FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Selecting Sensor Redundancies in Cyber Physical Systems for Fault Detection using Ontological Information

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Thomas Preindl, BSc
Matrikelnummer 00825346

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Mitwirkung: Univ.Ass. Dipl.-Ing.. Denise Ratasich, BSc

Wien, 3. September 2019

Thomas Preindl	Wolfgang Kastner

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Selecting Sensor Redundancies in Cyber Physical Systems for Fault Detection using Ontological Information

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computer Engineering

by

Thomas Preindl, BSc
Registration Number 00825346

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Assistance: Univ.Ass. Dipl.-Ing.. Denise Ratasich, BSc

Vienna, 3rd September, 2019

Thomas Preindl	Wolfgang Kastner

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Thomas Preindl, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 3. September 2019

Thomas Preindl

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

First of all, I would to thank my advisor Wolfgang Kastner and my second advisor Denise Ratasich, for their enduring support, feedback and patience throughout the elaboration of this thesis.

Further I would like to thank my colleges and friends, especially Stefan and Martin, for the support, understanding and their humor which made the time so much more enjoyable.

Last but not least, I would like to thank my girlfriend Nina for her never ending encouragement and my parents Frieda and Toni as well as my siblings Gisela and Ralf, for their their mental and financial support throughout my life and this thesis.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

In recent years, the advances in computing performance and energy efficiency have led to the introduction of Cyber Physical Systems (CPSs) in every aspect of life and business. In concurrence with this trend, the dependency on such CPSs increased accordingly. For this reason, it is necessary that CPSs are designed with a high safety threshold which among other things leads to high dependability. To achieve these properties, different measures have to be taken. The availability and correctness of the sensors of a CPS, its eyes to the world, is increased to the necessary levels using redundancy as one of those measures. This redundancy can be achieved by adding additional sensors for the same measurement domain. Another approach is to use redundancies in the measurements that occur naturally because of the physical interconnection between different measurements.

In the underlying thesis, an approach is presented, that allows to derive these redundancies from ontological information of CPSs and to monitor sensors for faults in an efficient way. For this purpose, a simple modelling language is presented that enables the description of sensors and their physical connection. It is shown that the search for a group of redundancies, called a Grouping Solution (GS), for the purpose of monitoring is an NP-complete problem. Finally, an algorithm is presented that finds valid GSs and their quality is discussed as well as the complexity of the algorithm.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

In den letzten Jahren haben die Fortschritte in den Bereichen Rechenleistung und Energieeffizienz zur Einführung von CPSs in jedem Aspekt des Lebens und der Warenproduktion geführt. Im Einklang mit diesem Trend hat sich die Abhängigkeit von solchen CPSs entsprechend erhöht. Aus diesem Grund ist es notwendig, dass CPSs für hohe Sicherheitsanforderungen ausgelegt sind, was unter anderem zu einer hohen Zuverlässigkeit führt. Um diese Eigenschaften zu erreichen, müssen verschiedene Maßnahmen getroffen werden. Als eine dieser Maßnahmen kann die Verfügbarkeit und Korrektheit der Sensoren eines CPS auf das erforderliche Niveau gebracht werden, indem auf Redundanz gesetzt wird. Diese Redundanz kann durch Hinzufügen zusätzlicher Sensoren für die gleiche Messdomäne erreicht werden. Ein weiterer Ansatz ist die Verwendung von Redundanzen in den Messungen. Diese Redundanzen können auf natürliche Weise aufgrund physikalischer Verbindungen zwischen verschiedenen Sensoren vorhanden sein.

In dieser Arbeit wird ein Ansatz vorgestellt, der es ermöglicht, diese Redundanzen aus ontologischen Informationen über ein CPS abzuleiten und durch Vergleich Sensoren auf Fehler effizient zu überprüfen. Hierfür wird eine einfache Modellierungssprache vorgestellt, die es ermöglicht, die Beschreibung der Sensoren und deren physikalische Verbindung durchzuführen. Weiters wird gezeigt, dass die Suche nach einer Gruppe von Redundanzen, genannt Grouping Solution (GS), für den Zweck der Überwachung ein NP-vollständiges Problem darstellt. Zum Schluss wird ein Algorithmus präsentiert, der gültige GSs findet und deren Qualität sowie die Komplexität des Algorithmus diskutiert.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 1

Introduction

1.1 Problem Statement

In recent years, the advances in mobile computing led to Cyber Physical Systems (CPSs) with greater processing power, more degrees of freedom as well as sensory inputs and therefore greater complexity. In major industries such as the automotive sector, the development is decomposed into subsystems created by different suppliers due to the difficulties of the design and manufacturing of such a system. This approach has many advantages from reducing complexity for the individual engineer to lower production costs.

The new possibilities of these advanced CPSs however demand for strict safety considerations as more and more physical aspects of the systems are controlled by the cyber part and not by a user, e.g. in autonomous driving. Failures of the CPS have to be prevented with different measures. For the physical part, redundancy is used in most cases if the reliability of a subsystem or -module cannot be assured with a high enough probability.

Redundancy however can be very expensive. For instance, in a car the number of additional sensors, and the necessary communication network supporting them, increase the cost of a vehicle in many ways. Not only the production costs for the additional hardware but also the higher weight that has to be compensated by lighter construction materials or better and more effective technologies have to be considered.

However, in most of today's modular designed CPSs some kind of redundancies are present implicitly, although not in the form of modular replication. Often sensors measure values in different domains that are interconnected by the physical behavior of the system or its environment. Many systems are deliberately designed this way to use such redundancies for sensor fusion to enhance measurements, monitoring or fault tolerance. Nonetheless for systems not designed this way, such redundancies can be present. Finding such interconnections in modular CPSs can be a tedious and difficult task without some systematic and automated approach.

The main hypothesis of this thesis is that, a method for deriving these redundancies from ontological information and using them for fault detection would solve the problems mention so far.

1.2 Aim of the Work

The main objective of the thesis is to find ways to open up the potential of implicit redundancies in a CPS for fault detection. For this, a formal description of a system model shall be developed focusing on possibilities to describe interconnections between sensors.

Further algorithms shall be proposed for the derivation of hidden redundancies and the detection of faults using these. The challenges thereby shall be to find groups of sensors to detect faults in an efficient way including how to select the sensors to compare and the comparison method such that an algorithm can be run on a high rate without using too much system resources.

Additionally, a detection method for missing fault detection capability shall be developed which would allow to alert a developer if the fault detection is not covering all sensors of the system and additional sensors would be necessary.

1.3 Methodological Approach

The methodological approach consists of the following steps:

- Research Literature:

  A deeper research in the field of sensor faults and their detection as well as on the topic of ontological modelling shall be done. Further, the literature on graph search for efficient grouping of related sensors shall be studied.

- Modelling:

  Before any implementation, a model representation of the problem and its solution shall be developed and the difficulties and problems for a potential implementations shall be analysed.

- Implementation:

  An algorithm for finding groups of sensors for fault detection shall be implemented as a proof of concept.

- Evaluation:

  The quality of the solutions generated by the implementation shall be analysed in a generic way as well as the complexity of the presented implementation.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

1.4 Structure of the Work

The structure of the work is aligned to the methodological approach. The findings of the literature review are presented in chapter 2. It gives an overview on the basic concepts that are important for the analysis of the problem, the implementation and the evaluation. In chapter 3, the problem of selecting senors redundancies is analysed. First an ontology for the description of sensors and their connections is presented. Based on this model, a possible representation of a solution is discussed and finally the complexity of computing such a solution is analysed. In chapter 4, an algorithm for the selection of sensor redundancies is presented. This algorithm is analysed in chapter 5 in regards to the quality of the solutions it can generate and the worst case running time that is needed to complete the computation. Finally, in chapter 6 a summery of the presented findings is given and possible topics of interest for future work are described.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 2

Basic Concepts and Related Work

In this section, we give an overview on the important basic concepts and related work. First faults, errors, failures and fault-diagnosis systems are introduced, as they form the basis of the topic. Then several concepts that are used throughout the work are discussed. Ontologies are described because a description language of the instances of our problem is needed. Then set theory and graph theory are presented. They represent the mathematical basis for representation of the model during the discussion. Thereafter computation and its complexity are discussed before describing how the mathematical concepts can be used during computation by using certain algorithms and data structures. The final section of this chapter will present related work.

2.1 Fault-Diagnosis Systems

In the last 50 years automation found its way in almost any part of industrial production and daily life. The advances in computation technologies facilitated this process and in return pushed the advances even further with the demand of more and cheaper computational performance. The automation process was at first an industrial one and later entered the daily life of most humans helped along by the capabilities of cheap and energy efficient micro-controllers that power most of our appliances and most importantly everyone's smartphone [1].

Most of these system have a common architecture we will describe in terms of process automation. However, similar abstractions can be found in any CPS. In Figure 2.1, a simplified schema of process automation is shown. The purpose of every automation system obviously is the automation of a physical process. To achieve this, the process plant or the machinery is equipped with sensors and actuators. The automation system uses the readings from the sensors to compute the appropriate actions and uses the actuators for the execute thereof. A supervision system checks the actions of the control system continuously and may adapt or stop the control system in case the process enters an undesired or dangerous state. Such states can arise from faults and errors that can be caused by any influence on the system [1].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

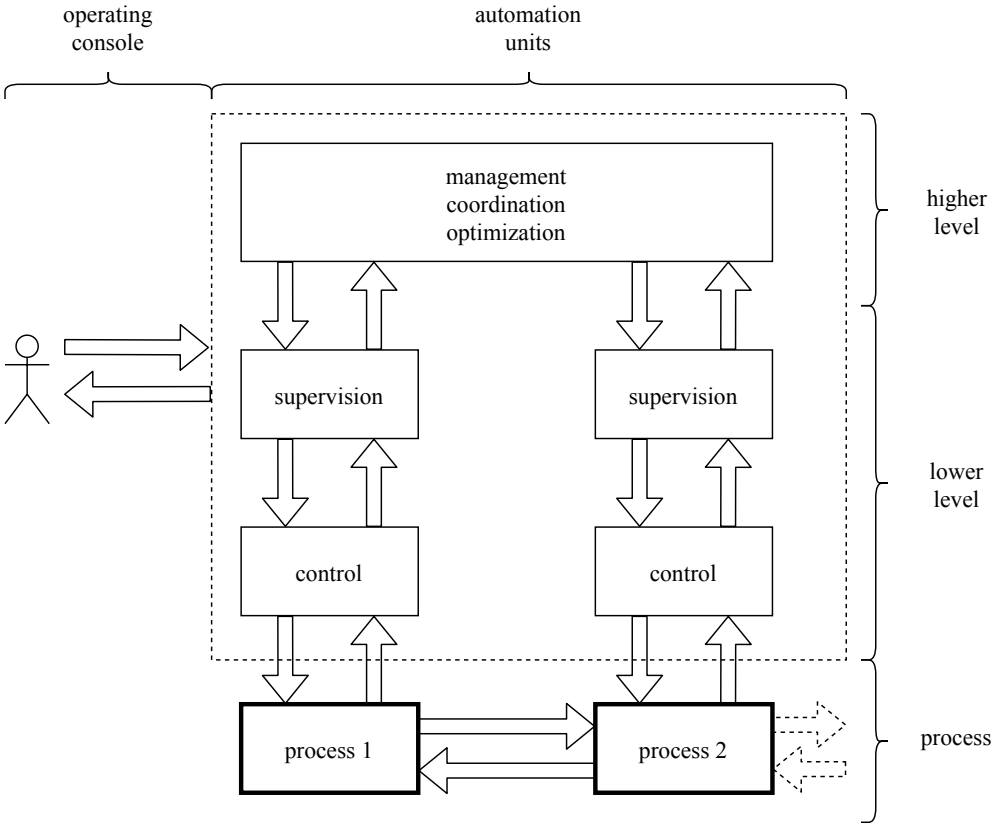

Figure 2.1: Simplified schema of process automation [1].

It is important to distinguish between faults, failures, and errors or malfunctions. Faults are defined as any deviation of a property of a system that is not usual nor acceptable. Such a fault, if it is not mitigated, will cause an error in the system i.e. the entering of the system into a not permitted state. If the system is not reset to a normal state, this error may cause the interruption of the designated operational behavior which is called a failure. The quality of a system to abstain from failure modes is called reliability and is measured in the Mean Time To Failure (MTTF). A certain reliability is needed if a CPS is able to cause harm. The ability to prevent harm and danger is called safety and has a certain risk threshold. There are several possible methods to assure a certain safety of a system. One example is to go into a safe state upon the detection of a failure that may be dangerous. The drawback of this approach is however, that the system cannot provide its service while in the safe state. The probability of a system to be operational, called availability, is therefore an important measure of quality for a CPS. A system that is reliable, safe, secure and available is called a dependable system. A major role of the supervision system in a CPS is to diagnose fault or errors and prevent failures from manifesting. Further, it can be designed to mitigate failures in a self healing manner [1, 2, 3].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

2.2 Ontologies

The term 'ontology' originally describes a branch of metaphysics, concerned with the description of nature, the parts in nature and their relations. In the effort of teaching the world to computers, the field of Artificial Intelligence (AI) adopted the term 'ontology'. In this field, 'ontology' is defined as "a formal, explicit specification of a shared conceptualisation" [4, 5]. 'Conceptualisation' in this context means an abstract model of some phenomenon or aspect of the world accompanied by the description of relevant concepts and relationships thereof. The term 'formal' means that the ontology has to be machine readable and further 'explicit' means that, the means of concept description are well defined and are governed by a fixed set of rules. In union, they assure that the ontology is understood in the same way by humans and machines. The description of phenomenon or aspect in an ontology has to be 'shared' i.e. representing consensual knowledge in a group or even the world [4, 6, 7].

However, this group or world needs a method to define the consensual knowledge. Hence a consensual language has to be defined first. There are many such ontology languages. Examples are the Resource Description Framework (RDF), the Resource Description Framework Schema (RDFS), and its successor the Web Ontology Language (OWL). With the OWL it is possible to describe classes, their connections, relations and properties in a rich, formal and expressive way. Using these concepts a consensual knowledge representation can be built and transcribed using the OWL standard. This process results in an eXtended Markup Language (XML) document, which is readable perfectly for a computer and rather hardly for a human. To counter this problem, a graphical representation of the concepts form the OWL was defined in the Visual Notation for OWL Ontologies (VOWL) [8, 9].

The described formality of the language allows for reasoning on the aspects, their concepts and relationship and therefore the extraction of additional facts not expressed in the ontology. This allows for a minimal representation of the knowledge without the need for the encoding of redundancies. The formality additionally allows for the combination of multiple ontologies, a so called matching. A matching describes the connection of concepts in one ontology to concepts in another. Given an instance of an ontology, such a matching can also be used to create an instance of the other ontology. This process is called a model transformation [6, 7, 10].

2.3 Set Theory

In this work, it is often necessary to talk about unknown collections of sensors and collections of functions, that can be used to transform measurements from the domain of on sensor to the one of another. Collections or groups, used to reason about problems in an abstract and generalized way, are called sets in mathematical theory and the objects that are grouped together in a set are called its members or elements. Formally, we write $x \in R$ ($\in$ reads "is an element of" or "is in") to describe the fact that object x is an element of set R and $\notin$ to describe the opposite. The definition of a set can be any precise description. This can either be an enumeration of the elements e.g. the set containing the elements $2, 4, 6, 100$ and 20423, or a general description e.g. the set of even integers. The description of sets using prose is one possibility to define sets. As

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

these definitions can be rather long curly braces are used as notation. For example, we can express the sets from above as $\{2, 4, 6, 100, 20423\}$ and $\{x \in \mathbb{Z} \mid x \pmod 2 = 0\}$. The vertical bar in the second definition is read as "such that". In this way, it is possible to describe a set by filtering the elements of another set according to a certain condition, which in our case is the property of evenness [11].

Now that we can define sets, a basic question that comes to mind is the one of its size i.e. the number of elements. Given a set S the number of elements or its cardinality is denoted by $|S|$. In the case of our example sets, the cardinality is 5 and infinity. It is however not necessary to count the element of a thread to argue about its cardinality. For example, given two sets A and B and a mapping from each element of set A to exactly one element in B, also called a one-to-one mapping, then $|A| \leq |B|$. If such a mapping does also exist for the opposite direction then $|A| = |B|$ [11].

The cardinality is only one property to reason on the differences or commonalities of two sets. For example, if all elements of a set A are also elements of a set B then A is a subset of B, also denoted by $A \subseteq B$. If however $A \neq B$ then A is a proper subset of B, which is written as $A \subset B$. In a more general case, in which sets A and B are not subsets of each other, other elementary operations are needed to argue about these sets. We call the set of all elements that are members to both A and B their intersection and write $A \cap B$. The set of elements that are in either A or B or in both is called union, denoted by $A \cup B$ and the set of elements that are members of A but not members of B is called the difference, denoted by $A \setminus B$. Finally, the set of elements that are in either A or B but not in both is called symmetric difference and is denoted by $A \oplus B$ or $A \triangle B$. In Figure 2.2, Venn diagrams for these operations are shown for better understanding [11].

These elementary operations on two sets is however not enough for the models presented in this work as we will reason on sets of sets. Such sets of sets are also called collections, systems, or families of sets. An example of a family of sets is the power set, the set of all subset of a set A which is denoted by 2^A or $\mathcal{P}A$. The notation hints on the fact that the power set for any set A has a cardinality of $2^{|A|}$ [11, 12].

Although the set is a simple concept, it is one of the most fundamental in mathematical theory as most of the basic concepts such as counting, functions or even the natural numbers can be defined using set theory. This topic is however out of scope for this work. What is needed for the reasoning in the following chapters is the notion of order. The basic building block of an ordered set is the ordered pair written as (a, b), which is defined as the family of sets $(a, b) = \{\{a\}, \{a, b\}\}$. Using the ordered pair, it is possible to define ordered triples $(a, b, c) = ((a, b), c))$, ordered quadruples $((a, b, c, d) = ((a, b, c), d))$ and so on for longer sequences [11].

2.4 Graph Theory

The mathematical modelling of the problem of selecting sensor redundancies is heavily based on graph theory. Therefore, an introduction to the important concepts and the terminology is given in this section. A graph is a pair $G = (V, E)$ of sets with the property that $E \subseteq V^2$. Please note that we will use the notation $G(V, E)$ as a shorthand for $G = (V, E)$ throughout this work. V^2 is the family of sets containing pairs of elements from the set V, thus each element in E is a pair of

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

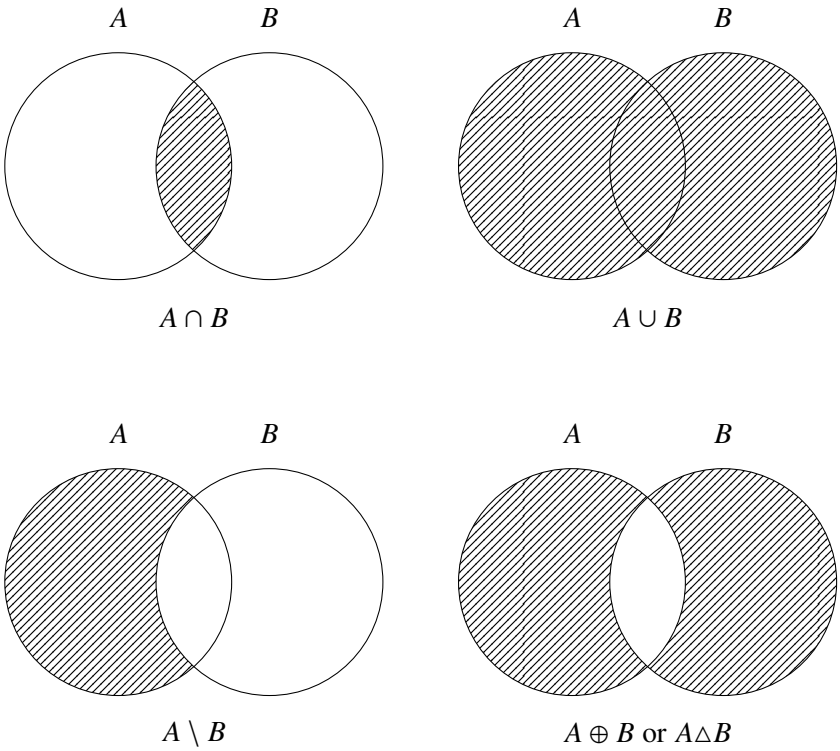

Figure 2.2: Venn diagrams for the intersection, union, difference and symmetric difference of sets A and B.

elements in V. We call the elements of V vertices or nodes of the graph G and the elements in E edges. In Figure 2.3, a graphical representation of a graph is shown with vertices drawn as dots and edges as lines between the pair of vertices they represent. The representation can however take many forms as long as the concept of objects and their pair-wise association is communicated [13].

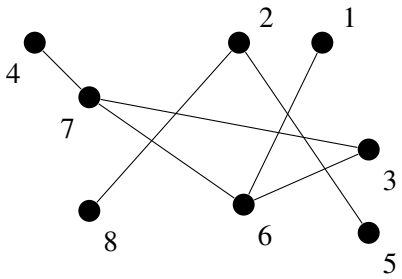

Figure 2.3: Graphical representation of graph $G(V,E)$ with $V = \{1,\ldots,8\}$ and $E = \{\{4,7\},\{2,8\},\{3,6\},\{2,5\},\{1,6\},\{3,7\},\{6,7\}\}$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

This pair-wise association encoded via an edge can be described using many terminologies. If an edge between two vertices in a graph exists then the two vertices are adjacent or neighbours, if two edges have a vertex in common the term adjacent is used as well and if all vertices in a graph are pair-wise adjacent than the graph is called a complete graph. Hence the case of a complete graph every vertex has an edge to all other vertices. The number of edges containing a vertex is therefore $|V| - 1$. This number is called the degree of the vertex and if a vertex is of degree 0 it is isolated. Given two adjacent vertices of degree 1 the two are not connected to the rest of the graph despite them not being isolated. If there exists a partition of the set of vertices in a graph, such that there is no edge with a vertex in each part, then the graph is disconnected. If no such partition exists the graph is connected [13, 14].

