**Master Thesis**

# Remote Fault Diagnosis and Health Monitoring of Robots Using a Robotic Black Box

*Author:*
Youssef Mahmoud Youssef
youssef.youssef@smail.inf.h-brs.de
Matrikel Nr: 9024421

*Supervisors:*
Prof. Dr. Paul Plöger
paul.ploeger@h-brs.de
Prof. Dr. Sascha Alda
sascha.alda@h-brs.de
Dipl.-Inform. Daniel Ota
daniel.ota@fkie.fraunhofer.de

*A report submitted in fulfilment of the requirements for the degree of Master of Autonomous Systems*

*in*

Hochschule Bonn-Rhein-Sieg
Department of Computer Science

Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Fraunhofer
FKIE

April 2018

# Declaration of Authorship

I, Youssef Mahmoud, declare that this master thesis project report titled, 'Remote Fault Diagnosis and Health Monitoring of Robots Using a Robotic Black Box' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a master degree at this University.
- Where any part of this report has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this report is entirely my own work.
- I have acknowledged all main sources of help.
- Where the report is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

*"Fear's a powerful thing. I mean it's got a lot of firepower. If you can figure out a way to wrestle that fear to push you from behind rather than stand in front of you, that's very powerful"*

- Jimmy Iovine

Hochschule Bonn-Rhein-Sieg

# *Abstract*

Department of Computer Science

Master Thesis

**Remote Fault Diagnosis and Health Monitoring of Robots Using a Robotic Black Box**

by Youssef Mahmoud Youssef

With the current increasing complexity of tasks that robots have to execute, the occurrence of faults is inevitable. Faults can occur in different components throughout the robotic system ranging from sensors, actuators, hardware or software components. Therefore, it is crucial for robot designers to consider health monitoring and fault diagnosis of these different components. Usually, faults are handled by adding redundant hardware components, in order for them to operate in case one of the components fails, or, by modelling specific components or behaviours mathematically or statistically and comparing them with the actual system. Adding redundant components is usually expensive and sometimes not feasible. Moreover, modelling of complex components is an exhaustive procedure, and in many cases, the models are not accurate enough to mimic the behaviour of the component. Therefore, a generalised method for fault diagnosis and health monitoring that takes into consideration different components can be helpful for robot designers. Another aspect of health monitoring and fault diagnosis of robots, is the standardisation of messages.

Bearing in mind the range of different components available to robots, it would be helpful to have a standard method of representing the health and faults of these components to aid the user in having a comprehensive overview of the status of the robot. Looking into NATO Generic Vehicle Architecture (NGVA), where different standards are developed and deployed for vehicle components in order to achieve quicker upgrades and interoperability, a standard for usage and condition monitoring systems (UCMS) was developed but not deployed yet.

In this study, a system architecture for a non-intrusive, health monitoring and fault diagnosis system is designed and implemented on a mini-PC (black box). The different fault diagnosis methods available are reviewed and analysed. A comparison between a hybrid fault diagnosis method using a data-driven and model-based approach, and consistency-based diagnosis is made. Consistency-based diagnosis is selected to be evaluated within the implemented system. The output of the diagnosis engine is mapped

to the UCMS data model module for standardization and transmission to a remote-PC using Data Distribution Services (DDS). A complete diagnosis system has been deployed on a Raspberry Pi 3, using only 30% of its computational powers. Experiments were conducted on a ROS-based robot, and hardware and software components are modelled for consistency-based diagnosis. Faults are injected into the system and the results of the diagnosis engine are recorded. Consistency-based diagnosis showed an accuracy of 88%, and improvements are proposed on how to improve the existing method for higher accuracy.

# *Acknowledgements*

I would first like to thank my thesis advisor Prof. Paul G. Plöger for all his encouragement, guidance and motivation throughout the masters programme. His door was always open for me whenever I faced any problems or needed any guidance during the course of my studies at Hochschule Bonn-Rhein-Sieg. I am very grateful for the time he invested in me, and for that, I cannot thank him enough.

I would also like to thank my second advisor Dipl.-Inform. Daniel Ota at the Fraunhofer Research Institute FKIE for giving me his time and guidance throughout my thesis period. His comments and remarks regarding my work have been very helpful, and I am thankful for everything I learnt from him. Also, I would like to thank the NGVA team at FKIE, for providing me with all the assistance I needed during my thesis.

I would also like to thank my close friends for being there for me through it all, for believing in me and for giving me constant motivation throughout my studies to succeed. Without them I would not have been able to get here. Thank you.

I also would like to thank my beautiful wife Kay for supporting and believing in me. She has always been there for me and has provided me with all the motivation I needed to get to where I am today. Thank you for everything.

I would like to thank my father, whom without him, I would have not been able to make my academic journey in Germany possible in the first place. Thank you for always being supportive of me and my goals.

Finally, I would like to thank my mother, who passed away courageously battling cancer two years ago. She is the reason for everything good in me. I owe her everything, and without her motivation, even in her hardest times, I would not have come as far as I have. Thank you mum.

# Contents

# Chapter 1

# Introduction

In the fast developing world of today, robots are being designed and manufactured to aid humans in tasks which are repetitive, dangerous or tedious. One of the main fields of development for robotics is Unmanned Ground Vehicles (UGVs). UGVs are fast growing and are of high interest to both the commercial and defence industries because of their practicality in different use-cases. Scenarios such as search and rescue, explosive ordnance disposal, mine clearing, or perimeter monitoring are attractive for using technologies such as UGVs. Some of the reasons why this technology is continuing to grow is because of the increasing availability of low-cost sensors and microcontrollers, increased labour cost due to globalization, change of the acceptability of human causalities and loss of life and many more reasons [1]. Recent studies have shown that the UGV market is projected to grow from USD 1.49 Billion in 2016 to USD 2.63 Billion by 2021 [1].

An interesting aspect of the UGV market survey is that based on mode of operation, teleoperated robots lead the market among tethered, semi-autonomous, and autonomous robots. This is because of the unreliability of autonomous robots, and the nature of the dynamic environments that humans want to deploy robots in. Unreliability is an aspect of being autonomous; since there is no human intervention in truly autonomous robots, the internal communications and decision-making processes of the robot are not known to the user. Due to these reasons and the unreliability which comes from the robot being autonomous, one can expect faults to occur frequently.

A fault is something that changes the behaviour of any given system such that the system can no longer satisfy its purpose [2]. Therefore, it is crucial for robot designers and developers to consider faults during their design process, either in building systems that are fault tolerant, or in creating systems that have fault diagnosis capabilities. Fault

[1] https://www.marketsandmarkets.com/Market-Reports/unmanned-ground-vehicles-market-72041795.html

tolerant systems are systems that are designed to continue operation in the presence of faults and execute their tasks successfully to some degree. Fault tolerant systems usually include redundant components which can operate once a fault occurs. An example of a fault tolerant system is an aircraft where multiple sensors, actuators and landing gear tires are installed in order to assure the aircraft's operation in the existence of faults. In the robotic domain, fault tolerance can be achieved by adding redundant sensors, actuators, or even redundant software components that make sure some computations are always executed. Fault tolerance is usually required in systems which are safety-critical or systems that require low downtime. Moreover, adding redundant components increases the total cost of the system, or adds greater computational complexity to the system.

Fault diagnosis includes fault detection, isolation and identification. Fault diagnosis is the process of determining the kind, size and time of occurrence of a fault. Examples of systems that include fault diagnosis capabilities are vehicles. Vehicles nowadays usually have a central computing system which collects signals and data from different components of the vehicle such as engine oil and fuel level, or indicator light signal health and so on in order to notify the driver when something is wrong. Adding fault diagnosis capabilities to a system usually requires the addition of some computing system in order to collect the signals and analyse them, or have components that include self-monitoring capabilities, that can notify the driver or technician about the faults that have occurred.

Another aspect that comes to mind regarding designing systems that are fault tolerant or have fault diagnosis capabilities, is the tracking of the status of components, and storage of important information in order for the users or experts to be able to assess and analyse the status of a system over time when required. One of the most notable systems that do so are aircrafts. Aircrafts have dedicated data recorders, famously known as black boxes, which store important information and data logs of multiple signals coming from different components. Over the years, flight data recorders (FDRs) have proven to be crucial, in analysing the status of aircrafts after accidents or faults have occurred.

Flight data recorders were introduced in 1958 and were mainly used for large aircrafts only. In the early years of FDRs, a low number of parameters were only logged [2]. As the years progressed and unfortunately, aircraft accidents occurred, the flight data recorder became a crucial part in understanding the behaviour of the aircraft during the accident and helped experts during investigations pin point the exact reasons behind failures.

In [3], an argument is made on the importance of keeping a data record of a robot's behaviour in the near future, in a form similar to flight data recorders. Since more robots and autonomous vehicles are being deployed to the market, one can also expect accidents

[2]https://www.skybrary.aero/index.php/Flight_Data_Recorder_(FDR)

to occur. Therefore, health monitoring and fault diagnosis would be fundamental in understanding a robot's behaviour.

Data representation and standardisation is also an issue that needs to be considered while developing robotic systems. Since robotic platforms nowadays are highly customizable, and a range of different components are available to a developer, one can expect such complex systems to exchange data in both high volume and rate. It therefore makes sense to have a generalized method to represent the health of different components in a way that is thorough enough to give the user or developer an overview of the robot's health, and be able to cope with the different components running on the robot.

Bearing in mind the previous mentioned points, developing robust fault diagnosis systems for autonomous robots can be split up into three main branches; namely:

- Appropriate fault diagnosis methods
- Efficient data logging methods
- A thorough representation of the health and faults.

## 1.1 Goals

The goals of this study is to develop a fault diagnosis system that can do the following:

- **Non-intrusively monitor a robot's health.**

  In order not to burden the robot's computational capabilities with more tasks, the addition of an external mini-PC (black box) with limited computational capabilities, dedicated for fault diagnosis and health monitoring is proposed.

- **Detect and diagnose faults of the monitored robot.**

  In order to detect and diagnose faults, while bearing in mind the limited computational capabilities of the black box, an overview of fault diagnosis methods should be considered and the appropriate methods should be selected and evaluated.

- **Safely log robot's health data.**

  To store health data of the robotic platform overtime, which can come in very handy for developers and users to analyse their robot's components over time, and in case of total failures of robots, a log of the robot's behaviour would be present, and will facilitate the understanding of why the failure has occurred. Therefore, adding a storage device that would be connected to the mini-PC, independent of

the robotic platform, would be helpful. For this, efficient data logging methods should be deployed.

- **Exchange standardized messages with a remote PC regarding the robot's health.**

  To give the user or developer a thorough overview of their platforms health status, since in many cases the robots have wireless capabilities, an appropriate method of communication should be selected, preferably independent from the robot's operating system, in order to add more robustness.

## 1.2 Use Cases & Scenarios

To give the reader a better understanding of the applications of such a project, some use cases and scenarios are described.

### 1.2.1 Use Case 1 - Military vehicles deploying robots for area exploration

The first use case is a military use case where a military vehicle is in an unexplored region and the crew wants to send out robots to explore, seeing as, progressing forward with the vehicle might cause harm to the soldiers. While the robots are being controlled by the soldiers or are exploring autonomously, information on the robot's health status would be crucial for the crew, to give them an overview of how the robot is operating, and if faults occurs, give them details on what component was responsible for it. In this case, having a fault diagnosis system which is independent from the robot operating system would add robustness to the robot in the sense that the crew would be more informed on what the current status of the robot is and how to commence with the mission at hand.

### 1.2.2 Use Case 2 - Robots assisting in hospital logistics

The second use case would be in hospitals, where multiple robots would aid doctors, nurses and patients in, for example, moving beds, lifting heavy objects, and many more cases. Managing a fleet of robots with regards to maintenance and reliability would be one of the main pillars of keeping the robot's operation for as long as possible with lowest down times. Therefore, if a black box is deployed on each robot in order to monitor the robot's condition, it would deem helpful. This would also reduce the cost of sending in an expert to solve the robot's failures, as the data can be transmitted through the internet

to the manufacturer's company, and repair actions can be suggested to non-experts, if the faults and failures are simple to fix.

### 1.2.3 Use Case 3 - Robots deployed for search & rescue in areas affected by disasters

From the literature, [4], UGVs have already been deployed to many areas where disasters occurred, and have been essential in exploration and aiding humans in areas where it would be too dangerous for a human to face. However, Murphy also mentioned the many failures and faults that occur during the robot's operation, and in many scenarios, the robots stop being used because of reliability issues. Therefore, deploying a black box that is dedicated for health monitoring and fault diagnosis in search and rescue missions in disastrous areas would be crucial in informing the users of the robot's health status.

## 1.3 Problem Statement

From the presented arguments, goals, and different use cases for this study, it can be deduced that health monitoring and fault diagnosis of robots is crucial for their success in missions and in increasing their autonomy.

The following questions present the problems that need to be addressed in order to create a health monitoring and fault diagnosis system for robotic platforms:

- Which fault diagnosis methods should be used to monitor the health of a wide range of different components that are deployed on a robotic platform?
- How well do these fault diagnosis methods perform under the constraints of the goals of this project?
- What is the system architecture that can help realize the goals of this project?
- What are the existing standards for health representation and standardization that can be applicable to robotic platforms?
- Can a complete fault diagnosis system be deployed on a mini-PC?

The rest of this study will focus on answering the given problems, and present solutions through implementations and evaluations.

## 1.4 Contributions

The contributions of this study can be summarized as follows:

- A generalized architecture for non-intrusive fault diagnosis and health monitoring of UGVs.
- An evaluation of different fault diagnosis methods suitable to be deployed to the designed architecture, taking into consideration the limitations of the black box.
- Implementation of a complete fault diagnosis system on a mini-PC.
- Evaluation of the performance of consistency-based diagnosis in the implemented system.
- A mapping and implementation of a health standard.
- Publishing of health data on the network to be available to the remote-PC.

## 1.5 Report Outline

The report is outlined as follows; in chapter. 2, the related work is presented with regards to common faults in robotics, fault diagnosis methods, and data standardization and communication with remote-PC. In chapter. 3, an Attribute-Driven-Design approach is used to develop the system architecture, and an insight of the different components within the system is given. Chapter. 4 gives a detailed analysis of the selected fault diagnosis methods that were chosen from the related work chapter. Next, the Usage and Condition Monitoring System (UCMS) standard, which is used for standardizing the messages produced by the fault diagnosis methods is reviewed in chapter. 5. The implementation of the designed system, and the experimental evaluation along with the results of the experiments are presented in chapter. 6. Finally, the conclusion, limitations and future work proposed for this project are discussed in chapter. 7.

# Chapter 2

# Related Work

In this chapter, an overview of the related work to this project will be discussed. The goals of this study as discussed in chapter 1, is to develop a non-intrusive fault diagnosis and health monitoring system for UGVs. This chapter will focus on discussing work related to common faults in UGVs, fault diagnosis methods, existing standards for health data standardization, and finally, discuss some of the existing implementations, related to this project.

In many of the works regarding fault diagnosis, the authors use similar terms but with different meanings. For example fault diagnosis in some work is referred to including fault isolation & identification only, and in other work is referred to including fault detection, isolation & identification. Therefore, it would be helpful to have some terms explained in order to make the report understandable. The following definitions will be used throughout this report and are derived from [2] and [5].

- Fault: “An unpermitted deviation of at least one characteristic of the system" [5].
- Failure: It is a break in the continuity of a system’s ability to perform a required function.
- Error: A state within the system that can lead to a failure.
- Fault Detection: Determining the time at which a fault occurred in the system.
- Fault Diagnosis: It is the determination of kind, size, location and time of detection of a fault. Fault diagnosis includes fault detection, isolation & identification.
- Fault Isolation: To determine in which component the fault occurred.
- Fault Identification: To identify the fault, estimate its magnitude and determine the severity of the fault.

## 2.1 Common faults in UGVs

Since the aim of this project is to develop a non-intrusive health monitoring and fault diagnosis for UGVs that can take into consideration a wide range of hardware and software components to monitor, an overview of common faults that affect UGVs would be helpful in later selecting the appropriate method for fault diagnosis.

Two main studies will be discussed in this section, by namely; G. Steinbauer [6] and Murphy [4]. Both studies give interesting insight into the frequent and common faults that occur to robots during operation.

In [4], the authors present a study focusing particularly on how UGVs fail in the field, with respect to two field applications, Urban Search & Rescue (USAR) and military. The authors collect information about failures from a number of studies and represent the results in a taxonomy which categorises failures based on their cause; either physical or human related failures.

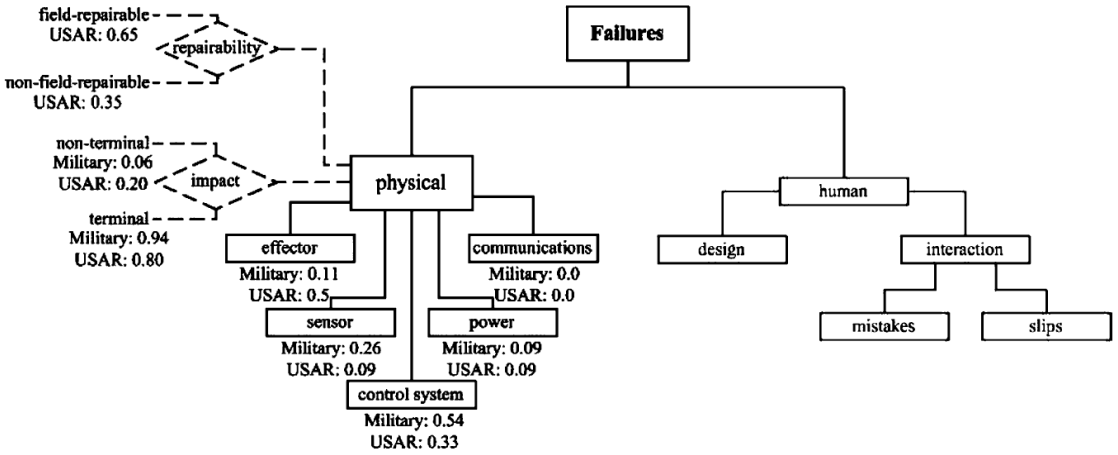



FIGURE 2.1: Classification of faults including probabilities of failures for military and USAR applications [4].

In Fig. 2.1, the taxonomy presented from [4] is shown. Physical failures are divided based on common subsystems usually found in robots. Human related failures are divided into design or interaction faults. Design related faults arise from, for example, defects in design or post production-modifications. Interaction faults are further divided into mistakes or slips. Mistakes occur when users deliberately perform an action that leads to a fault, and slips occur when users try to perform the correct action but, a fault still occurs.

The authors also state that physical failures have two attributes, namely, impact and repairability. The impact attribute states the degree the physical fault affects the success of the mission. The repairability attribute states the degree of repairability of physical faults when they occur.

To evaluate the results from the gathered studies about the physical failures, the authors use some metrics such as:

- Mean time between failure (MTBF), which is the frequency at which a failure occurs.
- Mean time to repair.
- Availability.
- Average downtime, which is the time it takes to repair a component or subsystem after a fault has occurred.

From the gathered studies and the applied metrics for evaluation, the authors present the probability of each fault occurring with regards to each category presented in Fig. 2.1. It can be noted that with regards to the military applications, control system faults and sensor faults are the most common faults, and with regards to impact, it is found that physical failures usually end in the failure of the overall mission with an impact probability of 0.96. As for USAR applications, effector and control system faults are the most common faults, and similar to military applications, physical faults have a high impact rate on the failure of the mission with a probability of 0.8. The authors also find that repairability of physical failures with regards to USAR applications has a moderate probability of 0.65.

In [6], the authors present the result of a survey which focuses on faults of robots that are used in RoboCup [1]. RoboCup aims to promote artificial intelligence to the robotics field by promoting relative challenges. The study focuses on gathering information on faults from different robotic platforms and use the taxonomy seen in Fig. 2.2.

The taxonomy in Fig. 2.2 divides faults in robotic platforms into four main categories, which are hardware, software, algorithms, and interaction faults. Hardware faults are faults that are caused by physical components deployed on the robots. Software faults can be caused by design and implementation of different algorithms on the robot. Algorithm faults may imply faults that are rooted in the theory of the algorithm itself. Finally, interaction faults can be caused by the nature of the dynamic environment the robots are deployed in or from human interaction.

Some of the results of the survey are discussed here, which are relevant to our study. In table. 2.1, different platform components are gathered and ranked based on the occurrence of faults.

[1] http://www.robocup.org

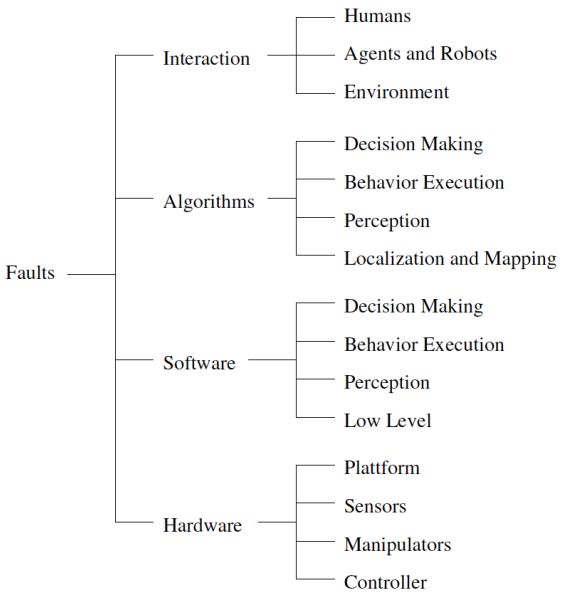



FIGURE 2.2: Fault taxonomy used for the survey. [6].

| Platform Part | Ranking (0 .. not affected. 5 .. much affected) | | | | | | |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | avg. |
| | % | | | | | | |
| Batteries | 29,4 | 11,8 | 41,2 | 5,9 | 11,8 | 0 | 1,59 |
| Motor Drivers | 33,3 | 20,0 | 26,7 | 6,7 | 13,3 | 0 | 1,47 |
| Controller Boards | 29,4 | 41,2 | 11,8 | 5,9 | 11,8 | 0 | 1,29 |
| Tracks | 62,5 | 0 | 12,5 | 12,5 | 0 | 12,5 | 1,25 |
| Motors | 46,7 | 33,3 | 0 | 0 | 20,0 | 0 | 1,13 |
| Gears | 66,7 | 6,7 | 13,3 | 0 | 13,3 | 0 | 0,87 |
| Wheels | 68,8 | 6,3 | 12,5 | 6,3 | 6,3 | 0 | 0,75 |
| Flipper | 75,0 | 12,5 | 0 | 0 | 0 | 12,5 | 0,75 |
| Chassis | 75,0 | 18,8 | 0 | 0 | 0 | 6,3 | 0,5 |

TABLE 2.1: Platform component faults as shown in [6].

| Causes | Ranking (0 .. not relevant. 5 .. much relevant) | | | | | | |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | avg. |
| | % | | | | | | |
| Connectors | 11,8 | 5,9 | 23,5 | 17,6 | 23,5 | 17,6 | 2,88 |
| Communication | 17,6 | 17,6 | 23,5 | 17,6 | 11,8 | 11,8 | 2,24 |
| Physical Impact | 31,5 | 12,5 | 25,0 | 12,5 | 6,3 | 12,5 | 1,88 |
| Wear | 35,3 | 5,9 | 29,4 | 17,6 | 5,9 | 5,9 | 1,71 |
| Vibration | 29,4 | 17,6 | 23,5 | 17,6 | 11,8 | 0 | 1,65 |
| Damage | 23,5 | 41,2 | 11,8 | 11,8 | 5,9 | 5,9 | 1,53 |
| Configuration | 35,3 | 23,5 | 23,5 | 5,9 | 5,9 | 5,9 | 1,41 |
| Temperature (Overheat) | 52,9 | 5,9 | 11,8 | 11,8 | 11,8 | 5,9 | 1,41 |
| Short Circuits | 37,5 | 25,0 | 18,8 | 12,5 | 0 | 6,3 | 1,31 |
| Environmental Conditions | 76,5 | 5,9 | 11,8 | 5,9 | 0 | 0 | 0,47 |

TABLE 2.2: Causes of platform component faults as shown in [6].

From table. 2.1, we can see that batteries and motor drivers deployed on robotic platforms are moderately affected. From table. 2.2, it is found that the main causes that create faults for platform components are connector problems and communication problems. From the survey gathered, it was also found that connector faults affect mostly batteries, controller boards, and motor drivers. The authors claim that battery, motors and controller boards highly affect the success rate of the robot's missions and frequently lead to termination of the mission.

Tables. 2.3 & 2.4 present the ranking of sensor faults on robotic platforms and the main causes that attribute to these faults. It is found from the study that laser range finders (LRF) and cameras are the most faulty components and that IMUs and Odometry have higher reliabilities. The most common causes for these failures were found to be either connector, configuration or communication problems. The authors also state the

| Sensor | Average Score |
|---|---|
| Hokuyu LRF | 1,78 |
| Swiss Rangers | 1,25 |
| Directed Cameras | 1,23 |
| Sonars | 1,2 |
| IMU | 0,6 |
| Odometry | 0,65 |

TABLE 2.3: Sensor faults average score [6].

| Cause | Average Score |
|---|---|
| Connectors | 2,73 |
| Configuration | 2,54 |
| Communication | 2,5 |

TABLE 2.4: Causes of sensor faults [6].

symptoms that arise from these sensor failures or faults which are mostly either that the sensor does not produce data or the sensor produces incorrect data.

As for manipulator faults, the authors find from the survey that the most common faults that occur are from either collisions or damaged components on the manipulator. For controllers, such as the computers running the robots, the authors state that the most common causes for faults are also either connector or configuration problems.

