotics and Automation*, pages 1350–1356. IEEE Comput. Soc. Press, 1991.

[47] Fraunhofer-Institut für Produktionstechnik und Automatisierung. The mobile robot assistant care-o-bot, 4 2013. http://www.care-o-bot.de/.

[48] Janos Galambos and Samuel Kotz. *Characterizations of probability distributions*, volume 2. Springer-verlag New York, 1978.

[49] Bill Gates. A Robot in Every Home. *Scientific American*, 296:58–65, January 2007.

[50] David Gilbert. The JFreeChart class library-developer guide. Website. http://www.jfree.org/jfreechart/.

[51] C. Goerick, H. Wersing, I. Mikhailova, and M. Dunn. Peripersonal space and object recognition for humanoids. In *5th IEEE-RAS International Conference on Humanoid Robots*, pages 387–392. IEEE, 2005.

[52] Joseph A Goguen. The logic of inexact concepts. *Synthese*, 19(3):325–373, 1969.

[53] Raphael Golombek, Sebastian Wrede, Marc Hanheide, and Martin Heckmann. Learning a Probabilistic Self-awareness Model for Robotic Systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Taipei, Taiwan, 2010.

[54] David Gouaillier, Vincent Hugel, Pierre Blazevic, Chris Kilner, Jerome Monceaux, Pascal Lafourcade, Brice Marnier, Julien Serre, and Bruno Maisonnier. Mechatronic design of NAO humanoid. In *2009 IEEE International Conference on Robotics and Automation*, pages 769–774. IEEE, May 2009.

[55] H-M Gross, C. Schroeter, S. Mueller, M. Volkhardt, E. Einhorn, A. Bley, C. Martin, T. Langner, and M. Merten. Progress in developing a socially assistive

mobile home robot companion for the elderly with mild cognitive impairment. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2430–2437. IEEE, September 2011.

[56] Object Management Group. Common Object Request Broker Architecture (CORBA) Specification, Version 3.3.

[57] F. Gustafsson, F. and Gustafsson. *Adaptive filtering and change detection*. Wiley Online Library, 5 edition, 2000.

[58] Robert Gwadera, Mikhail J Atallah, and Wojciech Szpankowski. Reliable Detection of Episodes in Event Sequences. In *Knowledge and Information Systems*, pages 67–74, 2004.

[59] Axel Haasch, Sascha Hohenner, Sonja Hüwel, Markus Kleinehagenbrock, Sebastian Lang, Ioannis Toptsis, Gernot A Fink, Jannik Fritsch, Britta Wrede, and Gerhard Sagerer. BIRON – The Bielefeld Robot Companion. In E Prassler, G Lawitzky, P Fiorini, and M Hägele, editors, *Proceedings International Workshop on Advances in Service Robotics*, pages 27–32, Stuttgart, Germany, May 2004. Fraunhofer IRB Verlag.

[60] A. Halme, J. Selkäinaho, and J. Soininen. Adaptive control with nonlinear filtering. *Automatica*, 21(4):453–463, July 1985.

[61] Marc Hanheide. A Cognitive Ego-Vision System for Interactive Assistance. Master's thesis, Technische Fakultät – Universität Bielefeld, 2006.

[62] Nick Hawes, Marc Hanheide, Kristoffer Sjöö, Alper Aydemir, Patric Jensfelt, Moritz Göbelbecker, Michael Brenner, Hendrik Zender, Pierre Lison, Ivana Kruijff-Korbayov, Geert-Jan M Kruijff, and Michael Zillich. Dora The Explorer: A Motivated Robot. In *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, 2010.

[63] Nick Hawes, Michael Zillich, and Jeremy Wyatt. BALT & CAST: Middleware for Cognitive Robotics. In *Proceedings of IEEE RO-MAN 2007*, pages 998–1003, August 2007.

[64] George T Heineman and William T Councill. *Component-based software engineering: putting the pieces together*, volume 17. Addison-Wesley USA, 2001.

[65] T. Höfling and R. Isermann. Fault detection based on adaptive parity equations and single-parameter tracking. *Control Engineering Practice*, 4(10):1361–1369, October 1996.

[66] Shunsuke Ihara. *Information theory for continuous systems*, volume 2. World Scientific Publishing Company Incorporated, 1993.

[67] R Isermann. Process fault detection based on modeling and estimation methods A survey. *Automatica*, 20(4):387–404, 1984.