In a connected graph $G(V, E)$, there exists a sequence of edges for any two vertices in the graph such that the any successive edges in the sequence are adjacent. This sequence can be described by a sub-graph $P(V_P, E_P)$, which means that $V_P \subseteq V$ and $E_P \subseteq E$, with $V_P = \{x_0, x_1, \ldots, x_k\}$ and $E_P = \{\{x_0, x_1\}, \{x_1, x_2\}, \ldots, \{x_{k-1}, x_k\}\}$. This sub-graph is call a path which connects the vertices x_0 and x_k and is of length k with is determined by the number of edges $|E_P|$. For any vertex in the path there are at most two edges hence any vertex is only passed once on the path. If we construct a graph given a path from x_0 to x_{k-1} by adding the edge $\{x_{k-1}, x_0\}$ then the new graph is called a cycle and its length is k [13].

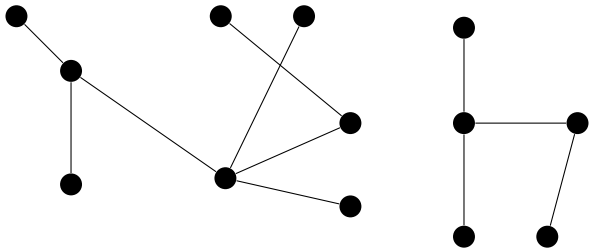

Figure 2.4: A forest

If a graph does not contain any cycles or equally if it is an acyclic graph, it is called a forest. In Figure 2.4, such a forest is presented which is comprised of two components that are connected sub-graphs of the forest and therefore forests as well. Connected forests are called trees and the vertices of degree 1 are called leaves. Other important graphs are for example the complete graph, which contains an edge between any two vertices, the empty graph, which contains no edges and the bipartite graph, which can be partitioned in two. What this means is that the vertex set of the graph can be partitioned in two sub-sets such that any edge in the graph connects a vertex from one partition with a vertex of the other partition, hence the vertices contained in one partition are not adjacent to each other. An important concept for such graphs is the vertex cover. A vertex cover is a subset of vertex set such that at least one vertex of any edge is a member of the vertex cover [13, 14, 15, 16].

Although there are these specific graph types, for some problems, these are not suitable as a representation. For example, when modeling the flow traffic in a network of streets or the flow of water in a network of rivers there arises the need to encode the notion of direction. This is the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

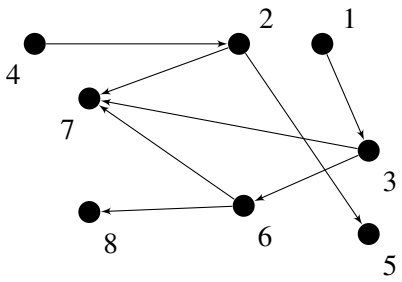

Figure 2.5: A digraph

reason a directed graph or digraph is needed. Such a graph is similar to a normal graph. The only difference is that the edges have a direction and are therefore encoded as an ordered two-tuple of vertices rather than a set. Edges of directed graphs are often called arcs in the literature, we will however use the term edge for the directed and undirected versions throughout this work. Because of the direction of the edges the neighbours of a vertex have to be distinguished between in-neighbours and out-neighbours. Similarly the degree of a vertex is split between the in-degree and the out-degree [17].

2.5 Computation and Computational Complexity

Through the history of the past two millennia, the idea of computation was known in different forms. A simple meaning in everyday life is associated to the process of taking some input information and producing output information by performing a finite number of steps. In the 20th century, more formal models for the notion of computation were introduced. Turing machines, lambda calculus, and cellular automate are only some examples of those models created to argue about computation. Following the definition of those models, it was shown that they are equivalent i.e. that any computation in one model can be simulated in the other models as well. However, the major breakthrough in the understanding of computation was the discovery, that all these models are universal which means that any computation that can be conceived of in any other model can be implemented in those models [18].

The discovery of the idea of universality was followed by extensive research on the topic of computability. An important result of this research was the fact that there exist computational tasks that are uncomputable, as any computer will end up in an infinite loop when processing these tasks. Another research topic, that arose from the discovery of computational universality is the issue of computational efficiency. For tasks that are computable, which naturally are of greater interest to the field of software engineering, the question what amount of computational resources is necessary to produce a result is of great interest. The theory on computational complexity is concerned with this question and its many nuances [18].

Much like the computational models were introduced for the purpose of having concepts to discuss the problems at hand, the theory of computational complexity needs such models and concepts as well. In a precise approach, the resources utilization can be described giving the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

sum of the cost of all computation steps. For example, given the instance x a problem with size $n = |x|$ the cost could be expressed by the function $an^2 + b \log_2 n + cn$ where a, b, c are constants that depend on the cost of the steps in the computation. As the cost of computation steps may vary, a comparison with other approaches in different models is not possible and hence some simplifications are necessary to get a more abstract representation. For this purpose, the cost of the individual computation steps is ignored, counting each step as a unit [19].

The comparison of different implementations of algorithms can remain difficult however, as the number of terms in the function may be extensive. Therefore, a further abstraction is necessary. This is achieved by only considering "the rate of growth or order of growth" [19, p.28] of the computational resources relative to the size of the instance of the problem. In the case of our example from above only the term an^2 is considered due to the fact that the other terms become insignificant relative to it when n is large. The same fact holds for the constant coefficient a leading to the term n^2. To formalize these abstractions the asymptotic functions O, Ω and Θ are used for the notation. The O-notation (pronounced "big-oh") describes the upper bound of a function. Formally if $f(n) = O(g(n))$ then there exist constants $c, n_0 \geq 0$ such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$. Analogously the Ω-notation (pronounced "big-omega") describes the lower bound. In case a function is bound from below and above i.e. $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$, then the Θ-notation is used. Therefore, the computational expenses $T(n)$ of our example can be expressed as $T(n) = \Theta(n^2)$ [18, 19].

As already mentioned in the discussion on the order of growth, only the term that grows the most is important for the discussion of the complexity of an implementation. This term can be used to classify problems and algorithms according to the other of growth, which can be for example constant, logarithmic, linear, linearithmic, polynomial or exponential. While the running time of an algorithm with a constant order of growth can be solved as the name states in constant time, the running time gets longer proportional to the size of the instance for each element in the list of examples. All algorithms with an order of growth in running time up to a polynomial order of growth in the size of the input can be executed practically also with huge inputs. For algorithms with an exponential order of growth the computation of larger instances will not complete in a trivial amount of time and hence the algorithm will be run to completion only for small instances [20].

As mentioned before, the running time of an algorithm is only a metric for the efficiency of the implementation and an upper bound for the complexity of a problem. If for example the best algorithm for a problem is known to grow exponentially relative to the size of the input, there could none the less exist a better algorithm with a running time that is polynomial relative to the input. On the other hand, the known algorithm could already be the most efficient algorithm possible, if the running time of an instance has a lower bound that is determined by the problem and not by the algorithm. This lower bound for the running time of an algorithm can be used to classify the complexity of the problem. All of the named examples of the orders of growth can be used to define a complexity class as all problems that are solvable in a running time that is bound by the same order. For example, the class P is the class of all problems that are solvable in polynomial time i.e. $T(n) = O(n^c)$ and the class EXP consists of problems solvable in exponential time i.e. $T(n) = O(2^{n^c})$ [18, 21].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

While the aforementioned classes are defined based on the order of growth, there exist certain problems that seem to belong to a class of lower complexity, however only algorithms of higher complexity are known. One such set of problems is the class NP. It is defined as the class of problems for which a polynomial-time algorithm exists, that can check the correctness of a solution. Although it is conjectured by many researchers that P≠NP it has not been proven to this date. The definition of the class NP does not say anything about the actual running time that is required to compute a solution. It can be shown for a program however, that it is at least as complex as any other program in the class NP. This property is called NP-hardness and has major consequences. In the case of P=NP it follows from this fact, that every problem in NP is in fact solvable in polynomial time. If a problem is in the class NP and is shown to be NP-hard it is NP-complete [18, 22].

As mentioned, it is necessary for the proof of NP-hardness to show that a problem is at least as hard as all problems in NP. Fortunately, there is a set of problems that are known to be NP-complete. Therefore, it remains to show that a problem is at least as hard as one of these problems and NP-hardness does follow immediately. The proof itself can be done by using a polynomial-time reduction from a NP-complete problem A to a problem under investigation B, i.e. if there exist a polynomial-time algorithm that can transform any yes-instance of A to an instance of B such that a solution can be found and translated back to A again in polynomial-time. If such a polynomial-time reduction exists problem A is said to be polynomial-time reducible to B and because of A being NP-complete it follows that problem B is NP-hard. However, this proving technique does only work if there exists a problem, that is known to be NP-complete. Fortunately it was shown using an independent proof that boolean satisfiability is NP-complete without the use of a recursive definition. Soon after this proof Karp showed that many problems are NP-complete. We will use **vertex cover**, one of these problems, to prove NP-hardness of our problem at hand. It is defined as follows [18, 22].

Vertex cover. Given an integer K and an undirected graph $G(V, E)$, then the problem is to decide, whether there is a vertex cover of size K or less. More formally, the problem is to decide, whether there is a subset C of the vertices V such that $|C| \leq K$ and that for every edge $\{x, y\} \in E$ either $x \in C$ or $y \in C$ or both [18].

2.6 Algorithms and Data Structures

In the previous sections, we described the important mathematical structures that are needed for the discussion in the following chapters. As already mentioned before, the most important concept will be the set. For the analysis of the complexity of the problem and the implementation we need to know how sets can be represented as data structures and how the operations on the sets can be implemented. In the programming language Python, we use for the proof of concept, the set lookup is implemented according to Algorithm "*Open addressing with double hashing*" from *The Art of Computer Programming, Volume Three: Sorting and Sarching*. The set elements are stored in a table. The position of each element is determined using two hash functions. The first one is used to determine an absolute position and the second one to iterate through alternative positions until an empty slot is found. The use of the hashing function allows for insertion and

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

membership tests that will be constant on average. In the worst case however, the slots of all the other members are hit before the insertion and therefore the running time is linear in the number of members. It has been shown that it is possible to define the hash functions in a way such that the insertion pattern is random enough that the worst case is very uncommon. The set operations use the lookup and/or insert method. For example, for the union the elements of both sets are simply inserted in a new set, for the intersection, the membership of each element in one set is tested on the other set and all positive candidates are added to a new set. For the other operations the procedures are similar [23, 24].

Another concept that is important for our implementations is the coroutine. A coroutine is generalization of the subroutine. While a subroutine has is called by a main program or a higher level subroutine a coroutine becomes its own program in a sense. The concept can be used for non-preemptive multitasking. A coroutine interrupts its execution by raising an exception or by calling a scheduler and resumes at a later point in time when another coroutine gives up control as well. The state of a coroutine is saved when it is interrupted and later restored when it is resumed. In contrast to a thread or a process the amount of memory that needs to be handled is very low. This makes coroutines a prime candidate to use for concurrent processing. In Python special syntax was introduced for coroutines. The "async def" defines a coroutine and the "await" statement can be used to wait for the result of another coroutine [25, 26].

The concept of coroutines is used together with another concept called generators. Generators are specialization of coroutines that implement an iterator pattern. The statement "yield" was added to Python for this purpose. When executed, it interrupts the execution of the containing subroutine and returns a value. When the next value is requested the execution resumes from the point where it was interrupted. Generators allow for lazy execution which helps to keep a low memory footprint, as every value is calculated when it is needed and consumed instantly. In the implementation, we combine these two concepts to return multiple values from coroutines [27, 28, 29].

2.7 Related Work

Ontological modelling is an important topic in the design of CPS. This can be seen in the case of the IEEE Autonomous Robotics Subgroup (AuR) aiming for a standardization of an ontology for the design of autonomous robots, a sub-field of CPS design. They present their work in progress on the requirements for such an ontology for autonomous robots in [30] including important core concepts needed and practical implications.

A great amount of work has further been done in the last years to apply ontology modelling to adaptive CPS. Martí et al. present an adaptive sensor fusion architecture using ontology modelling and automatic reasoning. The architecture consists of virtual sensors and widgets. They abstract the process of capturing sensor values and give a standard interface to components of the fusion system. They propose a description ontology for the problem space consisting of class objects to define class hierarchies and class properties to describe fusion-problems. Further, a context ontology is used to describe possible contexts and context constraints are defined using a rule

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

engine. The information from the different ontology models and the rule list are used to adapt the sensor fusion architecture according to which sensors are disposable in the current context. [6]

Similarly, Höftberger et al. propose a framework for re-configurable embedded real-time systems. Using knowledge about the system structure and semantics in the form of a system ontology the framework enables dynamic replacement of failed services by finding redundancies. Ratasich et al. present an extended framework for self healing by structural adaptation and show how an optimal replacement can be found using a property guided search [31, 32, 33, 34].

On the topic of sensor faults, Sharma et al. give an overview over different sensor fault modes and possible detection methods as well as a comparison of the different approaches. Although the work is based on wireless sensor networks the fault models do also apply for other sensor configurations. Verma et al. give a similar overview about the detection and diagnosis of faults in real-time [35, 36].

The topic off fault detection and isolation has seen a great amount of work over the last years. A widely used method is the observer based method. Boulaabi et al. as well as Loza et al. propose fault detection using a sliding mode observer over a group of sensors [37, 38].

The topic of selection sensors and fusion algorithms to achieve the biggest reliability is studied by Cohen and Edan. They present a framework for the selection of the most suitable sensors during the mapping of an unknown environment. Their approach for the selection is statistical in contrast to our work which is based on logical deductions [39].

Goel et al. propose a method for the detection and identification of faults in wheelded mobile robots using multiple model estimation and a neural network. Several Kalman filters are used as estimators for each failure mode and the neural network is used for the decision of which failure mode is imminent. Hashimoto, Kawashima, and Oba preset a similar approach using different estimators for the detection [40, 41].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 3

Selecting sensor redundancies for fault detection

In this chapter, the problem of selecting sensor redundancies in CPS for fault detection is discussed. In section 3.1, a minimal ontology for the description of physical quantities of a CPS and the connections thereof is presented. Based on this ontology, a formal definition of a System Description (SD) is given. Further in section 3.3, the concept of Voting Groups (VGs) is introduced that determine how measurement values of different sensors are used to detect discrepancies. Their features and limits are discussed which leads to section 3.4 where the combination of VGs to a so called GS for the detection of faults and their attribution to sensors is proposed, formal constraints concerning the correctness and the operational cost of a selection of VGs are given and their validity is proven. Finally in section 3.5 a formal definition of the problem of selection sensor redundancies is given and its computational complexity is analysed. With this, this chapter sets the theoretical foundation for the implementation presented in chapter 4.

3.1 System Description

CPSs generally consist of many parts that together enable the fulfillment of a designated purpose the system was designed and built for. All these parts have a state that may be of interest to the control-module. Therefore, sensors are installed to measure the different physical quantities of the individual parts of the system. Due to the fact that a CPS is part of the real world, all physical laws apply to the parts and the system as a whole. This confines the values of the physical quantities of the individual parts. For example, imagining an autonomous robot, the position of a module for GPS-based localization and the charging port may be fixed on said robot and hence there relative position does never change during the operation. This makes it unnecessary to measure the absolute position of the charging module as the relative position is known and can be accounted for when the robot drives into the charging station. Similarly, two parts of a robot arm may be connected via a hinge. Their positions and orientations are therefore interlocked and

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

only free in one axis, the axis of the hinge. Given the position of the first part and the relative angle of the second part, the absolute position of the second part can be determined and the robot can for example pick up some object with a gripper fixed to the second part of the arm making additional sensors unnecessary.

Where the inevitable influence of the laws of physics allow for the reduction of necessary sensors, by contrast, our approach aims to use such unnecessary sensors to detect the faults in these sensors. The presented aspects of physical quantities that are transformed into other physical quantities remain the same. The abstract nature of this approach makes it necessary to use a more formal language. Therefore, we call the physical quantities connected to parts of the system Value Domain (VD) and the transformations Value Relation (VR) and introduce the ontological model.

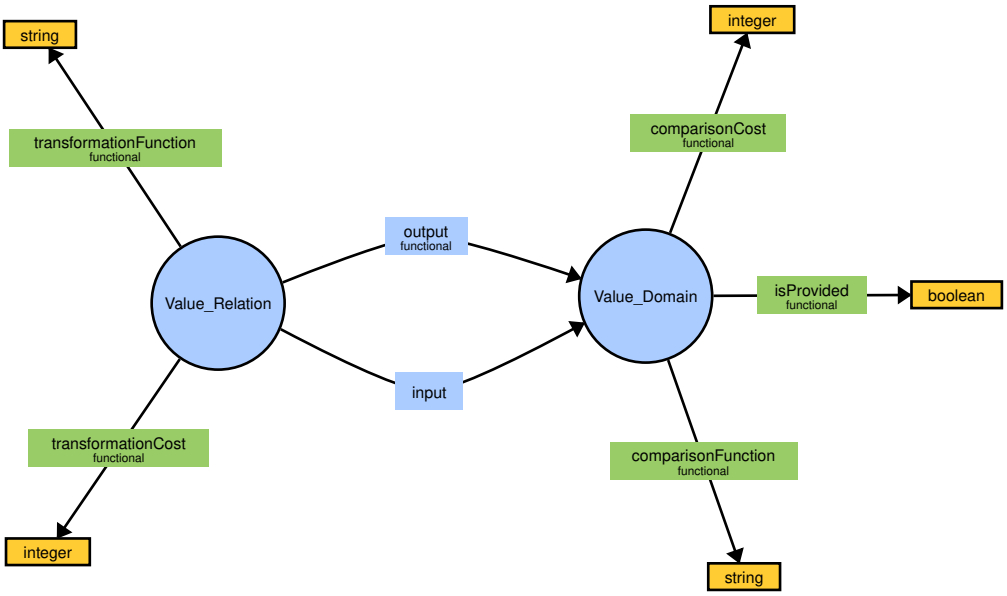

Figure 3.1: Classes of the ontological model using VOWL2 [9].

As can be seen in Figure 3.1, two classes are defined, one for VD and VR, respectively, together with some properties. A VD represents a state or physical quantity of a part or module of a CPS. Applied to the above examples, the position and orientation of the localization module, of the charging port, of both arm parts as well as the position of the grabber and further the aperture angle of the hinge between the arms are all instances of the class VD. The property *comparisonFunction* describes the function to be used to compare measurements in the VD. It is represented by a string that can be a formula or source code depending on the implementation of the monitoring system. Further, the property *comparisonCost* describes the computational cost to execute the comparison function and is represented as an integer value. Formally, given a VD d, we denote the property *comparisonFunction* as f_d and the property *comparisonCost* as c_d respectively. Lastly, the Boolean property *provided* of a VD indicates whether values are measured by a sensor. Formally, we denote the property as $\overline{p}_d$ given a VD d, i.e. if we name the hinge angle, the position of the localization module and the position of the first arm part as VDs

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

a, b and c respectively then because all three are measured by some sensors, the terms $\overline{p}_a, \overline{p}_b, \overline{p}_c$ are all equal to true. On the other hand, for all the other VDs, such as the position of the second part of the arm as well as the position of the gripper, the property is false. It is however possible to determine the respective positions using VR that describe how the positions can be determined using the sensor values of the provided VDs.

The class VR encodes the connection between VDs i.e. VR objects describe the relation between different VDs and how values can be transformed from the input VDs to the output VD. Giving an example, the information that the charging port is located at a certain distance in a certain direction from the localization module can be encoded into a function to calculate the absolute position of the charging port using the position and the orientation of the localization module as input. That function is represented by the property *transformationFunction* and can be a mathematical formula or program code that should be executed to get the transformed values. It is modeled as a simple string that may contain such formulas or code in a format that can be interpreted when calculating the resulting value. Note that the calculation does naturally incur costs for the CPS running it. The property *transformationCost* describes this cost in a quantified way so that comparisons of different VR are possible. The integer value of this property can represent execution time as well as memory usage or energy usage or even the result of a cost function combining these and more performance metrics. This concludes the internal properties of the class VR.

On the other hand, the properties *inputs* and *output* are defined to represent the connections of the different VDs. The property *inputs* is a set of VDs that represent the inputs to the transformation function. Similarly the property *output* represents the output VD. In the case of the mentioned transformation from localization module position to the absolute charging port position, the property *inputs* consists of one VD, the localization module position and the property *output* is equal to the VD that represents the absolute position of the charging port. If on a different robot the relative position is not fixed that information too would be a VD that is provided. Then the property *inputs* of the VR would include this additional VD. Similarly in the case of the robot arm in which the position of the VDs representing the first part and the hinge angle constitute the value of the property *inputs* and the VD representing the position of the second arm part, the value of the property *output* of the VR describes the connections between the arm parts. In composition, these concepts enable the definition of physical quantities of individual parts of a CPS and the encoding of individual physical connections thereof.

To make this ontological model useful in the reasoning process following in the next chapters and sections, it is necessary to define formal denotations for the presented model. Therefore, given some VRs named r and with input VDs named a and b and output VD named c the term $\mathcal{I}_r$ shall denote the property *inputs* and the term o_r shall denote the property *output* and therefore in this case $\mathcal{I}_r = \{a, b\}$ and $o_r = c$. In a similar manner, the terms c_r and f_r shall denote the value of the properties *transformationCost* and *transformationFunction*, respectively. Similar to a VR a VD does have inputs and outputs as well in the sense that if a VD is an output of a VR this VR is an input to said VD. Hence, for sake of the readability of the definitions following in this chapter we define shorthand terms. Given a VD d the term $\mathcal{I}_d$ shall denote the input VR of the VD i.e. given a set $\mathcal{R}$ of all VRs defined in a system then $\mathcal{I}_d = \{r \in \mathcal{R} | d = o_r\}$. In the case of

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

the example $\mathcal{I}_c = \{r\}$. Analogously the shorthand $\mathcal{O}_d$ shall denote the output VRs of VD d i.e. $\mathcal{O}_d = \{r \in \mathcal{R} | d \in \mathcal{I}_r\}$ or in the case of the example $\mathcal{O}_a = r$ while $\mathcal{O}_c = \emptyset$. It can be seen that these shorthand terms are only usable if the set of all VRs is determined and therefore make a SD necessary.