The rest of the survey covers information on software and algorithm faults where it was found that the most faulty softwares are softwares related to computer vision, behaviour execution and inter-robot communications. The causes of these faults are mostly related to configuration faults or performance leaks, where the software does not execute its objective within the required time.

From the common faults discussed in this section, it can be seen that faults that occur in robotic platforms can be caused from varying components deployed throughout the platform and can highly affect the success of the missions that are intended for the robots. Taking these faults into consideration, the selection of a fault diagnosis method is then crucial for any robotic platform.

## 2.2 Fault Detection & Diagnosis

From the different components that faults can occur from, and from the wide variety of robotic platforms, a suitable fault diagnosis method should be selected that can incorporate, if possible, a wide variety of components. For this, a study of different fault diagnosis methods has been conducted to select a suitable method.

Different studies have already been made that give an overview of the different methods of fault diagnosis [7], [8], [9]. In this section, a summary of the different fault diagnosis methods will be given and some of their applications with regards to robotics will be discussed.

Fault diagnosis methods can be split into four branches, namely; model-based, signal-based, knowledge-based, and hybrid. Fig. 2.3 shows a taxonomy of the fault diagnosis methods excluding the hybrid branch, since the hybrid methods are a combination of any of the mentioned methods together.

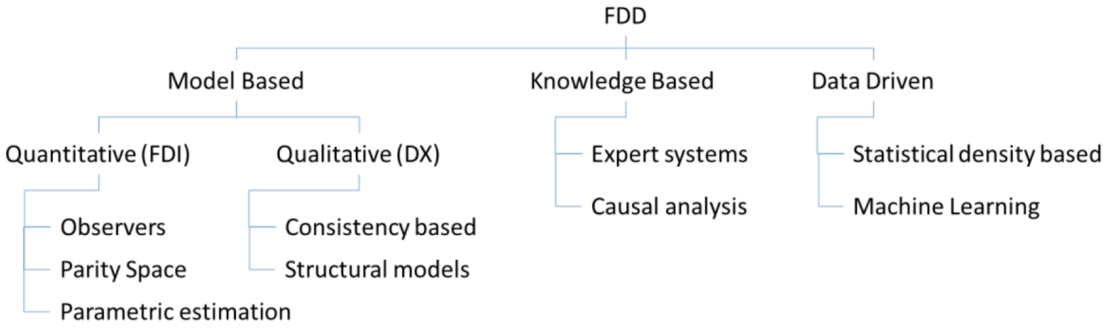



FIGURE 2.3: Taxonomy of fault diagnosis methods [7].

### 2.2.1 Model-based Diagnosis

Model-based diagnosis (MBD) relies on creating a model of a system by either having a set of analytical equations or a set of logical formulas that describe the behaviour of the system. When an inconsistency is found between the actual output of the system and the expected output from the model, the system is considered then to be faulty. Analytical or logical equations usually describe a specific behaviour of some phenomena. Therefore they usually require prior knowledge of the system.

Structural models however, describe the dependency of components to each other, which is by comparison to behavioural models, easier to derive since describing relations between components is not as complicated as describing phenomena through analytical or logical relations.

As seen from the taxonomy in Fig. 2.3, model-based diagnosis is split into two branches, FDI (Fault Detection and Identification) [10], and DX (Principles of Diagnosis) [2], which are two research communities that have different approaches to model-based diagnosis. The FDI community derive their models from control theory and statistical decision making, while the DX community rely on artificial intelligence and computer science.

#### 2.2.1.1 DX Model-based Diagnosis

DX-MBD usually relies on consistency-based approaches, using logical relations to form a model of a system. Faults are diagnosed when inconsistencies are detected between

[2]https://dx18.sciencesconf.org

the model and the actual system. However, some challenges arise when applying DX-MBD to robotic systems which is discussed in [7], such as difficulties in modelling the dynamic environment, difficulties in modelling noise coming from sensors or components, and representing interactions between the robot and environment. Some studies have been conducted using DX-MBD approaches for diagnosing faults on robots. In [11], the authors discuss the importance of diagnosis in robots, and how faults can affect the robot's beliefs based on the fault model it uses which is described in logical relations. The authors propose that it is important to be able to detect when faults occur in order to be able to update the robot's belief of the environment around it, since robots operate in dynamic environments and need to be able to adapt to the environment. However, this approach is only limited to what is modelled by the logical relations, meaning that the robot can only detect certain faults, and for the robot to be able to detect more faults, more behaviours need to be added to the model.

Since one of the main problems faced while trying to apply MBD to any domain is the formulation of a model, some studies have been conducted, which try to generate models automatically. In [12], models of a robot's behaviours are semi-automatically generated, by finding out which sensors and actuators are used when a certain behaviour is invoked. In [13], Zaman et al. use the robot operating system (ROS) to automatically diagnose and repair software or hardware faults based on an observer model. However, their approach was found to overload the robotic system because of the amount of observers being deployed, adding more computational complexity to the system.

Only a few studies have been made with regards to applying DX-MBD to robotic systems. This is mainly because of some challenges such as [7];

- Constructing a model that represents the behaviour of components on the robots such as sensors, that generate noise and operate in continuous time,
- Presenting the dynamic nature of the environment which the robot is deployed in, or representing the robot-environment interaction.

#### 2.2.1.2 FDI Model-based Diagnosis

FDI model-based diagnosis methods usually rely on developing numerical analytical relations which describe certain behaviours of any given system. The goal of using model-based diagnosis here is to replace hardware redundancy with analytical redundancy. For example, instead of adding an additional sensor to a system to increase its fault tolerance, an analytical model that represents the sensor is instead developed, and the same inputs which are passed to the real sensor are passed to the model, and a real value is

compared to a predicted value. In [8], the authors state that these models can either be deterministic or stochastic, continuous-time or discrete-time, linear or non-linear. As seen from the taxonomy in Fig. 2.3, the FDI models usually lie in one of three categories, either observers, parity space or parametric estimation.

- Observers play a major role in model-based diagnosis, where the inputs and outputs of a monitored process or system are modelled, and when a difference between the actual inputs and outputs and the predicted observations is found, a fault can be detected and diagnosed.
- Parity relations is the process of finding the redundancies in analytical relations and using these redundancies to detect and diagnose faults.
- Finally, parameter estimation is the process of representing physical features of a system in a model.

Generally, FDI approaches are widely used in fault diagnosis of internal components of robotic systems such as electric motors. For this reason, many work can be found where components of the robots are modelled analytically and these models are used to detect and identify faults. The interested reader can go through the following studies for more insight [14], [15] & [16].

### 2.2.2 Knowledge-based Diagnosis

Knowledge-based approaches are used in a way that mimics human reasoning where it can diagnose a fault based on known symptoms or based on previous knowledge. Knowledge-based diagnosis branches mainly into causal analysis and expert systems.

#### 2.2.2.1 Causal Analysis

Causal analysis is the process of assigning symptoms to faults [17]. Two methods of causal analysis are widely used which are symptom trees and signed directed graphs. Symptom trees are trees where causality is represented between symptoms and faults. On the other hand, the upper most node of the tree represents a symptom and the bottom nodes consist of possible faults. On the other hand, signal directed graphs represent the relation between system components and known fault symptoms. Nodes in the graph can represent the faulty symptoms of different system components. Once one of the symptoms is detected, the origin of the fault can then be found.

#### 2.2.2.2 Expert Systems

Expert systems try to mimic the human behaviour by having a knowledge base which is either represented in logical rules (such as IF statements), or fuzzy logic. Expert systems are sometimes represented as fault trees, which can be used to determine the root cause of a fault [18].

One of the main disadvantages of expert systems is that they are domain dependent. Meaning that they can only diagnose faults which are present in their knowledge base about a specific system or subsystem, therefore, not allowing them to detect unknown faults.

### 2.2.3 Data-Driven Approaches

Data-driven approaches are mainly classified into two main branches, statistical approaches and machine learning approaches. These methods rely on monitoring the online data produced by different components in a system to detect and diagnose faults.

#### 2.2.3.1 Statistical Approaches

Statistical approaches try to detect anomalies in the data based on statistical properties of the signal, such as deviations from a mean or standard deviation. Statistical approaches work best on data which is low in dimensionality and noise-free. However, this is not the case in real world scenarios, and thus one of the disadvantages of using data-driven approaches is that, usually some form of preprocessing to the data needs to occur before using the data for fault detection or diagnosis [19]. Using statistical methods, one can detect normal patterns in signals, and then detect outliers, which may then be interpreted as faults.

Some statistical methods such as Kalman filters and Particle Filters can be used to filter statistical noise, and then estimate the expected value of the signal. Using thresholding, one can then classify when the data being monitored falls within the expected range, or when the data falls outside the threshold range, an anomaly or fault can be detected. Application of Kalman filters or Particle Filters to the robotics field can be found in abundance in many studies.

In [20], the authors propose a method for detecting and identifying faults in robot sensors using multiple model adaptive estimation. Kalman filters are used as estimators with a specific embedded failure model. The Kalman filters here are used to give a probabilistic

estimation of the sensor's behaviour based on the operation of the robot, and based on the estimation, anomalies can be detected.

In [21], Kalman filters are applied to mobile robots for fault tolerance. Multiple fused sensors are used for localization, and the Kalman filter provides an estimation of the expected output of the sensor fusion, which adds fault tolerance to the system.

The use of Particle Filters can be found also in [22], where a Gaussian particle filtering algorithm is used for estimation of failure models in an autonomous underwater vehicle. Also in [23], particle filters are used in an effort to detect and diagnose faults in mobile robots; however, the authors stress that the computational complexity of particle filters depend on the amount of particles deployed. Ergo, the authors propose using an external computational device to reduce the burden on the robot's computational powers.

Other statistical methods also include Gaussian Mixture Modelling, Hidden Markov Models, Kernel Density Estimation or Mahalanobis-distance. The interested reader can go through the following studies for more insight; [24], [25], [26], [27].

#### 2.2.3.2 Machine learning Approaches

The other branch of data-driven approaches is machine learning. Machine learning in robotics is a well established field, where it has been applied to many problems such as object classification, speech recognition, or facial recognition. Machine learning approaches include methods such as neural networks, decision trees, self organizing maps and many more.

Neural networks present a great advantage to the robotics field, which is the ability to create models for non-linear systems, this is generally difficult to develop using traditional analytical methods. This in particular is attractive to the field of fault diagnosis, since models created from neural networks can be used to classify whether a data stream is normal or faulty.

Machine learning approaches can be supervised or unsupervised, where supervised approaches require a data set which is classified, to be learnt in order for a model to be created. Unsupervised approaches do not have this requirement. A problem that arises from applying a supervised approach is that they rely on the availability of classified data to learn. This is a problem when regarding fault diagnosis, as sometimes there is low availability of data that represents faults. Also, unknown faults can occur in the real system which would not be classified then by the supervised approach. Therefore, unsupervised approaches may be regarded. However, unsupervised approaches are generally less accurate than supervised approaches.

The use of machine learning for fault diagnosis can be found in many studies and brief examples are given here. In [28], support vector machines are used to detect anomalies, and an enhancement to one-class support vector machines is proposed for better detection. In [29], a neural network based fault estimation is developed to detect faults in robot manipulators. In [30], a novel adaptive sliding mode control method is developed to improve fault tolerance in robot actuators, where the method was then deployed to a spacecraft for testing.

Data-driven approaches can be also divided to univariate or multivariate approaches. Univariate implies that a data stream depends on another data stream's attribute, and multivariate implies that the data stream depends on more than one data stream attributes [7]. In [31], it is suggested that multivariate approaches are more superior to univariate approaches because they consider the correlation between other variables in the system to the monitored attribute.

Some problems which arise from selecting data-driven approaches are discussed in [7] and are summarised in the following points:

- Data-driven approaches usually disregard available information within the robotic system.
- Robots usually produce substantial amounts of data, and processing them for data-driven approaches can burden the computational powers of the robot.
- It is not easy to find classified data with regards to faulty and healthy classes. For this reason, developers resort to unsupervised methods which may be less accurate than supervised methods.

For these problems, the authors suggest using coupling the data-driven approaches to other approaches such as model-based diagnosis in order to enhance the diagnosis.

### 2.2.4 Hybrid Approaches

Since model-based, knowledge-based, and data-driven approaches all have advantages and disadvantages, hybrid approaches are developed to try to cope with a disadvantage of an approach by coupling it with another. Some studies that have combined different fault diagnosis methods can be found in [27], [31], [32].

### 2.2.5 Selection of Fault Diagnosis Method

As seen from this brief overview on different fault diagnosis methods, there exists a large number of methods to select from. For this, a qualitative analysis of the different methods can be derived, and based on the application, a method can be selected. As seen from table 2.5, the different methods discussed in this section are listed and advantages and disadvantages of each method are given. Using this table, one can get a sense of which fault diagnosis approach is appropriate for their application.

To aid developers select an appropriate fault diagnosis method, a diagram from [7], also serves as a guide to select the appropriate fault diagnosis method, which is aimed at robotic systems. The figure is given here in Fig. 2.4.

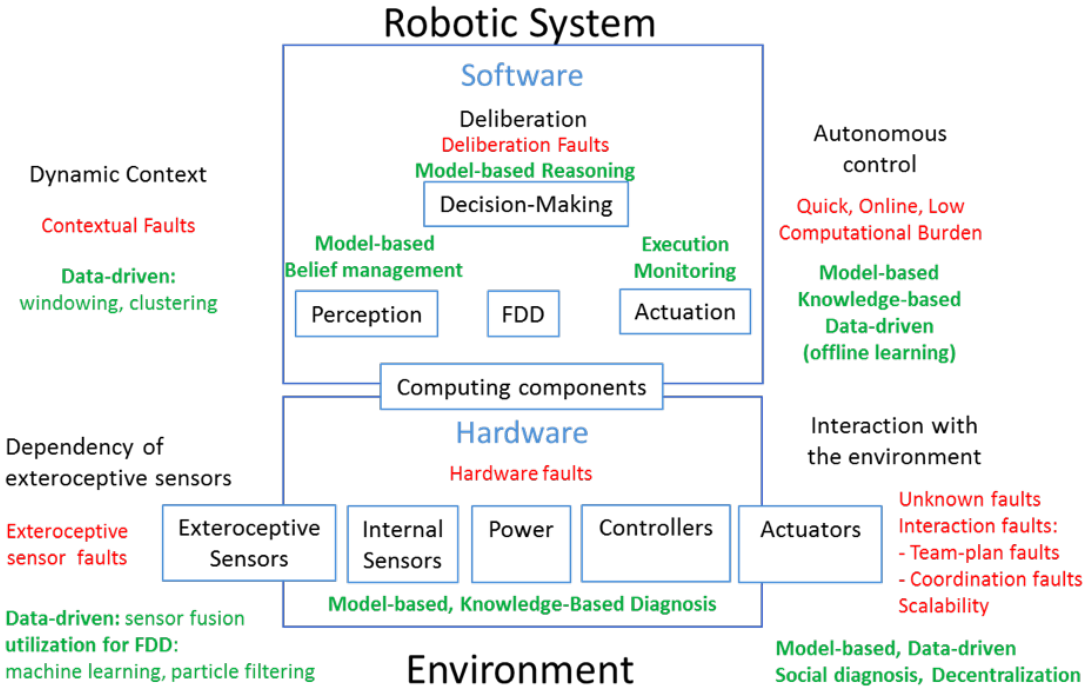



FIGURE 2.4: Fault diagnosis methods for robotic systems [7].

TABLE 2.5: Overview of different fault diagnosis methods.

| | Advantages | Disadvantages |
|---|---|---|
| **Model-Based: (Qaulitative DX)** | | |
| **Consistency-Based Diagnosis** | - Online.<br>- Fast reaction to faults.<br>- Can detect unknown faults.<br>- Low computational complexity.<br>- Can describe simple to complex behaviours. | - Can not express dynamic environment by using logic relations.<br>- Difficulties in modelling noise from components.<br>- Difficulties in representing interaction between robot and environment. |
| **Structural Models** | - Online.<br>- Easy to construct.<br>- Low computational complexity. | - Can not express complex behaviours.<br>- Only describes relation between different system components. |
| **Model-Based: (FDI)** | | |
| **Observers** | - Online.<br>- Fast reaction to faults.<br>- Can be used for diagnosis.<br>- Low computational complexity. | - Limited to what can be observed from the system.<br>- Developing an accurate model is time consuming.<br>- Difficulty in diagnosing faults caused by interaction with the dynamic environment. |
| **Parity relations** | - Online.<br>- Fast reaction to faults.<br>- Low computational complexity. | - Some phenomenas can not be represented analytically.<br>- Knowledge of the components to be monitored is required. |
| **Parameter estimation** | - Online.<br>- Low computational complexity. | - Prior knowledge of monitored components is required.<br>- Some phenomenas can not be represented. |
| **Knowledge-Based** | | |
| **Causal Analysis** | - Online.<br>- Fast reaction to faults.<br>- Low computational complexity. | - Requires prior knowledge of the system.<br>- Unknown faults can not be detected.<br>- Domain dependent |
| **Expert Systems** | - Online.<br>- Fast reaction to faults.<br>- Low computational complexity. | - Requires prior knowledge of the system.<br>- Unknown faults can not be detected.<br>- Domain dependent |
| **Data-Driven** | | |
| **Statistical** | - Online.<br>- Can utilise correlation of components.<br>- Easier to construct than model-based approaches.<br>- Simple to implement. | - Weak performance with noisy data.<br>- Weak performance with highly-dimensional data.<br>- Can be computationally complex if large number of data streams needs to be monitored. |
| **Machine learning** | - Can model non-linear systems.<br>- Can have high accuracy when classified data is available. | - Sometimes requires classified data sets.<br>- Can not classify unknown faults.<br>- Unsupervised methods usually have low accuracy. |

As seen from Fig. 2.4, the robotic system is comprised mainly of hardware, software and computing components. Hardware components include different sensors, actuators and other components such as power or controller components. The authors propose using model-based or knowledge-based approaches for diagnosing hardware faults. As for extroceptive sensors, the authors suggest taking advantage of the the increase of sensors deployed on the robots and use data-driven approaches for diagnosing sensor faults.

Robotic systems use sensors to sense the environment, and then use the extracted data to perform some computations which are represented in the software component section of the diagram. Deliberation represents the planning part of the software components and the authors suggest that model-based diagnosis would be a suitable method to diagnose faults in the execution of a plan, meaning, that when the observed behaviour does not match the expected behaviour based on the plan, a fault can be detected.

The actuation subcomponent of the software component represents the commands that are passed from high-level software components to low-level hardware components. For this, the authors suggest the use of model-based, data-driven, or knowledge-based approaches. As for the perception subcomponent, it represents the use of the sensor data to update the robot's beliefs about the environment. For this, the authors suggest model-based diagnosis for belief management.

The authors also stress that the more the robots are used for autonomous control, the higher the demand of fast fault diagnosis and online fault diagnosis becomes. In this case, the authors recommend using model or knowledge-based approaches, since data-driven approaches might not be able to meet these demands. If the robot operates in a dynamic context, then data-driven approaches are preferred.

Finally, since actuators interact with the dynamic environment, one can expect unknown faults to occur. For these faults, the authors suggest the use of either model-based or data-driven approaches, since knowledge-based approaches are not optimal for detecting unknown faults.

From this brief overview on different fault diagnosis methods, the developer can now select appropriate fault diagnosis methods with the aid of both table 2.5 and Fig. 2.4.

## 2.3 Data Standardization & Communication

The importance of standardizing health messages was discussed earlier in chapter 1. Also, from the overview given in this chapter on the many different types of fault diagnosis

available to the developer, one can expect different outputs for the different methods. However, all the diagnosis methods point to one direction, which is giving an estimation of the health of the monitored components on the robot. Another point to consider is the complexity of the robotic platform, and the amount of components that can be deployed on each robot. This usually means that there is a substantial amount of data generated by these components, which may be subject to health monitoring. This means as well that a large amount of data may be generated from the fault diagnosis methods, which, if not well represented to the user, can cause confusion, and the user may miss that a fault has occurred if the health data is published at a high rate.

Since robotic platforms are highly customizable with regards to their function and number of components deployed on them, it is difficult to find a data standard that is imposed on robotic platform components and accepted widely by the robotics research community. Nevertheless, this is not the case with military vehicles, which is a more well developed field, where data standards already exist for electric and electronic components representation and communication [33], [34]. Taking inspiration from military vehicles, one can take advantage of already well developed standards and apply them to the robotics field. Of course, there exists a huge difference between components deployed on vehicles and robots. However, since fault diagnosis methods which can be deployed on robots can also be deployed on vehicles, one can benefit from using standards related to fault diagnosis and health monitoring of vehicle components.

For this study, access to the NATO Generic Vehicle Architecture (NGVA) was granted, and will be discussed here in this section.

### 2.3.1 NATO Generic Vehicle Architecture (NGVA)

NGVA [3] is based on the UK Generic Vehicle Architecture [34]. It is an open architecture design approach to land platforms. GVA uses open standards to software and hardware interfaces, in order to ease upgrading or replacing electric or electronic components on a vehicle, rather than being dependant on third party hardware or software interface providers. GVA was later adopted and enhanced by European nations as NATO GVA (NGVA).

With the rise of components available to vehicles, such as sensors, actuators or even software components, the need to have agile military land platforms to cope with technological advancements became more crucial. With that also came the need to lower the overall cost of integrating new technologies, upgrading components or configuring land platforms to specific missions. NGVA tries to address these needs by standardising data

[3] https://natogva.org/

infrastructures for subsystems and components, making it easier to upgrade or integrate new technologies.

Fig. 2.5 gives an overview of the NGVA data infrastructure. The NGVA data model contains a collection of data modules, that represent different subsystems or components of a system. NGVA uses Data Distributed Services (DDS) [35], which is a machine-to-machine middle-ware. DDS has the advantage of providing reliable real-time information exchange between system components or subsystems. DDS works in a publish-subscribe manner, where it allows applications to publish or subscribe to topics on different computers within a network or on the same computer simultaneously.

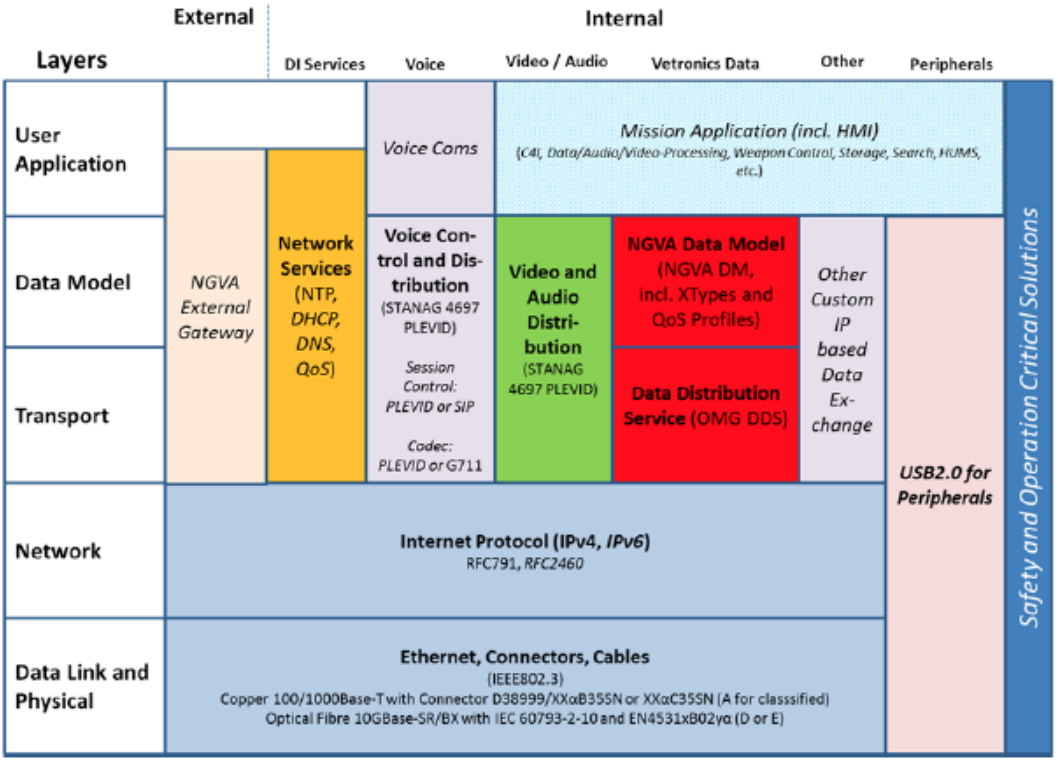



FIGURE 2.5: NGVA data infrastructure layered view.

Adopting such a standard to the robotics field with regards to fault diagnosis can be of great advantage, especially with regards to the goals of this research, since it provides an independent platform for communicating with a remote-PC, giving a higher robustness overall to the system, by not depending on the robot's operating system. Furthermore, since NGVA enforces a data infrastructure for the data being exchanged, one can benefit from having a standard to represent the health of components regardless of the fault diagnosis method used. From having an overview of the different data standards available from NGVA, the Usage and Condition Monitoring System (UCMS) module was selected

as the most appropriate standard to represent the health of subsystems and components deployed on the robot.

Since the UCMS data module has already been developed but not yet implemented, this also gave motivation to investigate if this data module could represent health of components thoroughly, and how it performs within our suggested approach. The UCMS data module will be discussed and evaluated later in chapter.5.

## 2.4 Approach

After having an in-depth study of the different aspects that are of importance to this project, some of the problem statement questions posed in chapter 1 can be addressed and an approach to implementing this project can be derived.