[68] R Isermann. Trends in the application of model-based fault detection and diagnosis of technical processes. *Control Engineering Practice*, 5(5):709–719, May 1997.

[69] Rolf Isermann. *Identifikation dynamischer Systeme*. Springer-Verlag, 1988.

[70] Rolf Isermann. *Fault-diagnosis systems: an introduction from fault detection to fault tolerance*. Springer, 2005.

[71] Nathalie Japkowicz. *Concept-learning in the absence of counter-examples: an autoassociation-based approach to classification*. PhD thesis, Rutgers, The State University of New Jersey, 1999.

[72] Artur Pereira João Cunha, António J. R. Neves, José Luis Azevedo, Bernardo Cunha, Nuno Lau. A mobile robotic platform for elderly care. In *BIOSTEC*, 2011.

[73] Harold Lee Jones. *Failure detection in linear systems*. PhD thesis, 1973.

[74] K-Team. Khepera. Webpage, 2013. http://www.k-team.com.

[75] Rudolph Emil Kalman et al. A new approach to linear filtering and prediction problems. *Transaction of the American Society for Mechanical Engineering, Series D, Journal of basic Engineering*, 82(1):35–45, 1960.

[76] ShehrozS. Khan and MichaelG. Madden. A Survey of Recent Trends in One Class Classification. In Lorcan Coyle and Jill Freyne, editors, *Artificial Intelligence and Cognitive Science*, volume 6206 of *Lecture Notes in Computer Science*, pages 188–197. Springer Berlin Heidelberg, 2010.

[77] J.-H. Kim, S. Sam Ge, P. Vadakkepat, Norbert Jesse, Ulrich Al Mamun, A. Puthusserypady, S. Rückert, Joaquin Sitte, Ulf Witkowski, R. Nakatsu, Th. Braunl, J. Baltes, J. Anderson, C.-C. Wong, I. Verner, and D. Ahlgren. Progress in Robotics. In *Proceedings of the FIRA RoboWorld Congress*, 2009.

[78] David Kortenkamp and Reid Simmons. *Springer Handbook of Robotics*, chapter Robotic Sy, pages 187–206. Springer, 2008.

[79] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *In ICML*, 2007.

[80] RoboCup League's committees. RoboCup@Home Rules & Regulations 2009. Technical report, RoboCup, 2009.

[81] RoboCup League's committees. RoboCup@Home Rules & Regulations 2010. Technical report, RoboCup, 2010.

[82] Manja Lohse. Investigating the influence of situations and expectations on user behavior - empirical analyses in human-robot interaction. Master's thesis, Bielefeld University, 2010.

[83] Jianhui Luo, Madhavi Namburu, Krishna R Pattipati, Liu Qiao, and Shunsuke Chigusa. Integrated Model-Based and Data-Driven Diagnosis of Automotive

Antilock Braking Systems. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 40(2):321–336, March 2010.

[84] Ingo Lütkebohle, Jan Schaefer, and Sebastian Wrede. Facilitating Re-Use by Design: A Filtering, Transformation, and Selection Architecture for Robotic Software Systems. In *ICRA Workshop on Software Development in Robotics SDIR*, number section III, 2009.

[85] Jason Maassen, Rob Van Nieuwpoort, Ronald Veldema, Henri Bal, Thilo Kielmann, Ceriel Jacobs, and Rutger Hofman. Efficient Java RMI for parallel programming. *ACM Transactions on Programming Languages and Systems*, 23(6):747–775, November 2001.

[86] Nader Mohamed, Jameela Al-Jaroodi, and Imad Jawhar. Middleware for Robotics: A Survey. In *2008 IEEE Conference on Robotics, Automation and Mechatronics*, pages 736–742. IEEE, September 2008.

[87] F. Mondada, L.M. Gambardella, D. Floreano, S. Nolfi, J. Deneubourg, and M. Dorigo. The cooperation of swarm-bots - Physical interactions in collective robotics. *IEEE Robotics & Automation Magazine*, 12(2):21–28, June 2005.

[88] M. A. Moni and A. B. M. Shawkat Ali. HMM based hand gesture recognition: A review on techniques and approaches. In *2009 2nd IEEE International Conference on Computer Science and Information Technology*, pages 433–437. IEEE, 2009.

[89] Chen Mou-Yen. Off-line handwritten word recognition using a hidden Markov model type stochastic network. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):481–496, May 1994.