A SD has at least to contain all the physical quantities that its sensors can measure. These quantities are represented by a set of provided VDs that shall be called $\mathcal{P}$. Further, the physical connections between those provided VDs themselves and other VDs can be included. They are represented by the set $\mathcal{R}$ of all VRs. Therefore, we define the System Description (SD) to be denoted by $\mathsf{S}(\mathcal{P}, \mathcal{R})$ consisting of the set $\mathcal{P}$ and the set $\mathcal{R}$. Note that the set of all VDs $\mathcal{D}$ of a SD is composed of the set $\mathcal{P}$ and VDs linked via the VRs *inputs* and *output*. Due to the definition of the VR the SD forms a directed bipartite graph of VDs in one part and VRs in the other part as the inputs and the output of a VR can only be VDs.

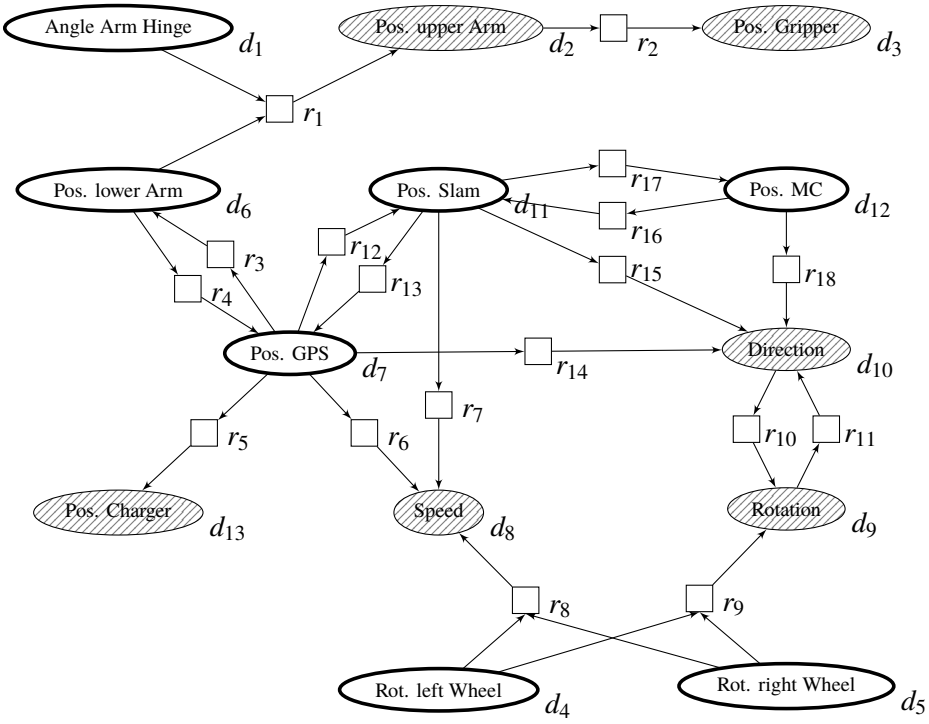

Figure 3.2: SD of a robot with multiple localization systems. Bold borders mark provided VDs.

A graphical representation although not in a bipartite layout of a SD can be seen in Figure 3.2 that includes the examples of positions of the arm and the charging port. Additionally, multiple localization systems are built into the robot and therefore also present in the SD. The localization module (d_7) described before is one of them. A localization module using a Simultaneos localization and mapping (SLAM) (d_{11}) approach and one based on IR motion capturing (d_{12}) are also integrated. It is possible to transform values of the VDs corresponding to these modules to the VDs corresponding to speed (d_8) or direction (d_{10}) using VRs r_6, r_7, r_{14}, r_{15} or r_{18}. The

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

transformed values can than be compared to determine if a sensor is faulty.

Further, an alternative way to determine speed and also direction is the usage of the rotational sensors of the right (d_5) and the left (d_4) wheel widening the possible solutions to the selection problem. It can be therefore seen that the detail of the description influences the performance of the monitoring system as more information about the connection of physical quantities can enable more pairs of sensor measurements to be compared. On the other hand, the process of selecting pairs or groups of sensors for the comparison will be more computational intensive as the number of possible solutions grows.

Another aspect of the SD as defined here has to be noted. The fact that VR have defined inputs and an output implies that the connection described by a VR is inherently directional i.e. the existence of a VR between two VDs does only describe how values can be transformed in one direction. If for example the transformation function is reversible than the direction of the transformation can be changed by solving the transformation equation for a certain input. This is however outside the capacity of this model and has to be done on a higher abstraction layer allowing for an automatic derivation of VRs for each additional direction. The VRs r_3 and r_4 are one example of such a bidirectional relation represented by two VRs. Although this property makes additional VRs necessary for the description of a system, it enables the usage of individual transformation directions in the monitoring system.

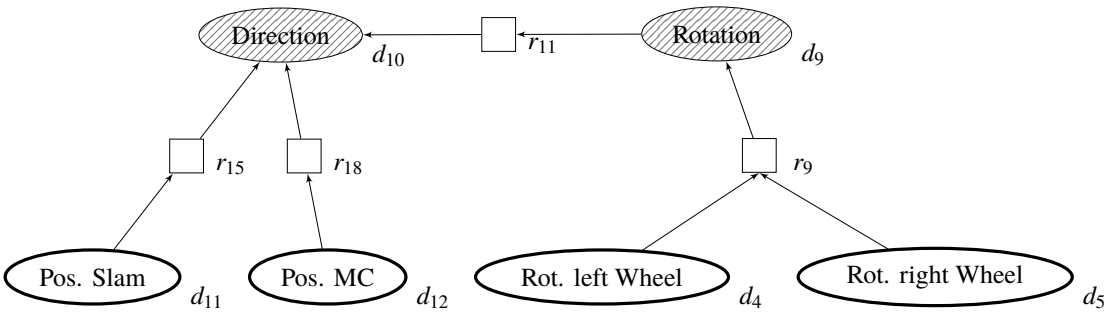

Figure 3.3: Sub-SD of the SD in Figure 3.2.

From the definition of the SD it may have been noticed that it would be possible to describe the structure of a SD only by the set of VRs $\mathcal{R}$, due to the fact that the set of VDs can be determined from the inputs and the output of all VRs. As the monitoring system can only transform measurements between VDs if VRs exist, not connected VDs can not be used in the monitoring system. However, if such unconnected VDs are part of the SD a system engineer can be warned that the sensors providing measurements to such VDs can not be monitored. Hence, the VDs of all sensors used in the CPS shall be in the set of used VDs $\mathcal{P}$. Provided VDs can also be unused and hence not part of $\mathcal{P}$. For an overall SD this will not be the case as it foremost describes the available resources of a system for parts of the system this can however be the case.

More specifically, the monitoring system uses such parts of a SD to transform measurements. Such a subset can be denoted as a sub-SD. We define the symbol $\subseteq$ to denote the subsystem

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

relation of two SDs i.e. given SDs $\mathsf{S}(\mathcal{P}_\mathsf{S}, \mathcal{R}_\mathsf{S})$ and $\mathsf{T}(\mathcal{P}_\mathsf{T}, \mathcal{R}_\mathsf{T})$ then $\mathsf{T} \subseteq \mathsf{S}$ iff $\mathcal{P}_\mathsf{T} \subseteq \mathcal{P}_\mathsf{S}$ and $\mathcal{R}_\mathsf{T} \subseteq \mathcal{R}_\mathsf{S}$. In Figure 3.3, a sub-SD of the example SD can be seen. A subset of the VDs is connected via a subset of the VRs. A sub-SD similar to the one depicted in Figure 3.3 is an useful representation of the flow of value transformations in the monitoring system as it is presented in the following sections.

3.2 Measurement Transformation

For the implicit redundancies of the system to be used, measurements from multiple sensors have to be compared. However, in contrast to the usage of explicit redundancies in which case multiple measurements are provided to a single VD and can be compared without transformation, in the case of implicit redundancies the measurements have to transformed first. More specifically, the values provided via the sensors of VDs have to be transformed using the transformation functions of VRs such that the resulting values can be compared in a single VD. This VD is called the root of the comparison and can be either one of the provided VDs or an unrelated third VD as shown in Figure 3.4 a) and b) respectively with two example SDs.

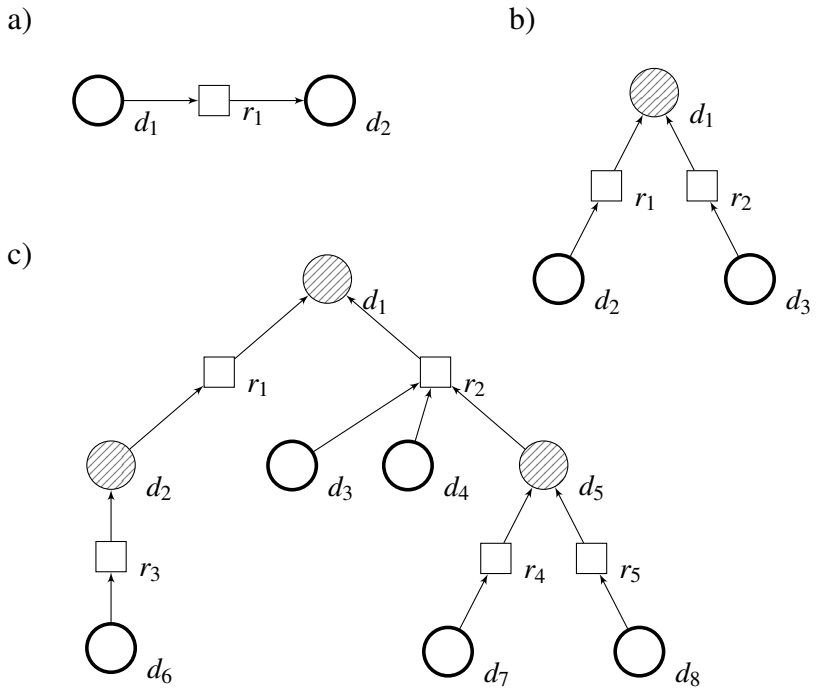

Figure 3.4: Transformations for voting

Figure 3.4 c) depicts a SD that can be used for voting as the measurements from the provided VDs d_3, d_4, d_6, d_7, d_8 marked with bold borders are transformed to the common root VD d_1 marked shaded such as all unprovided VDs. It is not always possible to directly transform measurements via a single VR, as no VR exists that can transform a measurement to VD where it can be

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

compared. Despite of this, it is possible to transform a measurement value multiple times using the transformation functions of several VRs i.e. given a SD $\mathsf{S}(\mathcal{P},\mathcal{R})$ with $\mathcal{D}$ representing the set of all VDs in S, the VDs $d_s, d_r \in \mathcal{D}$ and suppose there is no VR with input d_s and output d_r there could be multiple VRs and VDs that form a path on which the values can be transformed from VD d_s to the root VD d_r. Equation 3.1 gives a formal definition of a path p from VD d_0 to the destination VD d_n.

$$p_{d_0,d_n} = (d_0, r_1, d_1, \ldots, d_{n-1}, r_n, d_n) \left| \begin{array}{l} 0 \leq j \leq n,\ d_j \in \mathcal{D} \\ 1 \leq k \leq n,\ r_k \in \mathcal{R} \\ d_{k-1} \in i_{r_k} \\ d_k = o_{r_k} \end{array} \right. \tag{3.1}$$

In Figure 3.5 a), a SD comprised of such a path from a provided VD to a root VD is shown. These paths represent the most simple transformations where the transformation functions of all VRs on the path have only one input variable. The monitoring system should however be capable to handle more complex cases. The definition of the VR does support this via multiple input VDs which corresponds to a transformation function with multiple inputs. This allows to construct transformations that use the sensor measurements of multiple provided VDs to derive a measurement in the root VD. Such a multi-input measurement transformation naturally does not correspond to a path but rather to multiple paths with the same destination. The simplest instance of this can be a tree. As can be seen in Figure 3.5 b), the provided VDs correspond to the leafs of the tree where the root VD corresponds to the root of the tree. More specifically, the corresponding tree is an in-tree i.e. a directed tree where all edges are directed to the root vertex or in our case all provided VDs have a path to the root VD [42].

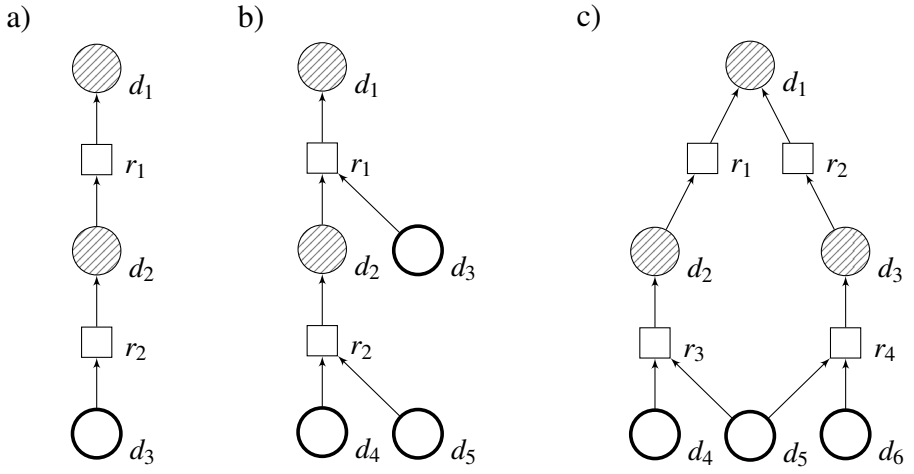

Figure 3.5: Possible transformation graphs

In some scenarios, however, even the tree structure is too restrictive, as it can be necessary that multiple provided VDs can only be compared if a third VD is combined in the transformation

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

to the common VR. Therefore, that VD has multiple paths to the root VD. The corresponding graph of an example SD can be seen in Figure 3.5 c). The corresponding graph representation is a rooted, connected, directed acyclic graph (DAG). Again, the leaf vertices correspond to the provided VDs and the root vertex to the root VD such as in the case of the tree. Further, every provided VD has at least one path to the root VD. Due to the fact that the path as well as the tree are directed acyclic graphs (DAGs) themselves, we will only consider DAGs when talking about the transformation of measurements.

Moreover, it can not be ignored that not every SD that can be represented as a rooted and connected DAG represents a valid measurement transformation. An example of this can be seen in Figure 3.4. The three depicted SDs are all rooted and connected DAGs. However, multiple measurements are transformed to the root VDs and not a single one as it is defined for Measurement Transformations (MTs). The SDs have to be divided into sub-SDs to describe every transformation individually. Figure 3.6 shows three SDs corresponding to the individual measurement transformations of the SD depicted in Figure 3.4 c). It has to be noted that the individual SDs share the root VD and in the case of SDs b) and c) also the VDs d_3, d_4 as they are shared by all transformations i.e. the SD is not partitioned literally but rather a set of sub-SDs is formed.

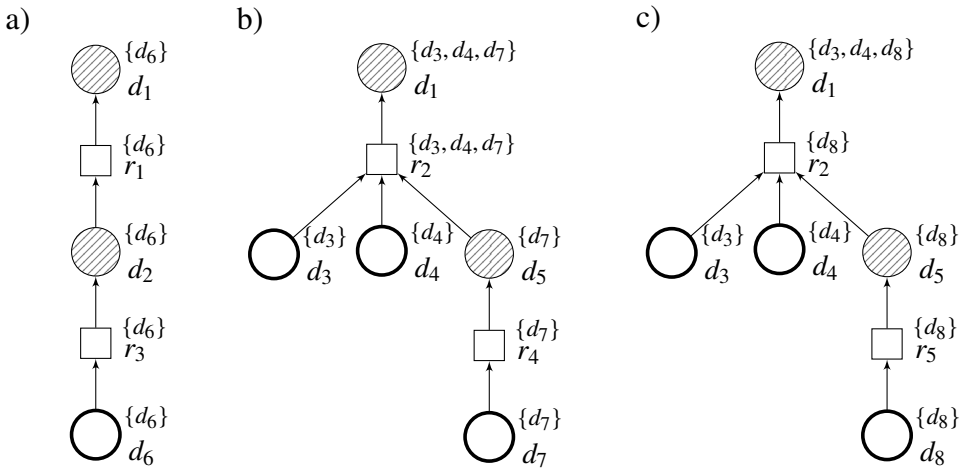

Figure 3.6: The measurement transformation graphs for the SD in Figure 3.4 c).

The annotations, in the form of sets of VDs, adjacent to the VDs and VRs in Figure 3.6 show which provided VDs influence the transformed measurements along the way to the root VD. Considering the transformation of multiple transformed and/or provided measurements via a VR, the resulting transformed measurement is clearly influenced by the combination of the influencing VDs of the individual input measurements. Formally, this corresponds to the union of the influence sets i.e. given two VDs with two measurements influenced by $\mathcal{M}_a = \{d_{a_1}, \ldots, d_{a_n}\}$ and $\mathcal{M}_b = \{d_{b_1}, \ldots, d_{b_n}\}$ respectively and a VR that has these two VDs as inputs, then the influence set that corresponds to the transformed measurement is $\mathcal{M}_r = \mathcal{M}_a \cup \mathcal{M}_b$ as can be seen in DAG b) of Figure 3.6 where the transformed measurement in VR r_2 is influenced by the VDs in the set $\{d_3, d_4, d_7\} = \{d_3\} \cup \{d_4\} \cup \{d_7\}$.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

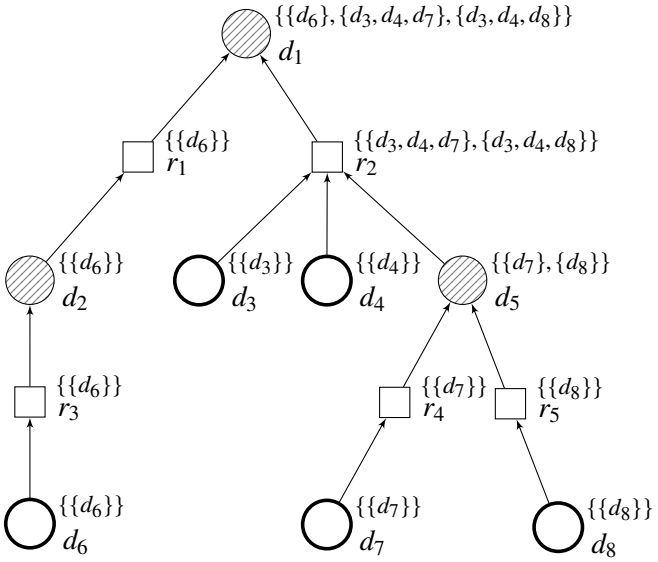

Figure 3.7: Voting group figure annotated with the measurement influences.

In the case of a SD not corresponding to a single transformation, multiple measurements can be provided and transformed to a single VD. Figure 3.7 shows the combination of the three DAGs and their individual transformed measurements. Correspondingly, the annotations show the influence sets of the measurements available in the VDs and transformed via the VRs. Formally, the sets can be determined recursively. For VDs two cases have to be distinguished depending on the number of inputs to a VD and whether a sensor provides measurements to the VD. The set of influencing VDs is determined by all the measurements transformed to the VD via VRs and the measurement provided by a sensor to the VD itself if the VD is provided, i.e. given a SD $\mathsf{S}(\mathcal{D},\mathcal{R})$ let $\mathcal{M} : (\mathcal{D} \times S \to \mathcal{P}(\mathcal{P}(\mathcal{D}))$ be the function defined in Equation 3.2.

$$\mathcal{M}(d,\mathsf{S}) = \begin{cases} \bigcup\limits_{r \in (\mathcal{I}_d \cap \mathcal{R}_\mathsf{S})} \mathcal{M}(r,\mathsf{S}) \cup \{\{d\}\} & \text{for } \overline{p}_d = \text{true} \\ \bigcup\limits_{r \in (\mathcal{I}_d \cap \mathcal{R}_\mathsf{S})} \mathcal{M}(r,\mathsf{S}) & \text{for } \overline{p}_d = \text{false} \end{cases} \tag{3.2}$$

The function $\mathcal{M}(r,\mathsf{S})$ determines all possible sets of influencing VDs for measurements transformed via VR r. It yields all combinations of influence sets of the measurements of the input VDs of the VR. In terms of the identifier sets, this corresponds to the unordered Cartesian product. More formally, given a SD $\mathsf{S}(\mathcal{D},\mathcal{R})$ let $\mathcal{M} : (\mathcal{R} \times S \to \mathcal{P}(\mathcal{P}(\mathcal{D})))$ be the function defined in Equation 3.3. The semantics of the unordered Cartesian product makes it necessary to take the union of elements of the resulting set, denoted by $\bigcup \mathcal{M} = \bigcup_{m \in \mathcal{M}} m$, to get a correct structure as a result. For example, in the case of r_2, the unordered Cartesian product of the measurements in the input VDs is $\{\{\{d_3\},\{d_4\},\{d_7\}\},\{\{d_3\},\{d_4\},\{d_8\}\}\}$ and the union of the set's innermost sets

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

gives $\{\{d_3, d_4, d_7\}, \{d_3, d_4, d_8\}\}$, the set of the transformed measurements [43].

$$\mathcal{M}(r, \mathsf{S}) = \left\{ \bigcup M \;\middle|\; M \in \prod_{d \in \mathcal{I}_n} \mathcal{M}(d, \mathsf{S}) \right\} \tag{3.3}$$

It can be seen that the given definitions are infinite in the case of a cycle in the SD. This is one further reason why the transformation has to be a DAG. Otherwise the monitoring system would not be able to determine the measurement transformation as it would not terminate. If a SD is a DAG we can further determine the influence of the contained measurements and therefore given a rooted acyclic SD $\mathsf{S}(\mathcal{P}, \mathcal{R})$ the following shorthand can be used to describe these sets.

$$\mathcal{M}(\mathsf{S}) = \mathcal{M}(d_{\text{root}}, \mathsf{S}) \tag{3.4}$$

Going on with the discussion, it is now possible using the presented concepts to give a precise definition of a MT. If a SD corresponds to a rooted DAG and only one measurement can be transformed to the root VD, then it is a MT. More formally given a SD $\mathsf{T}(\mathcal{P}, \mathcal{R})$ it is a MT iff the SD structure is a DAG, with root VD $d_{\text{root}} \in \mathcal{D}_\mathsf{T}$, and the set of influence sets has only one element, the set of provided VDs i.e. $\mathcal{M}(\mathsf{T}) = \{\mathcal{P}\}$. From this definition it follows that for all provided and used VDs the number of inputs is zero and that for all other VDs contained in a MT that the number of inputs is exactly 1. Otherwise two measurements would be provided to these VDs and therefore to the root which contradicts the definition. Hence for a MT $\mathsf{T}(\mathcal{P}_\mathsf{T}, \mathcal{R}_\mathsf{T})$ the properties in Equation 3.5 hold.