- Which fault diagnosis methods should be used to monitor the health of a wide range of different components that are deployed on a robotic platform?
  - The common faults in robotics and fault diagnosis methods sections discussed in this chapter helped in narrowing down the selection of the diagnosis methods that can be suitable for this study. Two methods have been chosen to be analysed and evaluated, which are Consistency-based diagnosis (CBD) and Sensor Fault Detection and Diagnosis (SFDD) [32]. The analysis of both methods will be presented in chapter 4.
- What are the existing standards for health representation and standardization that can be applicable to robotic platforms?
  - The UCMS data model module has been selected to be evaluated with regards to robotic platforms. The analysis can be found in chapter 5.

Finally, in order to be able to implement this system, an architecture needs to be designed that satisfies the goals of this project. The following chapter will present the system design and address the different constraints that need to be taken into consideration.

# Chapter 3

# System Design & Software Architecture

As mentioned previously in the beginning of chapter 2, the idea of the project is that an external computing device including a mini PC and a storage would be attached to robots in order to be used for health monitoring and fault diagnosis. Taking into consideration the many different components that would contribute to such a system, a conceptual system design has to be derived first and then, the different components within this system need to be analysed.

In this chapter, we present a detailed look at the requirements to develop such a system and discuss the different components that contribute to the system.

To come up with a conceptual design of a system, one must follow the guidelines of designing systems and take into consideration the constraints that come along with any system such as available computational resources or financial constraints or available/existing software/frameworks.

Following an Attribute-Driven-Design Method (ADD), the design process of the system is described as a set of steps to follow in order to try and manage all the system requirements and constraints. The requirements of the system are selected based on the Architecturally Significant Requirements (ASRs). The steps of ADD are as follows [36]:

1. Choose an element of the system to design.
2. Identify the ASRs for the chosen element.
3. Generate a design solution for the chosen element.
4. Inventory remaining requirements and select the input for the next iteration.

5. Repeat 1-4 until all ASRs have been satisfied.

To start, the first step would be selecting the system as a whole as the element to design. Therefore, the ASRs for the whole system should be taken into consideration. For that, quality attributes of a system are studied and the attribute requirements are then derived.

A brief overview of system quality attributes are given:

- Availability: It is the ability of a system to be used in the presence of faults, and how the system should deal with a fault once it has occurred.
- Interoperability: It is the ability of a system to exchange information.
- Modifiability: The ability of changing/modifying the system, and how that would affect the system as a whole including downtime, cost and effect on other system components.
- Performance: It is the ability of a system to manage its resources, and how it operates when these resources are unavailable or overloaded.
- Security: It is the ability of a system to manage its confidentiality, integrity and availability.
- Testability: The ease at which a software can be demonstrated for testing.
- Usability: The ability of a system to allow users to intervene in some processes of the system, such as

## 3.1 Requirements & Constraints

From the goals discussed in chapter 1, and the quality attributes that are used to describe the requirements of the system, a utility tree was developed, shown in table 3.1. The utility tree gives more insight into the requirements of the system and helps in the development of the system design.

The initial schematic of the system was as proposed in Fig. 3.1. However, based on the goals and requirements of the system, iterations over the schematic were needed to be performed with regards to the design of the system in order to develop a more concrete system design.

Another aspect which had to be taken into consideration as well while developing the system design, was the available frameworks, software and technologies. From the related

Table 3.1: Tabular Utility Tree for the Non-intrusive Fault Diagnosis & Health Monitoring System.

| **Quality Attribute** | **Attribute Refinement** | **ASR** |
|---|---|---|
| Configurability | User-defined model | A configuration file containing the necessary parameters required for fault diagnosis & health monitoring. Users can change or update the configuration file without the need of changing the source code. |
| Extensibility | Adding new fault diagnosis methods | The system should have independent components to allow for easy addition of new fault diagnosis methods. |
| Performance | Low computational complexity | The system is to be run on a mini-PC with limited resources; therefore, the computational complexity of the fault diagnosis methods selected should be low. |
| | Throughput | At peak network load, the system should be able to notify the remote user with a notification. Having self-monitoring ability. |
| Interoperability | Communication with remote PC | The system should have the ability to communicate with a remote PC to inform the user of the health status of the monitored robot. |
| Security | Store health data safely | The system should have a record of the robot's health. Therefore, an external storage device should be deployed along with the black box. It would also be useful in case of loss of communication with the remote PC. |

work discussed in chapter 2, and the available robots for testing, ROS was used as the main robot framework. ROS is based on publish-subscribe principles, where nodes, that are software components are used to perform computations, and publish or subscribe

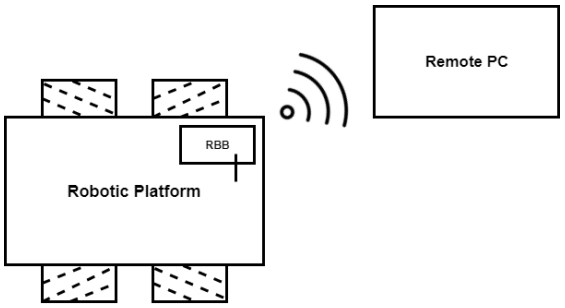



FIGURE 3.1: Schematic of black box connected to robot.

to data through topics. Nodes, for example, can be used to collect data from sensors; usually through drivers, and publish the extracted data through topics. Other nodes can then subscribe to the published topics and perform further operations on the extracted data. An example of how data is transmitted through ROS is given in Fig. 3.2.

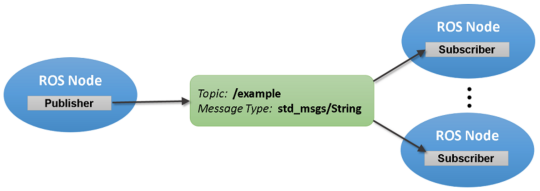



FIGURE 3.2: Illustration of nodes, topics, publishers & subscribers in ROS.

Another middle-ware that is also to be considered while developing this system is Data Distributed Service (DDS), which was mentioned also in the related work in chapter 2. DDS is a reliable machine-to-machine middle ware that is aimed at systems that require real-time data exchange. Using the discussed data model module UCMS in chapter 2, DDS will be used for communicating with the remote PC that will give an overview of the monitored robots health to the users.

## 3.2 Views

From the goals, requirements and constraints discussed, a system design and software architecture was developed. Since " a software architecture is a complex entity that can not be described in a simple one-dimensional fashion" [36], a more detailed description will be given in this section.

Fig. 3.3 - 3.6 show different diagrams of the designed system architecture. Each diagram has the purpose of conveying a different message to the reader in order to clarify the different aspects of the system. The first figure, Fig. 3.3, shows a use-case system architecture diagram. This diagram is a high-level system architecture diagram that presents the system in relation to it's actors, namely the robot and the remote user. The

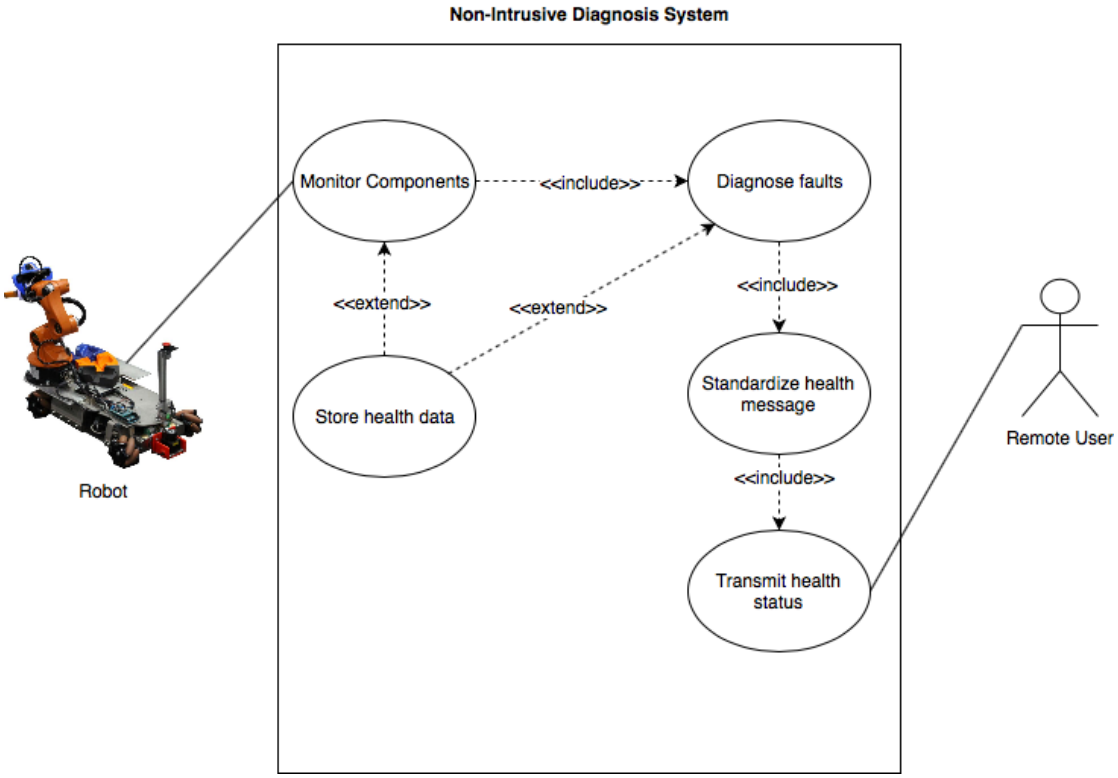



FIGURE 3.3: Use case diagram of non-intrusive fault diagnosis system

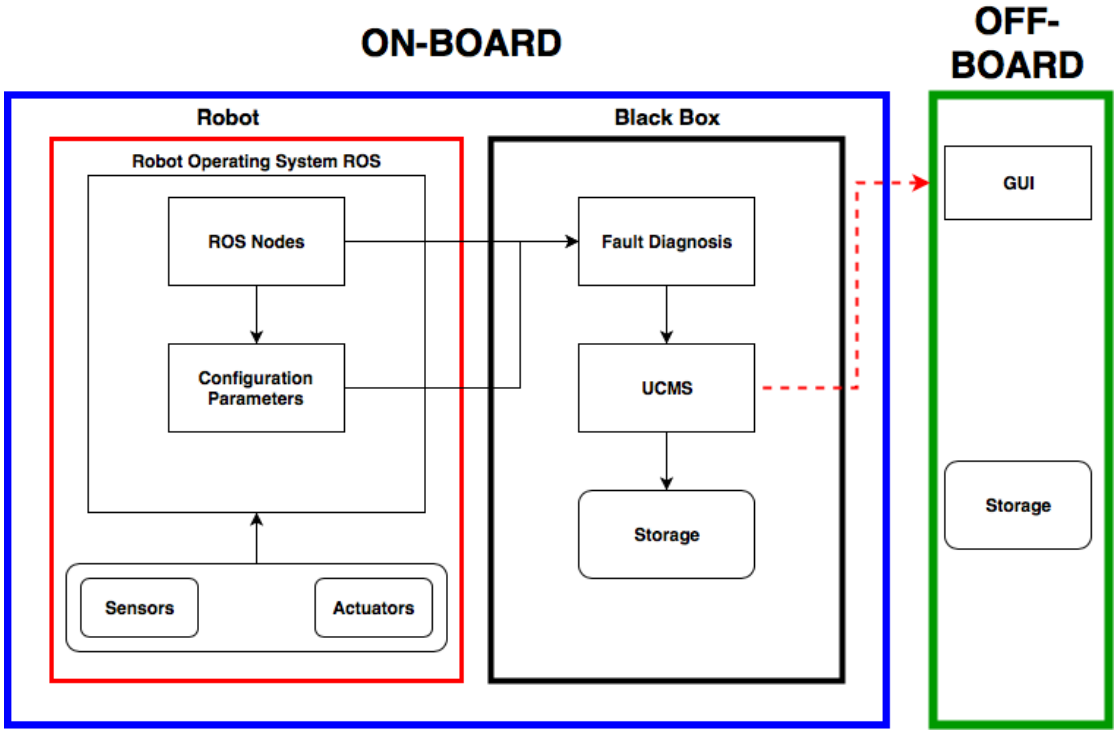



FIGURE 3.4: Conceptual system architecture of the non-intrusive black box connected to the robot.

system comprises of different use-cases, that represent actions, which are tasks executed within the system. The actions are initiated by the robot, where when the robot starts,

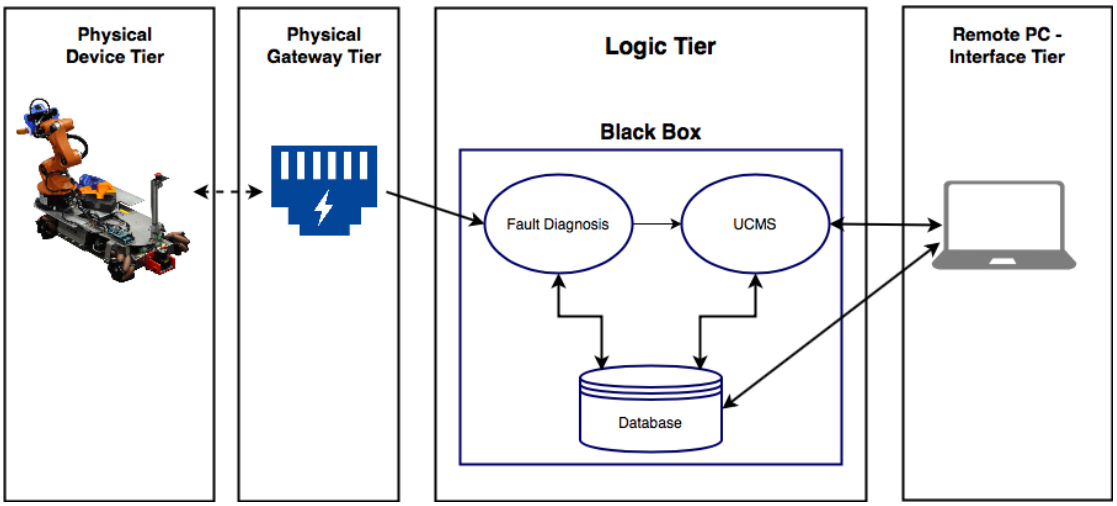



FIGURE 3.5: Tier view of system architecture containing physical, physical gateway, logic and remote PC - interface tiers.

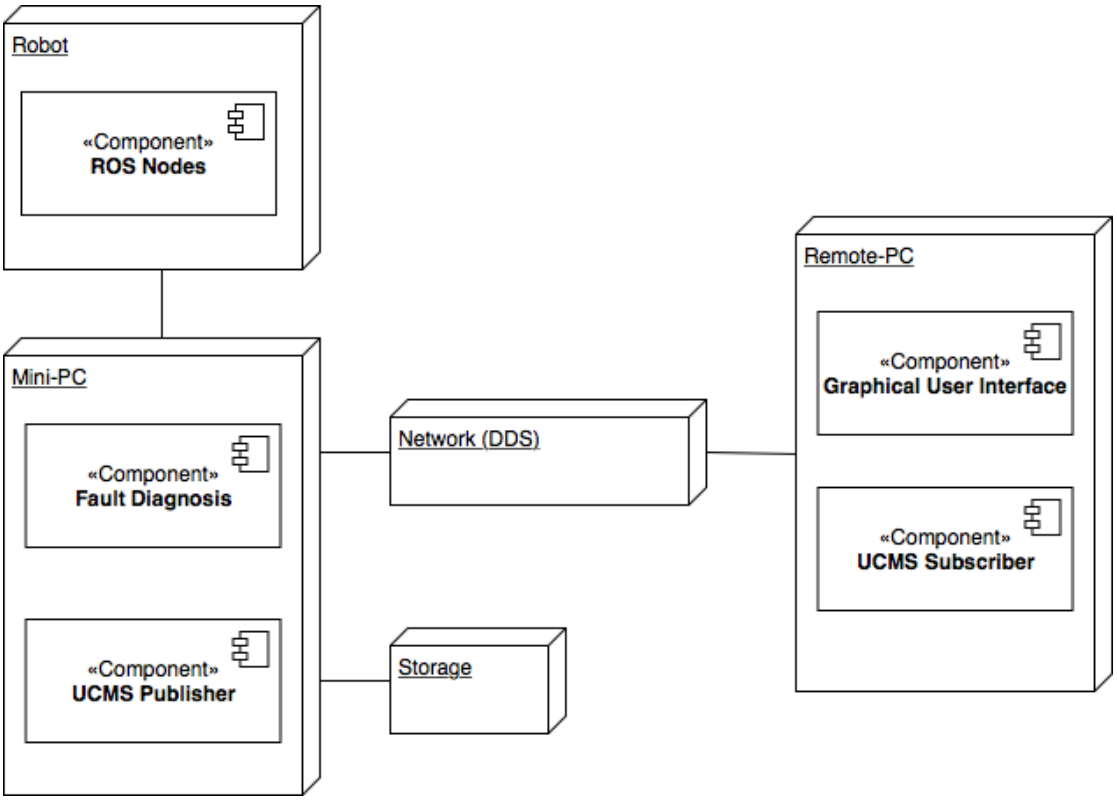



FIGURE 3.6: Deployment diagram of non-intrusive fault diagnosis system.

the components of the robot are monitored. This task initiates other use-cases such as “Diagnose faults", and “Store health data". The remote user can then view the health status of the monitored component once they are transmitted by the “Transmit health status" use-case. This diagram is used to give an abstract overview of the different actions that are included within the system.

Fig. 3.4 shows a conceptual system architecture diagram that can help clarify the different software and hardware components within the system. The blue box contains the on-board components which are the robot and the black box. The robot (red box) has different hardware components mounted on it such as sensors and actuators. These components interact with the robotic platform through the Robot Operating System (ROS). The robotic platform contains different software components such as ROS nodes and configuration parameters. ROS nodes can use the hardware components such as sensors to extract data, then process it. These nodes can also use the actuators to perform an action in the environment. The configuration parameters is a requirement of this system, which is a software component containing all the important information required for fault diagnosis. The black box (black box) contains the fault diagnosis and UCMS components, which are software component. The fault diagnosis component monitors the different ROS nodes and uses the configuration parameters to gather information on these components such as a model of the system, and produces a diagnosis of the system. The UCMS component, contains the standard that the fault diagnosis messages will be mapped to. The storage device in the black box will be used to store health messages that are produced by the diagnosis system. The off-board component (green box) will contain the graphical user interface, and storage if required. The red arrow used here represents that health data being transmitted using the Data Distribution Services (DDS) wirelessly.

Fig. 3.5 show the different tiers involved within the system, which can help understand how the system will be deployed. Finally, Fig. 3.6 shows a deployment diagram of the designed system architecture, where different hardware components of the system are presented containing the relevant software components.

### 3.2.1 On-board Component

- Using the ethernet network interface, the black box connects to the robot's ROS core, and extracts the required data from ROS nodes and topics, that is used by the fault diagnosis component of the black box.
- The output of the fault diagnosis component will depend on the chosen diagnosis method which is discussed later in chapter 4. However, based on the expectations of the fault diagnosis component, the output should give an overview of the health of different components that are monitored on the robot.
- The output of the fault diagnosis component will then be mapped to the standard data model module, UCMS, and prepared to be transmitted to the off-board PC.

- The standardized messages will then be transmitted to the of-board PC using the mini-PC wireless network interface, using DDS as the middle-ware for communication.

- As the health of the robot's components is being observed, data is logged to the storage of the black box using methods such as periodic data logging and event-based logging. The storage will also act as a fault tolerant component to the fault diagnosis system, such that, in the case of loss of communication to the remote PC, a log of the health of the robot will still be stored and can be later retrieved if required.

### 3.2.2 Off-board Component

The off-board component will comprise of a remote PC, that receives the health data from the black box through DDS. The remote PC contains a GUI which displays the health status to the user, and should also enable the user to request data logs of specific components or all the components if required.

### 3.2.3 Graphical User Interface

Based on the UCMS standard that will be used for this study, a conceptual graphical user interface (GUI) was developed, giving an insight on how the messages will be displayed to the remote user. Since DDS allows for cross-platform communications, the GUI concept was developed using a windows-PC. Fig. 3.7 and Fig. 3.8 show the log-in page and the main window of the interface itself respectively.

Fig. 3.7 shows the log-in page which will help add security to the system, meaning that only users with permission will be allowed access to the diagnosis information. In Fig. 3.8, the main window of the GUI will display the monitored components and show important information to the user. Based on the health status of each component, a color warning may be displayed to the user such as yellow for degraded performance and red for non-operational component. The GUI will be communicating directly with the data transmitted from the black-box, making the GUI independent of ROS and the robotic platform itself.

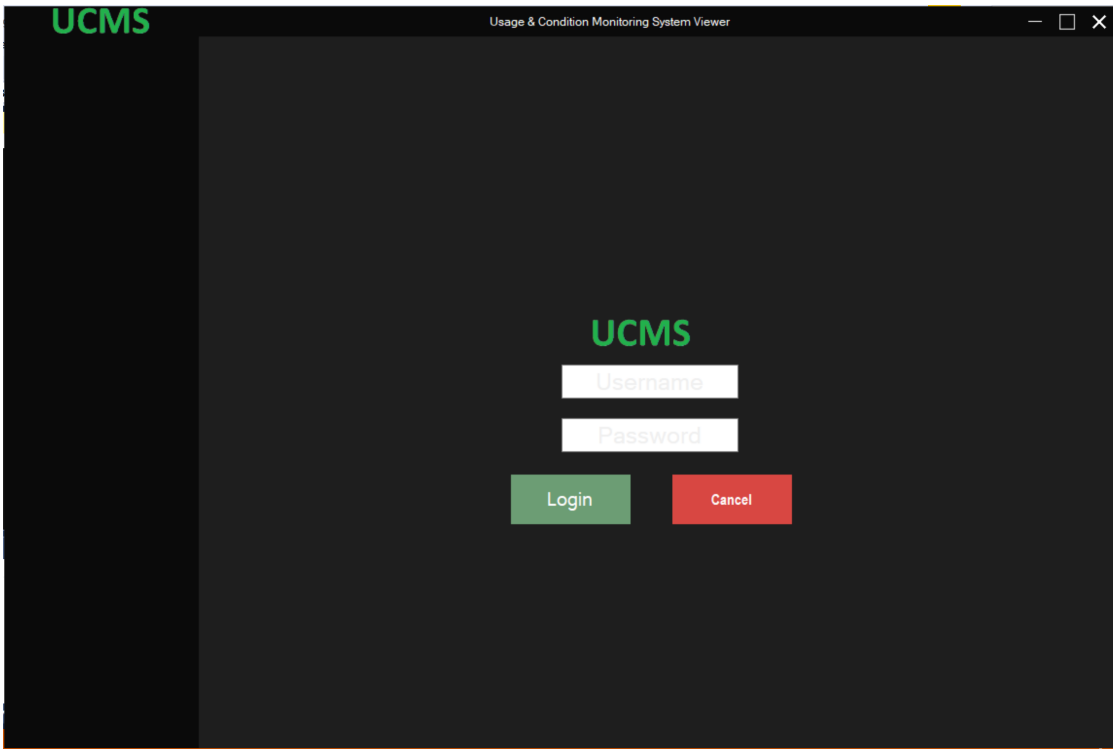



FIGURE 3.7: Log-in page of conceptual GUI.

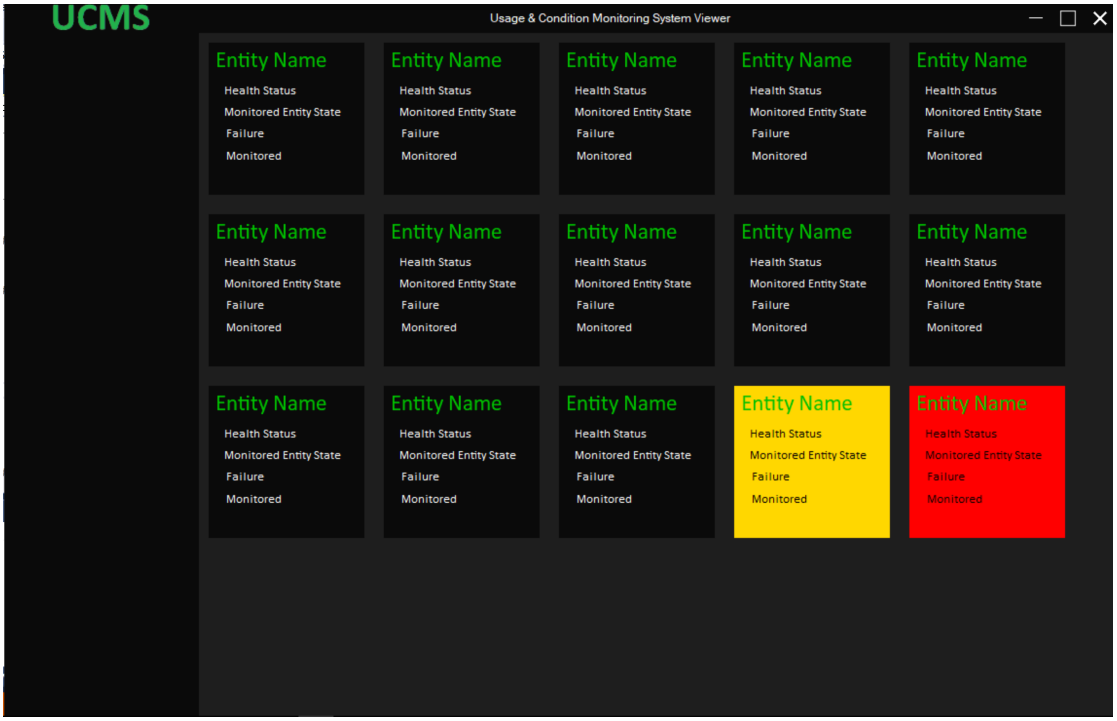



FIGURE 3.8: Main window of GUI showing different monitored entities along with important information.