[90] Thomas J Mowbray and William Ruh. *Inside CORBA: distributed object standards and applications*. Addison-Wesley Longman Publishing Co., Inc., 1998.

[91] Gero Mühl, Ludger Fiege, and Peter Pietzuch. *Distributed Event-Based Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[92] Maurice Naftalin and Philip Wadler. *Java generics and collections*. O'Reilly Media, Incorporated, 2006.

[93] Sriram Narasimhan and Lee Brownston. HyDE - A General Framework for Stochastic and Hybrid Model-based Diagnosis. In *Proceedings of the 18th International Workshop on Principles of Diagnosis*, 2007.

[94] Tim Niemueller, Alexander Ferrein, Daniel Beck, and Gerhard Lakemeyer. Design Principles of the Component-Based Robot Software Framework Fawkes. In *Simulation, Modeling, and Programming for Autonomous Robots*, 2010.

[95] Tim Niemueller, Gerhard Lakemeyer, and Siddhartha S. Srinivasa. A generic robot database and its application in fault analysis and performance evaluation. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 364–369. IEEE, October 2012.

[96] Simon Nolet, Edmund Kong, and David W. Miller. Autonomous docking algorithm development and experimentation using the SPHERES test-bed. In Jr. Tchoryk and Melissa Wright, editors, *Defense and Security*, pages 1–15, August 2004.

[97] A. Nordmann and S. Wrede. A Domain-Specific Language for Rich Motor Skill Architectures. In *3rd International Workshop on Domain-Specific Languages and models for Robotic systems*, Tsukuba, 2012.

[98] Mattias Nyberg, Erik Frisk, and Mattias Krysander. A Data-Driven and Probabilistic Approach to Residual Evaluation for Fault Diagnosis. In *50th IEEE Conference on Decision and Control*, Orlando, Florida, USA, 2011.

[99] Oracle. Java, 2013.

[100] Nikunj C. Oza, Kagan Turner, Irem Y. Turner, and Edward M. Huff. Classification of aircraft maneuvers for fault detection. *Lecture notes in computer science*, 2709:375–384, 2003.

[101] H. Park, R. Mackey, M. James, M. Zak, M. Kynard, J. Sebghati, and W. Greene. Analysis of Space Shuttle main engine data using beacon-based exception analysis for multi-missions. In *Proceedings, IEEE Aerospace Conference*, volume 6, pages 6–2835–6–2844. IEEE, 2002.

[102] F PREVIDI and T PARISINI. Model-free actuator fault detection using a spectral estimation approach: the case of the DAMADICS benchmark problem. *Control Engineering Practice*, 14(6):635–644, June 2006.

[103] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully B Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: an open-source Robot Operating System. In *Proc. of the Int. Conf. on Robotics and Automation*, Open-Source Software workshop, 2009.

[104] L. Rabiner and B. Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, 3(1):4–16, January 1986.

[105] L R Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Readings in speech recognition*, 53(3):267–296, 1990.

[106] L.R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[107] M Rosenblatt. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, 27(3):832–837, 1956.

[108] M. Saif and Y. Guan. A new approach to robust fault detection and identification. *IEEE Transactions on Aerospace and Electronic Systems*, 29(3):685–695, July 1993.

[109] Soumik Sarkar, Xin Jin, and Asok Ray. Data-Driven Fault Detection in Aircraft Engines With Noisy Sensor Measurements. *Journal of Engineering for Gas Turbines and Power*, 133(8):081602, 2011.

[110] C. Schlegel and R. Worz. The software framework SMARTSOFT for implementing sensorimotor systems. In *Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 1610–1616. IEEE, 1999.

[111] Dietmar Schreiner. *Component Based Communication Middleware for AUTOSAR*. PhD thesis, 2009.

[112] Mark Schwabacher. Machine learning for rocket propulsion health monitoring. *SAE transactions*, 114(1):1192–1197.

[113] J. Selkainaho and A. Halme. An adaptive filter based fault detector applied to dynamic positioning system. In *1986 25th IEEE Conference on Decision and Control*, pages 1598–1602. IEEE, 1986.

[114] J. Siegel. *CORBA 3 fundamentals and programming*. John Wiley & Sons, 2000.

[115] T. L. Casavant Singhal and M. Scale in Distributed Systems. In *In Readings in Distributed Computing Systems*. IEEE Computer Society, 1994.

[116] E.N. Skoundrianos and S.G. Tzafestas. Fault diagnosis on the wheels of a mobile robot using local model neural networks. *IEEE Robotics & Automation Magazine*, 11(3):83–90, September 2004.