$$\begin{aligned} &\exists! d_{\text{root}} \in \mathcal{D}_\mathsf{T} && : |\mathcal{O}_{d_{\text{root}}}| = 0 \\ &\forall d \in \mathcal{P}_\mathsf{T} \setminus d_{\text{root}} && : |\mathcal{I}_d| = 0 \\ &\forall d \in \mathcal{D}_\mathsf{T} \setminus (\mathcal{P}_\mathsf{T} \setminus d_{\text{root}}) && : |\mathcal{I}_d| = 1 \end{aligned} \tag{3.5}$$

Of course every MT T has to be a sub-SD of an overall SD S, i.e. $\mathsf{T} \subseteq \mathsf{S}$, to be used in a monitoring system of the SD S. In a similar way, a MT can also be a sub-SD of another MT. There are however much more sub-SDs that are not MTs. To distinguish these sub-MTs from all other sub-SDs, we introduce the subset relation for MTs. Given two SDs $\mathsf{T}_1, \mathsf{T}_2$ this relation is defined in Equation 3.6.

$$\mathsf{T}_1 \sqsubseteq \mathsf{T}_2 \iff \begin{pmatrix} \mathsf{T}_1 \subseteq \mathsf{T}_2 \\ \mathsf{T}_1 \text{ and } \mathsf{T}_2 \text{ are MTs} \end{pmatrix} \tag{3.6}$$

Using the subset relation, a MT can be decomposed into its sub-MTs. This is shown in Figure 3.8 where the MT T_1 and all sub-MTs are depicted. This can be a recursive process such as in the case of MTs T_3 and T_4, which are both subsets of MT T_1 but T_4 is also subset of T_3. This decomposition is important for the calculation of the cost of the monitoring system presented in subsection 3.4.2. For this purpose, we denote the function $\mathcal{T}$ defined in Equation 3.7 as the decomposition function.

$$\mathcal{T}(\mathsf{T}) = \{\mathsf{U} \sqsubseteq \mathsf{T}\} \tag{3.7}$$

Applied to MT T_1 the function returns the set $\mathcal{T}(\mathsf{T}_1) = \{\mathsf{T}_1, \mathsf{T}_2, \mathsf{T}_3, \mathsf{T}_4\}$. It has to be noted that T_1 is contained in the set as it is part of the decomposition. This holds as well for the other MTs in

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

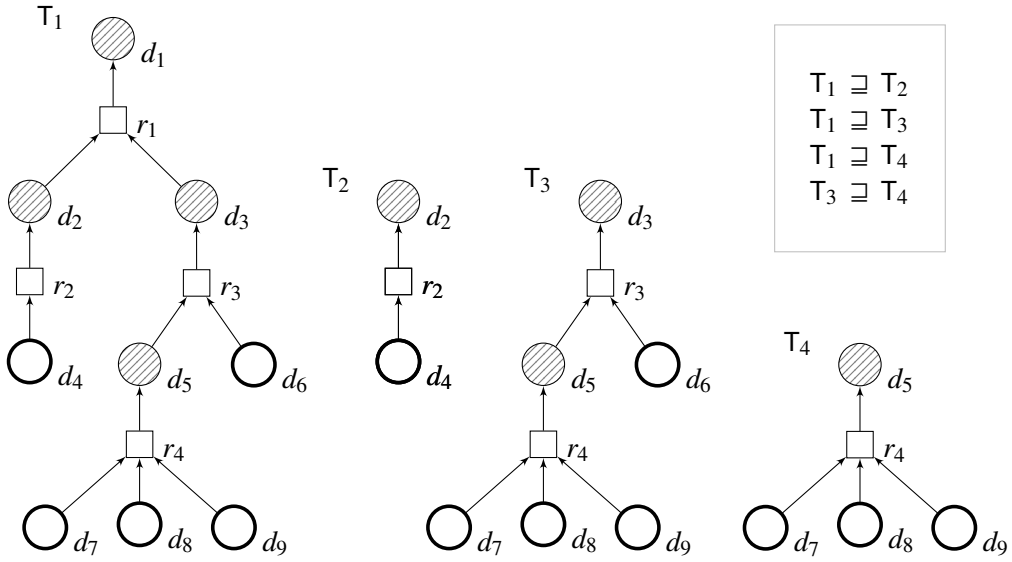

Figure 3.8: Decomposition of a MT T_1 into the sub-MT $\mathsf{T}_2, \mathsf{T}_3, \mathsf{T}_4$,.

Figure 3.8. For MT T_4 as well as for MT T_2 the result is the unit set as they can not be divided further and hence represent the smallest building blocks of the monitoring system. These building blocks can be used to create other MTs and several MTs can be used to vote on the correctness of measurements and thereby on the failure state of sensors as it is presented in the following section.

3.3 Voting Group

As already mentioned, the monitoring system has to determine if any sensor is providing faulty measurements and if so which one does so. The first part of this task can be achieved by transforming a group of provided measurements using two MT to a common root VD and comparing the transformed measurements. We call such a group a Voting Group (VG). More formally, given a SD $\mathsf{S}(\mathcal{P}, \mathcal{R})$ we define $\mathsf{V}(\mathsf{T}_1, \mathsf{T}_2)$ to be a VG according to Equation 3.8.

$$\mathsf{V}(\mathsf{T}_1, \mathsf{T}_2) \text{ is a VG in SD } \mathsf{S} \iff \begin{pmatrix} \mathsf{T}_1, \mathsf{T}_2 \subseteq \mathsf{S} \\ \mathsf{T}_1 \text{ and } \mathsf{T}_2 \text{ are MTs} \\ d_{root,\mathsf{T}_1} = d_{root,\mathsf{T}_2} \\ \mathcal{M}(\mathsf{T}_1) \neq \mathcal{M}(\mathsf{T}_2) \end{pmatrix} \tag{3.8}$$

From this definition it follows that a VG is itself again a SD and a sub-SD of SD S. Furthermore its graph representation is again a connected and rooted DAG. The common root of the MTs is the root of the VG, the union of the provided VDs of the MTs are the provided VDs of the VG and the union of the VR of the MTs are its VRs. In contrast to the MT, the provided VDs do not have to be leaf vertices. The two MTs could overlap and a provided but unused VD in one can be a used one in the other. Figure 3.4 a) is an example of this. SD b) is a case where the provided VDs do

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

not overlap. SD c) on the other hand is not a VG as it consists of more than two MTs as depicted in Figure 3.6.

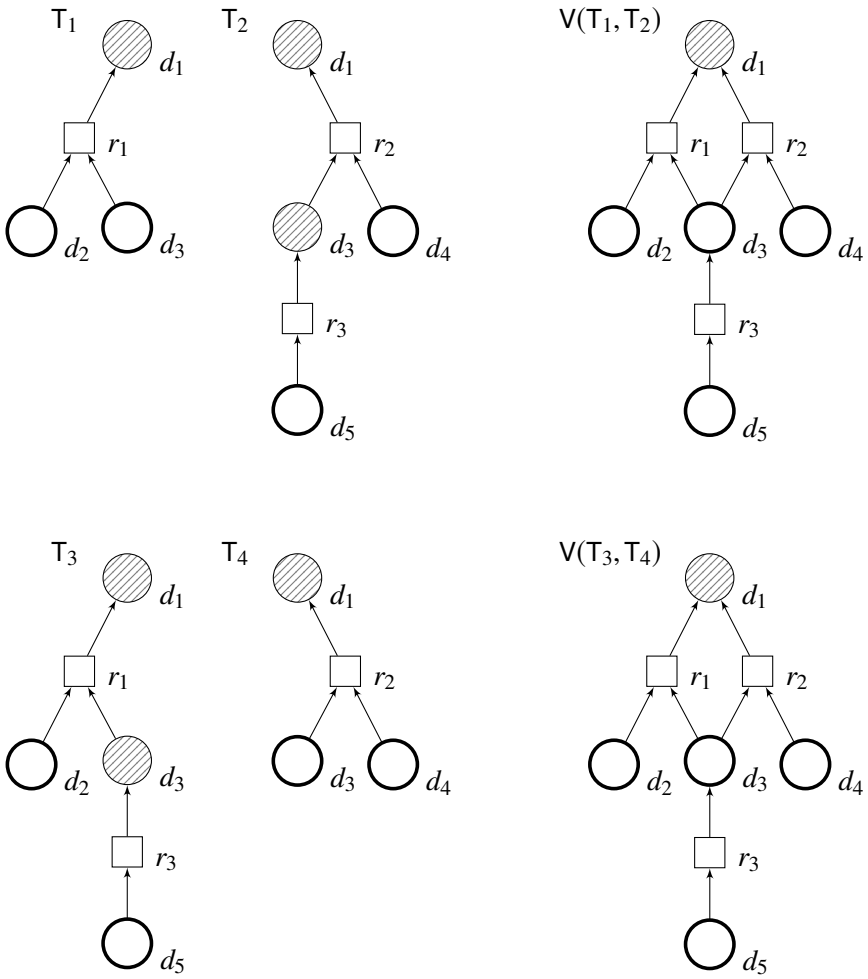

Figure 3.9: Voting group vs measurement transformation

Despite the similarities with MTs, a VG does not describe transformations in an unambiguous way. This can be seen in Figure 3.9. The VGs $V(T_1, T_2)$ and $V(T_3, T_4)$ are identical in their representation as a SD, however the underlying MTs are different. Further, the number of measurements that can be transformed to the root is more than the two expected from the MTs i.e. given a VG $V(T_1, T_2)$ it holds that $\mathcal{M}(V) \neq \mathcal{M}(T_1) \cup \mathcal{M}(T_2)$. Therefore, the information contained in the MTs has to be preserved. This is the reason for the definition of the VG as combination of two MTs and not as a SD itself.

After these transformations, the next step for the monitoring system is to compare measurements. For this, a comparison function is needed. For the purpose of reducing complexity and to provide a more general framework instead of reasoning on a specific implementation of such a function,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

it will be shown that it is enough to make a few general assumptions on the properties of such a function to show the correctness of this approach. We call such a function an approximate-comparison function $\overline{\mathbf{AP}} : \mathcal{M}^* \times \mathcal{M}^* \rightarrow \{\text{true}, \text{false}\}$ where $\mathcal{M}^*$ denotes the domain of all possible measurement values. The function is defined by the following assumptions:

1. Given two measurements, if none of them is influenced by a faulty sensor the function will return true.
2. Given two measurements, if only one is influenced by a faulty sensor the measurements are eventually detected as not approximate i.e. the function returns false after some random but finite time δ_t.

Assumption 1 is trivial as two correct working sensors will give back the same information by the definition of the word correct. In the case of one valid and one faulty measurement, a discrepancy has to be measurable for the fault to be a fault. However, the time between the beginning of the fault and the detection can vary depending on the delay introduced by the transformation function on the path to the root VD. Therefore, it may take a different amount of time in the individual MT for the error to propagate and ultimately for a difference to be detected. Assumption 2 takes this into account. In the case of both measurements influenced by a faulty sensor, the result can be either true or false because the fault could be transformed to the root VD in the same way and no difference could be found. However, it could as well be the other way around that the transformation of the faulty sensor value influences the measurements in different ways and the comparison would show a difference. Therefore, in this case no assumption on the output can be made. Table 3.1 shows the possible combinations and the result. The X in the last row represents the uncertainty if a sensor is influencing both sides of a comparison is providing faulty measurements.

Table 3.1: Eventual results of the functions $\overline{\mathbf{AP}}$ depending on whether a measurement is valid (V) or influenced by a faulty sensor (I).

m_{T_1}	m_{T_2}	$\overline{\mathbf{AP}}(m_{T_1}, m_{T_2})$
V	V	T
I	V	F
V	I	F
I	I	X

From the perspective of the monitoring system, the value X does not exist, as the output of the comparison function is either equal true or false i.e. the measurements are approximate or not. Therefore, several cases have to be distinguished.

1. The output is true and all sensors work properly
2. The output is true and a sensor influencing both measurements is failing.
3. The output is false then one of the sensors influencing one or both measurements is failing.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

These cases can now be used to determine the set of potentially faulty sensors from the return value of the approximate-comparison function. In Table 3.2, these potential faulty sensors are shown. It can be seen that the detection of faults in the sensors that influence both measurements represented by the set $\mathcal{P}_{\mathsf{T}_1} \cap \mathcal{P}_{\mathsf{T}_2}$ can not always be detected.

Table 3.2: Potential faulty VDs depending on output of $\overline{\mathbf{AP}}(\mathsf{T}_1, \mathsf{T}_2)$

$\overline{\mathbf{AP}}(m_{\mathsf{T}_1}, m_{\mathsf{T}_2})$	VDs
T	$\mathcal{P}_{\mathsf{T}_1} \cap \mathcal{P}_{\mathsf{T}_2}$
F	$\mathcal{P}_{\mathsf{T}_1} \cup \mathcal{P}_{\mathsf{T}_2}$

This observation makes it necessary to distinguish between the VDs whose faults can be concluded from a detected discrepancy but do not cause one reliably and the VDs where the VG stands witness for their fault i.e. where every fault leads to a detection after some finite amount of time. We call the first the blamed VDs and denote the function to evaluate them as $\boldsymbol{\mathcal{D}}_{\mathrm{B}}$ defined in Equation 3.9 and the second as witnessed VDs and denote the function to evaluate them as $\boldsymbol{\mathcal{D}}_{\mathrm{W}}$ defined in Equation 3.10. $\oplus$ stands for the symmetric difference of the sets that comprises all elements that are in one of the sets but not in both or more formally $\mathcal{A} \oplus \mathcal{B} = (\mathcal{A} \cup \mathcal{B}) \setminus (\mathcal{A} \cap \mathcal{B})$.

$$\boldsymbol{\mathcal{D}}_{\mathrm{B}}(\mathsf{V}) = \mathcal{P}_{\mathsf{T}_1} \cup \mathcal{P}_{\mathsf{T}_2} \tag{3.9}$$

$$\boldsymbol{\mathcal{D}}_{\mathrm{W}}(\mathsf{V}) = \mathcal{P}_{\mathsf{T}_1} \oplus \mathcal{P}_{\mathsf{T}_2} \tag{3.10}$$

As defined in the beginning of this section, the VG is used for the description of the comparison of transformed measurements. Therefore, we introduce a shorthand for the approximate-comparison function as it is always applied to the measurements transformed by the two MTs of a VG. Hence given a VG $\mathsf{V}(\mathsf{T}_1, \mathsf{T}_2)$ and the transformed measurements m_1, m_2 corresponding to the MTs $\mathsf{T}_1, \mathsf{T}_2$ respectively, we define $\overline{\mathbf{AP}}(\mathsf{V}) = \overline{\mathbf{AP}}(m_1, m_2)$. This makes the following formulas more clear and the reasoning easier to follow.

In the monitoring system, faults can be detected by checking the result of $\overline{\mathbf{AP}}(\mathsf{V})$. If a discrepancy is found using this approach, it is possible to restrict the set of possibly-faulty VDs using Equation 3.9. However, to determine which VD is causing the discrepancy multiple VGs have to be used to narrow down the set until only a single VD is left. The following section will describe how VGs have to be selected from an overall SD to build a robust monitoring system.

3.4 Grouping Solution

As mentioned in the earlier section, the monitoring system has to determine whether all sensors are working correctly and if not, which sensor introduces faulty measurements into the system. Therefore, several VGs are used to compare all sensors. We call this set of VGs a Grouping Solution (GS). More formally given a SD $\mathsf{S}((\mathcal{P}, \mathcal{R}))$ Equation 3.11 defines the GS X as a subset of the set of all possible VGs in S.

$$\mathsf{X} \subseteq \{\mathsf{V}(\mathsf{T}_1, \mathsf{T}_2) \mid \mathsf{V} \text{ is a VG in } \mathsf{S}\}. \tag{3.11}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Similar to the VG, the GS itself forms a sub-SD of an overall SD. An illustration of this can be seen in Figure 3.10. For the SD $S(\mathcal{P},\mathcal{R})$, $X(V_1,V_2,V_3)$ is a possible GS that itself is constructed from the VGs $V_1(T_1,T_2), V_2(T_1,T_3), V_3(T_2,T_3)$ which use the MTs T_1,T_2,T_3. It can be seen that the MTs are shared between the VGs. How this can be used to improve the overall efficiency of the monitoring system is discussed in subsection 3.4.2.

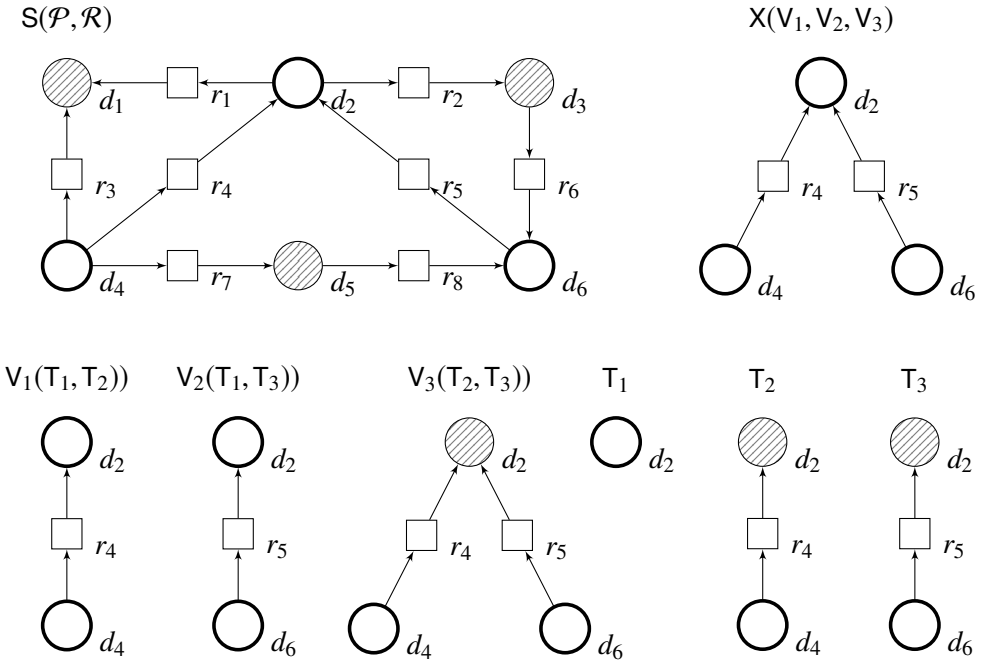

Figure 3.10: Example SD with a possible GS and its VG.

How the VGs are combined in the GS. As discussed in the previous sections, a faulty sensor will cause the detection of discrepancies between the compared measurements in the VGs its corresponding VD is providing to. The sets of the blamed VDs can be derived with the function $\mathcal{D}_B(V)$ for every VD V where the approximate-comparison function detected a discrepancy. The resulting sets each represent possibly-faulty sensors. However, not every set does contain the same VDs and therefore further VDs can be excluded from the search. Only the VDs that are part of every VGs that was triggered are part of set of remaining VDs. More formally this operation does correspond to the set intersection of all sets of the blamed VDs. In Table 3.3 the results of the approximate-comparison function for every voting group from Figure 3.10 as well as the corresponding sets of blamed VDs and their intersection is shown for a hypothetical fault in the sensors. It can be seen that the resulting set does indeed only contain the VD of the faulty sensor hence the localization is working.

The example GS in Figure 3.10 is capable to detect a single fault under any circumstance. This simple example however hides several issues with more complex systems. The following subsection will discuss the issues and how they can be mitigated. The final subsection will discuss

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Table 3.3: Result of the approximate-comparison function in case of a fault in one of the provided VDs from autoref:fig:grouping-solution

Faulty VD	$\overline{\mathbf{AP}}(\mathsf{V}_1)$	$\overline{\mathbf{AP}}(\mathsf{V}_2)$	$\overline{\mathbf{AP}}(\mathsf{V}_3)$	$\{\mathcal{D}_\mathrm{B}(\mathsf{V}) \mid \neg\overline{\mathbf{AP}}(\mathsf{V})\}$	$\bigcap$
d_2	F	F	T	$\{\{d_2, d_4\}, \{d_2, d_6\}\}$	$\{d_2\}$
d_4	F	T	F	$\{\{d_2, d_4\}, \{d_4, d_6\}\}$	$\{d_4\}$
d_6	T	F	F	$\{\{d_2, d_6\}, \{d_4, d_6\}\}$	$\{d_6\}$

how the selection of MTs does influence the performance cost and thereby the efficiency of the system.