# Chapter 4

# Health Monitoring & Fault Diagnosis

Fault diagnosis and health monitoring is the core of this study, since it will provide the intelligence required to the system to determine whether or not the monitored components are healthy. The fault diagnosis and health monitoring method selected in this study should have the ability of incorporating as many components as possible, such as hardware and software components. Moreover, the selected diagnosis method should have a low computational complexity since it will be deployed on a mini-PC having limited computational capabilities. From the related work presented in chapter 2, and the available diagnosis methods that can be applied to robotic platforms, two approaches were selected to be analysed. Consistency-based diagnosis (CBD), and a hybrid approach presented by [32], Sensor Fault Detection and Diagnosis (SFDD), using a combination of data-driven and model-based approaches, were found to be suitable for this study. CBD is a suitable method since it is low in computational complexity and can be used to describe the behaviour of different components. It is also simple to construct models for simple behaviours, therefore having the ability to quickly set up a diagnosis system using CBD. SFDD was also attractive because it does not require an in-depth model of the system to be built, however, takes into consideration the connections only between sensors and components, and tries to find the correlation between different components.

In this chapter, a detailed look on both mentioned fault diagnosis methods will be presented. A comparison of both methods will be given with regards to how they can be implemented for this intended study.

## 4.1 Consistency-based Diagnosis

Founded by Reiters [37], consistency-based diagnosis is used in diagnosing systems by describing the correct behaviour of the components and the way components interact. CBD uses First Order Logic (FOL) to describe the behaviour of the system including its components. A model of a system is then a collection of the FOL statements describing the behaviour of the components. These models are used to diagnose the system based on observations of the real system. Describing a system in terms of its components means that the system can be decomposable, which can help in fault isolation. A description of a system can then be split into three main parts [38]:

- Behaviour of component types.
- List of Components.
- Component structure.

The following example will be used in describing the modelling and diagnosis process of Consistency-based diagnosis. Usually, logic circuits are used to describe CBD, such as the circuit shown in Fig. 4.1.

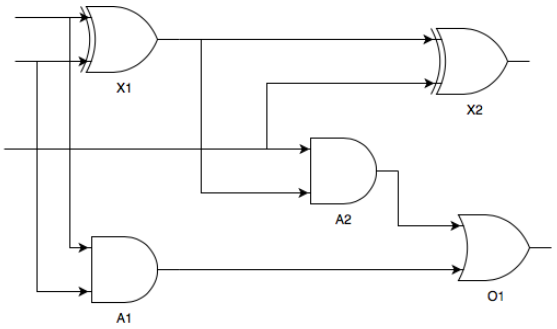



FIGURE 4.1: Structure of a circuit.

In Fig. 4.1, components of type AND, OR, XOR gates are connected together. The description of behaviour of component type will be in the following form:

$$type_i \wedge ok(x) \rightarrow \Phi(x) \tag{4.1}$$

In equation 4.1, $\Phi$ represents a generic formula that states how the component behaves, $type_i$ represents the types of components and $x$ is the component that belongs to type x. For example, the behaviour of component type statement for components in Fig. 4.1 is as follows:

$$\begin{aligned} TYPES = \{&ANDG(X) \wedge \neg AB(X) \rightarrow out(x) = and(in1(x), in2(x)), \\ &ORG(X) \wedge \neg AB(X) \rightarrow out(x) = or(in1(x), in2(x)), \\ &XORG(X) \wedge \neg AB(X) \rightarrow out(x) = xor(in1(x), in2(x))\} \end{aligned} \tag{4.2}$$

where ANDG, ORG, XORG represent AND, OR and XOR gates respectively, and AB means abnormal.

After describing the behaviour of component types, the components in the system need to be listed along with their types. Therefore for Fig. 4.1, the COMPTYPES list will be as follows:

$$COMPTYPES = \{ANDG(A1), ANDG(A2), ORG(O1), XORG(X1), XORG(X2)\} \tag{4.3}$$

Finally, the system structure needs to be given, stating the connections between the components.

$$\begin{aligned} CONN = \{&out(X1) = in2(A2), \\ &out(X1) = in1(X2), \\ &out(A2) = in1(O1), \\ &in1(A2) = in2(X2), \\ &in1(X1) = in1(A1), \\ &in2(X1) = in2(A1), \\ &out(A1) = in2(O1)\} \end{aligned} \tag{4.4}$$

After listing the connections between the components (CONN), the system description (SD) becomes complete. Therefore a complete system description is:

$$SD = TYPES \cup COMPTYPES \cup CONN \tag{4.5}$$

This system description is now the model of the system that can be used for diagnosis. In order to diagnose a system, observations (OBS) are required to be drawn from the real system. A diagnosis problem is defined as a triple:

$$(SD, COMPS, OBS) \tag{4.6}$$

where SD is the system description, COMPS is the set of suspected components and OBS is a finite set of first-order sentences containing the observations made of the system. As the system is running, consistency is checked between the model and the observations. If an inconsistency exists, a diagnosis is called for. In a CBD problem, a diagnosis assumes that only a subset of the components are faulty and the rest are normal. Therefore:

$$D \subset COMPS \tag{4.7}$$

where D is minimal set of components such that:

$$SD \cup OBS \cup \{AB(c)|c\ in\ D\} \cup \{\neg AB(c)|c\ in\ COMP - D\} \tag{4.8}$$

is consistent. Meaning that the system description along with the observations are consistent with the system components contain a set of faulty components or in other words abnormally behaving components.

Finally, to find the faulty components in D, a hitting set algorithm is used. This is done by assuming that one of the components in D is faulty, and then checking for consistency with the system description, this is done repeatedly until a component can be declared as faulty.

Using CBD to diagnose faults in robots has been done previously in different studies [13], [39], [40]. The advantage of using CBD for robots, is that it can be applied to different levels of the system. For example, CBD can be used in diagnosing faults in a wheel motor by modelling the components of the motor and checking for consistency, or it can be used in diagnosing the whole robotic system by modelling all the available components and their connections on a system level, rather than a component level. The limitations of CBD is that it can only diagnose what is stated in the models, therefore, if a fault occurs in a component which is not modelled, it will not be detected. Another limitation is that modelling a system thoroughly takes a substantial amount of time, and the diagnosis will only be as accurate as the modelling is.

## 4.2 Sensor Fault Detection & Diagnosis

In [32], Khalastchi et al. derive a method for sensor fault detection and diagnosis for autonomous systems. The fault detection method used in this paper is a combination of a data driven approach with a model-based approach. The advantage of this method is that it can be used online, as the data is being produced by the sensors.

The study focuses mainly on two kinds of sensor faults, which are difficult to detect; stuck and drift. Stuck is when a sensor is producing the same value regardless of the state of the system, drift is when the readings being produced are gradually increase or decrease over time and deviate from the real reading. It is difficult to imply that a sensor is faulty if it is stuck, since it might be operating within its normal range, therefore using minimum and maximum values as thresholds will not detect that the sensor is in a faulty state. As for drift, it is also difficult to imply that a sensor is faulty when there is drift because of the same mentioned problem.

To address this problem, the authors combine a data driven approach that determines the correlation of sensors to each other using Pearson Correlation, and a structural model that represents which sensor relies on which internal hardware component. The advantage of using such a model over other methods such as mathematical models is that it is easier to derive and represent.

The authors describe their process as follows:

- A sliding window of size m $\times$ n, named $H^t$ is taken as input to the process. Where m is the number of time steps per window containing the reading of the sensor, and n is the number of sensors to be monitored.
- The sliding window is split into two halves. The first half of the matrix, is subjected to a correlation test to determine which sensors are correlated to each other. The output of this test, for each monitored sensor, is a set of correlated sensors.
- After the correlation is determined, the second half of the sliding window $H^t$, is checked to see if the correlation of the sensors hold. If there exists a correlation between two sensors in the first half, and the correlation does not hold in the second half, the sensor is marked as uncertain, and a pattern is assigned to the sensor.
- Using the structural model of the system, and the marked sensor, along with its set of correlated sensors, a check is performed to find out if this change in correlation is due to a fault or a response of the system.
- If it is found that a fault has occurred, the structural model is again used to determine which component or sensor of the system is responsible for the fault.

This process is done repeatedly online, while updating the window every t time steps. A problem arises from online sensor reading is the noise produced by the sensors. This noise needs to be filtered first using some low-pass filters. The output of the Pearson Correlation test for each two compared sensors is a value between -1, and 1. The authors

state that determining whether two sensors are correlated or not depends on a user defined threshold between 0 and 1. When checking if the correlation holds in the second half of the input matrix, patterns are assigned to each sensor after the check. The state patterns defined by the authors are as follows:

- OK: Correlation holds.
- Stuck: The sensor produces the same value even after its correlated sensors have changed behaviour.
- Drift: The sensor produces an output which has deviated over time, either to lower or higher values.

The authors also advise to use simple linear regression to detect the drift or stuck. After states are assigned to each sensor, the set of sensors showing suspicious patterns are placed in a set. Having the set of suspicious sensors, and the sets of sensors correlated to each sensor, the structural model is then required to detect and diagnose the faults, if they have occurred. The structural model of a system can be in the following form, as shown in Fig. 4.2.

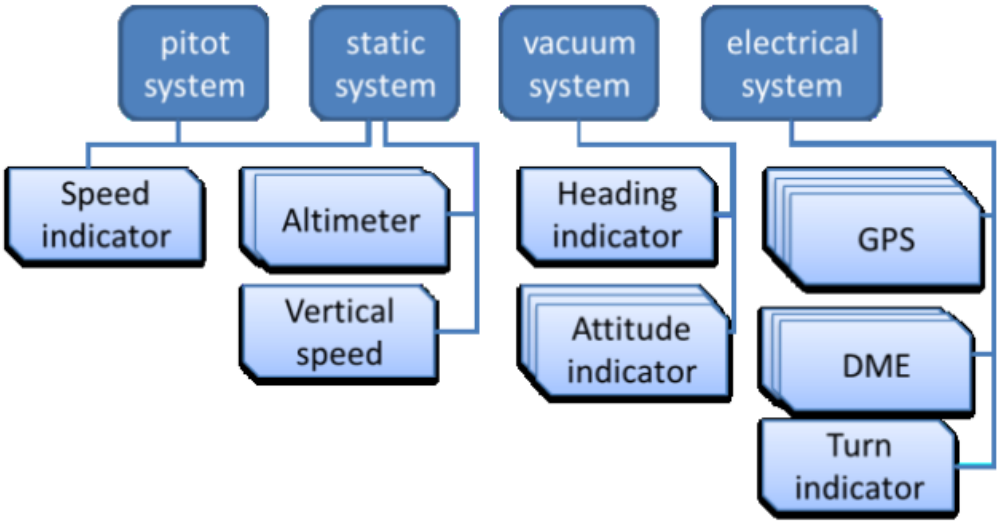
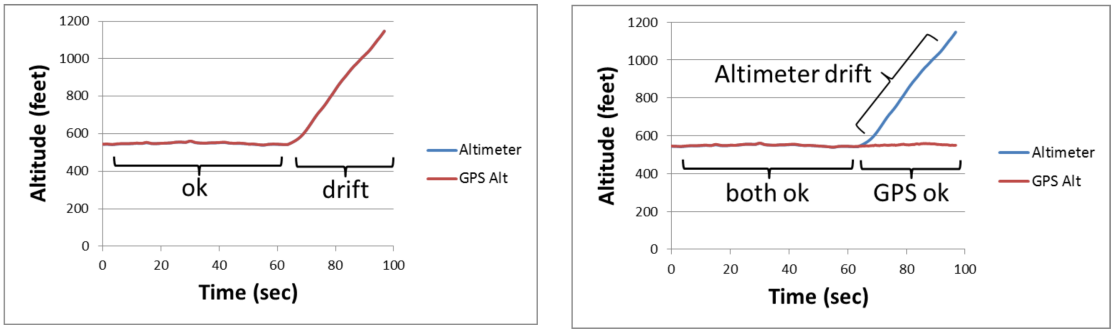



FIGURE 4.2: Structural model of a Cessna 172p, as shown in: [32].

The structural model shown in Fig. 4.2, is from the example used in the study by Khalastchi et al. of a Cessna 172p airplane. The model shows how each sensor is connected to its subsystem or hardware component.

Fault detection is done by checking for each uncertain sensor, if there exists another sensor from a different component or subsystem that shares the same behaviour. If a

sensor is found to be suspected of a fault, a diagnosis is then called for. The fault diagnosis process takes the suspected sensor, its state, the list of uncertain sensors, and the structural model of the system as input. The subsystem or hardware component that the sensor is connected to is then examined, to determine whether the sensor only is faulty, or the component that it connects to is faulty. If the component is faulty, all its connected sensors are assumed to be faulty as well. This is done by checking if the other sensors connected to the component with the suspected sensor are also included in the list of uncertain sensors. Based on the number of suspected sensors in a component, a probability of the health of the component can be derived. The following example, which is taken from the study as well, shows the behaviour of the altimeter in relation to the altitude reading from a GPS sensor.




(A) Both sensors behaving the same. (B) Sensors not behaving the same.

FIGURE 4.3: Altimeter and GPS sensor behaviour in different cases as shown in:[32].

After applying the correlation test to the sensors in the system included from Fig. 4.2, it is found that both the altimeter from the static system and the GPS altitude reading from the electrical system are correlated. Fig. 4.3a shows both the sensors behaving in the same manner. When the GPS altitude sensor drifts, the altimeter readings also drift. This means that no fault exists. However, in Fig. 4.3b, it can be seen that both sensors behave in the same manner, in the period marked “both ok", but then the altimeter reading starts to drift while the GPS altimeter reading is marked as “ok". This is an example of a suspicious behaviour which needs to be investigated using the fault detection and diagnosis methods mentioned earlier. If the altimeter is drifting due to a system response, meaning that another sensor from a different component, that was correlated to the altimeter when the correlation test was applied, is behaving in the same manner, then the fault is cleared. If not, then the diagnosis method is called, to determine whether the altimeter is faulty or the static system is faulty. The same will also be applied to the GPS altitude sensor, since both sensors will be marked as suspicious after they are found to be not behaving the same.

The algorithms developed in this [32] study were tested on a laboratory wheeled robot, having three sonar range detectors and three infra-red range detectors, and on a flight

simulator which gives the authors the ability to inject faults and test their algorithms. The results of the tests were promising showing high fault detection rates and very low false alarm rates.

A drawback of the proposed approach is that it only targets sensors which provide single dimension data. Meaning that some complex sensors such as laser range finders can not be taken into consideration when using this algorithm.

The authors suggest as well that it would be useful to use calculated expected values, such as speed from a GPS module as an input to the algorithm as well so that there are more redundant sensors and values. They also discuss how the derived algorithm can be used recursively since in most cases, systems are never as simple as the provided example, where systems contain many subsystems with various components and sensors.

A recent study by Biswas [41], implemented this method to an omnidirectional robot. Some drawbacks were noted from the implementation which were not mentioned in the original study. It was noted that the proposed method does not take care of positive or negative correlations, and takes only the absolute value of the correlation. Since the positive and negative correlations are not taken care of, the drift pattern proposed in the study, can not also be detected. Finally, the computational complexity of the algorithm increases with the increase in number of sensors monitored, making the algorithm difficult to implement on a PC with limited resources. A modified version of the SFDD algorithm has been proposed by Biswas, where the signals are correlated with two defined signals, instead of correlating all sensor signals with each other, to lower the computational complexity. It was also proposed to only detect a change correlation, instead of checking for the correlation itself. This introduced a new set of patterns to detect, allowing for positive and negative correlations to be detected as well.

The results of the study showed an accuracy between 0.45-0.6 for different kinds of faults relating to the omnidirectional robot base. The study showed that the accuracy of the algorithm is much lower than what was stated in [32], and pointed out some disadvantages which need to be addressed such as the computational complexity and the effect of noise of the sensors on the Pearson Correlation algorithm.

## 4.3 Comparison Between CBD & SFDD

After investigating both methods, the following analysis can be made. Table. 4.1 summarizes the key differences between both methods. It can be seen from the comparison and the discussions presented on both methods, that consistency-based diagnosis is more suitable for our project since it is, most importantly, lower in computational complexity

Table 4.1: Comparison Between CBD and SFDD.

| Category | **Consistency-based Diagnosis** | **Sensor Fault Detection & Diagnosis** |
|---|---|---|
| **Operation** | online. | online. |
| **Modelling** | Requires model containing components, connections, and component types. | Requires structural model. |
| **Preprocessing** | Filter readings if required in windowed fashion. | filtering in a windowed fashion + correlation test + List of suspicious sensors. |
| **Monitored Components** | Can monitor different kinds of components. | Can monitor different kinds of components provided they generate one-dimensional data. |
| **Computational Complexity** | Low computational complexity. | Computational complexity increases with increase in components monitored. |
| **Comments** | Low initial set up time. Will use the available data from components for fault diagnosis. | Higher initial set up time, since one-dimensional data needs to be drawn from components, which may not be available to start with. |

and has the ability to cover a wider range of components without the need of extracting a special kind of data, such as one-dimensional data for SFDD. CBD also requires less pre-processing to the incoming data since the data only needs to be filtered from noise, if it exists. On the other hand, SFDD will require the signals from different components to be one-dimensional, filtered from noise, subjected to a correlation test, subjected to a test to check if the correlation holds, before being used for fault detection or diagnosis. For these reasons, the selected method to be tested for this study is CBD, and the results generated here will be compared to the results of the study presented by [32] and it's implementation by [41].

# Chapter 5

# Usage & Condition Monitoring System

This chapter will give an overview of the Usage and Condition Monitoring System (UCMS) data model module and the Health and Usage Monitoring System (HUMS) data standard which was developed by the UK GVA for vehicles. The UCMS data model module is derived from the HUMS data standard and is developed mainly for military vehicles.

With the fast growing technological advancements in military vehicles, and the need to constantly upgrade different electronic components on the vehicles to have tactical advantage on the field, the need to have an architecture that allows for easy upgrades and modifications to the vehicle platform grew. As explained in chapter. 2, these were some of the reasons for developing the NATO Generic Vehicle Architecture (NGVA). Different data model modules have already been deployed on military vehicles for components such as Laser Range Finders (LRF), video cameras and brake systems. Also, since many different components are now deployed on military vehicles, it became crucial to develop a data standard for fault diagnosis and health monitoring of these components, in order to increase the reliability of the vehicles and also give the operators an overview of their equipment's health status.

The HUMS standard was developed to be deployed on vehicles to give them the ability of monitoring the health status of the multiple components deployed on a platform. A generic system architecture of a vehicle including the HUMS data standard is presented in [42] and is shown below in Fig. 5.1.

As seen from Fig. 5.1, HUMS contains different components which can be useful for having a full diagnosis system running on a vehicle. The standard's architecture already

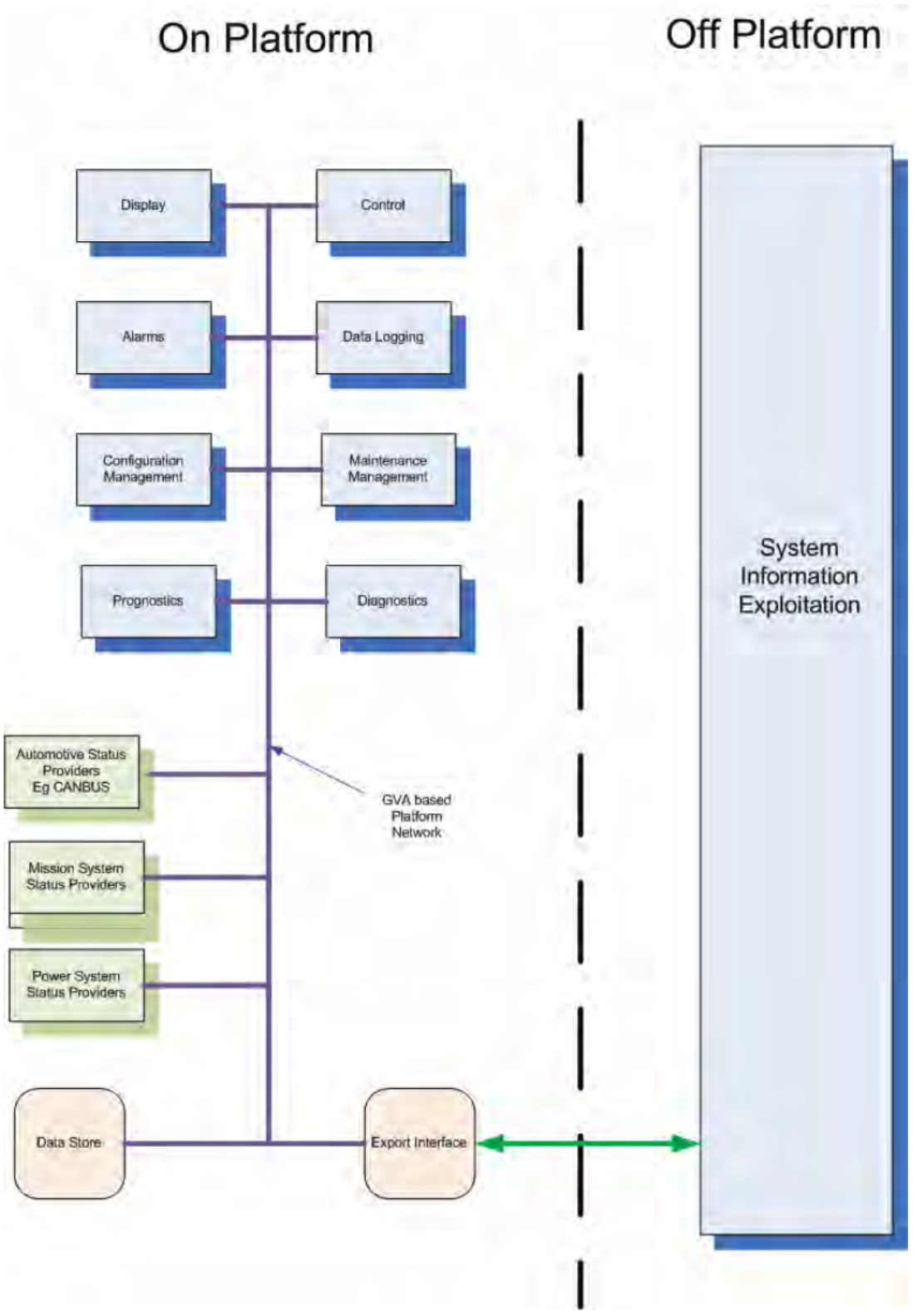



FIGURE 5.1: HUMS in a GVA context [42].

contains an on and off-platform components, which is a similarity between the architecture developed in this project for the black box. The on-platform side of the architecture contains components such as Diagnostics, Data Logging, Data Store, Prognostics &

Export interface. Some of these components are also a shared similarity between the architecture developed for this project in chapter. 3 (see Fig. 3.4). The requirements of each component is explained in the standard [42], and the relative aspects will be summarized in this chapter.

## 5.1 Data Storage

A main feature of the HUMS standard is the data storage, which is used to keep a record of the health status of different components of a vehicle. Some requirements of the data storage are that it should be capable of storing health data for a specific amount of hours, in the case of vehicles 4000 operational hours are required. The data storage should also be able to store different kinds of health data such as components conditions, usage data, configuration data, and important events regarding the health status of components such as failure events. For that, the data stored on the storage device entries should be uniquely identifiable and include a time stamp.

## 5.2 External Interfaces

A system deploying the HUMS standard should also have the ability to interface with an external system (Off-Platform), either to:

- Export data, in order to keep logs of the vehicle's status throughout a period of time.
- Empty the data storage of the HUMS system after the specified time requirement for storing data has ended.
- Importing new data to update some configuration parameters such as thresholds for example or adding information about new equipment.

The HUMS external interface should allow users to also export information about the vehicle's platform when required. The required data could be all the health data available about all the components, or data about specific components deployed on the platform, or data about specific events that occurred such as failures.

## 5.3 Data Logging Techniques

Some data logging techniques are proposed in the standard since it is expected that a substantial amount of data is to be generated from a platform containing multiple components that are required to be monitored. Some of these techniques include:

- Periodic logging: Can be useful to track the status of a component over time and reduce the amount of data stored by only logging data at a periodic time value.
- State-driven logging: When a certain state is entered, it could be helpful to increase the amount of logging if required.
- Event-driven logging: If an event occurs, such as a fault, it can also be useful to increase the logging of the monitored component, and possibly take a snapshot of the whole system state for further analysis, which can be used by the diagnosis component to find the root cause of the fault.

Logging generally is dependant on the component that is being monitored and the system it is deployed in. It is up to the designers of the system to specify the appropriate techniques for each component or subsystem based on the requirements. Here, the HUMS standard only gives implications of what possible techniques for data logging can be used for different use cases.

## 5.4 Failure Data

A system deploying the HUMS data standard should also have the ability to collect failure information that occur on different components throughout the system. Failure information in the standard is split into four main categories namely; failure detection, failure acknowledgement, failure rectification and predicted failure.

## 5.5 Monitored Parameters

In order to have an overview of the monitored components deployed on a platform, different parameters of each component have to be monitored. Parameters which are to be monitored can include an absolute limit, a limit that is exceeded within a specified range of time, a limit that has been exceeded multiple times or a drift of a value over a period of time.

## 5.6 Examples of Platform Status Data Collection

The remaining of the standard contains a logical breakdown of a vehicle platform and a selection of data which can be useful to each aspect of the vehicle. A vehicle platform is broken down into the following structure; core platform network including usage data, navigation, communications, video, power, HUMS, automotive, survivability, lethality and special-to-role components.