[117] G Steinbauer, M Mörth, and F Wotawa. Real-time diagnosis and repair of faults of robot control software. *Lecture Notes in Computer Science*, 4020:13, 2006.

[118] U M Stocker and Karl-Heinz Waldmann. Stochastische Modelle: Eine anwendungsorientierte Einführung, 2003.

[119] Jorg Stuckler and Sven Behnke. Following human guidance to cooperatively carry a large object. In *2011 11th IEEE-RAS International Conference on Humanoid Robots*, pages 218–223. IEEE, October 2011.

[120] S.Y.H. Su and E. DuCasse. A Hardware Redundancy Reconfiguration Scheme for Tolerating Multiple Module Failures. *IEEE Transactions on Computers*, C-29(3):254–258, March 1980.

[121] Clemens Szyperski, Dominik Gruntz, and Stephan Murer. *Component software: beyond object-oriented programming*. Addison-Wesley, 2002.

[122] Howard M Taylor and Samuel Karlin. *An introduction to stochastic modeling*, volume 3. Academic Press New York, 1984.

[123] S. Thrun. Simultaneous Localization and Mapping with Sparse Extended Information Filters. *The International Journal of Robotics Research*, 23(7-8):693–716, August 2004.

[124] Mikaelian Tsoline and Williams Brian C. Model-based monitoring and diagnosis of systems with software-extended behavior. In *Proceedings of the national conference on artificial intelligence*, 2005.

[125] H. Utz, S. Sablatnog, S. Enderle, and G. Kraetzschmar. Miro - middleware for mobile robot applications. *IEEE Transactions on Robotics and Automation*, 18(4):493–497, August 2002.

[126] Jeremiah Via. A data-driven self-awareness model for robotics systems. Bachelor thesis, University of Birmingham, United Kingdom, 2012.

[127] Christopher M Vigorito and Andrew G Barto. Intrinsically Motivated Hierarchical Skill Learning in Structured Environments. *IEEE Transactions on Autonomous Mental Development*, 2(2):132–143, June 2010.

[128] Paul Viola and Michael J. Jones. Robust Real-Time Face Detection. *International Journal of Computer Vision*, 57(2):137–154, May 2004.

[129] N Viswanadham and R Srichander. Fault detection using unknown input observers. *Control–Theory and Advanced Technology*, 3(2):91–101, 1987.

[130] W3C. Extensible Markup Language (XML), 2013.

[131] Sven Wachsmuth, Frederic Siepmann, L. Ziegler, F. Lier, and Matthias Schöpfer. ToBI-Team of Bielefeld: The Human-Robot Interaction System for RoboCup@HOME 2012. Technical report, Applied Informatics Group at Bielefeld University, Mexico City, Mexico, 2012.

[132] Johannes Wienke and Sebastian Wrede. A middleware for collaborative research in experimental robotics. In *2011 IEEE/SICE International Symposium on System Integration (SII)*, pages 1183–1190. IEEE, December 2011.

[133] Brian C. Williams, Seung Chung, and Vineet Gupta. Mode estimation of model-based programs: monitoring systems with complex behavior. In *Proceedings of IJCAI-01*, pages 579–585, August 2001.

[134] Thomas Wisspeintner, Tijn Van Der Zant, Luca Iocchi, and Stefan Schiffer. RoboCup@Home: Scientific Competition and Benchmarking for Domestic Service Robots. *Interaction Studies*, 10(3):392–426, 2009.

[135] B. Wrede, M. Kleinehagenbrock, and J. Fritsch. Towards an Integrated Robotic System for Interactive Learning in a Social Context. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1817–1823. IEEE, October 2006.

[136] S. Wrede, J. Fritsch, C. Bauckhage, and G. Sagerer. An XML based framework for cognitive vision architectures. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 1, pages 757–760 Vol.1. IEEE, 2004.

[137] Sebastian Wrede. *An Information-Driven Architecture for Cognitive Systems Research*. PhD thesis, Technical Faculty – Bielefeld University, 2009.

[138] N Ye. A markov chain model of temporal behavior for anomaly detection. In *Proceedings of the 2000 IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop*, pages 171–174, 2000.

[139] Y.C. Yeh. Triple-triple redundant 777 primary flight computer. In *1996 IEEE Aerospace Applications Conference. Proceedings*, volume 1, pages 293–307. IEEE.