3.4.1 Issues and Countermeasures

To comprehend the issues at hand we have to change our perspective from the hypothetical final detection state to the sequence of events in the case of a fault in one sensor. Table 3.3 does hide the fact that the approximate-comparison function does detect a discrepancy only after some delay as defined in assumption 2. This delay is caused by several factors. Firstly the transformation functions introduce a natural delay due to the time it takes to compute the outputs. The values of these outputs are further influenced by the functions. If for example the function does integration on a faulty input then the fault will propagate only after the influence over the integrated values is big enough i.e. the propagation does incur a delay. Lastly the detection depends on the comparison of the measurements in the root VD of the VG using the approximate-comparison function, as different VGs run the comparison in different VDs where the transformed fault may take different shapes. The monitoring system hence will detect discrepancies one after another due to these circumstances i.e. the approximate-comparison functions run on the VGs affected by a faulty sensor will evaluate to false one after another. The set of voting groups in which a fault was detected will therefore grow with the progression of time. As mentioned this set can be derived by running the approximate-comparison function repeatedly in every VG and grouping the VGs where a discrepancy was detected. Formally the function of faulty VGs $\mathcal{F}_\mathsf{V} : \mathsf{GS}^* \to \mathsf{V}^*$ defined in Equation 3.12 describes this process.

$$\mathcal{F}_\mathrm{V}(\mathsf{X}) = \{\mathsf{V} \in \mathsf{X} \mid \overline{\mathbf{AP}}(\mathsf{V}) = \text{false}\} \tag{3.12}$$

Further, this set of faulty VGs can be used to determine the possibly faulty sensors by intersecting the sets of VDs blamed by the VGs. Formally we denote $\mathcal{F}_\mathrm{D} : \mathsf{GS}^* \to \mathcal{D}^*$ as the function to derive these blamed VDs as defined in Equation 3.13. From a chronological perspective this function returns the empty set until the first discrepancy is detected. At that time the blamed VDs of the detecting VG represent the possible candidates. Following this, VG after VG detects a discrepancy as well. The set of possibly faulty sensors is narrowed down and only a single VD remains, which corresponds to the senor at fault. However, for this to be true the GS has to be of a certain structure such that a fault can always be detected and a single source can be determined.

$$\mathcal{F}_\mathrm{D}(\mathsf{X}) = \bigcap_{\mathsf{V} \in \mathcal{F}_\mathrm{V}(\mathsf{X})} \mathcal{D}_\mathrm{B}(\mathsf{V}) \tag{3.13}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

To investigate this we have to change our perspective back to the hypothetical final detection state. We further introduce an example GS described in Table 3.4. Each row describes a VG via its two MTs and their respective influencing VDs shown in the second and third column. Note that the structure of the underlying SD is not important. The sets of influencing VDs are enough for the considerations of the correctness of the GS. The table further depicts the VDs that are blamed in case of a detected discrepancy in column four and in column five the VDs the VG stands witness for as discrepancies can be determined in all cases after some finite delay.

Table 3.4: Example VGs and their blame/witness sets.

VG	$\mathcal{M}(\mathsf{T}_{1,\mathsf{V}})$	$\mathcal{M}(\mathsf{T}_{2,\mathsf{V}})$	$\mathcal{D}_{\mathrm{B}}(\mathsf{V})$	$\mathcal{D}_{\mathrm{W}}(\mathsf{V})$
V_1	$\{d_1, d_4, d_5\}$	$\{d_2, d_5\}$	$\{d_1, d_2, d_4, d_5\}$	$\{d_1, d_2, d_4\}$
V_2	$\{d_1\}$	$\{d_3\}$	$\{d_1, d_3\}$	$\{d_1, d_3\}$
V_3	$\{d_2, d_4\}$	$\{d_3, d_4\}$	$\{d_2, d_3, d_4\}$	$\{d_2, d_3\}$
V_4	$\{d_4\}$	$\{d_5\}$	$\{d_4, d_5\}$	$\{d_4, d_5\}$
V_5	$\{d_2\}$	$\{d_5\}$	$\{d_2, d_5\}$	$\{d_2, d_5\}$
V_6	$\{d_1\}$	$\{d_4\}$	$\{d_1, d_4\}$	$\{d_1, d_4\}$

To investigate the hypothetical faults further, a function analogous to Equation 3.12 is needed but instead of determining the VGs that are detecting a discrepancy, it should provide all VGs that can show discrepancies when a hypothetical fault is present in a VD. Further, a function to determine the VGs that will in all cases detect a discrepancy. Similarly to the definitions in Equation 3.9 and Equation 3.10 from section 3.3, we formally the denote the functions $\boldsymbol{\mathcal{V}}_{\mathrm{B}} : \mathcal{D} \times \mathsf{GS}^* \to \mathsf{V}^*$ defined by Equation 3.14 and $\boldsymbol{\mathcal{V}}_{\mathrm{W}} : \mathcal{D} \times \mathsf{GS}^* \to \mathsf{V}^*$ defined by Equation 3.15 as the blaming VGs function and the witness VGs function respectively.

$$\boldsymbol{\mathcal{V}}_{\mathrm{B}}(d, \mathsf{X}) = \{\mathsf{V} \in \mathsf{X} \mid d \in \mathcal{D}_{\mathrm{B}}(\mathsf{V})\} \tag{3.14}$$

$$\boldsymbol{\mathcal{V}}_{\mathrm{W}}(d, \mathsf{X}) = \{\mathsf{V} \in \mathsf{X} \mid d \in \mathcal{D}_{\mathrm{W}}(\mathsf{V})\} \tag{3.15}$$

This functions can now be applied to the example GS as can be seen in Table 3.5 and Table 3.6 for function $\boldsymbol{\mathcal{V}}_{\mathrm{B}}$ and $\boldsymbol{\mathcal{V}}_{\mathrm{W}}$ respectively. However, rather then only showing the result of evaluating the whole GS, partial results for subsets of the GS are shown row by row starting with the set of VGs $\{\mathsf{V}_1, \mathsf{V}_2\}$ and adding a VG in every line. This representation shows three major issues in the selection of VGs for a GS highlighted using the three colors. These issues are described in the following enumeration and the necessary countermeasures are proposed.

1. The VG sets marked in red show that for the corresponding VD a detection is either impossible in case of no VG or that a detected discrepancy can not be attributed to the VD in case of a single VG influenced by the VD as any of the influencing VDs could be the cause. Due to this problem it is clearly necessary for a correct GS to include enough and the right VGs, such that each VD influences at least two VGs. Additionally, the influenced VGs have to be part of the witnessing VGs. Otherwise the only blaming VGs could miss a fault and make the detection impossible. Summing up, a GS X has to adhere to Equation 3.16 where $\mathcal{P}_{\mathsf{X}}$ denotes the set of provided and used VDs.

$$\forall a \in \mathcal{P}_{\mathsf{X}} \; : \; |\boldsymbol{\mathcal{V}}_{\mathrm{W}}(a, \mathsf{X})| \; > 1 \tag{3.16}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

2. The VG sets marked in blue highlight sets that overlap with the sets of the other VDs. For example, the sets for VDs d_2 and d_4 are identical i.e. fully overlapping if blaming is considered for VGs V_1, V_2, V_3. Although the detection of a fault is possible the attribution is not as both VDs are part of all VGs. The addition of VG V_4 depicted in the next row has the same issue as the VDs d_2, d_4 and d_5 can not be attributed in that case. To counter this issue, it is necessary that the sets of blaming VGs from the set $\mathcal{P}_X$ for all pairs of provided and used VDs are not subsets of each other. More formally a GS X has to adhere to Equation 3.17.

$$\forall a, b \in \mathcal{P}_X \ : \ \mathcal{V}_B(a, X) \not\subseteq \mathcal{V}_B(b, X) \tag{3.17}$$

3. The VG sets marked in green show that overlapping can also occur between the blaming VGs and the witnessing VGs of different VDs. In the example the witnessing VGs of VD d_4 is a subset of the blaming VGs of VD d_5 if VGs $V_1, \ldots, V_5$ are considered and this can prevent the attribution of the fault in certain cases. For example, lets assume that VD d_4 provides faulty measurements and the fault is not detected in VG V_3 as both sides of the comparison are influenced the same way by VD d_4 and the remaining VGs V_1 and V_4 are detecting a discrepancy. The application of Equation 3.13 to attribute the fault will return the set $\{d_4, d_5\}$ as both VDs are part of both VGs. To counter this problem it is necessary that for all pairs of provided and used VDs from the set $\mathcal{P}_X$ it holds that the set of witnessing VGs of one VD is not a subset of the set of blaming VGs of the other VD i.e. a GS X has to adhere to Equation 3.18.

$$\forall a, b \in \mathcal{P}_X \ : \ \mathcal{V}_W(a, X) \not\subseteq \mathcal{V}_B(b, X) \tag{3.18}$$

Table 3.5: Sets of blaming VGs for each VD of the example for the different sub-GS.

$\mathcal{V}_B(d, X)$	d_1	d_2	d_3	d_4	d_5
$X = \{V_1, V_2\}$	$\{V_1, V_2\}$	$\{V_1\}$	$\{V_2\}$	$\{V_1\}$	$\{V_1\}$
$X = \{V_1, V_2, V_3\}$	$\{V_1, V_2\}$	$\{V_1, V_3\}$	$\{V_2, V_3\}$	$\{V_1, V_3\}$	$\{V_1\}$
$X = \{V_1, \ldots, V_4\}$	$\{V_1, V_2\}$	$\{V_1, V_3\}$	$\{V_2, V_3\}$	$\{V_1, V_3, V_4\}$	$\{V_1, V_4\}$
$X = \{V_1, \ldots, V_5\}$	$\{V_1, V_2\}$	$\{V_1, V_3, V_5\}$	$\{V_2, V_3\}$	$\{V_1, V_3, V_4\}$	$\{V_1, V_4, V_5\}$
$X = \{V_1, \ldots, V_6\}$	$\{V_1, V_2, V_6\}$	$\{V_1, V_3, V_5\}$	$\{V_2, V_3\}$	$\{V_1, V_3, V_4, V_6\}$	$\{V_1, V_4, V_5\}$

Table 3.6: Sets of witnessing VGs for each VD of the example for the different sub-GS.

$\mathcal{V}_W(d, X)$	d_1	d_2	d_3	d_4	d_5
$X = \{V_1, V_2\}$	$\{V_1, V_2\}$	$\{V_1\}$	$\{V_2\}$	$\{V_1\}$	$\{\}$
$X = \{V_1, V_2, V_3\}$	$\{V_1, V_2\}$	$\{V_1, V_3\}$	$\{V_2, V_3\}$	$\{V_1\}$	$\{\}$
$X = \{V_1, \ldots, V_4\}$	$\{V_1, V_2\}$	$\{V_1, V_3\}$	$\{V_2, V_3\}$	$\{V_1, V_4\}$	$\{V_4\}$
$X = \{V_1, \ldots, V_5\}$	$\{V_1, V_2\}$	$\{V_1, V_3, V_5\}$	$\{V_2, V_3\}$	$\{V_1, V_4\}$	$\{V_4, V_5\}$
$X = \{V_1, \ldots, V_6\}$	$\{V_1, V_2, V_6\}$	$\{V_1, V_3, V_5\}$	$\{V_2, V_3\}$	$\{V_1, V_4, V_6\}$	$\{V_4, V_5\}$

Before going on with the discussion it has to be noted that the rules can be simplified and combined. From the definition of the functions $\mathcal{D}_B$ and $\mathcal{D}_W$ in Equation 3.9 and Equation 3.10 it

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

follows that $\forall \mathsf{V} \in \mathsf{X} \ : \ \mathcal{D}_\mathrm{W}(\mathsf{V}) \subseteq \mathcal{D}_\mathrm{B}(\mathsf{V})$. Hence by definition of functions $\mathcal{V}_\mathrm{B}$ and $\mathcal{V}_\mathrm{W}$ it follows that $\forall d \in \mathcal{P}_\mathsf{X} \ : \ \mathcal{V}_\mathrm{W}(d,\mathsf{X}) \subseteq \mathcal{V}_\mathrm{B}(d,\mathsf{X})$ because for every VG V in $\mathcal{V}_\mathrm{W}(d,\mathsf{X})$ the VD d has to be in the set $\mathcal{D}_\mathrm{W}(\mathsf{V})$ and therefore due to the subset relation also in the set $\mathcal{D}_\mathrm{B}(\mathsf{V})$ which implies that VG V is in the set $\mathcal{V}_\mathrm{B}(d,\mathsf{X})$. Using this property we can show that Equation 3.18 does imply Equation 3.17. Given VDs $a, b \in \mathcal{P}_\mathsf{X}$ where it holds that $\mathcal{V}_\mathrm{W}(a,\mathsf{X}) \nsubseteq \mathcal{V}_\mathrm{B}(b,\mathsf{X})$ then there exist at least one VG in $\mathcal{V}_\mathrm{W}(a,\mathsf{X})$ that is not in $\mathcal{V}_\mathrm{B}(b,\mathsf{X})$ and hence it has to be an element of $\mathcal{V}_\mathrm{B}(a,\mathsf{X})$ due to the subset-relation shown before. Therefore, the rule defined in Equation 3.17 is not needed.

Going further Equation 3.16 can be simplified as well. Due to the definition of the VG, at least two VDs are influencing a VG such that a comparison can be done. Therefore, if a VD is in exactly one VG its set of witnessing VGs is a subset of the set of blaming VGs of the other VD and hence the rule stated in Equation 3.16 enforces that at least a second VG influenced by the VD is added to a GS. Summing this up it is therefore enough that every VD is part of at least one VG or stated otherwise that the set of witnessing VGs is not empty. Formally Equation 3.19 and Equation 3.20 define the rules for a correct GS.

$$\forall a \in \mathcal{P}_\mathsf{X} \ : \ \mathcal{V}_\mathrm{W}(a,\mathsf{X}) \neq \emptyset \tag{3.19}$$

$$\forall a, b \in \mathcal{P}_\mathsf{X} \ : \ \mathcal{V}_\mathrm{W}(a,\mathsf{X}) \nsubseteq \mathcal{V}_\mathrm{B}(b,\mathsf{X}) \tag{3.20}$$

It remains to show formally that the rules do enforce a correct GS. Therefore, it has to be shown that in the case of a faulty sensor a discrepancy is detected and fault is attributed to the right sensor i.e. assuming that VD d does provide faulty measurements to the system then it holds after the faults have been propagated that

$$\mathcal{F}_\mathrm{D}(\mathsf{X}) = \{d\}.$$

From the definition of the approximate-comparison function it follows that after the propagation of the fault all witnessing VGs detect a fault when it is propagated to the root VD but that the blaming-only VGs may not. Hence it follows in the case of $\mathcal{F}_\mathrm{V}(\mathsf{X}) = \mathcal{V}_\mathrm{W}(d,\mathsf{X})$ that

$$\mathcal{F}_\mathrm{D}(\mathsf{X}) = \bigcap_{\mathsf{V} \in \mathcal{F}_\mathrm{V}(\mathsf{X})} \mathcal{D}_\mathrm{B}(\mathsf{V}) = \bigcap_{\mathsf{V} \in \mathcal{V}_\mathrm{W}(d,\mathsf{X})} \mathcal{D}_\mathrm{B}(\mathsf{V}) = \{d\}.$$

Note that in the case of $\mathcal{V}_\mathrm{W}(d,\mathsf{X}) \subseteq \mathcal{F}_\mathrm{V}(\mathsf{X})$, i.e. some of the blaming-only VGs detect a discrepancy, the result will not change as the intersection of $\{d\}$ with any set containing d will always result in $\{d\}$ again. Therefore, we can deduce that if an Equation 3.21 does hold for a GS, then the attribution will always yield a single result. In Table 3.7 the application of Equation 3.21 on the example can be seen with the valid sets colored in green and the invalid in red. When compared to the tables 3.5 and 3.6 it can be seen that the same sets are identified as being invalid. Therefore, we propose in lemma 1 that the different rules are equivalent.

$$\forall d \in \mathcal{P}_\mathsf{X} \ : \ \bigcap_{\mathsf{V} \in \mathcal{V}_\mathrm{W}(d,\mathsf{X})} \mathcal{D}_\mathrm{B}(\mathsf{V}) = \{d\} \tag{3.21}$$

Lemma 1. *Given a GS with the properties that every provided and used VD is part of at least one VG and that the set of witnessing VGs of every provided and used VD is not a subset of the set*

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

of blaming VGs of another provided and used VD, then and only then the set of intersections of all blaming sets of any provided and used VD is the unit set containing only the VD itself or more formally given a GS X *then the following equation holds.*

$$\left(\begin{array}{ll} \forall a \in \mathcal{P}_{\mathsf{X}} & : \mathcal{V}_W(a,\mathsf{X}) \neq \emptyset \\ \forall a,b \in \mathcal{P}_{\mathsf{X}} & : \mathcal{V}_W(a,\mathsf{X}) \not\subseteq \mathcal{V}_B(b,\mathsf{X}) \end{array}\right) \iff \forall d \in \mathcal{P}_{\mathsf{X}} \; : \bigcap_{\mathsf{V} \in \mathcal{V}_W(d,\mathsf{X})} \mathcal{D}_B(\mathsf{V}) = \{d\}$$

Proof: "$\Longrightarrow$". The proof is done by contradiction. We assume $\forall a \in \mathcal{P}_{\mathsf{X}} \; : \; \mathcal{V}_{\mathrm{W}}(a,\mathsf{X}) \neq \emptyset$ and $\forall a,b \in \mathcal{P}_{\mathsf{X}} \; : \; \mathcal{V}_{\mathrm{W}}(a,\mathsf{X}) \not\subseteq \mathcal{V}_{\mathrm{B}}(b,\mathsf{X})$ to be true. Further, we assume

$$\exists d \in \mathcal{P}_{\mathsf{X}} \; : \bigcap_{\mathsf{V} \in \mathcal{V}_{\mathrm{W}}(d,\mathsf{X})} \mathcal{D}_{\mathrm{B}}(\mathsf{V}) \neq \{d\}$$

for the purpose of establishing a contradiction. It follows that either

1. d is not an element of the set, i.e. more formally it holds that $d \notin \bigcap_{\mathsf{V} \in \mathcal{V}_{\mathrm{W}}(d,\mathsf{X})} \mathcal{D}_{\mathrm{B}}(\mathsf{V})$, or that
2. there exists a VD e that is part of the set, i.e. more formally it holds that $\exists e \neq d \in \mathcal{P}_{\mathsf{X}} \; : \; e \in \bigcap_{\mathsf{V} \in \mathcal{V}_{\mathrm{W}}(d,\mathsf{X})} \mathcal{D}_{\mathrm{B}}(\mathsf{V})$.

We will bring each case to a contradiction hence also the case of both is covered.

Case 1: We assume that d is not in the set $\bigcap_{\mathsf{V} \in \mathcal{V}_{\mathrm{W}}(d,\mathsf{X})} \mathcal{D}_{\mathrm{B}}(\mathsf{V})$. Further, we assumed $\forall a \in \mathcal{P}_{\mathsf{X}} \; : \; \mathcal{V}_{\mathrm{W}}(a,\mathsf{X}) \neq \emptyset$ hence the set is the result of an intersection. This implies that d is not an element of the blaming set of one VG in $\mathcal{V}_{\mathrm{W}}(d,\mathsf{X})$ and therefore not in every set of the intersection i.e. $\exists \mathsf{V} \in \mathcal{V}_{\mathrm{W}}(d,\mathsf{X}) \; : \; d \notin \mathcal{D}_{\mathrm{B}}(\mathsf{V})$. This is a contradiction due to the definition of the function $\mathcal{V}_{\mathrm{W}}$ and the subset relation $\mathcal{D}_{\mathrm{W}}(\mathsf{V}) \subseteq \mathcal{D}_{\mathrm{B}}(\mathsf{V})$ and therefore d has to be in the set.

Case 2: We assume that there exists a VD e that is an element of $\bigcap_{\mathsf{V} \in \mathcal{V}_{\mathrm{W}}(d,\mathsf{X})} \mathcal{D}_{\mathrm{B}}(\mathsf{V})$. Due to the definition of the set intersection this implies that e is an element of all blaming sets of d i.e. $\forall \mathsf{V} \in \mathcal{V}_{\mathrm{W}}(d,\mathsf{X}) \; : \; e \in \mathcal{D}_{\mathrm{B}}(\mathsf{V})$ and further that $\forall \mathsf{V} \in \mathcal{V}_{\mathrm{W}}(d,\mathsf{X}) \; : \; \mathsf{V} \in \mathcal{V}_{\mathrm{B}}(e,\mathsf{X})$ due to the definition of the function $\mathcal{V}_{\mathrm{B}}$. Therefore, it can be deduced that $\mathcal{V}_{\mathrm{W}}(d,\mathsf{X}) \subseteq \mathcal{V}_{\mathrm{B}}(e,\mathsf{X})$ which is a contradiction as we assumed that such a pair of VDs does not exist.

As both cases lead to a contradiction it is not possible that the set is not equal to $\{d\}$ contraditing our first assumtion. Therefore, the assumption has to be wrong and in deed it holds that

$$\left(\begin{array}{ll} \forall a \in \mathcal{P}_{\mathsf{X}} & : \mathcal{V}_{\mathrm{W}}(a,\mathsf{X}) \neq \emptyset \\ \forall a,b \in \mathcal{P}_{\mathsf{X}} & : \mathcal{V}_{\mathrm{W}}(a,\mathsf{X}) \not\subseteq \mathcal{V}_{\mathrm{B}}(b,\mathsf{X}) \end{array}\right) \Longrightarrow \forall d \in \mathcal{P}_{\mathsf{X}} \; : \bigcap_{\mathsf{V} \in \mathcal{V}_{\mathrm{W}}(d,\mathsf{X})} \mathcal{D}_{\mathrm{B}}(\mathsf{V}) = \{d\}.$$

□

Proof: "$\Longleftarrow$". It has to be shown that

$$\left(\begin{array}{ll} \forall a \in \mathcal{P}_{\mathsf{X}} & : \mathcal{V}_{\mathrm{W}}(a,\mathsf{X}) \neq \emptyset \\ \forall a,b \in \mathcal{P}_{\mathsf{X}} & : \mathcal{V}_{\mathrm{W}}(a,\mathsf{X}) \not\subseteq \mathcal{V}_{\mathrm{B}}(b,\mathsf{X}) \end{array}\right) \Longleftarrow \forall d \in \mathcal{P}_{\mathsf{X}} \; : \bigcap_{\mathsf{V} \in \mathcal{V}_{\mathrm{W}}(d,\mathsf{X})} \mathcal{D}_{\mathrm{B}}(\mathsf{V}) = \{d\}$$

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Table 3.7: Sets of blamed VDs in case of the individual fault of a VD of the example for the different sub-GS. Valid results are marked in green, invalid ones in red.