The types of data that are to be collected from each branch of the vehicle platform include:

- Periodic Data
- Threshold Exceeded Data
- Event Data
- Fault and Failure data

Table 5.1 below will give some examples of different components and the expected data to be collected from them.

Table 5.1: Example of components Platform Status Data.

| **Component** | **Periodic Data** | **Threshold Exceeded data** | **Event Data** | **Fault Codes** |
|---|---|---|---|---|
| **Network Ethernet Switch** | | - High traffic | - State changes<br>- Cumulative hours | - Loss of communications |
| **Satellite GPS** | - Position<br>- Time<br>- Accuracy | | - State changes<br>- Standby<br>- Low power mode | - All fault codes |
| **Driver's Camera** | | | - Status<br>- Usage<br>- Channel information | |
| **Battery Management System** | - Voltage<br>- State of charge<br>- Temperature | - Voltage<br>- State of charge<br>- Temperature | - Cumulative hours | - Battery faults |
| **Engine Control Unit** | - Engine Speed<br>- Torque<br>- Oil level<br>- Oil Pressure | - Engine Speed<br>- Torque<br>- Oil level<br>- Oil pressure | - Mode of operation<br>- Cumulative hours<br>- Cumulative power cycles | - All engine faults<br>- Databus failure |

The components displayed in table 5.1 are from different subsystems of a vehicle platform. The table is meant to give some examples of what data could be collected from across the platform regarding the health of the components.

## 5.7 Usage & Condition Monitoring System

Based on the HUMS standard developed by the UK GVA, the NATO GVA developed a data model that is to be used on military vehicle platforms called Usage and Condition Monitoring System (UCMS). The data model is to be used to represent the health status and faults and failures that occur on different components across a military vehicle platform. A UML diagram of the data model is presented in Fig. 5.2.

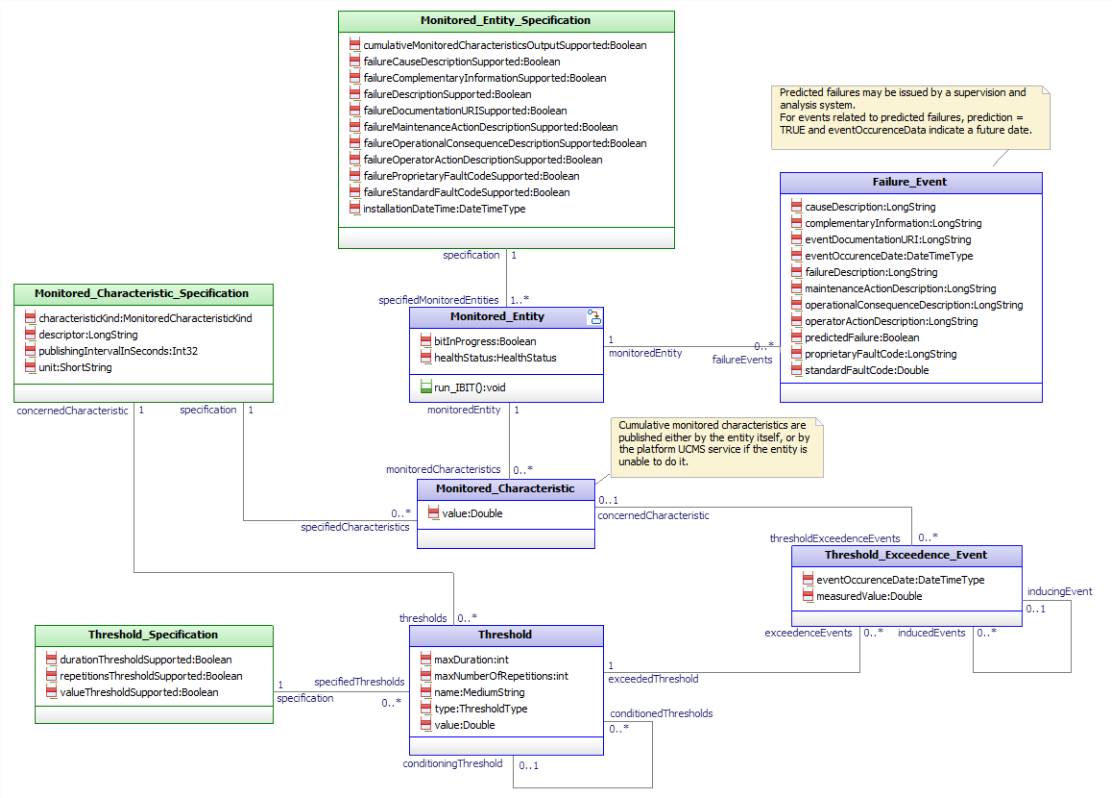



FIGURE 5.2: Class Diagram of UCMS[1].

The UCMS data model module contains multiple classes that are to be used for health representation of components and subsystems across a military vehicle platform. The following classes are currently part of the data model:

- **Monitored Entity**: Represents what component or subsystem is being monitored and contains information on the health status of the component or subsystem.
- **Monitored Entity Specification**: Contains information on the specifications or static data that are related to the monitored entity.
- **Monitored Characteristic**: Represents which aspect of the monitored entity or subsystem is being monitored. It contains information on the value of the aspect that is being monitored.

[1]https://natogva.org

- **Monitored Characteristic Specifications**: Contains the static data that is related to the monitored characteristic, such as the unit and description of the monitored characteristic.
- **Threshold**: Contains information on the thresholds that are related to the monitored characteristics, such as the value of the threshold and the threshold type.
- **Threshold Specification**: Contains the static information related to the threshold.
- **Threshold Exceeded Event**: Represents when a threshold is exceeded and records the time that this event occurred and the value that was recorded of this event.
- **Failure Event**: Represents an event where a failure of the monitored entity has occurred and contains a description of the failure along with other information that can be useful to the operators.

The UCMS data model has been developed; however, it has not been deployed yet to vehicles. For that, an evaluation of this data model would prove useful from a health monitoring and fault diagnosis point of view since it can give a complete overview of the data that can be expected from different fault diagnosis techniques. The evaluation can illustrate whether data model is generic enough to be able to represent the health status of a wide variety of different components.

Naturally, vehicle components are fundamentally different than components that are to be deployed on robotic platforms. Vehicle components, especially military vehicle components, are much more durable and are manufactured for higher reliability since they are to be deployed in harsh environments and also, to give confidence to the crew operating these components since they may be putting their lives in danger. However, faults may still occur and they need to be handled quickly to avoid breakdowns or complete failures of components or subsystems. Moreover, as mentioned in the beginning of this chapter, military vehicles are constantly upgraded to cope with the technological advancements and to have tactical advantage being on the field. Therefore, adopting a health standard that enables easier and faster upgrades to the vehicle platform would be crucial. This requirement is also shared with robotic platforms. Because of the higher availability of components such as sensors and actuators in the market, robotic platforms receive component upgrades and get reconfigured frequently to adopt to different use cases.

The data model that represents the health of components on vehicle platforms or robotic platforms needs to be flexible enough to represent this wide range of different components

and also be capable of being used with different types of fault diagnosis methods. The fault diagnosis methods discussed in chapter 2 can be applied to both vehicle components or robotic platform components. For example, one can use model-based diagnosis to create an analytical model of a mechanical engine and use it to diagnose engine faults. At the same time, one can also use model-based diagnosis to create an analytical model of an electric motor deployed on a robot and use the developed model for fault diagnosis as well.

Furthermore, since one of the main goals of NGVA is interoperability [33], and since more and more robots are being deployed in the field, applying the relative data models to robotic platforms would definitely bring a great advantage to the robots operators, and would enable easy integration between both the vehicle and robotic systems, in the case where, for example, military vehicles are deploying robots for specific missions.

For these reasons, the UCMS data model module was chosen to be deployed and evaluated on a robotic platform for this study, which will be discussed later in chapter 6.

# Chapter 6

# Experimental Evaluation

So far, we have discussed and presented the different aspects required to develop a non-intrusive remote fault diagnosis system for unmanned ground vehicles. A system architecture that satisfies the requirements of the system has been designed in chapter 3, Fig. 3.4. The different components within the architecture; fault diagnosis and health messages representation and standardization, have been also addressed throughout chapters 4 & 5. In this chapter, the implementation of the system architecture and the evaluation of the system performance is presented.

## 6.1 Implementation

The implementation of this project can be split into the following categories:

- Hardware selection: The black box is the heart of this project, since it will be responsible for handling the computations regarding health monitoring and fault diagnosis, logging of health data and communicating with the off-board remote PC.
- Software implementation of selected diagnosis method: Based on the capabilities of the selected black box, and the available software implementations that can help realize the selected diagnosis methods from chapter. 4, the software implementation should be carefully selected and evaluated.
- Mapping from diagnosis to UCMS: From the presented UCMS data model in chapter. 5, the messages produced by the fault diagnosis component of the black box should be mapped to the UCMS standard.

- Communication with remote PC: The selected method for communication to the remote-PC which was discussed in chapter. 2 and 3, Data Distribution Services (DDS) is to be implemented and tested within our proposed architecture.

### 6.1.1 Hardware Selection

The hardware required to implement this system were a robot running ROS, a mini-PC to be attached to the robot with a storage device that can be connected to it for data logging, and a remote-PC to receive updated health data about the robots. For the scope of this project, the selection of both the robot to be used and the remote-PC were irrelevant, since the goal of this project is to create a general solution for fault diagnosis that can integrate easily with any robotic system. The available robots provided by Hochschule Bonn-Rhein-Sieg were suitable to be used for this project and can serve the purpose of being the test robots. As for the remote-PC, any PC can be used while having the ability to run the required software which will be discussed in the following sections.

For this, the most relevant hardware components to be selected for this project were both the mini-PC and the storage device. Seeing that there are many available mini-PCs in the market currently, the selection was made based on the specifications of the mini-PC, its compatibility with storage devices, power consumption, the ability to run both ROS, to connect to the robot, and DDS. A summary of the different mini-PCs considered for this project are given below.

TABLE 6.1: BeagleBone Black specifications[1].

| **Mini-PC** | **Specificatoins** | **Power Consumption** | **ROS** | **DDS** |
|---|---|---|---|---|
| BeagleBone Black | - ARM Cortex-A8<br>- 512MB DDR3 RAM<br>- 4GB 8-bit eMMC<br>on-board flash storage<br>- 3D graphics acceleratorNEON<br>- 10/100 Ethernet<br>floating-point accelerator<br>- 2x PRU 32-bit microcontrollers | 210-460mA @5V | Yes | Yes |

TABLE 6.2: Raspberry Pi 3 Model B specifications[2].

| **Mini-PC** | **Specifications** | **Power Consumption** | **ROS** | **DDS** |
|---|---|---|---|---|
| Raspberry Pi 3B | - 4x ARM Cortex-A53, 1.2GHz<br>- GPU: Broadcom VideoCore IV<br>- RAM: 1GB LPDDR2 (900 MHz)<br>- Networking: 10/100 Ethernet,<br>2.4GHz 802.11n wireless, Bluetooth<br>4.1 Classic, Bluetooth Low Energy<br>- Storage: microSD | 260-730mA @5V | Yes | Yes |

[1]https://beagleboard.org/black

Table 6.3: UDOO x86 specifications[3].

| Mini-PC | Specificatoins | Power Consumption | ROS | DDS |
|---|---|---|---|---|
| UDOO x86 | - 32-bit ARC core 32 MHz<br>- Intel HD Graphics 405<br>Up to 700 MHz<br>- 8 GB DDR3L Dual Channel<br>- 32GB eMMC Storage | 2A @12V | Yes | Not supported |

Tables 6.1, 6.2 and 6.3 give an overview of some of the available mini-PCs that were considered for this study. Seeing that the UDOO x86 has not been tested with DDS, and that it draws more power than the other two selections, it was disregarded for this project. As for comparing the BeagleBone with the Raspberry Pi Model 3, it is obvious from the specifications that the Raspberry Pi has an advantage. Also, looking into storage devices that would be compatible with both mini-PCs, it would have been advantageous to have been able to attach solid state drives (SSDs) to them because of their fast reading and writing speeds, however, it was found that both mini-PCs do not support SSDs. For this Hard-Disk drives (HDDs) were considered to be attached to the mini-PC. Looking into possibilities of attaching HDDs to the mini-PCs, it was found that, since it is not possible to attach the HDDs to the mini-PCs directly through the serial attachment (SATA) link because of limited power capabilities, the HDDs needed to be attached through USB. This still sometimes produces problems in the mini-PCs if low power is provided to the HDD. This problem was addressed by the Western Digital (WD) and Raspberry Pi community by having a WD HDD that is optimized for usage with the Raspberry Pi. These reasons made the selection of the Raspberry Pi and a WD HDD with a capacity of 375GB to be used as the black box for this project [4].

### 6.1.2 Software Implementation of Selected Diagnosis Method

The two selected fault diagnosis methods were consistency-based diagnosis (CBD) and sensor fault detection and diagnosis (SFDD) as presented in chapter 4. As discussed earlier, CBD will be examined in this study and compared to the results of other implementations of SFDD. Looking into existing implementations of CBD that are based on ROS, some studies were found. As discussed earlier in chapter 2, Zaman et al. [13], developed a model-based diagnosis system for ROS-based robots using CBD. However, it was found that their implementation was computationally expensive, since it deployed many observers as nodes in ROS on top of the robotic system which caused it to overload. The experimental results of this implementation were discussed previously in [39]. Another implementation mentioned in [39] was also a ROS model-based CBD diagnosis

[2]https://www.raspberrypi.org/products/raspberry-pi-3-model-b/
[3]https://www.udoo.org
[4]https://www.wdc.com/products/wdlabs.html

tool which addresses the problem of overloading the CPU of the robot with many nodes by publishing all observations onto one node, lowering the computational complexity of the method. This work is addressed in [40]. This method is more suitable for our project since the whole diagnosis system is to be deployed on the black box and needs to be low in computational complexity. A detailed explanation of the tool is given here since it will be used for modelling our components for the experimental evaluation.

#### 6.1.2.1 Model-based Diagnosis Tool

As explained earlier in chapter. 4, consistency-based diagnosis is comprised of three main components, a system description (SD), components (COMP), and observations (OBS). Since the diagnosis is to be applied to a ROS-based system, the system description will be the model of the expected behaviour of ROS components such as nodes or topics. The components will be the ROS nodes and topics. Finally the observations will be the actual data produced by these nodes and topics.

To create a system description, the tool deploys different observers to the robotic system that are modelled using a configuration file. The following observers are available to the tool[5], and if required more observers can be added to the system as plug-ins.

- Activated: This observer is used to check whether a node in ROS is active or not. When a node is active, the observer produces an observation that is greater than or equal to 0. If not, the observation produced is less than 0.
- Hz: A frequency observer that is used to monitor the rate at which a topic is published. Based on the modelled description, a hypothesis check can be used to verify whether the frequency of a topic is less than or greater than a specified value.
- Time-out: An observer that is used to trigger a negative observation if a topic does not publish a message within a specified time threshold. This is useful for when a node is active and behaving correctly, but the topic being published by that node is behaving incorrectly.
- Resource: This observer can be used to monitor the CPU and memory usage of each specified node. This can be useful to make sure that the CPU and memory of the robot do not pass a certain threshold that can slow down the robot's performance.
- Network: A monitor for the usage of the network. Both the transmission and receiving rates of the network can be modelled and observed using this observer. This can be useful to make sure the network on the robot is not overloaded.

[5] http://www.ist.tugraz.at/ais-wiki/model_based_diagnosis

- Score/Value: An observer that is used to monitor if a ROS topic is publishing a float value between a certain range or threshold. A hypothesis check can also be used here to produce an observation.
- Moving: This observer takes in a movement topic as input and checks to see if the robot is moving or not. States can be assigned to different observations for example if a robot is idle or moving.
- Movement: Another observer to check movement in a robot. However, this robot takes in two movement topics as an input, for example, odometry produced by the robot, and IMU sensor values. The observer then differentiates both readings until both readings are accelerations and checks if the difference between both topics is close to 0, based on the noise produced by each sensor.
- Timing: Is used to check if a timing between one message from a certain topic and another message from another topic is between a certain time threshold. This can be used for example when a goal is supposed to be reached for example within a certain time period.

As noted, some observers can be used to observe nodes and others can be used to observe topics. Furthermore, as stated some observers require a hypothesis check to be performed in order to check if the observed value is between a certain threshold or at an exact value. For this, the available hypothesis checks are as follows:

- Student-T test: This hypothesis check is used to perform a Student-T test on the observation, it requires a mean value and a standard deviation. If the observation is within the range of this mean and standard deviation, the observation produced is equal to or greater than 0. If not, the observation is less than 0.
- Gauss: A test is performed on an observation to check if the observation is within the range of the Gaussian curve.
- Exact: Checks if the observation matches an exact value.
- Not: Checks if a value is not what is specified in this hypothesis check.
- Greater/Less than: Performs a test on the observation to check if it is greater or less than a specified value.
- In between: Performs a check to see if the observation is between the specified interval.
- Not in between: Performs a check on the observation to make sure the value is not in between the specified range.

Naturally, if sensors are connected to the robot and their data is being published on ROS topics, the data tends to be noisy. Therefore, subjecting these readings directly to a hypothesis check will definitely produce many false positives, which is an unwanted behaviour. These readings need to be filtered in order to approximate the true value which these sensors are producing. The tool used for this project allows for different filters to be applied to the readings and also, if required, apply these filters on a specified window size by the user.

The filters available are the following:

- Mean filter: The mean value of the given readings are produced after applying this filter.
- Median filter: A median filter can be applied to the readings, with a specified window size if required.
- K-means filter: A K-means filter can be applied while providing a K value and a window size.
- Exponentially Weighted Moving Average (EWMA) filter: An EWMA filter can be applied to readings while providing a decay rate and a window size

So far, the observers used to create a description of the system have been discussed along with their required hypothesis checks and their filters. An example of how a system description is developed using a configuration file is given in Fig. 6.1.

```
- type: scores
  main_loop_rate: 1.0
  use_global_subscriber: true
  topics:
  - name: /signal
    filter:
      type: ewma
      window_size: 20
      decay_rate: 0.05
      deviation_type: std_deviation
    states:
    - state: 'ok'
      number: 1
      score:
        type: nominal_value
        value:
          type: gauss
          mean: 15.0
          std_deviation: 0.5
```

FIGURE 6.1: Example of partial system description.

A system description will contain all the expected behaviours of the ROS nodes and topics that are required to be monitored. In Fig. 6.1, the topic with the name signal is

being monitored with the score observer while applying an EWMA filter with a window size of 20 and a decay rate of 0.05. A state named OK is assigned to the observation if it is within the range of the hypothesis check that is applied to it. In that case, a gaussian hypothesis check is applied on the ROS topic. If the observation of the topic signal is within the specified Gaussian range then the output of the observation will be equal to or greater than 0, otherwise, the observation will be less than 0.

When the nodes are started on the robot, data starts being produced on ROS shared network through topics. To start collecting the observations, the observer node must be started as well which uses the configuration file containing the system description. As all the ROS nodes and topics contained in the system description file are being observed, the observations are being produced on a ROS topic called observations. The observation message produced is in the following form shown in Fig. 6.2

```
tug_observers_msgs/observation_info[] observation_infos
  std_msgs/Header header                  # ros header message
    uint32 seq
    time stamp
    string frame_id
  string type                             # observation type, e.g. hz
  string resource                         # observed topic and/or node name
  tug_observers_msgs/observation[] observation
    int32 GENERAL_OK=0
    int32 GENERAL_ERROR=-1
    int32 NO_STATE_FITS=-2
    int32 NOT_AVAILABLE=-3
    int32 TIMEOUT=-4
    string observation_msg                # a brief message
    string verbose_observation_msg        # description of taken obs.
    int32 observation                     # observation result
```

FIGURE 6.2: Observation message produced by observer node [43].

The observation message contains information such as a time stamp, what type of observer is deployed, which node or topic are being observed under the resource field, and the observation message itself. For our project, the observation message will need to be later mapped into the UCMS standard to be transmitted to the remote-PC.

The flow of information from ROS nodes and topics to observations are shown in the following Fig. 6.3.

It is noteworthy to mention that in order for the observers that monitor the resource usage of the nodes such as CPU or memory usage, a separate node must be run named resource monitor that uses the robot's system own tools to gather information on the usage of the required observers.

After an observation is produced, the diagnosis engine can be triggered. As mentioned earlier, a CBD problem consists of three components, a system description (SD), a list

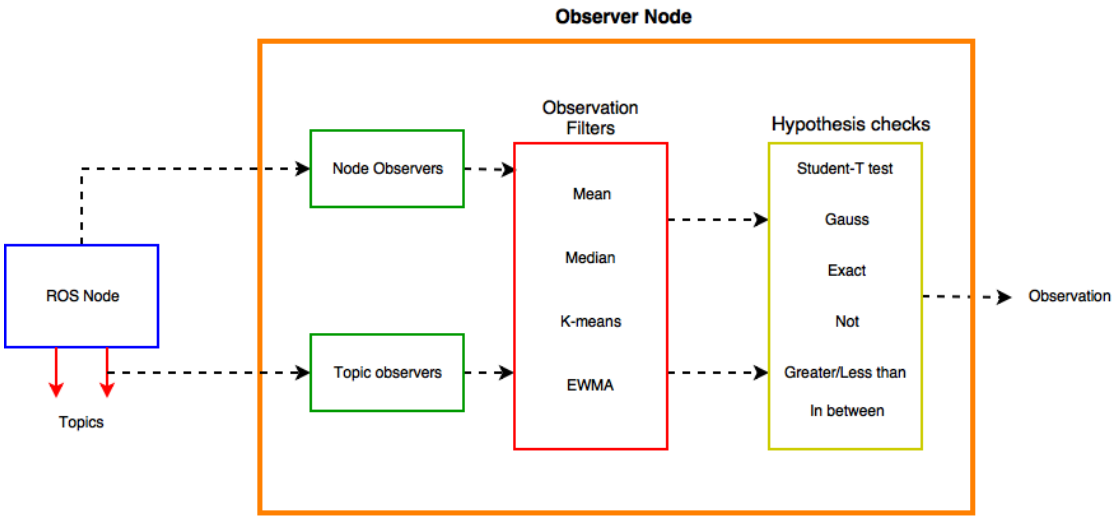



FIGURE 6.3: Information flow from ROS nodes and topics to observations.

of components (COMPS), and observations (OBS). So far, we have gathered the system description in the configuration file mentioned, and the observations are generated from the actual robot. The only thing missing is the components description and the connection between the components. Using this model-based diagnosis tool, one can provide another configuration file containing the components of the system and their connections. In this case, the components of the system are the names of the nodes and their related topics represent the connections between the nodes. The output of the diagnosis is a faulty node once a faulty observation is produced.

Having all the required components of a consistency-based diagnosis system, one can now use this tool to model and diagnose a robotic system. Both the observation and diagnosis nodes will be run on the black box and will gather information on the robot non-intrusively. The evaluation of this method will be presented in the experimental evaluation section of this chapter.

### 6.1.3 Data logging

To store the health data of the monitored components, the storage device was utilized. An already existing method to log data produced by the ROS nodes, ROSbag [6], was considered. ROSbag is a tool developed by ROS, where the data produced by the different ROS topics, can be logged in a binary format. This tool can be used for different use-cases, such as recording motion of a robot, or recording sensor readings for simulation purposes. The drawback of using this tool for our system is that it will record all the data produced by the monitored components at its operating frequency. This can fill

[6] http://wiki.ros.org/rosbag

the data storage device in a small period, if the data produced by the components has a high frequency and band width. Also, since the components are being monitored, there is no need to record all the healthy data that is being produced. Taking inspiration from the HUMS standard and UCMS data model module, periodic logging and event-based logging of the monitored components can be more practical. This can be helpful for reducing the volume of data stored since the data produced by different components is being recorded periodically. When a fault or failure occurs, the messages produced by the fault diagnosis tool can also be logged, therefore having a complete overview of the monitored components. For this, a simple python script can be deployed on the Raspberry Pi, that contains the different topics that are required to be monitored and a timer, which can be used to set the value of the periodic logging.

### 6.1.4 Mapping from Diagnosis to UCMS

Having a ROS-based diagnosis system now running, the observations and messages produced have to be mapped onto the selected standard UCMS, which was discussed in chapter 5. The advantage of mapping messages to such a standard is that it helps in creating interoperability between different system components. For example, in our proposed system, if the diagnosis method is to be changed, this change will not affect the operation of the rest of the system, since the output of the other diagnosis methods will also be mapped to the existing standard, keeping the on-board and off-board components independent.

In this section we present the proposed mapping from the output of the diagnosis system to the UCMS standard. This mapping will also help in evaluating the standard since it has not been deployed yet.

The following classes are available from the UCMS standard and can also be seen in Fig. 5.2 (the full data model can be seen in appendix A):

- Monitored Entity/Monitored Entity Specification
- Monitored Characteristic/Monitored Characteristic Specification
- Threshold/Threshold Specification
- Threshold Exceedence Event
- Failure Event
- Installed Software

- Preventive Maintenance Recommended Event

To help understand the mapping from the observations and diagnosis, The following example diagram of ROS nodes and topics is given in Fig. 6.4.