$\bigcap_{V \in \mathcal{V}_W(d,X)} \mathcal{D}_B(V)$	d_1	d_2	d_3	d_4	d_5
$X = \{V_1, V_2\}$	$\{d_1\}$	$\{d_1, d_2, d_4, d_5\}$	$\{d_1, d_3\}$	$\{d_1, d_2, d_4, d_5\}$	$\{\}$
$X = \{V_1, V_2, V_3\}$	$\{d_1\}$	$\{d_2, d_4\}$	$\{d_3\}$	$\{d_1, d_2, d_4, d_5\}$	$\{\}$
$X = \{V_1, \ldots, V_4\}$	$\{d_1\}$	$\{d_2, d_4\}$	$\{d_3\}$	$\{d_4, d_5\}$	$\{d_4, d_5\}$
$X = \{V_1, \ldots, V_5\}$	$\{d_1\}$	$\{d_2\}$	$\{d_3\}$	$\{d_4, d_5\}$	$\{d_5\}$
$X = \{V_1, \ldots, V_6\}$	$\{d_1\}$	$\{d_2\}$	$\{d_3\}$	$\{d_4\}$	$\{d_5\}$

The proof can be split up, as each part of the left side is implied individually by the right side and therefore it remains to show that the following two equations are valid.

$$\forall d \in \mathcal{P}_X \; : \; \bigcap_{V \in \mathcal{V}_W(d,X)} \mathcal{D}_B(V) = \{d\} \implies \forall a \in \mathcal{P}_X \; : \; \mathcal{V}_W(a, X) \neq \emptyset \tag{3.22}$$

$$\forall d \in \mathcal{P}_X \; : \; \bigcap_{V \in \mathcal{V}_W(d,X)} \mathcal{D}_B(V) = \{d\} \implies \forall a, b \in \mathcal{P}_X \; : \; \mathcal{V}_W(a, X) \not\subseteq \mathcal{V}_B(b, X) \tag{3.23}$$

To show that Equation 3.22 is valid we show that if the antecedent is true the consequent has to be true as well. For this lets assume the antecedent i.e. that the intersection of the witnessing VGs of a VD does contain only the VD itself. From this and the definition of the set intersection it follows that there has to be at least one VG in the witnessing set and hence it can not be equal to the empty set.

We proof Equation 3.23 by contradiction. Lets assume the antecedent is true and that the consequence is false i.e. that it holds that $\exists a, b \in \mathcal{P}_X \; : \; \mathcal{V}_W(a, X) \subseteq \mathcal{V}_B(b, X)$. Due to this and the definitions of the functions $\mathcal{V}_W$ and $\mathcal{V}_B$ it follows that $\forall V \in \mathcal{V}_W(a, X) \; : \; b \in \mathcal{D}_B$ and therefore also that $b \in \bigcap_{V \in \mathcal{V}_W(a,X)} \mathcal{D}_B(V)$. However, due to assuming the antecedent it has to hold that $\bigcap_{V \in \mathcal{V}_W(a,X)} \mathcal{D}_B(V) = \{a\}$ which is a contradiction as b is not in the set and hence Equation 3.23 has to be valid.

Due to the fact that both equations are valid the overall statement is valid as well. □

Due to the correctness of lemma 1 both sides of the equivalence can be used by an implementation to assure that any found GS will yield valid results. However, one property needed for an implementation is a way to compare different solutions. Therefore, the computational cost of running a grouping solution is presented in the following section.

3.4.2 Cost and Optimization

The execution of the monitoring system does incur computational cost. This cost is composed of the cost to evaluate the comparison functions in root of the VGs and the cost of running the MTs. The two parts can be analysed individually as they do not influence each other.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

The cost of the approximate-comparison functions run in the root VDs of the VGs is a property of those root VDs. Hence given as VG $\mathsf{V}(\mathsf{T}_1, \mathsf{T}_2)$ with the root VD $d_{\text{root},\mathsf{V}} = d_{\text{root},\mathsf{T}_1} = d_{\text{root},\mathsf{T}_2}$ the marginal cost is defined in Equation 3.24.

$$\mathbf{C}_{\text{M}}(\mathsf{V}) = c_{d_{\text{root},\mathsf{V}}} \tag{3.24}$$

The cost is marginal as it only includes the cost for the comparison and not the cost for the transformations to the root VD. This is necessary due to the possibility that multiple VGs can use the same MT hence the cost would be counted several times. The same is possible for multiple MTs that can contain the same sub-MT. This can be seen in Figure 3.11 where the MTs T_1 and T_2 share the common sub-MT T_3.

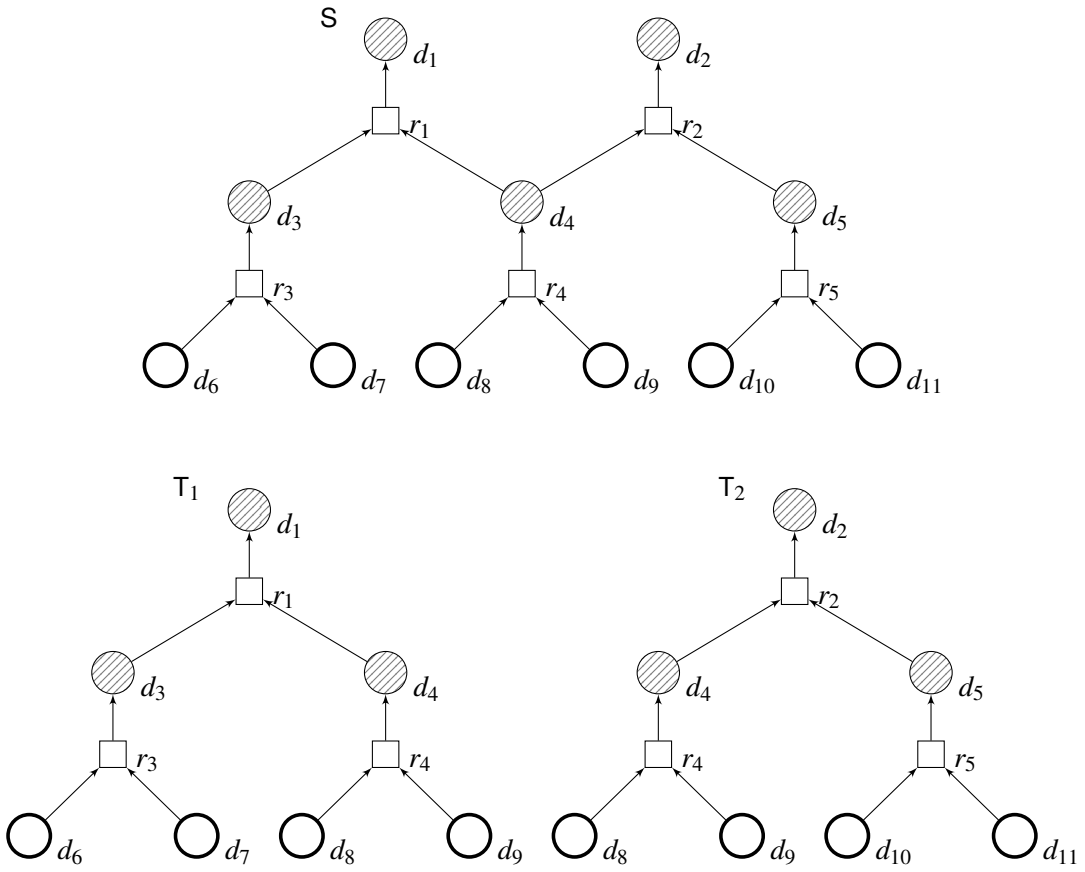

Figure 3.11: Overlapping of two MTs.

To counter this problem the whole GS has to be decomposed into the individual MTs much like the decomposition of the MT itself such that every MTs is only counted once. Hence similarly to the definition of the decomposition set on MTs in Equation 3.7 we define the decomposition sets for the VG and the GS in Equation 3.25 and Equation 3.26 respectively as the union of the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

decomposition of the underlying structures.

$$\mathcal{T}(\mathsf{V}) = \mathcal{T}(\mathsf{T}_{1,\mathsf{V}}) \cup \mathcal{T}(\mathsf{T}_{2,\mathsf{V}}) \tag{3.25}$$

$$\mathcal{T}(\mathsf{X}) = \bigcup_{\mathsf{V} \in \mathsf{X}} \mathcal{T}(\mathsf{V}) \tag{3.26}$$

These sets contain every transformation only once, which prevents counting of the cost of a single MT multiple times if only the marginal cost is considered. Marginal in this case means that only the cost of the VR that is the input to the root VD of the MT is considered. More formally we define the marginal cost of a given MT T with the root VD d_{root} and its single predecessor VR r_{pred} to be the transformation cost of r_{pred} such as given in Equation 3.27.

$$\mathbf{C}_{\mathrm{M}}(\mathsf{T}) = c_{r_{\text{pred}}}. \tag{3.27}$$

Due to the decomposition of all GSs, VGs and MTs into all sub-MTs every marginal cost can be accounted for when the individual costs of these MTs is summed up. The overall cost of a MT is therefore determined by the sum of the marginal cost of each sub-MT such as defined formally in Equation 3.28. In the case of the VG the overall cost is as well comprised of the sum of the marginal cost with the addition of the cost for the approximate-comparison function executed in the root VD. Equation 3.29 does provide a formal definition of this cost. Finally for the GS the cost of the approximate-comparison function of all VGs are added to the cost of the MTs as can be seen in Equation 3.30.

$$\mathbf{C}(\mathsf{T}) = \sum_{\mathsf{T}_{\text{sub}} \in \mathcal{T}(\mathsf{T})} \mathbf{C}_{\mathrm{M}}(\mathsf{T}_{\text{sub}}) \tag{3.28}$$

$$\mathbf{C}(\mathsf{V}) = \sum_{\mathsf{T}_{\text{sub}} \in \mathcal{T}(\mathsf{V})} \mathbf{C}_{\mathrm{M}}(\mathsf{T}_{\text{sub}}) + \mathbf{C}_{\mathrm{M}}(\mathsf{V}) \tag{3.29}$$

$$\mathbf{C}(\mathsf{X}) = \sum_{\mathsf{T}_{\text{sub}} \in \mathcal{T}(\mathsf{X})} \mathbf{C}_{\mathrm{M}}(\mathsf{T}_{\text{sub}}) + \sum_{\mathsf{V} \in \mathsf{X}} \mathbf{C}_{\mathrm{M}}(\mathsf{V}) \tag{3.30}$$

With these formulas in combination of the rules for the validity of a GS a monitoring system can be defined and multiple GS can be compared. However, before discussing how a monitoring system could be implemented we will analyse the computational complexity of our problem to guide us in the implementation. This is done in the following subsection.

3.5 Complexity of the Problem

Given the structure of our problem at hand, the selection of sensors for the purpose of monitoring, the hypothesis arises that the problem might be a member of the class NP. This hypothesis follows from the observation that given a GS X for a SD S it seems trivial to check if X is valid. This impression is however not enough as we have to show that the checking part can be done in polynomial time relative to the size of S, as we described in section 2.5. Before delving into the proof we have to refine the definition of our problem to make it more suitable for the techniques presented and hence we define our problem as a decision problem as follows. The optimization problem can than always be solved by asking for a larger solution until one is found.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Select sensors. Given a SD S and an integer C, find a valid and minimized GS X with cost of at most C, or report that none exists. Minimized in this case means that removing any VG would render X invalid.

Lemma 2. ***Select sensors*** *is a member of the class NP.*

Input: A SD S $(\mathcal{P}, \mathcal{R})$, a GS X and cost C
Output: True if X is valid, False otherwise

```
1  B ← dictionary with {d : P} for all d ∈ P                    /* Θ(|P|) */
2  /* Check for the upper bound on the size of the GS                 */
3  if |X| > |P|²/4 then                                          /* Θ(1) */
4  |  return False                                               /* Θ(1) */
5  end
6  foreach VG V (T1, T2) ∈ X do                                  /* Θ(|P|²) */
7  |  if d_root,T1 ≠ d_root,T2 then                              /* Θ(1) */
8  |  |  return False                                            /* Θ(1) */
9  |  else if not check-MT(T1) then                              /* Θ(|R| + |D|) */
10 |  |  return False                                            /* Θ(1) */
11 |  else if not check-MT(T2) then                              /* Θ(|R| + |D|) */
12 |  |  return False                                            /* Θ(1) */
13 |  end
14 |  D_blame ← P_T1 ∪ P_T2                                      /* Θ(|P|) */
15 |  D_witness ← P_T1 ⊕ P_T2                                    /* Θ(|P|²) */
16 |  foreach VD d ∈ D_witness do                                /* Θ(|P|) */
17 |  |  B[d] ← B[d] ∩ D_blame                                   /* Θ(|P|²) */
18 |  end
19 end
20 foreach VD d ∈ P do                                          /* Θ(|P|) */
21 |  if B[d] ≠ {d} then                                        /* Θ(1) */
22 |  |  return False                                            /* Θ(1) */
23 |  end
24 end
25 if C(X) > C then                                             /* Θ(|P|² * |R| + |P|² * |D|) */
26 |  return False                                               /* Θ(1) */
27 end
28 return True                                                  /* Θ(1) */
```

Algorithm 1: check-GS

Proof. The proof is done by showing that given any positive instance of **select sensors** a given solution can be checked in polynomial time for validity. More concretely given a SD S, an integer C and a GS X in S we show that the function check-GS shown in algorithm 1 checks whether X is a valid GS in S with cost less or equal to C in polynomial time. The algorithm first checks whether

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

X is inside of the upper bound for a minimized GS. Further, every VG is checked for validity and the intersection-sets for the blaming VD are computed. Finally it is checked using these sets whether Equation 3.21 is fulfilled and if the overall cost is smaller than C. The comments on the right hand side on each line mark the worst case execution time of the line, in the case of loop the worst case number of loop iterations is marked.

However, before looking at the individual lines and the respective running times lets first look at the size of the possible solutions to the problem. As GSs are sets of VGs we have to first determine how many possible VGs could exist. A major characteristic for every VG, at least for the determination of the validity, is the set of blamed VD. This set is a subset of the set of provided VDs $\mathcal{P}$ or equivalently it is an element of the power set and hence there exist $2^{|\mathcal{P}|}$ possible VGs. The size of GSs could therefore be exponential. Fortunately the definition of the problem allows only for minimized GS i.e. any VG that is not needed has to be removed otherwise the GS is not valid and therefore we have to determine what the largest minimized GS is. From the last observation it follows that the GS is the larger the more individual VG are necessary for each VD to be distinguishable [19].

For this reason we have to consider the individual sets and how they interact. For any blaming set $\mathcal{B} = \{d_1, \ldots, d_n\}$ in the solution, for every d_i there has to be family of sets where the intersection of all the sets with $\mathcal{B}$ does give the set $\{d_i\}$. This is necessary to fulfilled Equation 3.21. In other words there have to be sets that leave out every element besides d_i. However, every set for d_i is also a set for all other elements. A maximal solution has to concentrate as many sets on a small group of VD such that, more sets are needed. We have to distinguish between the size of the sets to go further on in this discussion. Therefore, lets consider that $|\mathcal{P}| = n$ and sets of size k. Then we can fix $k-1$ elements in the sets and combine them with all remaining elements. If we use k elements from $\mathcal{P}$ as the fixed ones, then there are k possible fixed parts of the sets of size $k-1$. Combined with the remaining elements there are $k(n-k)$ sets that can be build this way. The fixed elements maximize the sets that are needed in this way. This is maximized if $k = n/2$ which can be determined by setting the first derivation equal to zero. Plugging this result back in, we can see that the worst case number of sets is $\frac{n^2}{4} = \Theta(n^2)$. Given this result we discuss the running time of each line in the following list.

- **Line 1.** When implementing the dictionary as an array of pointers this operations comes down to writing a pointer to the set $\mathcal{P}$ for each element in $\mathcal{P}$.
- **Line 3.** The calculation and comparison do not depend on the size of the inputs and can be done in constant time.
- **Lines 4,8,10,12,22,26,28.** All return statements can be executed in constant time.
- **Line 6.** The worst case number of loop iterations depends on the size of the GS and its worst case is $\Theta(|\mathsf{P}|^2)$ as was shown above.
- **Line 7.** The comparison between two objects is done in constant time.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

- **Lines 9 and 11.** The function check-MT does check whether a MT is a connected and rooted DAG and that the set of provided and used VDs does only consist of leaves. It can be checked if a directed graph $G(V, E)$ is acyclic by running a depth-first search algorithm which has a worst-case running time of $\Theta(|V| + |E|)$. During the execution of the search every VD and VR is examined once, hence the other properties necessary for the validity can be checked at that point as well [19].

- **Lines 14, 15, 17.** Given two sets, the union can be computed by adding all elements from both sets to a new set in constant time each. Therefore, the running time is $\Theta(|S_1| + |S_2|)$. For intersection and symmetric difference it suffices to check for each element in one set whether it exists in the other. As lookup can take $\Theta(|S|)$ in the worst case the overall running time is $\Theta(|S_1| * |S_2|)$.

- **Line 25.** To calculate the cost of the GS a set of all sub-MT has to be created. This can be done using a depth-first-search on all MTs of the VGs, hence the running time is $\Theta(|\mathcal{P}|^2 * |\mathcal{R}| + |\mathcal{P}|^2 * |\mathcal{D}|)$ because of the worst case number of VGs and the running time of the depth-first search. Due to the fact that $\mathcal{R} \subseteq \mathcal{D}$ the running time can be expressed as $\Theta(|\mathcal{P}|^2 * |\mathcal{D}|)$.

The parts of the algorithm that are most expensive are the loop in line 6 and the check for the cost. The running time of the loop is $\Theta(|\mathcal{P}|^4)$ while it is $\Theta(|\mathcal{P}|^2 * (|\mathcal{D}| + |\mathcal{R}|))$ for the check. The overall running time is however dominated by the loop. The running time of the algorithm is therefore $\Theta(|\mathcal{P}|^4)$. This is clearly polynomial which implies that **select sensors** is a member of class NP. □

Now that we know that finding sensor redundancies is in NP we have established an upper bound for the complexity of the problem. It could however still be a problem in the class P. On the other hand there exist exponentially many possible solutions as any GS can be seen as a subset of the power set of the provided VDs which itself is already exponential in size relative to the number of provided VDs. Despite of the upper bound on the size of GS the possible combinations are still super-polynomial. The search for a solution is therefore most likely non-polynomial or rather NP-hard and therefore we propose and proof the following lemma.

Lemma 3. ***Select sensors*** *is a NP-hard problem.*

Proof. The proof is done by reduction from **vertex cover** to **select sensors**. Given an instance of **vertex cover** i.e. an integer K and an undirected graph $G(V, E)$ we have to show that there exists a polynomial-time reduction to an instance of **select sensors** such that this instance is a yes-instance if and only if the other instance is as well. We define the reduction as follows.

Reduction. Given an undirected graph $G(V, E)$ and a integer K i.e. an instance of **vertex cover**. We create an instance of **select sensors**, a SD S, as shown in Figure 3.12. First of all we add an unprovided VD s_1 and the two provided VDs s_2, s_3. Then we add a provided VD u and a VR r_u from s_2 to u with cost 1 for each $u \in V$. The next step is to add a provided VD e_{uv} and an unprovided VD e_{uv_s} for each edge $(u, v) \in E$ and connect s_3 and e_{uv} via a 0-cost VR to e_{uv_s}.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

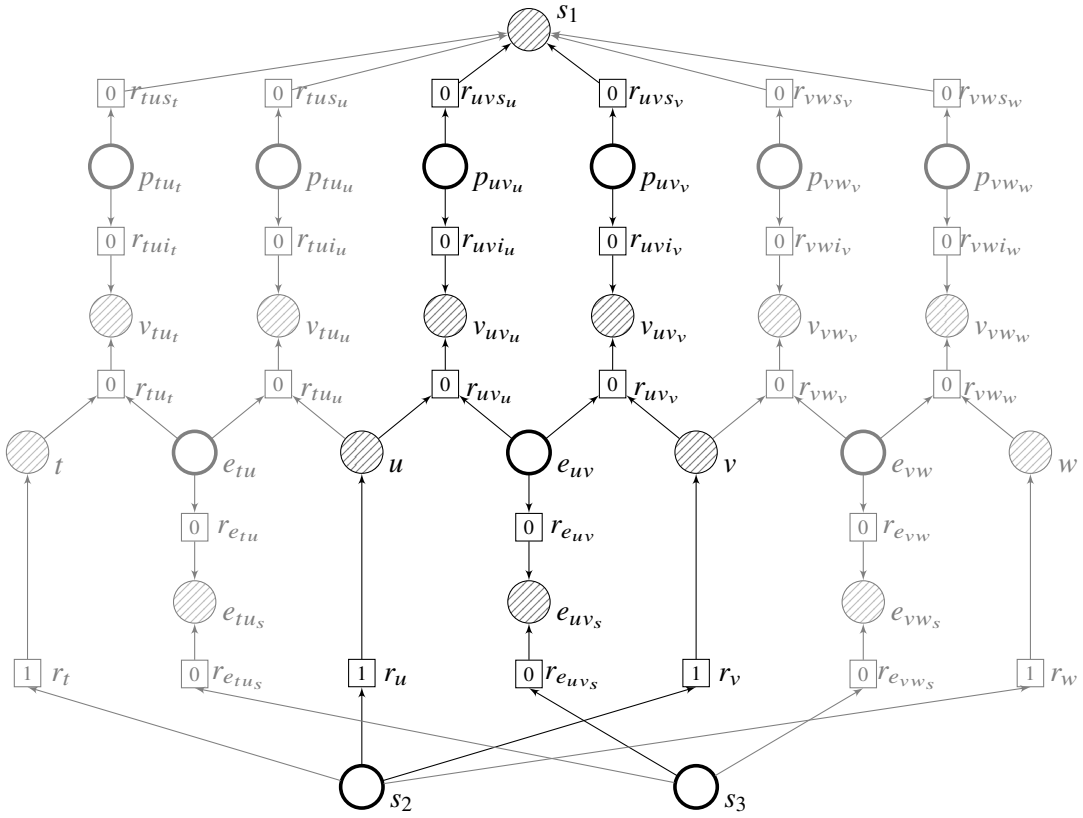

Figure 3.12: Part of the reduced instance for edge (u, v) and its coadjacent edges $(t, u), (v, w)$.

Further, we add unprovided VDs v_{uv_u}, v_{uv_u} and provided VDs p_{uv_u}, p_{uv_v} and connect the p's to the corresponding v's and to s_n via 0-cost VRs. Lastly we add a VR r_{uv_u} with e_{uv} and u_n as inputs and r_{uv_u} as output for the side of vertex u and the same for the side of vertex v. All of these VR are 0-cost as well. The resulting instance of **select sensors** is comprised of the SD S and the integer K.

The running time of the reduction is $\Theta(|V| + |E|)$ as a constant number of VDs and VRs are added for each vertex and each edge. Therefore, the reduction is clearly computable in polynomial-time. It remains to show that the reduced instance is a yes instance if and only if the given instance of **vertex cover** is one as well. We will proof the two implications of the equivalence separately.