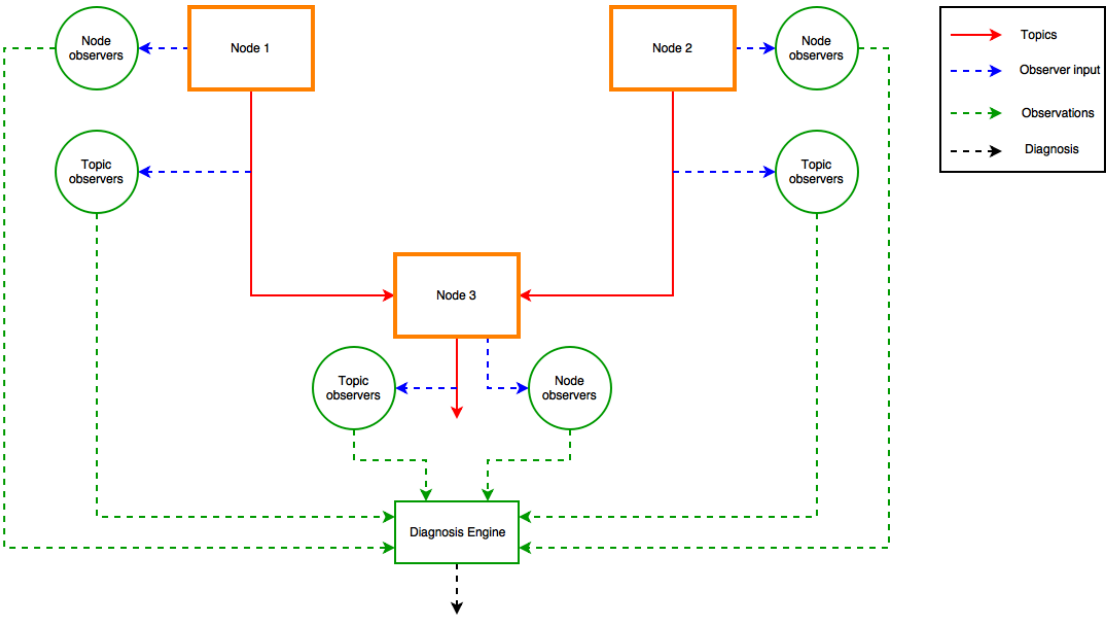



FIGURE 6.4: Example of ROS-based systems including nodes and topics along with observers and diagnosis engine.

Fig. 6.4 presents an example of a robotic ROS-based system containing three nodes along with the fault diagnosis system. The observers are used to monitor the nodes and topics, and the output of the observers are used as inputs to the diagnosis engine to produce a diagnosis once a faulty observation occurs. Nodes can represent either hardware or software components. For example, a camera can be connected to a robotic platform, which will be presented by a ROS node, and the video data output of the camera would be published on a topic that contains the video information. Our goal now is to use the UCMS data model to represent the health of these components so that they can be later transmitted to a remote-PC. The proposed mapping is shown in Fig. 6.5 and table 6.4.

TABLE 6.4: Mapping of CBD output to UCMS topics.

| **Consistency-based Diagnosis output** | **UCMS mapping** |
|---|---|
| Observer Resource | Monitored Entity |
| Observer type | Monitored Characteristic |
| Observer threshold | Threshold |
| Observer threshold exceeded | Threshold Exceedence Event |
| Diagnosis | Failure Event |

From this mapping, each ROS node will be mapped as a Monitored Entity in UCMS along with Monitored Entity Specifications. Each ROS node produces topics that publish

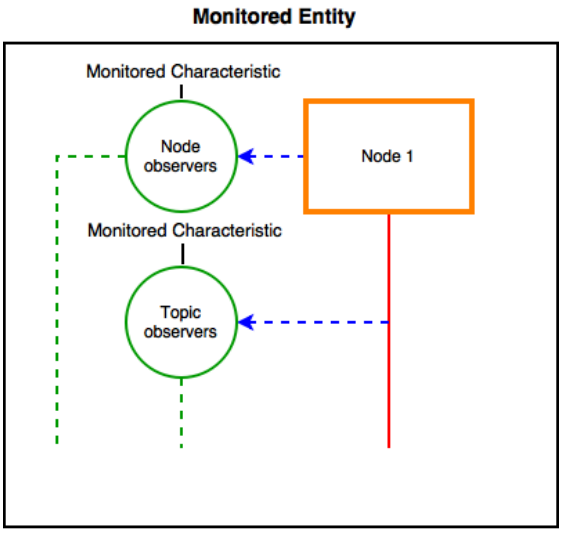



FIGURE 6.5: Mapping of nodes and observers.

information to the robot network that can be used by other nodes or can be used to activate some actuator for example. Each ROS node and its related topics can have multiple observers, which will be mapped to the Monitored Characteristic and Monitored Characteristic Specification topics. The different observers will be easily distinguishable since they will have different source IDs. The thresholds that will be applied to each observer (if applicable) will be mapped to the Threshold and Threshold Specification topic. Once an observer reports a threshold has been exceeded, this will be mapped to a Threshold Exceedence Event topic. If a faulty observation is observed, and the diagnosis engine produces a node that is marked as faulty, this will be mapped to the topic Failure Event.

### 6.1.5 Communication with Remote-PC

Once the observations and diagnosis are mapped to the UCMS data model, they are ready to be transmitted to the remote-PC. Since the NGVA community, which developed the UCMS data model module uses Data Distribution Services (DDS) for data exchange, it was also used in this project to communicate with the remote-PC. The specific DDS software used in this project was developed by RTI[7]. Some of the advantages and key features of using DDS will be explained in this section.

DDS provides a databus for applications which require real-time data exchange. Distributed systems, such as our project here, require real time data exchange, between the black box and the remote-PC for example. Applications such as this project as well, require reliability and independence from the robotic platform itself, since the goal is

[7]https://www.rti.com

to monitor the health of the robot, regardless of its state and always give the user an overview of the behaviour of the robot and its monitored components. Using DDS in this project helps achieve this goal, since it creates independence between the black box and the remote-PC. Moreover, since the data would be available on the databus, the applications will only deal with the data, instead of needing to deal with another application, along with its dependencies. Another advantage of using DDS, is the availability of Quality of Service (QoS) policies, which allow the publishers, subscribers, data readers or writers of the applications to adopt certain behaviours. Some of these attributes include:

- Durability: a parameter that can specify if the data that is produced by a certain entity, such as a data publisher, is available for new or late joiners such as applications requiring this data.
- History: it is a parameter that specifies the volume of data to store on the data bus to be available for other subscribers.
- Reliability: a parameter that deals with how the network deals with lost samples.

Furthermore, DDS is available to be deployed on different platforms such as Linux, Windows or MacOS, and also, can be programmed in multiple languages such as C++ and Java. Fig. 6.6 shows a conceptual diagram of how DDS works.

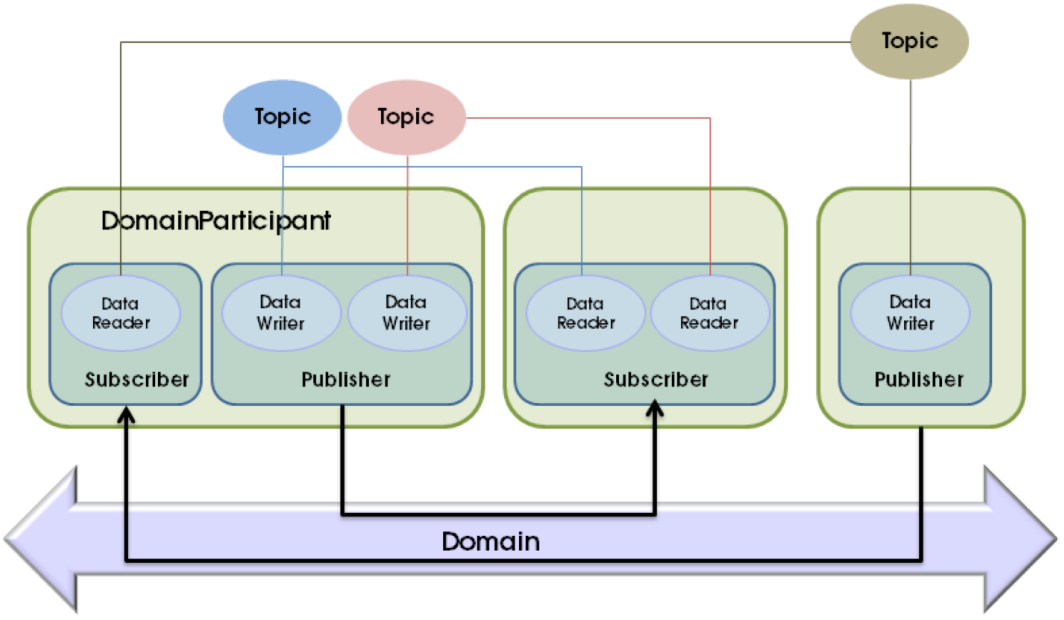



FIGURE 6.6: Schematic of DDS [44].

As mentioned earlier in chapter 2, DDS uses a publish-subscribe model to transfer data between applications. Publishers and subscribers have data-writers and readers respectively. A publisher uses the data writer to actually write the data to the data bus, and a subscriber uses a data reader in a similar way. For applications to communicate with each other, a publisher must write data using the data writer to a specific topic, and the subscriber can access this data using its own data reader by subscribing to the same topic. Both the publisher and subscriber must be on the same domain in order for them to communicate. Each application that utilizes DDS has a domain participant in order to enable them to communicate on the specified domain.

To summarize this section, DDS is a middleware that is used by applications that require real-time data exchange. DDS uses a publish-subscribe model to allow applications to communicate with each other. Each pair of publisher and subscriber can have Quality of Service attributes assigned to them. QoS attributes can be viewed as a contract between the publishers and subscribers, the publishers offer a level of service, and the subscribers can request also a level of service using the available QoS attributes. The DDS middleware is then responsible for determining if what the publisher offers is relevant to what the subscriber's requests. In order for publishers and subscribers to communicate, the applications need to participate in the same domain that is provided on the databus.

After the messages that have been generated from the diagnosis engine are mapped to the UCMS standard, they are published on the databus using the specified topics from the UCMS data model. Each node, which is represented as a monitored entity along with its specifications, characteristics, thresholds, and failure events can use the same data writer for each specified topic, since they can be written to the databus as different samples using unique source IDs. The details of what QoS attributes will be used for each topic will be discussed in the experimental setup section of this chapter.

## 6.2 Experimental Evaluation

Now that all the important elements of the designed system architecture have been implemented, the performance of the system needs to be evaluated. In this section, the experimental setup, results and evaluation will be presented. The goals of the experiments is to find out how well the system performs with regards to:

- The CPU and memory usage of the black box while the full system is running.
- The accuracy of the selected diagnosis method.
- The network usage.

- How accurate does the standard represent the health of the monitored components.

### 6.2.1 Experimental Setup

The experiments in this section were performed to answer the mentioned goals. The experiments contained the following components:

- A service robot running ROS: Care-o-bot 3 [8] provided by Hochschule Bonn-Rhein-Sieg.
- The black box: A Raspberry Pi 3 with a 375GB HDD
- A laptop running linux to receive the published health messages.

Since the goal of this project is to be able to monitor the health of both hardware and software components of a robot, three components were selected to be modelled and used in this experiment. A USB camera connected to the robotic platform that publishes raw frames on a ROS topic, a microphone that publishes data as well on a ROS topic, and a third node named signal, which is a software node that subscribes to both the camera and microphone topics and publishes a noisy signal with a certain amplitude. Fig. 6.7 shows the node connections and topics related to them.

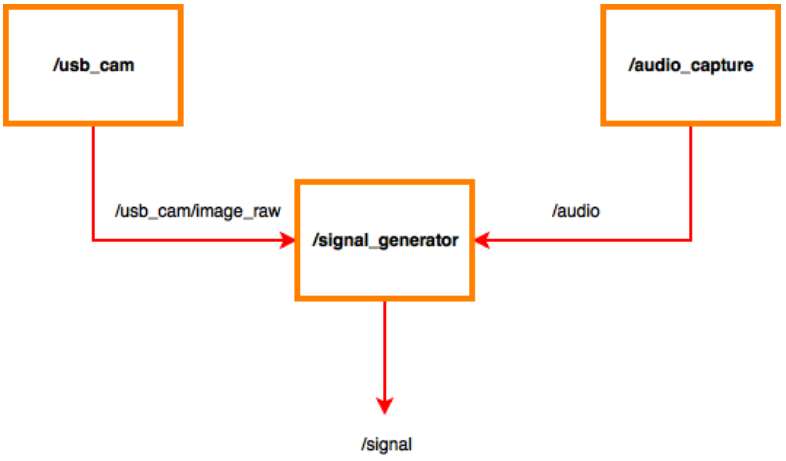



FIGURE 6.7: Setup of ROS nodes and topics for experiment.

Now that the three nodes to be monitored are set up, models need to be created using the CBD tool discussed earlier. Based on the available observers, the expected behaviour of the nodes can be modelled. The selected observers for the topics and nodes are shown in Fig. 6.8.

[8] https://www.care-o-bot.de/de/care-o-bot-3.html

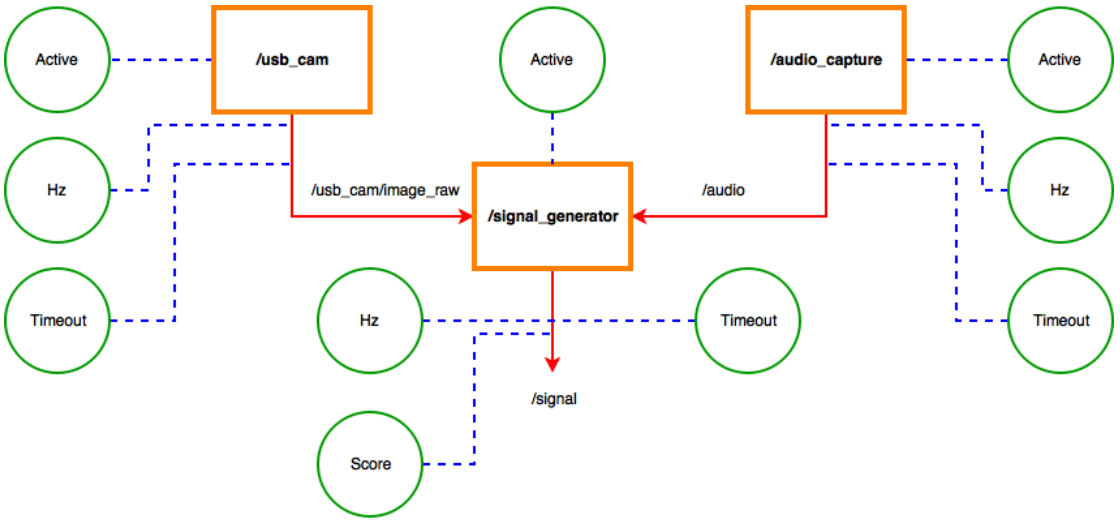



FIGURE 6.8: Observers applied on nodes and topics.

The setup so far has three nodes, two nodes that represent hardware components and one node represents a software component. All three nodes are run on the robotic platform. On the black box side, ten observers are deployed for the three nodes as seen in Fig. 6.8. The active observer monitors if the node is active or not. The timeout observer monitors the topics published by the nodes, and if the messages are not published within the specified threshold, a fault is declared. The timeout threshold here was set to 1 second for all nodes. As for the score and frequency observers, as mentioned earlier in the software implementation section, require a filter and a hypothesis check (see Fig. 6.3). To filter the inputs, in this case, the frequency reading of the topics for example, a window size and a type of filter needs to be selected. To find out the behaviour of each filter and how they react to the change in window size, the following experiment was performed.

#### 6.2.1.1 Filters Vs Window Size

The signal generator node was set to publish a signal with an amplitude of 15 with noise of +/- 0.5. A score observer was deployed on the node with the following parameters for the hypothesis check:

- Hypothesis check: Gaussian
- Mean: 15
- Standard deviation: 0.2

The variables of the experiment were the window size and the filters. The window sizes were varied from 2-100 samples per window. The experiment was ran for 100 seconds for

each filter. For each window size, the number of fault positives were recorded since no faults were injected. False positives occur when no fault is injected to the system, but the observers produce a fault. A sample of this test at window size 50 is shown in Fig. 6.9.

The parameters for the K-means and EWMA filter were set as follows:

- K-means filter K-size = 10
- EWMA filter decay rate = 0.05

The aim of this experiment is to find out what is the appropriate filter and window size to select based on the number of false positives. The lower the false positives the better, since we want to only detect when a true fault happens and filter the noise in the readings. The results of this test is shown in Fig. 6.10.

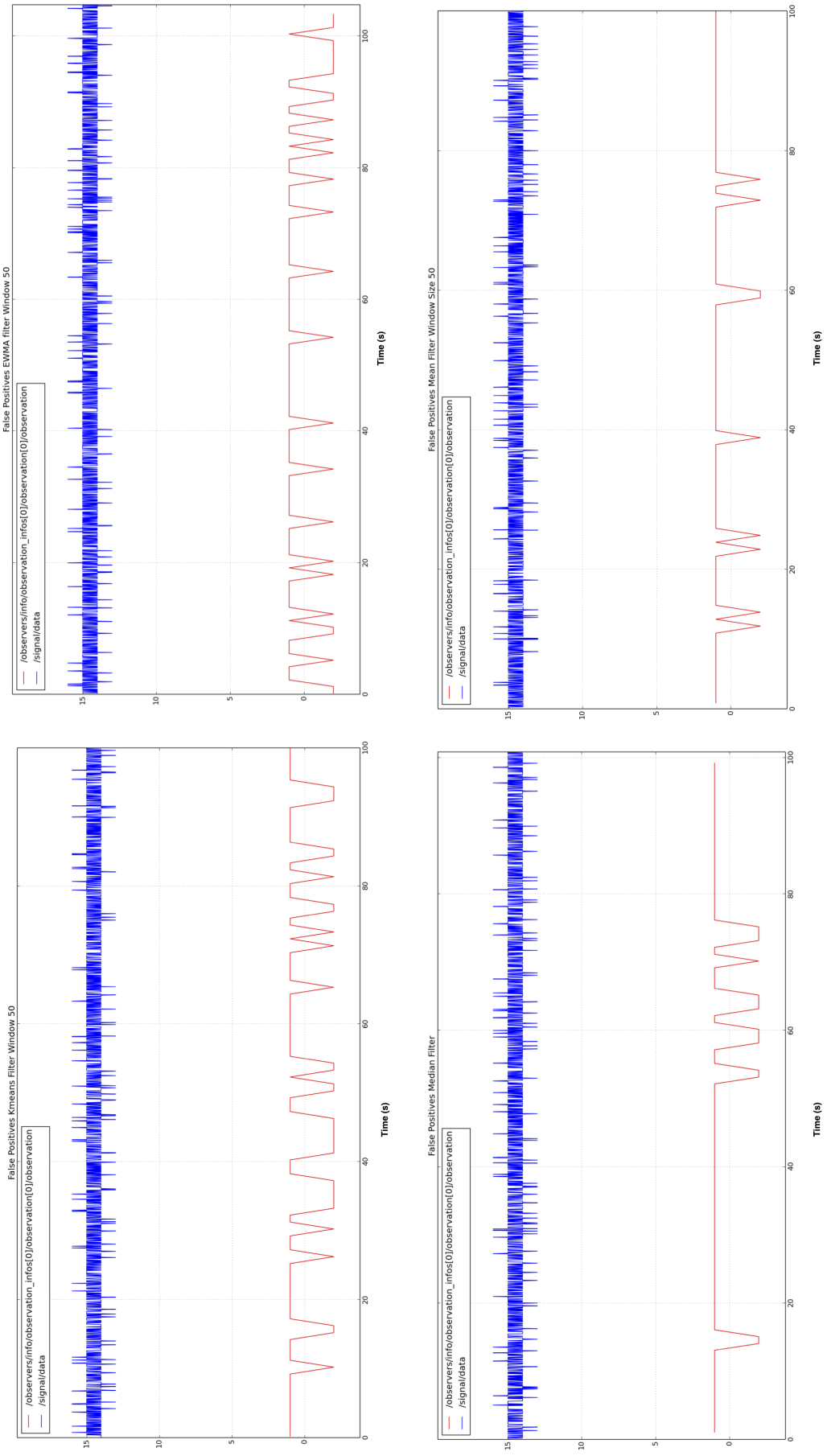



FIGURE 6.9: Different filters behaviour at window size 50.

The results show that as the window sizes increases, the false positives decrease for all the given filters. This is due to the fact that over a larger window, the subtle changes in the signal are not detected because more values are taken into consideration when applying each filter. The optimum case would be to have zero false positives, which based on this test alone would mean selecting a very large window size (>100). However, choosing a large window size causes another phenomena to occur, which is the delay in detection, or sometimes no detection at all. Fig. 6.11, shows the delay of detection with the mean filter at different window sizes. In this experiment, the hypothesis check's mean and standard deviation have been relaxed in order to account for large changes and not trigger any false positives.

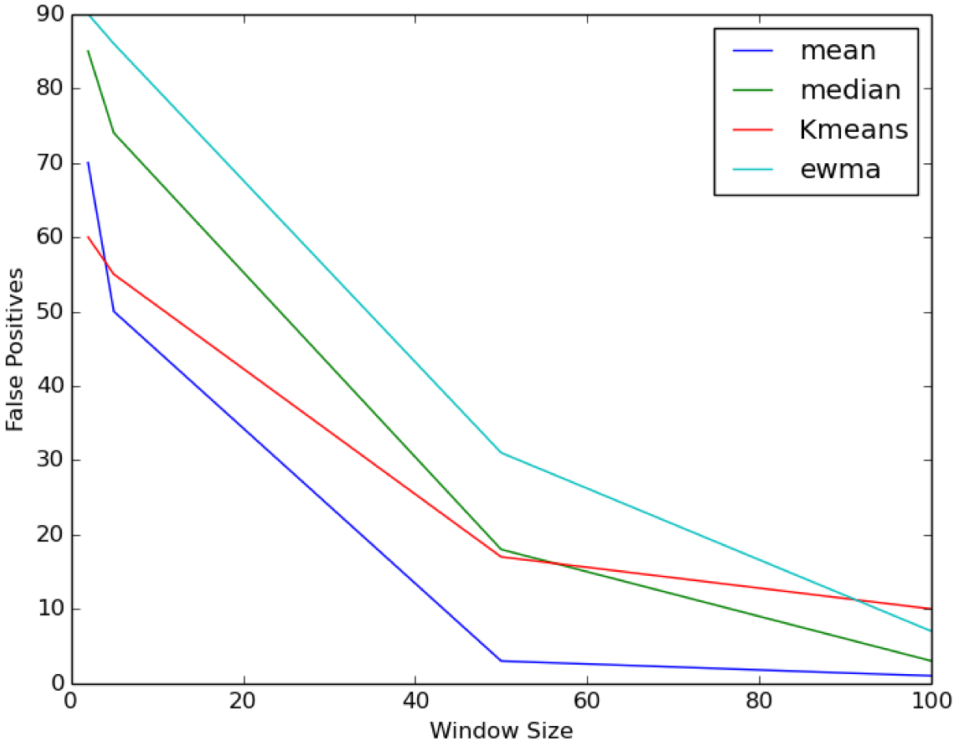



FIGURE 6.10: False positives vs window size for each filter.

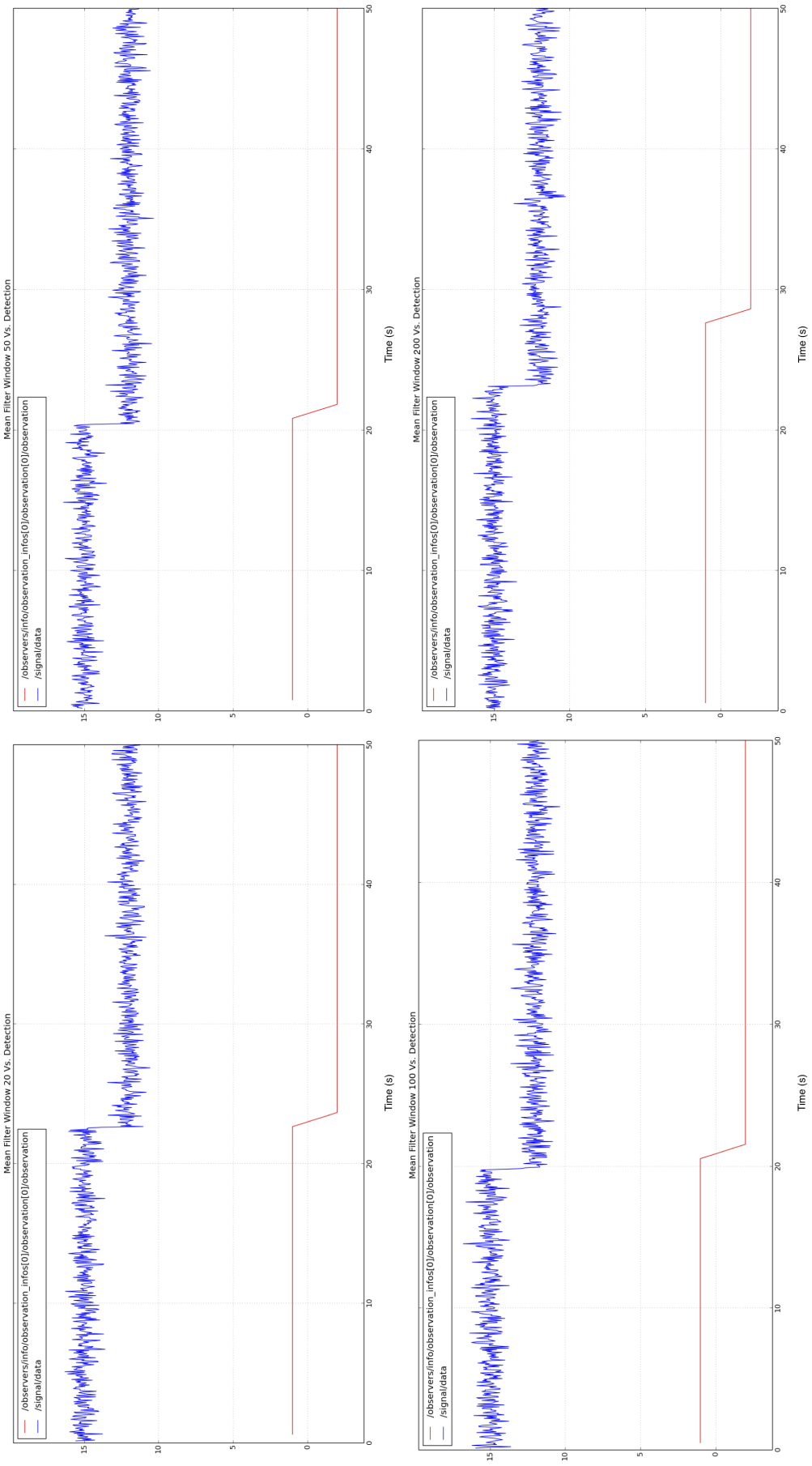



FIGURE 6.11: Delay in detection due to increase in window size.