"$\Longrightarrow$:" Given a yes-instance of **vertex cover** i.e. a graph $G(V, E)$ and an integer K then there exists a vertex cover $C \subseteq V$ with size $|C| \leq K$. Further, let SD S be the reduction of graph G to **select sensors** then there exists a minimized and valid GS X with a cost of $\mathbf{C}(\mathsf{X}) = |C| \leq K$. In the GS X for each edge $(u, v) \in E$ there exist two VGs. If $u \in C$ then there is a VG with blaming set s_2, e_{uv}, p_{uv_u} in X, additionally, if $v \in C$ then there is a VG with blaming set $\{s_2, e_{uv}, p_{uv_v}\}$ in X. From the definition of the vertex cover we know that there has to be at least one vertex in C. If only one is included, then a VG with blaming set $\{e_{uv}, s_3\}$ is included. In either case the

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

intersection of the two sets is $\{e_{uv}\}$ hence the solution is valid for e_{uv}. As it is necessary to use VRs r_u or r_v for the MTs if $u \in C$ or $v \in C$, cost of 1 unit is added for every vertex included in the vertex set. Due to the fact, that VD s_2 is part of all the VGs of representing the included vertices, its set of blamed VDs is s_2, as for every edge it has to be included at least once hence these different elements cancel each other out during the intersection. Similarly VD s_3 is part of all VGs that have to be included when only one vertex is part of the vertex cover. Again the different VDs e_{uv} cancel each other out in the intersection. For the VDs p_{uv_u}, p_{uv_v} for all edges $(u, v) \in E$, there can be found pairs as all of them, can be transformed to VD s_1 and therefore the solution is valid for them as well. Due to the fact, that only the VR r_u for $u \in C$ incur a cost it follows that $\mathbf{C}(\mathsf{X}) = |C| = K$.

"$\Longleftarrow$:" Given an instance of **vertex cover** i.e. a graph $G(V, E)$ and integer K as well as the a SD S reduced from G and a valid GS X of cost K. Then there exists a vertex cover which contains each vertex u whose corresponding VR r_u is part of the solution or more formally $C \subseteq V$ with $C = \{u \in V | r_u \in \mathsf{X}\}$. This follows from the fact that only VRs corresponding to the vertices incur any cost that can contribute to the cost of the solution, hence not more then K can be in the solution. Further, are they needed for a valid solution due to the fact that at least one has to be transformed to make the GS valid for VD e_{uv} corresponding to the edges $(u, v) \in E$. Hence C is a vertex cover.

From these results it follows that **select sensors** is NP-hard and therefore also NP-complete. □

The NP-completeness property of **select sensors** does provide an important peace information for the implementation of an algorithm for the problem. Unless P=NP there does not exist a implementation that can solve **select sensors** in polynomial time. Moreover any attempt of creating an implementation that gives optimal solutions will always take super-polynomial running time. In chapter 4 we present an implementation and analyse its complexity in chapter 5.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 4

Implementation

This chapter discusses the implementation of the monitoring system. The implementation uses knowledge in the form of a representation in a high-level, domain-specific language to find redundancies that can be used to detect sensor faults during operation. These tasks are done at different points in the life-cycle of a system. During the design phase or during the configuration phase when modules are added to a system, a GS is computed for the system. This GS is then used as the configuration for the run-time system that performs the monitoring itself. The following chapters describe these parts and their interfaces.

4.1 Generating Grouping Solutions

During the development phase or alternatively during the configuration of a CPS, redundancies have to be selected which the monitoring system can use for the comparison of sensor values to detect sensor faults. For this purpose, we propose an implementation of a two step system which can be seen in Figure 4.1. The input of the system consists of a high-level system description provided by the system designer. This high-level system description is then transformed to the low-level SD presented in chapter 3. This circumstance allows for the adaptation of the approach to any domain. Using a model transformation a system model described in a domain specific high level language can be transformed to the our basic meta model consisting of VDs and VRs. The low-level SD is then used to generate a GS that represents the output of the system.

The model transformation is part of each individual adaptation of the framework into a development process for a certain engineering domain. Therefore, it is not part of this work, will not be further described and considered as a given. The grouping process however represents the main part of the contribution that is presented in this work and will therefore be described in detail in this subsection.

However, before the description of the algorithm itself a short overview of the goals of the solution shall be given. As presented in section 3.4, the smallest MTs can be combined into bigger MTs

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

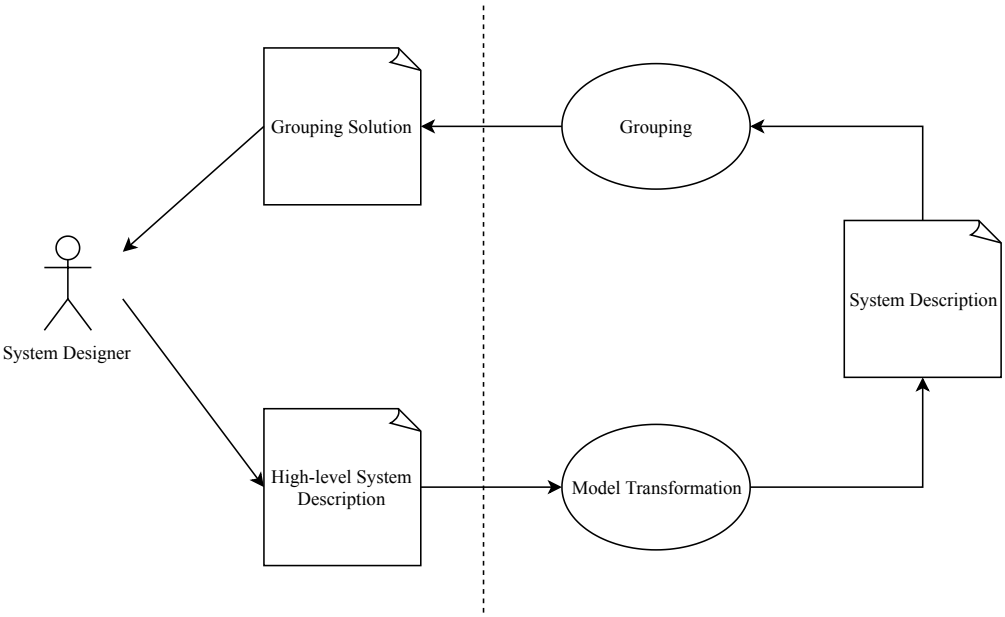

Figure 4.1: Overview of the implementation of the design-time part of the monitoring system

and VGs. These VGs again can be combined into GSs. The task of the algorithm is to find valid GSs according to Equation 3.21 for a low cost according to the cost-function defined in Equation 3.30. From an intuitive perspective, this is achieved by building such GSs starting from the provided VDs of a SD and incrementally building new MTs and VGs along all possible paths to other provided VDs in the graph corresponding to the SD. The cost metric can be used to guide the search i.e. the next step the algorithm will take is always the cheapest in comparison to all other possible ones. When a step was made in one branch however, the added cost will often lead to the next step being taken in another path that now has the cheapest next step. This shows that in an abstract sense the algorithm is comparable to a breadth-first search. With this fact in mind, the implementation was designed in a parallel manner such that the multi-processing capabilities of modern CPUs can be used for performance gains. To achieve this, a fork in the paths through the graph was implemented as a fork in the execution flow using coroutines, which are scheduled according to the associated cost of the corresponding partial solutions. This forking approach leads to an implementation that has a program flow that does correspond to a depth-first search. This search is however interrupted by the scheduler according to the cost of the partial solutions. The depth-first search and the scheduling infrastructure is presented in the following to subsections, respectively.

4.1.1 Depth-First Search

The entry point for the algorithm can be any provided VD in the SD. At the beginning, there is one measurement available at this VD namely the one provided via the corresponding sensor. The goal of the algorithm is now to find several VGs that form a GS for the VD. For this purpose,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

the provided measurement can either be transformed to a VD to be compared there to another measurement or another measurement is transformed to the entry VD for the comparison at that point. The algorithm hence has two possible directions to take either in the direction of the VRs or opposite to the direction. In the algorithm, these directions are called ascending and descending, respectively. From a data-structure point of view the ascending direction does correspond to creating new MT from the provided measurement where the descending direction searches for MTs that end in the initially provided VD.

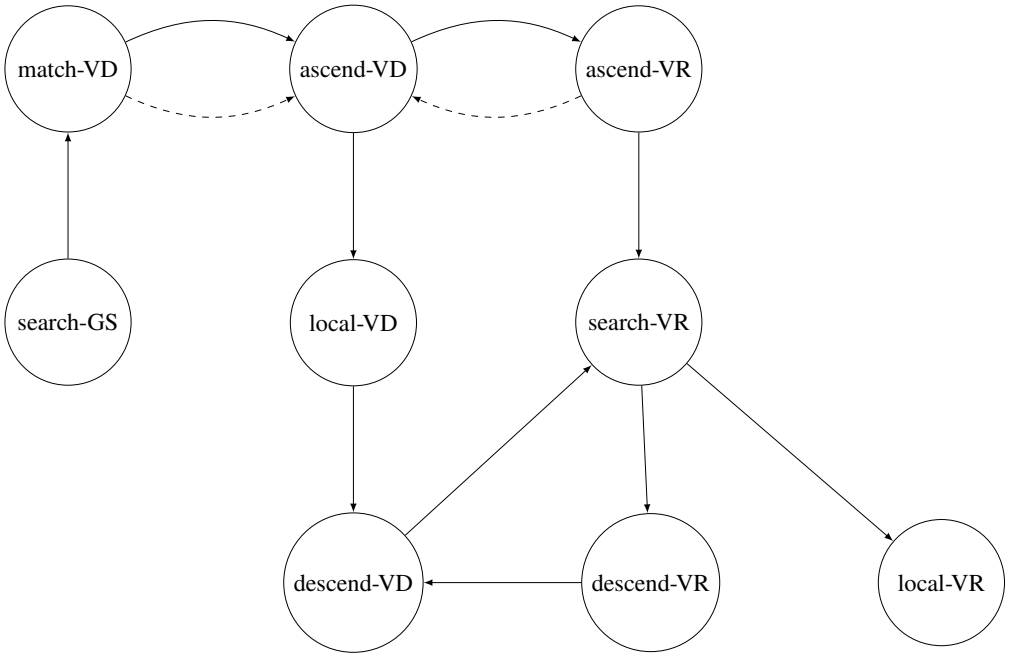

Figure 4.2: Call graph of the algorithm

The algorithm is comprised of a set of functions that call each other recursively. Figure 4.2 shows a call graph of these functions. From an abstract perspective, each function creates coroutines according to the call graph that report all found solutions back to it. These solutions are then possibly augmented by the calling function and passed back to its calling function recursively. Some functions however augment the solutions by passing them to another function before returning the result. These calls are drawn as a dashed arrow. Although similar in structure the functionality of the functions is adapted to the location in the graph they represent and the purpose can be described as follows.

- **search-GS:** This function represents the entry point to the algorithm. Its aim is to create a GS that enables the detection and attribution of every provided VD in a SD. For this, the function calls 'match-VD' repeatedly for every provided VDs passing the partial GS returned from previous calls as can be seen in algorithm 2.
- **match-VD:** The purpose of this function is to find a partial GS that enables a monitoring

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Input: A SD S $(\mathcal{P}, \mathcal{R})$
Output: A GS X that monitors all provided VGs
1 X $\leftarrow \emptyset$
2 **foreach** *VD d in* $\mathcal{P}$ **do**
3 | X $\leftarrow$ `match-VD(`d, X, S`)`
4 **end**
5 **return** X

Algorithm 2: search-GS

system to detect and attribute all faults that origin from the provided VD. If a partial GS is given as input it is extended to accommodate all VGs needed for the detection of the provided VD. In the case that no combination of VGs is found that covers the VD the input GS is returned with the VD marked as impossible to monitor correctly. The first step of the function is to launch the function 'ascend-VD' as a coroutine. All solutions returned by this coroutine are checked for validity. Not valid solutions are passed again to the function 'ascend-VD' by creating additional coroutines. If a valid solution is found however, the function terminates returning the new solution. All launched coroutines are aborted with the termination of the function. In the case that all coroutines terminate without yielding any valid solution, the whole graph was searched exhaustively and the VD d is impossible to monitor. Hence it is marked accordingly in the original GS provided as an input and returned together with it. In algorithm 3, a representation of the function is shown. The validity is determined according to Equation 3.21 with the VD d fixed to the VD provided to the function.

Input: A VD d, a partial GS X, a SD S
Output: A GS that can monitor VD d or that marks d as not monitored
1 **foreach** $\mathsf{X}_{\mathsf{new}}$ *yielded from* `ascend-VD(`d, X, S`)` *and further run coroutines* **do**
2 | **if** $\mathsf{X}_{\mathsf{new}}$ *is valid for* d **then**
3 | | **return** $\mathsf{X}_{\mathsf{new}}$
4 | **else**
5 | | `async-run(ascend-VD(`d, $\mathsf{X}_{\mathsf{new}}$, S`))`
6 | **end**
7 **end**
8 X $\leftarrow$ `mark-unmatched(`X, d`)`
9 **return** X

Algorithm 3: match-VD

- **ascend-VD:** Calling this function corresponds to the step from a VR to a VD while creating a MT. At this point the direction of the search splits in two. Part of the search does ascend further while the other part descends to find another MT to create a VG rooted in the VD the function was called on. Hence the function creates a coroutine for every VRs that is connected via an outgoing edge in the graph by calling the function 'ascend-VR'. In the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

case that taking a VR would create a looping MT such a VR is ignored to prevent endless loops in the algorithm. While these coroutines represent the ascending part there is another coroutine created that represents the descending part and executes the function 'local-VD'. As can be seen in algorithm 4, that represents this function, the results yielded by the coroutines are passed along to the calling function.

Input: A VD d, a partial GS X, a SD S
Output: Yields GSs containing a new VG
1 `/* Build a set of VRs` $\mathcal{R}_{\text{asc}}$ `that would not create a loop in the graph if taken as the next step. */`
2 $\mathcal{R}_{\text{asc}} \leftarrow \{r \in O_d \mid r \notin \mathsf{X}\}$
3 $\mathcal{F}_{co} \leftarrow \{\text{ascend-VR}(r, \mathsf{X}, \mathsf{S}) \mid r \in \mathcal{R}_{\text{asc}}\} \cup \{\text{local-VD}(d, \mathsf{X}, \mathsf{S})\}$
4 `async-run(`$\mathcal{F}_{co}$`)`
5 **foreach** X_{new} *yielded from the running coroutines* **do**
6 | **yield** X_{new}
7 **end**

Algorithm 4: ascend-VD

- **local-VD:** This function combines the MT created along the ascending path and combines it with all MTs found descending from the VD. The resulting solution contains a VG comprised from the MTs. This is achieved by creating a coroutine that executes the function 'descend-VD'. The resulting GS contain said MTs. If the combination with the MT from the ascending branch is a valid VG as well as the GS resulting from the input solution augmented by this new VG, then the new GS is yielded to the calling function. An abstract representation of this function can be seen in algorithm 5.

Input: A VD d, a partial GS X, a SD S
Output: Yields GSs containing a new VG
1 **foreach** X_{new} *yielded from* `async-run(descend-VD(`d, X, S`))` **do**
2 | $\mathsf{X}_{\text{new}} \leftarrow$ `combine-MTs(`X, X_{new}`)`
3 | **if** X_{new} *is valid* **then**
4 | | **yield** X_{new}
5 | **end**
6 **end**

Algorithm 5: local-VD

- **descend-VD:** This function is one of the two entry points into the descending path with the other being 'search-VR' as can be seen in Figure 4.2. Starting from the VD provided as input a partial solution is build containing a new MT that can be combined with the one of the ascending path in the case the function was called from 'local-VD' or the partial solution is passed to the calling function 'descend-VR' where a bigger MT is build from the one in the passed solution. To achieve this the function does two things. First of all, if

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

the VD is provided a partial solution is created with a MT representing the provided values by calling the function 'VD-solution'. This solution is yielded to the calling function. As the second part coroutines that execute 'search-VR' are launched for all inputs of the VD and all the yielded GS are passed along to the calling function. However, as can be seen in algorithm 6 that gives a pseudo-code representation of this function, not every input VR is considered. Some of the inputs may already be part of the descending path which would lead to a loop in the MT. These VR are therefore filtered out in line 5. Otherwise the algorithm would follow a loop indefinitely and hence never terminate.

Input: A VD d, a partial GS X, a SD S
Output: Yields GSs containing a new VG
1 **if** *d is provided* **then**
2 | **yield** `VD-solution(`d`,` X`)`
3 **end**
4 `/* Build a set of VRs` $\mathcal{R}_{\text{desc}}$ `that would not create a loop in the graph if taken as the next step. */`
5 $\mathcal{R}_{\text{desc}} \leftarrow \{r \in \mathcal{I}_d \mid r \notin \mathsf{X}\}$
6 $\mathcal{F}_{co} \leftarrow \{\text{search-VR}(r, \mathsf{X}, \mathsf{S}) \mid r \in \mathcal{R}_{\text{desc}}\}$
7 `async-run(`$\mathcal{F}_{co}$`)`
8 **foreach** X_{new} *yielded from the running coroutines* **do**
9 | **yield** X_{new}
10 **end**

Algorithm 6: descend-VD

- **ascend-VR:** This function is part of the ascending path. Its purpose is to create a valid MT on the way up to the next VD where new VG can be created. The MT is created from the MT that is part of the partial solution provided by the calling function combined with MTs that have to be created for all inputs other then the VD the calling function 'ascend-VD' was running on. Therefore, the function 'search-VR' is called by creating a coroutine to search for the MTs on the descending path. When such MTs are found and yielded contained in a partial GS the function creates a coroutine that executes 'ascend-VD' with the newly found GS. As can be seen in algorithm 7 the function has to distinguish between the partial GS yielded from the descending path and those returned from the ascending path. Where the former are passed to the ascending path as already mentioned, the latter are yielded to the calling function returning a new VG or even a complete GS to the caller.
- **search-VR:** This function is the other entry point into the descending path. Its purpose is to find MTs that can provide measurements to the output VD of the VR provided to the function. To achieve this the function creates two coroutines executing the functions 'descend-VR' and 'local-VR'. Where the former will search for the MTs the latter will return already found MTs from a cache. Although not necessary for the correctness this distinction enables the approach to be more efficient in terms of memory and processor usage. The yielded partial solutions yielded by the two functions are passed along to the calling function.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Input: A VR r, a partial GS X, a SD S
Output: A GS containing a new VG
1 **foreach** X_{new} *yielded from* `search-VR(`r`, X, S)` *and further run coroutines* **do**
2 **if** X_{new} *contains a new VG* **then**
3 **yield** X_{new}
4 **else**
5 `async-run(ascend-VD(`o_r`, `X_{new}`, S))`
6 **end**
7 **end**

Algorithm 7: ascend-VR

Input: A VD d, a partial GS X, a SD S
Output: Yields GSs containing a new VG
1 `/* Build a set of VRs` $\mathcal{R}_{asc}$ `that would not create a loop in the graph if taken as the next step. */`
2 $\mathcal{F}_{co} \leftarrow$ {descend-VR(r, X, S), local-VR(r, X, S)}
3 `async-run(`$\mathcal{F}_{co}$`)`
4 **foreach** X_{new} *yielded from the running coroutines* **do**
5 **yield** X_{new}
6 **end**

Algorithm 8: search-VR

- **local-VR:** This function does provide the cached partial GS for a given VR. As can be seen in algorithm 9 this is achieved by reading the cache and yielding each GS to the calling function. The cache has to be synchronized between this function and the search done by 'descend-VR' in parallel.

Input: A VR r, a partial GS X, a SD S
Output: Yields GSs already cached for the VR r
1 **foreach** X_{new} *in* `GSs-from-cache(`r`)` **do**
2 **yield** X_{new}
3 **end**

Algorithm 9: local-VR

- **descend-VR:** This function does create MTs for a VR. For this the function 'descend-VD' is executed in coroutines for the inputs of the VR unless the provided partial GS does provide a MT transformation for that input i.e. if the function is called from the ascending path. To produce the MT the function has first to wait until a solution was yielded for every input. The variable $\mathcal{D}_{\text{unprovided}}$ is used to track the missing inputs that are not already provided. Note that the root of the provided GS is excluded, as it is already known. This is done by calling the function 'root-VD' that provides this root as can be seen in lines 1 to 6 of the pseudo-code representation of the function in algorithm 10. Each time a solution

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

is found for an input it is removed from $\mathcal{D}_{\text{unprovided}}$ and the solution is added to the set of solutions found for the input in the dictionary $\mathsf{S}_{\text{input}}$. When a new solution is found and all inputs are already provided then the known solutions are combined in every possible way creating a GS with the corresponding MT. The GS are then put in the cache and further yielded to the calling function.

Input: A VR r, a partial GS X, a SD S
Output: A GS containing a new MT with r as the top most VR

```
1  if root-VD(X) ∈ I_r then
2  |  S_input ← dictionary with {root-VD(X): {X }}
3  else
4  |  S_input ← empty dictionary
5  end
6  D_unprovided ← I_r \ { root-VD(X)}
7  F_co ← {descend-VD(d, descending-GS(X, r, d), S) | d ∈ D_unprovided }
8  foreach X_new yielded from async-run(F_co) do
9  |  d_input ← root-VD(X_new)
10 |  /* If we found a solution from a jet unprovided input remove
   |     that input from the set                                    */
11 |  D_unprovided ← D_unprovided \{d_input}
12 |  /* Add the solution to the set of solutions for the
   |     corresponding input                                        */
13 |  S_input[d_input] ← S_input[d_input] ∪ {X_new }
14 |  /* If a partial GS was found for every input                 */
15 |  if D_unprovided = ∅ then
16 |  |  /* Generate new GSs for all possible combinations of the
   |  |     other inputs with X_new                                */
17 |  |  foreach X_new in combine-input-MTs(r, X_new, S_input) do
18 |  |  |  cache-GS(r, X_new)
19 |  |  |  yield X_new
20 |  |  end
21 |  end
22 end
```

Algorithm 10: descend-VR

- **combine-MTs:** This function is needed to combine two partial GSs representing two MTs into one GS with a VG. For this, the MTs are extracted from the GSs, combined into a VG. This VG is then used to create a new GS. The pseudo-code representation of this function can be seen in algorithm 11. According to the definition of the VG in Equation 3.8 the function checks for the common root of the MTs and whether the MTs are different. Hence, only valid VGs are created by this function.