After these tests were ran, the selected filter to be used for the main experiment was the mean filter, since it produced the lowest number of false positives, with regards to the set parameters of the previous tests presented. The selected window size was 50 samples, since it had a low number of false positives (see Fig. 6.10), and can report a fault within 1 second of its occurrence (see Fig. 6.11).

Finally, the chosen parameters for the observers deployed are shown in table 6.5. The nodes were mapped to the UCMS standard based on the mentioned mapping earlier in section 6.1.4. As for the Quality of Service attributes, it was proposed by the NGVA team to use the History, Reliability and Durability attributes for the specification topics. As for the failure event and threshold exceeded events, the recommended QoS attributes was History. The assignment of QoS attributes to UCMS topics can be seen in table 6.6.

Table 6.5: Assignment of observers to ROS nodes and topics.

| **Node/Topic** | **Observer parameters** | | | |
|---|---|---|---|---|
| | Active | Hz | Timeout | Score |
| /usb_cam | Applied | | | |
| /usb_cam/Image_raw | | Filter: mean<br>Window size: 50<br>Hypothesis: Gauss<br>mean: 15<br>std_dev: 0.01 | 1 second | |
| /audio_capture | Applied | | | |
| /audio | | Filter: mean<br>Window size: 50<br>Hypothesis: Gauss<br>mean: 30<br>std_dev: 0.01 | 1 second | |
| /Signal_generator | Applied | | | |
| /signal | | Filter: mean<br>window size: 50<br>Hypothesis: Gauss<br>mean: 15<br>std_dev: 0.01 | 1 second | Filter: mean<br>Window size: 50<br>Hypothesis: Gauss<br>mean: 15<br>std_dev: 0.5 |

Table 6.6: Qos attributes for UCMS topics.

| **UCMS topic** | **QoS attributes** |
|---|---|
| Monitored Entity | Durability, History |
| Monitored Entity Specification | Durability, History, Reliability |
| Monitored Characteristic | Durability, History |
| Monitored Characteristic Specification | Durability, History, Reliability |
| Threshold | Durability, History |
| Threshold Specification | Durability, History, Reliability |
| Threshold Exceedence Event | History |
| Failure Event | History |

#### 6.2.1.2 Experiment Scenario

The experimental scenario was as follows:

- The robot was ran with the 3 nodes discussed.
- The black box ran the observers and diagnosis nodes, along with the UCMS data publisher.
- Each Experiment lasted 5 minutes.
- Random faults were injected.
- True positives, false positives, false negatives, memory, and CPU usage were recorded.
- The experiment was repeated 10 times.

The faults injected were as follows:

- Killing of a node, results in an activated observer fault.
- unplugging the USB camera, which results in a timeout fault.
- Changing the frequency of publishing the camera, causes a Hz fault.
- Changing the frequency of publishing the signal, causes a Hz fault.
- Changing the magnitude of the signal value published, results in Score observer fault.

### 6.2.2 Experimental Results

After the 10 experiments were performed, the results were logged. The following metrics are used to measure the performance of the CBD algorithm in the system implemented for this study:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{6.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{6.3}$$

$$F1score = \frac{2 * (Recall * Precision)}{Recall + Precision} \tag{6.4}$$

Where TP (true positives), is the number of correctly detected faults. TN (true negatives) means when no fault is injected and no fault is detected. FP (false positives) is when no fault is injected but a fault is detected. FN (false negative) is when a fault is injected but not detected.

Accuracy determines the ratio between the correctly detected faults over the total observations. Precision calculates the ratio between the correctly detected faults over the total observations of faults detected. Recall determines the sensitivity of the CBD algorithm where it is the ratio between the correctly detected faults to the observations of detection. Finally, the F1 score is the ratio between the weighted average of precision and recall of the algorithm.

Table. 6.7, shows the final results. The complete configurations and results can be found in appendix B. It was noticed from the given results that the accuracy of the CBD Diagnosis was lower than expected, since the model parameters were based on the readings produced by the nodes and topics themselves. This low accuracy was found to be because of the connection method between the robot and the black box, and the hardware on the Raspberry Pi itself.

Table 6.7: Results of 10 Experiments.

| | |
|---|---|
| TOTAL Injected | 260 |
| TOTAL Detected | 390 |
| TOTAL True Positive | 240 |
| TOTAL false positive | 60 |
| TOTAL false negative | 35 |
| Accuracy | 71.64% |
| Precision | 0.8 |
| Recall | 0.88 |
| F1 score | 0.83 |
| Average CPU usage | 34% |
| Average Memory Usage | 145 MB |

Since the camera was publishing frames at a rate of 15 frames per second, it was using a bandwidth of 27 MB/s, and looking at the Raspberry Pi's maximum throughput, it was found to be 100Mb/s. Meaning that the Camera alone required 218 Mb/s at the minimum on the Raspberry Pi to be available, which is beyond it's hardware capabilities. It was found that using an Ethernet connection from the robot to the Raspberry Pi, the frames frequency of the camera dropped from 15 FPS to 12 FPS. Using wireless connection, the frames dropped from 15 FPS to 1 FPS.

After adjusting the parameters to accompany this drop of the frames, the tests were ran once more and a higher accuracy could be achieved which can be seen in table 6.8 (see appendix B.

Table 6.8: Results of Experiments 11-20 after parameter updates.

| | |
|---|---|
| TOTAL Injected | 290 |
| TOTAL Detected | 320 |
| TOTAL True Positive | 276 |
| TOTAL false positive | 23 |
| TOTAL false negative | 12 |
| Accuracy | 88% |
| Precision | 0.92 |
| Recall | 0.96 |
| F1 score | 0.93 |
| Average CPU usage | 32.5% |
| Average Memory Usage | 142.2 MB |

### 6.2.3 Evaluation

From the results achieved from the performed experiments, we can derive that deploying a complete diagnosis system on a mini-PC is achievable with a high degree of accuracy. The CPU and memory usage of the complete diagnosis system were found to be within satisfactory levels, this was due to the fact that consistency-based diagnosis is low in computational complexity. The only limitation that was found from this test was because of the hardware used, specifically, the network chip on the Raspberry Pi. It was found that using consistency-based diagnosis, we can model different hardware and software components which are deployed on a common robotic platform ROS. Consistency-based diagnosis has the advantage that it can be easily setup to monitor simple behaviours. However, if a complete robot model is to be developed, it would require a substantial amount of time to be able to model all the robots behaviours accurately. As for the goal of this project, which was to monitor the health of robot software and hardware components, this has been demonstrated through the given experiment.

Looking into the UCMS data model provided my NGVA, it was found that it can represent the health of the monitored components thoroughly, by the mapping that was proposed. However, some additions to the data model can be proposed. For components which produce noisy data, it would be useful to add a field to the monitored characteristics class mentioning the expected noise from that particular component. Also, it would be useful to have the filters used and their specified parameters also included in the standard as part of the characteristics as well. Furthermore, faults that occur on different components, are usually represented by the class “Threshold Exceedence Event". Thresholds can be sometimes exceeded due to a state change and not always due to a fault, therefore, a differentiation between state changes and faults should be considered. However, since the aim here is to only monitor the health of components, which usually means that components have a range to operate within, this did not affect the accuracy

of the representation. Finally, since part of keeping track of health of components over time is another goal of this project, it is recommended to also be able to represent the periodic logging value for each component for the user.

# Chapter 7

# Conclusion

In this study, a non-intrusive fault diagnosis system for unmanned ground vehicles was proposed, designed and implemented. The aim of the study was to create a diagnosis system that can monitor the health of different components, and that can be easily upgraded to accompany new hardware or software components being added to the robotic platform. The reason for choosing to non-intrusively diagnose the health of components deployed on the robot is to not burden the robot's limited computational resources with more tasks. Also, non-intrusive fault diagnosis helps in keeping the robotic system independent of the diagnosis system, which means that the diagnosis system can not induce faults to the robotic platform. The goals for creating such a system were as follows:

- Non-intrusively monitor a robot's health using a mini-PC.
- Detect and diagnose faults that occur on different components across the robotic platform.
- Safely log the robot's components health data.
- Use a common standard for representing the health of all monitored components, and send the data to a remote user.

These goals presented some challenges. These challenges can be summarized as follows:

- Developing a system design that satisfies the requirements of the project.
- Selecting a fault diagnosis method that is:
  - Low in computational complexity to be deployed on a mini-PC with limited resources.

  - Extensible to allow for easy updates and addition of new components.
  - Able to incorporate varying components of different natures.

- Investigation of existing health monitoring and fault diagnosis standards, to represent the health of varying components of a robot.

In this study, these challenges were addressed and the system was developed and tested. Based on the requirements of the system and following and Attribute-Driven-Design method, a system architecture was designed. This was addressed in chapter 3. The architecture developed took into consideration the different constraints imposed by the goals and requirements of the system. The system architecture was designed in a way that would also allow for extensibility, where more components can still be added to allow for further tasks to be executed. The architecture also helped keep the diagnosis module independent of the robotic platform. Therefore, if it is required to substitute the diagnosis method, this can be done easily. After looking at the state of the art in fault diagnosis, different methods were examined and table 2.5 was produced, where advantages and drawbacks of different methods of fault diagnosis were listed. From the state of the art, two methods of fault diagnosis were chosen to be examined, based on their computational complexity, applicability to varying components on a robotic platform, and extensibility to allow for new components to be added. Chapter 4 gives a detailed overview and comparison of the two selected fault diagnosis methods, consistency-based diagnosis (CBD) and sensor fault detection and diagnosis (SFDD). It was found that the SFDD algorithm had some draw backs and had a high computational complexity, unlike what was stated in the proposed study [32].

On the other hand, after the fault diagnosis methods were chosen and analysed, the standardization and representation of health and faults on the monitored components was examined. This was addressed in chapter 5, where based on the NATO Generic Vehicle Architecture, a standard for representing health and faults of vehicle components (Usage and Condition Monitoring System) was selected and analysed for it's applicability to robotic platforms, since similarities exist nowadays between vehicles and robotic platforms such as the deployment of multiple electronic components that need to be monitored. Chapter 6 presents the implementation and evaluation of the complete diagnosis system, which was deployed on a Raspberry Pi with limited computational complexity. It was found that CBD showed a higher accuracy than SFDD, after multiple tests were performed using the CBD algorithm. The CPU and memory usage of the complete system on the Raspberry Pi was found to be very suitable and could allow for more functionalities to be added to the existing system. The output of the fault diagnosis method was mapped to the UCMS standard and published over the network to be available to a remote system that can be used for monitoring the status of the robot remotely. In case

of loss of connection to the remote user, a redundant component was also considered and added to the system, which was the storage device, that keeps records of the health of the components in a periodic fashion. The rate of logging the messages is customizable, which is useful to accompany different sizes of storage devices. If a limited storage space is available, a low frequency of logging messages is advised to be able to record for the required operation period of the robot and vice versa.

## 7.1 Contributions

The contributions of this work are shown here:

- Design of system architecture for non-intrusive health monitoring and fault diagnosis of robots.
- Implementation of complete diagnosis system on a Raspberry Pi.
- Evaluation of consistency-based diagnosisIn comparison to SFDD
- Application of CBD to a ROS-based robotic platform.
- Implementation of UCMS standard to a robotic platform on specific components, and a mapping of diagnosis output which can be used for further components.

## 7.2 Limitations

Some limitations were noted after the implementation and testing of the system were performed.

With regards to the selected hardware, the Raspberry Pi was able to handle the computations of the complete system with acceptable CPU and memory usage. However, the limitation was with regards to the network adapter available on the Raspberry Pi itself, which limited the amount of data that can be received from the robot per second. This will hinder the fault diagnosis system from performing correctly due to drops in data packages, which can trigger false alarms that faults have occurred on the robot.

The fault diagnosis method selected, CBD, has many advantages as discussed earlier, with regards to deploying it on this system. The problem however, lies in the accuracy of the models that are developed for the components that are used later on for fault detection and diagnosis. The underlying problem of modelling is mainly the selection of the appropriate thresholds and window sizes, along with filters that are suitable for

each component. In this study, several filters were tested, and the effect of changing the size of the window sizes on the detection has been noted. Another draw back of CBD is that faults that occur outside of the models knowledge will not be detected, meaning that in order to monitor a complete robotic platform, the model needs to be inclusive of all components.

The UCMS standard presented, has shown that it is sufficient for representing the health of varying components on a robotic platform. Some minor updates though, are suggested to be applied to the standard, such as differentiation between a fault and a threshold that has been exceeded, the addition of an indicator of noise level produced by a certain component, the addition of the kind of filter applied to the component, and finally, the addition of the periodic logging value of the component.

All in all, these limitations are minor and due to the system architecture designed, changes can be applied to each individual component of the system without affecting the other components in the system.

## 7.3 Future Work

To improve the current system, the following ideas are proposed as future work.

With the current status of the implemented system, and the limitations discussed, the following can be suggested as future work for this project. Since the modelling is the most time-consuming part of applying this system to any robotic platform, and since the mini-PC allows for more processes to be deployed on it, an unsupervised machine-learning method is suggested to learn the values of the appropriate thresholds and window sizes to apply to each component. This would allow for different modes to be run on the black box, one would be a training mode where the behaviour of the monitored component is learnt, and the thresholds are obtained, and the other mode would be fault diagnosis.

Another suggestion is to couple both CBD and SFDD methods in a manner that would allow to verify if a fault has actually occurred. Since for both CBD and SFDD, the structural model of the system is required, it could be used as a common model, and when the CBD algorithm produces a fault from a certain component, it can be verified by the underlying SFDD algorithm if it produced a fault from the same component as well.

As for the current hardware limitation of the system, the mini-PC can be replaced with a newer version that has a better network adapter that can handle the bandwidth of the entire robot without having any package drops. In that sense, the black box is to be

thought of as an extension to the robot, and should not be limited by it's own hardware. Another suggestion would be to throttle the number of messages incoming from the robot to the black box. However, this would mean changing configurations on the robot itself which is intrusive to the robotic system.

This study has shown that it is possible to deploy a complete diagnosis system on a mini-PC, and can be used as a foundation for future work regarding fault diagnosis of robots. Health monitoring and fault diagnosis of robots is crucial to the development and advancement of robotic systems, since in the future, we will be more dependant on robots in executing more and more tasks, and reliability will play a key role in their acceptance into our everyday lives.

# List of Figures

# List of Tables

# Appendix A

# Usage & Condition Monitoring System Data Model

```
#include "LDM_Common.idl"

module P_UCMS_PSM
{
   enum T_Monitored_Entity_StateType
   {
      L_Monitored_Entity_StateType_ON,
      L_Monitored_Entity_StateType_OFF
   };

   enum T_HealthStatus
   {
      L_HealthStatus_OPERATIONAL,
      L_HealthStatus_DEGRADED,
      L_HealthStatus_NOT_OPERATIONAL,
      L_HealthStatus_UNKNOWN
   };

   enum T_MonitoredDataWarningLevel
   {
      L_MonitoredDataWarningLevel_NO_WARNING,
      L_MonitoredDataWarningLevel_PRE_WARNING,
      L_MonitoredDataWarningLevel_WARNING
   };

   enum T_ThresholdType
   {
      L_ThresholdType_MIN,
      L_ThresholdType_MAX
   };
```

```
enum T_BITType
{
   L_BITType_C_BIT,
   L_BITType_P_BIT,
   L_BITType_I_BIT
};

enum T_MonitoredCharacteristicKind
{
   L_MonitoredCharacteristicKind_INSTANT,
   L_MonitoredCharacteristicKind_CUMULATIVE
};

struct C_Monitored_Entity_run_IBIT
{
   P_LDM_Common::T_IdentifierType A_recipientID; //@key
   P_LDM_Common::T_IdentifierType A_sourceID; //@key
   P_LDM_Common::T_Int32 A_referenceNum;
   P_LDM_Common::T_DateTimeType A_timeOfDataGeneration;
};

// The status of system (platform or sub-system) whose
// usage and condition is being monitored.
struct C_Monitored_Entity
{
   P_LDM_Common::T_IdentifierType A_sourceID; //@key
   P_LDM_Common::T_DateTimeType A_timeOfDataGeneration;
   T_HealthStatus A_healthStatus;
   P_LDM_Common::T_Boolean A_bitInProgress;
   T_Monitored_Entity_StateType A_MonitoredEntityState;
   sequence <P_LDM_Common::T_IdentifierType> A_failureEvents_sourceID;
   P_LDM_Common::T_IdentifierType A_specification_sourceID;
   sequence <P_LDM_Common::T_IdentifierType>
   A_installedSoftwares_sourceID;
   sequence <P_LDM_Common::T_IdentifierType>
   A_preventiveMaintenanceRecommendedEvents_sourceID;
   sequence <P_LDM_Common::T_IdentifierType>
   A_monitoredCharacteristics_sourceID;
};

// An event indicating that a failure occurred in a sub-system
// or is expected to occur (predicted failure issued by a
// supervision and analysis system).
//
struct C_Failure_Event
{
   P_LDM_Common::T_IdentifierType A_sourceID; //@key
```

```
   P_LDM_Common::T_DateTimeType A_timeOfDataGeneration;
   P_LDM_Common::T_DateTimeType A_eventOccurenceDate;
   P_LDM_Common::T_Double A_standardFaultCode;
   P_LDM_Common::T_LongString A_proprietaryFaultCode;
   P_LDM_Common::T_Boolean A_predictedFailure;
   P_LDM_Common::T_LongString A_failureDescription;
   P_LDM_Common::T_LongString A_causeDescription;
   P_LDM_Common::T_LongString A_operationalConsequenceDescription;
   P_LDM_Common::T_LongString A_operatorActionDescription;
   P_LDM_Common::T_LongString A_maintenanceActionDescription;
   P_LDM_Common::T_LongString A_complementaryInformation;
   P_LDM_Common::T_LongString A_eventDocumentationURI;
   P_LDM_Common::T_IdentifierType A_monitoredEntity_sourceID;
};

struct C_Installed_Software
{
   P_LDM_Common::T_IdentifierType A_sourceID; //@key
   P_LDM_Common::T_DateTimeType A_timeOfDataGeneration;
   P_PredefinedTypesCpp::T_int A_serialNumber;
   P_PredefinedTypesCpp::T_int A_version;
   P_PredefinedTypesCpp::T_int A_configuration;
   P_LDM_Common::T_IdentifierType A_subSystem_sourceID;
};

struct C_Preventive_Maintenance_Recommended_Event
{
   P_LDM_Common::T_IdentifierType A_sourceID; //@key
   P_LDM_Common::T_DateTimeType A_timeOfDataGeneration;
   P_LDM_Common::T_MediumString A_dataDescriptor;
   P_LDM_Common::T_ShortString A_dataUnit;
   P_LDM_Common::T_Double A_dataValue;
   P_LDM_Common::T_Double A_preventiveMaintenanceThreshold;
   P_LDM_Common::T_DateTimeType A_occurrenceDateTime;
   P_LDM_Common::T_IdentifierType A_subSystem_sourceID;
};

// The static parameters of an entity (whole platform or sub-system)
// whose usage and condition is being monitored.
//
struct C_Monitored_Entity_Specification
{
   P_LDM_Common::T_IdentifierType A_sourceID; //@key
   P_LDM_Common::T_DateTimeType A_timeOfDataGeneration;
   P_LDM_Common::T_Boolean
   A_cumulativeMonitoredCharacteristicsOutputSupported;
   P_LDM_Common::T_Boolean A_failureStandardFaultCodeSupported;
   P_LDM_Common::T_Boolean A_failureProprietaryFaultCodeSupported;
```

```
    P_LDM_Common::T_Boolean A_failureDescriptionSupported;
    P_LDM_Common::T_Boolean A_failureCauseDescriptionSupported;
    P_LDM_Common::T_Boolean
    A_failureOperationalConsequenceDescriptionSupported;
    P_LDM_Common::T_Boolean A_failureOperatorActionDescriptionSupported;
    P_LDM_Common::T_Boolean
    A_failureMaintenanceActionDescriptionSupported;
    P_LDM_Common::T_Boolean
    A_failureComplementaryInformationSupported;
    P_LDM_Common::T_Boolean A_failureDocumentationURISupported;
    P_LDM_Common::T_DateTimeType A_installationDateTime;
    sequence <P_LDM_Common::T_IdentifierType>
    A_specifiedMonitoredEntities_sourceID;
};

// Instant or cumulative monitored usage and condition data of a
// sub-system, as calculated by the monitored entity itself or
//by the UCMS service.
// Ex. current temperature of an electronic component
// (instant monitored characteristic), vehicle mileage
// (cumulative monitored characteristic)
//
struct C_Monitored_Characteristic
{
    P_LDM_Common::T_IdentifierType A_sourceID; //@key
    P_LDM_Common::T_DateTimeType A_timeOfDataGeneration;
    P_LDM_Common::T_Double A_value;
    P_LDM_Common::T_IdentifierType A_monitoredEntity_sourceID;
    sequence <P_LDM_Common::T_IdentifierType>
    A_thresholdExceedenceEvents_sourceID;
    P_LDM_Common::T_IdentifierType A_specification_sourceID;
};

// The static parameters of a monitored usage and condition data of
// a sub-system.
// Ex. temperature of the crew compartment ; vehicle mileage.
struct C_Monitored_Characteristic_Specification
{
    P_LDM_Common::T_IdentifierType A_sourceID; //@key
    P_LDM_Common::T_DateTimeType A_timeOfDataGeneration;
    P_LDM_Common::T_ShortString A_unit;
    P_LDM_Common::T_LongString A_descriptor;
    P_LDM_Common::T_Int32 A_publishingIntervalInSeconds;
    T_MonitoredCharacteristicKind A_characteristicKind;
    sequence <P_LDM_Common::T_IdentifierType>
    A_specifiedCharacteristics_sourceID;
    sequence <P_LDM_Common::T_IdentifierType>
    A_thresholds_sourceID;
```

```
  };

  struct C_Threshold
  {
     P_LDM_Common::T_IdentifierType A_sourceID; //@key
     P_LDM_Common::T_DateTimeType A_timeOfDataGeneration;
     P_LDM_Common::T_MediumString A_name;
     P_LDM_Common::T_Double A_value;
     T_ThresholdType A_type;
     P_PredefinedTypesCpp::T_int A_maxDuration;
     P_PredefinedTypesCpp::T_int A_maxNumberOfRepetitions;
     sequence <P_LDM_Common::T_IdentifierType>
     A_exceedenceEvents_sourceID;
     P_LDM_Common::T_IdentifierType A_concernedCharacteristic_sourceID;
     sequence <P_LDM_Common::T_IdentifierType>
     A_conditionedThresholds_sourceID;
     P_LDM_Common::T_IdentifierType
     A_conditioningThreshold_sourceID;
     P_LDM_Common::T_IdentifierType A_specification_sourceID;
  };

  struct C_Threshold_Exceedence_Event
  {
     P_LDM_Common::T_IdentifierType A_sourceID; //@key
     P_LDM_Common::T_DateTimeType A_timeOfDataGeneration;
     P_LDM_Common::T_DateTimeType A_eventOccurenceDate;
     P_LDM_Common::T_Double A_measuredValue;
     P_LDM_Common::T_IdentifierType A_exceededThreshold_sourceID;
     P_LDM_Common::T_IdentifierType A_concernedCharacteristic_sourceID;
     sequence <P_LDM_Common::T_IdentifierType> A_inducedEvents_sourceID;
     P_LDM_Common::T_IdentifierType A_inducingEvent_sourceID;
  };

  struct C_Threshold_Specification
  {
     P_LDM_Common::T_IdentifierType A_sourceID; //@key
     P_LDM_Common::T_DateTimeType A_timeOfDataGeneration;
     P_LDM_Common::T_Boolean A_valueThresholdSupported;
     P_LDM_Common::T_Boolean A_durationThresholdSupported;
     P_LDM_Common::T_Boolean A_repetitionsThresholdSupported;
     sequence <P_LDM_Common::T_IdentifierType>
     A_specifiedThresholds_sourceID;
  };
};
```

# Appendix B

# Experimental Configurations & Results

Components and connections description for experimental setup (see Fig. 6.8).

```
nodes:

  - name: 'usb_cam'
    sub_topic: []
    pub_topic: ['/usb_cam/image_raw']

  - name: 'audio_capture'
    sub_topic: []
    pub_topic: ['/audio']

  - name: 'Signal_generator'
    sub_topic: ['/usb_cam/image_raw', '/audio']
    pub_topic: ['/signal']

observations:
  - type: "activated"
    nodes: ["/audio_capture", "/usb_cam", "/Signal_generator"]


  - type: "hz"
    topics: ['/audio']

  - type: "hz"
    topics: ['/usb_cam/image_raw']

  - type: "hz"
    topics: ['/signal']
```

```
  - type: "timeout"
    topics: ['/audio']

  - type: "timeout"
    topics: ['/usb_cam/image_raw']

  - type: "timeout"
    topics: ['/signal']

  - type: "scores"
    topics: ['/signal']
```

System desctiption for experiment 1-10.

```
setup:
  - type: scores
    main_loop_rate: 1.0
    use_global_subscriber: true
    topics:
    - name: /signal
      filter:
        type: mean
        window_size: 50
        deviation_type: std_deviation
      states:
      - state: 'ok'
        number: 1
        score:
          type: nominal_value
          value:
            type: gauss
            mean: 15.0
            std_deviation: 0.5

  - type: hz
    main_loop_rate: 1.0
    topics:
    - name: /audio
      callerids:
      - callerid: []
        filter:
          type: mean
          window_size: 50
          deviation_type: std_deviation
        states:
        - state: 'State_ok'
          number: 1
          frequenzy:
```

```
          type: nominal_value
          value:
            type: gauss
            mean: 0.03
            std_deviation: 0.01


  - name: /usb_cam/image_raw
    callerids:
    - callerid: []
      filter:
        type: mean
        window_size: 50
        deviation_type: std_deviation
      states:
      - state: 'State_ok'
        number: 1
        frequenzy:
          type: nominal_value
          value:
            type: gauss
            mean: 0.06666666
            std_deviation: 0.01


  - name: /signal
    callerids:
    - callerid: []
      filter:
        type: mean
        window_size: 50
        deviation_type: std_deviation
      states:
      - state: 'State_ok'
        number: 1
        frequenzy:
          type: nominal_value
          value:
            type: gauss
            mean: 0.06666666
            std_deviation: 0.01




- type: activated
  start_up_time: 1.0
  resource_topic: /diag/node_infos
```

```
    nodes:
    - name: /audio_capture
    - name: /usb_cam
    - name: /Signal_generator

  - type: timeout
    topics:
    - name: /audio
      callerids:
      - callerid: []
        timeout: 1.0
        max_timeouts_in_a_row: 20000
    - name: /usb_cam/image_raw
      callerids:
      - callerid: []
        timeout: 1.0
        max_timeouts_in_a_row: 2000
    - name: /signal
      callerids:
      - callerid: []
        timeout: 1.0
        max_timeouts_in_a_row: 2000
```

TABLE B.1: Results of tests 1-3.

| Observed Entity | Faults Injected | | | | Faults Detected | | | | TP | FP | FN | Av. CPU usage(%) | Av. Mem. Usage Mb |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Hz | Timeout | Kill | Value | Hz | Timeout | Kill | Value | | | | | |
| Test 1 | | | | | | | | | | | | | |
| Camera | 2 | 2 | 3 | 0 | 1 | 5 | 2 | 0 | 5 | 0 | 1 | | |
| Audio Microphone | 3 | 0 | 4 | 0 | 5 | 0 | 4 | 0 | 7 | 2 | 0 | | |
| Software Signal | 2 | 0 | 5 | 2 | 4 | 0 | 5 | 1 | 8 | 2 | 1 | | |
| TOTAL | 7 | 2 | 12 | | 10 | 5 | 11 | 1 | 20 | 8 | 2 | 39 | 140 |
| Test 2 | | | | | | | | | | | | | |
| Camera | 2 | 1 | 5 | 0 | 10 | 6 | 5 | 0 | 8 | 8 | 0 | | |
| Audio Microphone | 0 | 0 | 3 | 0 | 4 | 0 | 3 | 0 | 3 | 4 | 4 | | |
| Software Signal | 2 | 0 | 4 | 1 | 1 | 2 | 4 | 0 | 5 | 0 | 1 | | |
| TOTAL | 4 | 1 | 12 | 1 | 15 | 8 | 12 | 0 | 16 | 12 | 5 | 33 | 139 |
| Test 3 | | | | | | | | | | | | | |
| Camera | 2 | 3 | 8 | 0 | 1 | 11 | 8 | 0 | 0 | 0 | 1 | | |
| Audio Microphone | 2 | 2 | 5 | 0 | 0 | 0 | 5 | 0 | 5 | 0 | 2 | | |
| Software Signal | 2 | 0 | 10 | 3 | 1 | 10 | 10 | 2 | 11 | 0 | 2 | | |
| TOTAL | 6 | 5 | 23 | 3 | 2 | 21 | 23 | 2 | 16 | 0 | 5 | 30 | 140 |

TABLE B.2: Results of tests 4-6.

| Observed Entity | Faults Injected | | | | Faults Detected | | | | TP | FP | FN | Av. CPU usage(%) | Av. Mem. Usage Mb |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Hz | Timeout | Kill | Value | Hz | Timeout | Kill | Value | | | | | |
| Test 4 | | | | | | | | | | | | | |
| Camera | 3 | 1 | 5 | 0 | 3 | 6 | 4 | 0 | 8 | 0 | 1 | | |
| Audio Microphone | 3 | 3 | 10 | 0 | 10 | 11 | 10 | 0 | 16 | 0 | 2 | | |
| Software Signal | 4 | 0 | 5 | 3 | 3 | 5 | 5 | 2 | 8 | 0 | 1 | | |
| TOTAL | 10 | 4 | 20 | 3 | 16 | 22 | 19 | 2 | 32 | 0 | 4 | 32 | 140 |
| Test 5 | | | | | | | | | | | | | |
| Camera | 4 | 1 | 3 | 0 | 3 | 4 | 3 | 0 | 8 | 0 | 0 | | |
| Audio Microphone | 0 | 0 | 5 | 0 | 0 | 5 | 5 | 0 | 5 | 0 | 0 | | |
| Software Signal | 3 | 0 | 11 | 0 | 4 | 11 | 11 | 0 | 14 | 0 | 1 | | |
| TOTAL | 7 | 1 | 19 | | 7 | 20 | 19 | | 27 | 0 | 1 | 30 | 150 |
| Test 6 | | | | | | | | | | | | | |
| Camera | 3 | 1 | 5 | 0 | 2 | 6 | 0 | 0 | 8 | 0 | 0 | | |
| Audio Microphone | 0 | 0 | 5 | 0 | 0 | 6 | 0 | 0 | 5 | 0 | 5 | | |
| Software Signal | 4 | 0 | 3 | 1 | 4 | 3 | 3 | 1 | 8 | 0 | 0 | | |
| TOTAL | 7 | 1 | 13 | 1 | 6 | 15 | 3 | 1 | 21 | 0 | 5 | 31 | 140 |

TABLE B.3: Results of tests 7-10.

| Observed Entity | Faults Injected | | | | Faults Detected | | | | TP | FP | FN | Av. CPU usage(%) | Av. Mem. Usage Mb |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Hz | Timeout | Kill | Value | Hz | Timeout | Kill | Value | | | | | |
| Test 7 | | | | | | | | | | | | | |
| Camera | 1 | 2 | 10 | 0 | 0 | 13 | 10 | 0 | 12 | 1 | 1 | | |
| Audio Microphone | 0 | 0 | 6 | 0 | 0 | 7 | 6 | 0 | 6 | 1 | 0 | | |
| Software Signal | 2 | 0 | 2 | 2 | 2 | 3 | 2 | 1 | 4 | 1 | 1 | | |
| TOTAL | 3 | 2 | 18 | 2 | 2 | 23 | 18 | 1 | 22 | 3 | 2 | 30 | 150 |
| Test 8 | | | | | | | | | | | | | |
| Camera | 5 | 3 | 3 | 0 | 2 | 6 | 3 | 0 | 9 | 0 | 3 | | |
| Audio Microphone | 0 | 0 | 2 | 0 | 0 | 2 | 2 | 0 | 4 | 0 | 0 | | |
| Software Signal | 1 | 0 | 6 | 2 | 2 | 5 | 6 | 3 | 7 | 1 | 1 | | |
| TOTAL | 6 | 3 | 11 | 2 | 4 | 13 | 11 | 3 | 20 | 1 | 4 | 32 | 140 |
| Test 9 | | | | | | | | | | | | | |
| Camera | 2 | 2 | 2 | 0 | 2 | 4 | 2 | 0 | 8 | 0 | 0 | | |
| Audio Microphone | 0 | 0 | 5 | 0 | 0 | 5 | 6 | 0 | 5 | 1 | 0 | | |
| Software Signal | 5 | 0 | 7 | 6 | 6 | 6 | 7 | 8 | 12 | 3 | 1 | | |
| TOTAL | 7 | 2 | 14 | 6 | 8 | 15 | 15 | 8 | 25 | 4 | 1 | 30 | 140 |
| Test 10 | | | | | | | | | | | | | |
| Camera | 3 | 2 | 4 | 0 | 3 | 6 | 4 | 0 | 9 | 0 | 0 | | |
| Audio Microphone | 0 | 0 | 4 | 0 | 0 | 0 | 4 | 0 | 4 | 0 | 4 | | |
| Software Signal | 4 | 0 | 3 | 2 | 3 | 5 | 3 | 1 | 8 | 2 | 2 | | |
| TOTAL | 7 | 2 | 11 | 2 | 6 | 11 | 11 | 1 | 21 | 2 | 6 | 32 | 145 |

System description for Experiments 11-20.

```
setup:
  - type: scores
    main_loop_rate: 1.0
    use_global_subscriber: true
    topics:
    - name: /signal
      filter:
        type: mean
        window_size: 50
        deviation_type: std_deviation
      states:
      - state: 'ok'
        number: 1
        score:
          type: nominal_value
          value:
            type: gauss
            mean: 15.0
            std_deviation: 0.5

  - type: hz
    main_loop_rate: 1.0
    topics:
    - name: /audio
      callerids:
      - callerid: []
        filter:
          type: mean
          window_size: 50
          deviation_type: std_deviation
        states:
        - state: 'State_ok'
          number: 1
          frequenzy:
            type: nominal_value
            value:
              type: gauss
              mean: 0.03
              std_deviation: 0.01


    - name: /usb_cam/image_raw
      callerids:
      - callerid: []
        filter:
          type: mean
```

```
        window_size: 50
        deviation_type: std_deviation
      states:
      - state: 'State_ok'
        number: 1
        frequenzy:
          type: nominal_value
          value:
            type: gauss
            mean: 0.06666666
            std_deviation: 0.01


  - name: /signal
    callerids:
    - callerid: []
      filter:
        type: mean
        window_size: 50
        deviation_type: std_deviation
      states:
      - state: 'State_ok'
        number: 1
        frequenzy:
          type: nominal_value
          value:
            type: gauss
            mean: 0.083333
            std_deviation: 0.01




- type: activated
  start_up_time: 1.0
  resource_topic: /diag/node_infos
  nodes:
  - name: /audio_capture
  - name: /usb_cam
  - name: /Signal_generator

- type: timeout
  topics:
  - name: /audio
    callerids:
    - callerid: []
      timeout: 1.0
      max_timeouts_in_a_row: 20000
```

```
  - name: /usb_cam/image_raw
    callerids:
    - callerid: []
      timeout: 1.0
      max_timeouts_in_a_row: 2000
  - name: /signal
    callerids:
    - callerid: []
      timeout: 1.0
      max_timeouts_in_a_row: 2000
```

Table B.4: Results of tests 11-13.

| Observed Entity | Faults Injected | | | | Faults Detected | | | | TP | FP | FN | Av. CPU usage(%) | Av. Mem. Usage Mb |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Hz | Timeout | Kill | Value | Hz | Timeout | Kill | Value | | | | | |
| Test 11 | | | | | | | | | | | | | |
| Camera | 5 | 1 | 4 | 0 | 3 | 2 | 4 | 0 | 8 | 0 | 2 | | |
| Audio Microphone | 5 | 0 | 5 | 0 | 5 | 0 | 5 | 0 | 10 | 0 | 0 | | |
| Software Signal | 3 | 0 | 5 | 2 | 3 | 0 | 5 | 2 | 10 | 0 | 0 | | |
| TOTAL | 13 | 1 | 14 | | 10 | 5 | 11 | 1 | 20 | 0 | 2 | 35 | 139 |
| Test 12 | | | | | | | | | | | | | |
| Camera | 5 | 1 | 4 | 0 | 5 | 1 | 4 | 0 | 10 | 0 | 0 | | |
| Audio Microphone | 0 | 0 | 4 | 0 | 4 | 0 | 4 | 0 | 8 | 4 | 0 | | |
| Software Signal | 5 | 0 | 4 | 2 | 5 | 0 | 4 | 2 | 11 | 0 | 0 | | |
| TOTAL | 10 | 1 | 12 | 2 | 14 | 1 | 12 | 2 | 29 | 4 | 0 | 32 | 149 |
| Test 13 | | | | | | | | | | | | | |
| Camera | 6 | 1 | 3 | 0 | 6 | 1 | 3 | 0 | 10 | 0 | 0 | | |
| Audio Microphone | 7 | 1 | 6 | 0 | 7 | 1 | 6 | 0 | 14 | 0 | 0 | | |
| Software Signal | 4 | 0 | 2 | 2 | 4 | 0 | 2 | 2 | 8 | 0 | 0 | | |
| TOTAL | 17 | 2 | 11 | 2 | 17 | 2 | 11 | 2 | 32 | 0 | 0 | 31 | 141 |

Table B.5: Results of tests 14-16.

| Observed Entity | Faults Injected | | | | Faults Detected | | | | TP | FP | FN | Av. CPU usage(%) | Av. Mem. Usage Mb |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Hz | Timeout | Kill | Value | Hz | Timeout | Kill | Value | | | | | |
| Test 14 | | | | | | | | | | | | | |
| Camera | 2 | 1 | 6 | 0 | 2 | 1 | 6 | 0 | 9 | 0 | 0 | | |
| Audio Microphone | 0 | 0 | 6 | 0 | 3 | 0 | 6 | 0 | 6 | 0 | 0 | | |
| Software Signal | 3 | 0 | 9 | 4 | 3 | 0 | 9 | 3 | 15 | 0 | 1 | | |
| TOTAL | 5 | 1 | 21 | 4 | 8 | 1 | 21 | 3 | 30 | 0 | 1 | 34 | 150 |
| Test 15 | | | | | | | | | | | | | |
| Camera | 4 | 2 | 9 | 0 | 4 | 2 | 9 | 0 | 15 | 0 | 0 | | |
| Audio Microphone | 0 | 0 | 4 | 0 | 1 | 0 | 4 | 0 | 4 | 1 | 0 | | |
| Software Signal | 6 | 0 | 9 | 2 | 6 | 0 | 9 | 2 | 15 | 0 | 0 | | |
| TOTAL | 10 | 2 | 22 | | 11 | 2 | 22 | 2 | 34 | 1 | 0 | 34 | 140 |
| Test 16 | | | | | | | | | | | | | |
| Camera | 4 | 1 | 8 | 0 | 4 | 1 | 8 | 0 | 13 | 0 | 0 | | |
| Audio Microphone | 0 | 0 | 6 | 0 | 4 | 0 | 6 | 0 | 6 | 4 | 0 | | |
| Software Signal | 3 | 0 | 8 | 3 | 3 | 0 | 8 | 2 | 11 | 0 | 1 | | |
| TOTAL | 7 | 1 | 22 | 3 | 11 | 1 | 22 | 2 | 30 | 4 | 1 | 29 | 142 |

Table B.6: Results of tests 17-20.

| Observed Entity | Faults Injected | | | | Faults Detected | | | | TP | FP | FN | Av. CPU usage(%) | Av. Mem. Usage Mb |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Hz | Timeout | Kill | Value | Hz | Timeout | Kill | Value | | | | | |
| Test 17 | | | | | | | | | | | | | |
| Camera | 2 | 1 | 5 | 0 | 2 | 1 | 5 | 0 | 8 | 0 | 0 | | |
| Audio Microphone | 0 | 0 | 11 | 0 | 3 | 0 | 11 | 0 | 11 | 3 | 0 | | |
| Software Signal | 7 | 0 | 8 | 2 | 7 | 0 | 8 | 1 | 16 | 0 | 1 | | |
| TOTAL | 9 | 1 | 24 | 2 | 12 | 1 | 24 | 1 | 35 | 3 | 1 | 33 | 140 |
| Test 18 | | | | | | | | | | | | | |
| Camera | 4 | 1 | 6 | 0 | 4 | 1 | 6 | 0 | 11 | 0 | 0 | | |
| Audio Microphone | 0 | 0 | 7 | 0 | 2 | 0 | 7 | 0 | 7 | 2 | 0 | | |
| Software Signal | 2 | 0 | 4 | 3 | 2 | 3 | 4 | 3 | 9 | 0 | 1 | | |
| TOTAL | 6 | 1 | 17 | 3 | 8 | 4 | 17 | 3 | 27 | 2 | 1 | 34 | 140 |
| Test 19 | | | | | | | | | | | | | |
| Camera | 7 | 1 | 3 | 0 | 4 | 1 | 3 | 0 | 8 | 0 | 3 | | |
| Audio Microphone | 0 | 0 | 4 | 0 | 3 | 4 | 4 | 0 | 4 | 3 | 0 | | |
| Software Signal | 4 | 0 | 4 | 2 | 4 | 4 | 4 | 2 | 10 | 4 | 1 | | |
| TOTAL | 11 | 1 | 11 | 2 | 11 | 9 | 11 | 2 | 22 | 7 | 4 | 30 | 139 |
| Test 20 | | | | | | | | | | | | | |
| Camera | 5 | 2 | 3 | 0 | 7 | 2 | 3 | 0 | 10 | 2 | 0 | | |
| Audio Microphone | 0 | 0 | 2 | 0 | 5 | 2 | 2 | 0 | 2 | 0 | 0 | | |
| Software Signal | 3 | 0 | 1 | 3 | 3 | 1 | 1 | 1 | 5 | 0 | 2 | | |
| TOTAL | 8 | 2 | 6 | 3 | 15 | 5 | 6 | 1 | 17 | 2 | 2 | 33 | 142 |

# Bibliography


[1] D P. Sellers, A J. Ramsbotham, Hal Bertrand, and Nicholas Karvonides. International assessment of unmanned ground vehicles. page 76, 02 2008.

[2] Mogens Blanke, Michel Kinnaert, Jan Lunze, and Marcel Staroswiecki. *Diagnosis and Fault-Tolerant Control With contributions by Jochen Schröder*. 2006. ISBN 9783540356523.

[3] Alan Winfield and Marina Jirotka. The case for an ethical black box. 07 2017.

[4] Jennifer Carlson and Robin R. Murphy. How UGVs physically fail in the field. *IEEE Transactions on Robotics*, 21(3):423–437, 2005. ISSN 15523098. doi: 10.1109/TRO.2004.838027.

[5] Rolf Isermann. *Fault-diagnosis systems: an introduction from fault detection to fault tolerance*. Springer Science & Business Media, 2006.

[6] Gerald Steinbauer. A survey about faults of robots used in RoboCup. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7500 LNAI:344–355, 2013. ISSN 03029743. doi: 10.1007/978-3-642-39250-4_31.

[7] Eliahu Khalastchi and Meir Kalech. On fault detection and diagnosis in robotic systems. *ACM Computing Surveys (CSUR)*, 51(1):9, 2018.

[8] Zhiwei Gao, C Cecati, and S X Ding. A Survey of Fault Diagnosis and Fault-Tolerant Techniques Part I: Fault Diagnosis. *IEEE Transactions On Industrial Electronics*, 62(6):3768–3774, 2015. ISSN 0278-0046. doi: 10.1109/TIE.2015.2417501.

[9] Zhiwei Gao, Carlo Cecati, and Steven X Ding. A survey of fault diagnosis and fault-tolerant techniques-part II: Fault diagnosis with knowledge-based and hybrid/active approaches. *IEEE Transactions on Industrial Electronics*, 62(6):3768–3774, 2015. ISSN 02780046. doi: 10.1109/TIE.2015.2419013.

[10] Inseok Hwang, Sungwan Kim, Youdan Kim, and Chze Eng Seah. A survey of fault detection, isolation, and reconfiguration methods. *IEEE transactions on control systems technology*, 18(3):636–653, 2010.

[11] Gerald Steinbauer and Franz Wotawa. On the way to automated belief repair for autonomous robots. In *the 21st International Workshop on Principles of Diagnosis (DX-10)*, 2010.

[12] Eliahu Khalastchi, Meir Kalech, and Lior Rokach. Multi-layered model based diagnosis in robots. *23rd InternationalWorkshop on Principles of Diagnosis, UK*, 2012.

[13] Safdar Zaman, Gerald Steinbauer, Johannes Maurer, Peter Lepej, and Suzana Uran. An integrated model-based diagnosis and repair architecture for ros-based robot systems. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 482–489. IEEE, 2013.

[14] Ming Yu, Danwei Wang, Ming Luo, and Danhong Zhang. Fdi and fault estimation based on differential evolution and analytical redundancy relations. In *Control Automation Robotics & Vision (ICARCV), 2010 11th International Conference on*, pages 1341–1346. IEEE, 2010.

[15] Tesheng Hsiao and Mao-Chiao Weng. A hierarchical multiple-model approach for detection and isolation of robotic actuator faults. *Robotics and Autonomous Systems*, 60(2):154–166, 2012.

[16] Xiaodong Zhang. Sensor bias fault detection and isolation in a class of nonlinear uncertain systems using adaptive estimation. *IEEE Transactions on Automatic Control*, 56(5):1220–1226, 2011.

[17] Leo H Chiang, Evan L Russell, and Richard D Braatz. *Fault detection and diagnosis in industrial systems*. Springer Science & Business Media, 2000.

[18] Monica L Visinsky, Joseph R Cavallaro, and Ian D Walker. Expert system framework for fault detection and fault tolerance in robotics. *Computers & electrical engineering*, 20(5):421–435, 1994.

[19] Markos Markou and Sameer Singh. Novelty detection: a review—part 2:: neural network based approaches. *Signal processing*, 83(12):2499–2521, 2003.

[20] Stergios I Roumeliotis, Gaurav S Sukhatme, and George A Bekey. Sensor fault detection and identification in a mobile robot. In *Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on*, volume 3, pages 1383–1388. IEEE, 1998.

[21] Kaci Bader, Benjamin Lussier, and Walter Schön. A fault tolerant architecture for data fusion: A real application of kalman filters for mobile robot localization. *Robotics and Autonomous Systems*, 88:11–23, 2017.

[22] Yu-shan Sun, Xiang-rui Ran, Yue-ming Li, Guo-cheng Zhang, and Ying-hao Zhang. Thruster fault diagnosis method based on gaussian particle filter for autonomous underwater vehicles. *International Journal of Naval Architecture and Ocean Engineering*, 8(3):243–251, 2016.

[23] Michał Zając. Online fault detection of a mobile robot with a parallelized particle filter. *Neurocomputing*, 126:151–165, 2014.

[24] Victoria Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial intelligence review*, 22(2):85–126, 2004.

[25] Zoubin Ghahramani. An introduction to hidden markov models and bayesian networks. *International journal of pattern recognition and artificial intelligence*, 15(01): 9–42, 2001.

[26] Matt P Wand and M Chris Jones. *Kernel smoothing*. Crc Press, 1994.

[27] Eliahu Khalastchi, Gal A Kaminka, Meir Kalech, and Raz Lin. Online anomaly detection in unmanned vehicles. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 115–122. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[28] Mennatallah Amer, Markus Goldstein, and Slim Abdennadher. Enhancing one-class support vector machines for unsupervised anomaly detection. In *Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description*, pages 8–15. ACM, 2013.

[29] Mien Van and Hee-Jun Kang. Robust fault-tolerant control for uncertain robot manipulators based on adaptive quasi-continuous high-order sliding mode and neural network. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 229(8):1425–1446, 2015.

[30] Mien Van, Shuzhi Sam Ge, and Hongliang Ren. Robust fault-tolerant control for a class of second-order nonlinear systems using an adaptive third-order sliding mode control. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 47(2): 221–228, 2017.

[31] Ola Pettersson. Execution monitoring in robotics: A survey. *Robotics and Autonomous Systems*, 53(2):73–88, 2005.

[32] Eliahu Khalastchi, Meir Kalech, and Lior Rokach. Sensor fault detection and diagnosis for autonomous systems. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 15–22. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[33] NATO. STANAG 4754, NATO Generic Systems Architecture (NGVA) for Land Systems, Edition 1, Ratification Draft 1, November 2016.

[34] Generic Vehicle Architecture GVA. issue 1, 2010.

[35] Gerardo Pardo-Castellote. Omg data-distribution service: Architectural overview. In *Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on*, pages 200–206. IEEE, 2003.

[36] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, third edition, 2012.

[37] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57 – 95, 1987. ISSN 0004-3702. doi: https://doi.org/10.1016/0004-3702(87)90062-2. URL `http://www.sciencedirect.com/science/article/pii/0004370287900622`.

[38] Claudia Picardi. A short tutorial on model-based diagnosis. 2005.

[39] Ahmad Drak, Youssef Youssef, Anastassia Kuestenmacher, and Paul Plöger. Remote Fault Diagnosis of Robots Using a Robotic Black Box. In *The 27th International Workshop on Principles of Diagnosis: DX-2016*, Denver, Colorado, 2016.

[40] Stefan Loigge, Clemens Mühlbacher, Gerald Steinbauer, Stephan Gspandl, and Michael Reip. A model-based fault detection, diagnosis and repair for autonomous robotics systems. 5 2017.

[41] Saugata Biswas. Deployment of a Correlation Based Algorithm for a Robotic Black Box. 2017.

[42] Defence Standard 23-009 Part 3 Generic Vehicle Architecure ( GVA ) Part : 3 : Health and Usage Monitoring System ( HUMS ). (3), 2016.

[43] Stefan Loigge. Unified and Dependable Robot Control Architecture based on ROS. (September), 2016.

[44] RTI. RTI Connext DDS Professional Core Libraries Getting Started Guide. 2016. URL `https://www.rti.com/products/dds/`.