- **combine-input-MTs:** This function is used to generate all possible MTs from input MTs

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Input: Two GSs X_1 and X_2 representing a new MT each
Output: A GS X_{new} containing a VG fromed by combining the two MTs

```
1 T1 ← get-MT(X1)
2 T2 ← get-MT(X2)
3 if root-VD(X1) ≠ root-VD(X2) or T1 = T2 then
4 |   return None
5 end
6 V ← new-VG(T1, T2)
7 Xnew ← new-GS(V, X1, X2)
8 return Xnew
```

Algorithm 11: combine-MTs

of a VR. Therefore, all possible combinations of the GSs for the inputs other then the newly found GS are combined with that new GS or more precisely their individual MTs are combined to a new MT which can transform measurements to the output VD of the given VR.

Input: A VR, a newly found GS X_{new} and a dictionary of input-GSs S_{input}
Output: A GS containing the new MT

```
1 foreach X_group in the Cartesian Product of S_input \ root-VD(X_new) do
2 |   T_new ← new-MT(r, X_new, X_group)
3 |   X_out ← new-GS(T_new)
4 |   yield X_out
5 end
```

Algorithm 12: combine-input-MTs

In combination these functions provide an algorithm to find a valid GS for a given SD. Due to the construction of MTs and VGs in an additive procedure, the resulting solution is a GS. The solution is build in an iterative manner by adding the cheapest VG until the new solution is valid for the provided VD. The validity for a VD is checked in function 'match-VD' shown in algorithm 3 or if there is no valid solution that fact is marked in the solution. This process is repeated for all provided VDs in the SD. The resulting solution hence is valid for all provided VDs for which a solution does exist and marks the ones for which there does not. One thing that remains to be shown however is that the addition of VGs to a solution that is valid for a VD does not render the solution invalid.

The validity of a solution in regards to VD can be determined using Equation 3.21 with the VD fixed, hence a GS X is valid in regards to VD d if

$$\bigcap_{V \in \mathcal{V}_W(d,X)} \mathcal{D}_B(V) = \{d\}.$$

Given a GS X that is valid for VD d, if an additional VG V is added that is a witness of d then the equations holds as well as $\{d\} \cap \mathcal{D}_B(V) = \{d\}$ due to the definition of the set intersection and the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

fact that d has to be an element of the set $\mathcal{D}_B(V)$. Therefore, the addition does not change the validity and leads to the conclusion that all returned GS are valid. This concludes the correctness considerations of the algorithm in regards to the validity, the quality of the solution in terms of the cost is however determined by the scheduling part of the algorithm which is discussed in the following subsection.

4.1.2 Coroutine Scheduling and the Breadth-First Search

The presented algorithm in subsection 4.1.1 does not work as presented without a scheduling algorithm that is aware of the cost of partial GS. Only by the inclusion of that information, the algorithm does execute in a breadth-first manner and not in a depth-first one. As the additional code necessary for the scheduling would make the presented pseudo-code harder to comprehend it was intentionally left out. The implementation of the scheduling is however detrimental for the correctness and computational complexity of the algorithm. Hence it will be presented in this subsection.

The scheduling of the coroutines is done in a cooperative manner as no interruptions in the program flow are supported by their execution framework. Interruption is however not necessary for the implementation of the algorithm as only the cheapest solution should be processed at any time which makes interruption unnecessary as long as the processing proceeds. If an interruption becomes necessary, the interruption can be done in an interactive approach where the functions give back the execution flow to the scheduler at fixed points in the execution path. Due to the common structures in the functions presented in subsection 4.1.1 these points can be generalized. In Figure 4.3 such a generalization of the functions is shown in the form of a state transition diagram. The generalization is done by giving a super-set of state transition diagram of the functions. However, all function progress more or less along these state transitions which can be used as points for the cooperative scheduling. There are two states, where the functions have to wait. The first is the point after the start of each function where the cost of the partial GS can be compared to the cost of the other GS and the coroutine waits to be the one with the cheapest GS. Most of the functions executed in the coroutines will proceed to launch coroutines themselves and wait for results thereafter. At this point they give the execution back to the scheduler until a solution was returned by a called function and that solution is the cheapest one that hence can be scheduled. Using this approach only the cheapest GS is processed at any moment.

For the scheduling of the cheapest solution, the scheduling algorithm has to keep track of all solutions and their cost. For this purpose, priority queues are used, that sort the solutions based on the cost and always return the cheapest. However, before the sorting can take place, the cost of a solution has to be determined. We defined the cost of a complete GS in Equation 3.30. Although the algorithm does handle partial solutions which consist of a possibly empty list of VGs and a MT that is currently build into a VG, the defined function does provide a correct cost. It adds together the costs for all MTs and the cost of the comparisons in the GSs. Hence, when applied to a partial solution, all parts of the solution that occur costs are included. This follows from the fact, that in each step a new MT or VG is created which represents the current state of the algorithm. For the descending path however, the cost of the steps taken has to be taken into account before the MT can be created because it is build from the top to the bottom. Further,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

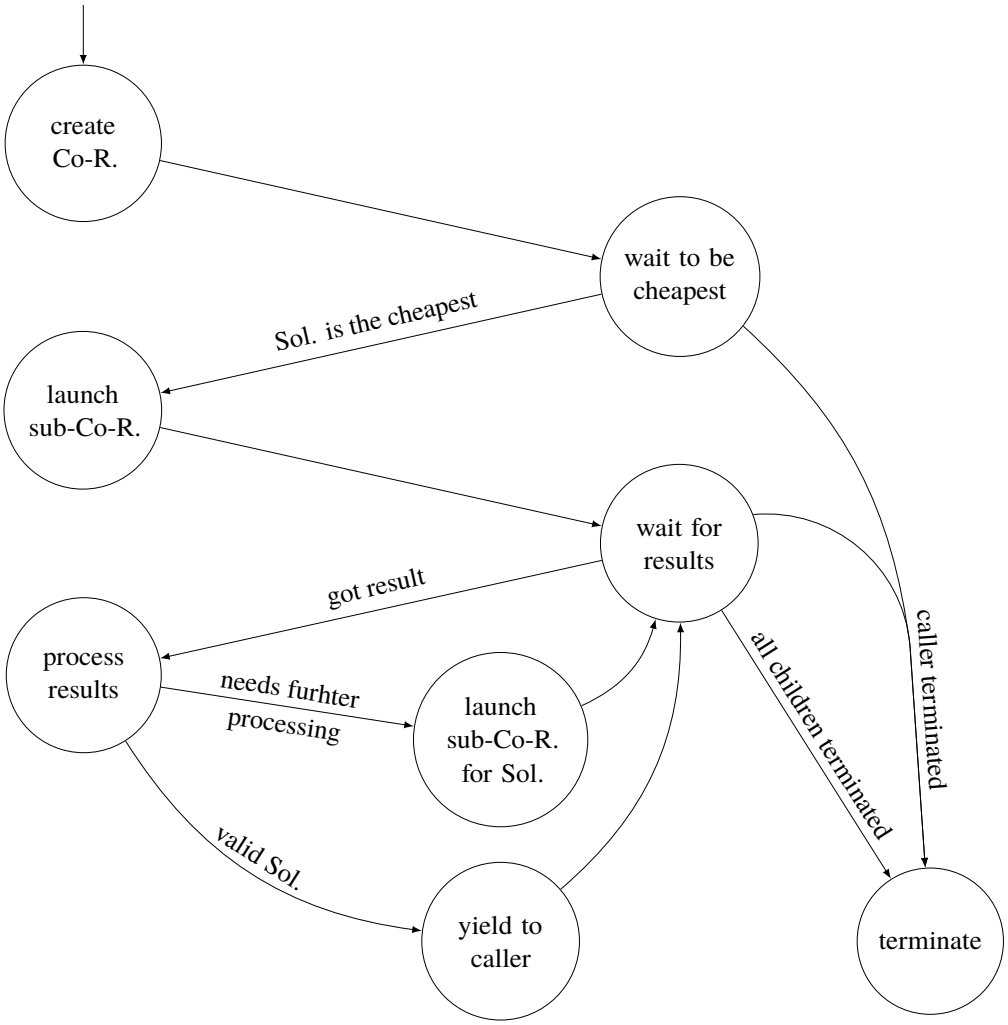

Figure 4.3: General abstract state transitions of the functions presented in subsection 4.1.1.

there is also the possibility that multiple MTs are found during the descend, hence the cost can not be determined beforehand. For these reasons, the cost is added to the partial GS during the descend. This happens in line 7 of function 'descend-VR' shown in algorithm 10 where a new solution is created for each descend path using the function 'descending-GS'. The cost of the VR is added to this GS such that it can be taken into account by the scheduler. This is however not the only reason for a respective GS for every path. It is also the solution for additional problems of the cost calculation on the descend path.

As already mentioned the search in the descending path is different in terms of the cost than the ascending path. Where the ascending path has only a single direction the new solution is build to, the descending path branches out in all directions. A possible MT that can be found while descending is a combination of all those paths and hence its cost is the sum of the cost of all the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

paths added together. For this reason the solutions processed on all the paths are sorted in a queue and only the combined cost is added to the higher level queue for the scheduler to consider. Using this approach the scheduling algorithm will first schedule a tree of descending branches because their overall cost is the cheapest, continuing by scheduling the cheapest of the paths recursively until a GS is found that can be scheduled on their own. The data-structure of the queues does form a tree structure that corresponds to the tree structure of the SD itself.

One further concern that has to be taken care of in regards to the correct calculation of the cost comes from the possibility of multiple solutions coming from the different paths. As mentioned in subsection 4.1.1 the function 'combine-input-MTs' used in algorithm 10 produces MTs using the Cartesian product. Hence when a new solution is returned form a path several new GS are produced. This has to be taken into account when calculating the cost of the solution in the path. From a global perspective the cost is equal to the sum of the cheapest partial solution of any path and the cost of the VR mentioned before. The function 'descending-GS' does take care of the note keeping for all the descending paths such that the GS does report back the right cost during the sorting process. Therefore, the property that only the cheapest partial solution is processed at any time can be maintained throughout the algorithm.

Coming back to the tree structure of the queues an additional aspect is important to note. The scheduler will walk the graph in a breadth-first manner. Given a VR and a partial solution on it that has been scheduled, it will progress on the paths descending from the VR until there is no partial solution left in that part of the tree that is cheaper than the next best solution in another part of the tree. The next step is to schedule the next cheapest solution until its descending paths do not contain any solution with the cheapest cost. Hence, the cost of the solution that is processed at any moment will only increase during the algorithm. Interestingly, the cost does represent the depth in the search and the algorithm will process all solutions with a cost of n before those with cost $n + 1$. Therefore, this algorithm corresponds to a breadth-first search.

4.2 Run-time Implementation

As already mentioned, the GS that is generated is used to configure the run-time part of the monitoring system. This run-time part takes the provided sensor measurements of its CPS as inputs and determines whether there is any measurement fault and if so which sensor is responsible. In Figure 4.4 a graphical representation of the monitoring system can be seen with its three components. The first component executes all the measurement transformations according to the GS whenever new sensor measurements become available. The second component executes all approximate comparison functions of all the VGs defined in the GS using the provided measurements as well as the transformed values from the previous component. The third component uses the results of the previous to determine the validity of the sensor measurements and in the negative case which sensor is to blame for a detected fault. The validity can simply be deduced from the results of the previous component. If no approximate comparison function detects a discrepancy then the system is running correctly or more formally the

$$\text{system is valid} \iff \bigwedge_{V \in X} \overline{\mathbf{AP}}(V).$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

If the system is not valid the sets of faulty sensors can be narrowed down using Equation 3.13, the intersection of the blamed VD. If a fault is present the resulting set will shrink until all VGs have detected the discrepancies and only one VD remains representing the faulty sensor.

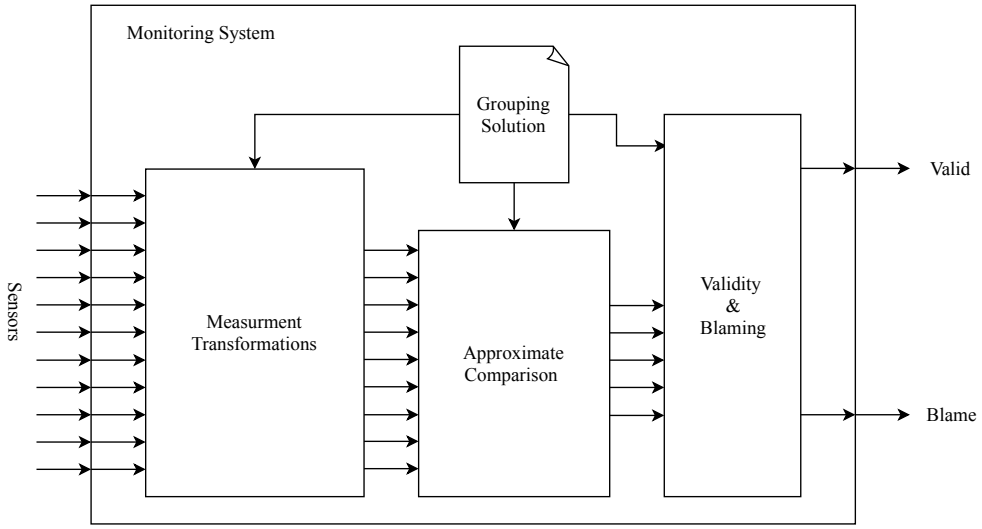

Figure 4.4: Overview of the monitoring system run-time implementation

This concludes the description of the implementation of the sensor monitoring system. In the following section the performance, efficiency as well as the shortcomings of the presented solution will be discussed.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 5

Evaluation

In the previous chapters, the problem of selecting sensors was analysed and an implementation in the form of an algorithm was presented. This chapter will cover the evaluation of this algorithm. First, the quality of the computed solutions is discussed and as a second step the running time is determined. However, before taking these considerations, it is investigated how the algorithm performs on the model of the robot presented in section 3.1. However, the SD of the model shown in Figure 3.2 does not include the transformation cost. Therefore, we reintroduce the model in Figure 5.1 including these costs shown as integers in the VRs.

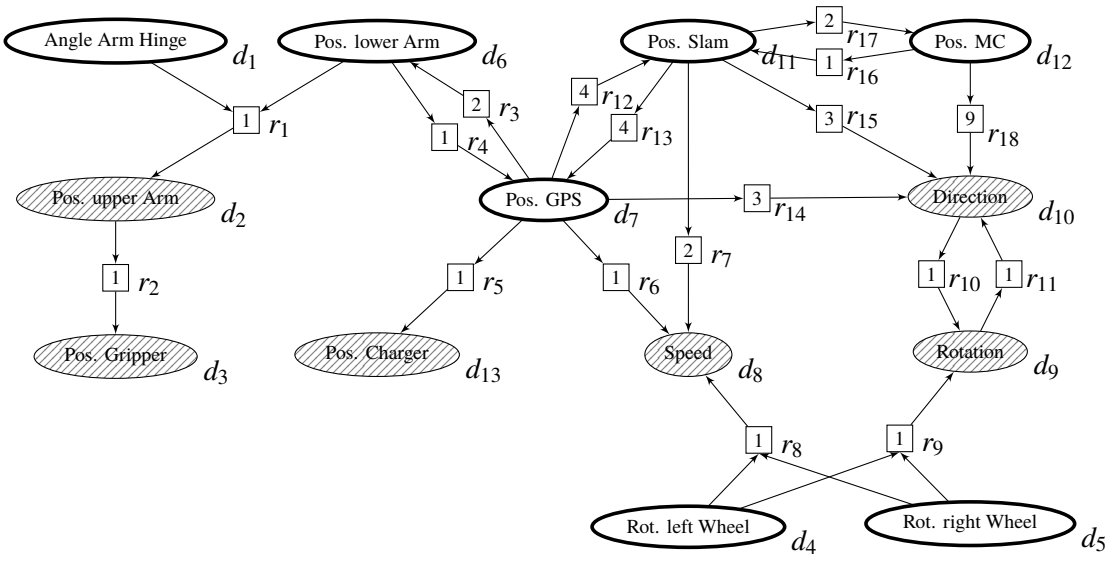

Figure 5.1: SD from Figure 3.2 with transaction costs shown in the VRs.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

As already mentioned in chapter 4, the algorithm was implemented in the programming language "Python". This allowed for a prototypical implementation in a short time but led to a longer execution time of the algorithm. Hence this results in a rather long execution time of 60 seconds when run on the robot model. The returned GS is shown in Figure 5.2. The dashed ellipses around VDs d_7, d_8 and d_{11} mark the roots of the VGs. It can be seen that VD d_1 is not connected to any of these as it is not part of any VG. Therefore, VD d_1 can not be monitored by the system. A developer would have to augment the SD with additional VR such that a VG can be found or if not possible additional sensors would have to be added to the system to enable the monitoring of the VD. Similarly VDs d_4, d_5 can not be monitored despite of being connected to the graph. The connection via VR r_8 is shared. This makes the attribution of a fault impossible, as the set of possibly faulty VDs will contain both VDs. A look at the model does show that there would be another possible VR (r_9) that could be used to monitor the two VDs. Due to the fact that both VRs are influenced by the two VDs adding r_9 does not improve the capabilities of the monitoring system to determine the origin of a fault. Again, only the addition of VRs or sensors would enable the monitoring of the two VDs.

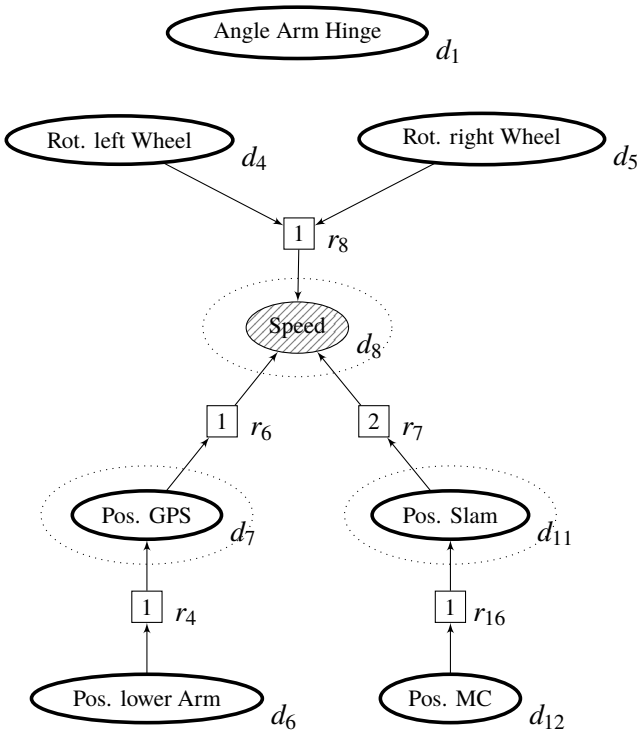

Figure 5.2: GS for the SD shown in Figure 5.1

We can formally verify these observations by looking at the individual VGs shown in Table 5.1, or more specifically the sets of blamed VDs of each VG. First of all, VD d_1 is not part of the blaming set of any VG and VDs d_4 and d_5 are only part of the blaming set of VG V_2. Hence, the monitoring is not possible for these VDs, because this violates Equation 3.21, which is necessary

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

for a valid solution. For the other VDs however, the monitoring is possible. To verify this, we have to check for the intersection of the blaming sets, which is shown in Table 5.2. For all the remaining VDs, the set consists only of themselves which implies that the solution is valid for each of them individually. Therefore, the given solution is also correct for the VDs where a monitoring solution is feasible.

Table 5.1: The VGs of the solutions with their root VD, their MTs and the set of blamed VDs.

VG	root	t_1	t_2	$\mathcal{D}_B(V)$
V_1	d_7	d_7	$r_4(d_6)$	$\{d_6, d_7\}$
V_2	d_8	$r_6(r_4(d_6))$	$r_8(d_4, d_5)$	$\{d_4, d_5, d_6\}$
V_3	d_8	$r_6(d_7)$	$r_7(d_{11})$	$\{d_7, d_{11}\}$
V_4	d_{11}	d_{11}	$r_6(d_{12})$	$\{d_{11}, d_{12}\}$
V_5	d_6	d_{12}	d_6	$\{d_6, d_{12}\}$

However, despite the correctness, the solution is not optimal. VG V_2 could be removed without affecting the validity of the solution. The set of blaming VDs would still be $\{d_6\}$ for VD d_6 and although VDs d_4 and d_5 would not be connected to the solution, it would still be valid because the monitoring of those VDs is infeasible nonetheless. However, the cost of the solution, which is 9 would reduce to 8, making the solution optimal for this instance. There exist several reasons for the sub-optimal solutions. We will discuss some of them in the following sub-section.

Table 5.2: The provided VDs of the solution with the set of witnessing VGs and the intersection of the blaming sets of those VGs.

VD	$\mathcal{V}_W(d, X)$	$\bigcap_{V \in \mathcal{V}_W(d,X)} \mathcal{D}_B(V)$
d_1	$\emptyset$	$\emptyset$
d_4	$\{V_2\}$	$\{d_4, d_5, d_6\}$
d_5	$\{V_2\}$	$\{d_4, d_5, d_6\}$
d_6	$\{V_1, V_2, V_5\}$	$\{d_6\}$
d_7	$\{V_1, V_3\}$	$\{d_7\}$
d_{11}	$\{V_3, V_4\}$	$\{d_{11}\}$
d_{12}	$\{V_4, V_5\}$	$\{d_{12}\}$

5.1 Quality of the Solutions

As described in the main section of this chapter, the returned GS is not optimal, when the algorithm is run on the example SD from section 3.1. VG V_2 is not necessary to monitor the VDs for which the monitoring is feasible. In this section, we will therefore discuss the reasons for the inclusion of the VG and other reasons for sub-optimal results. For this let's first take another look on the VGs returned by the algorithm. A graphical representation of the individual VGs can be seen in Figure 5.3 in which the VGs are numbered in order of addition to the solution during the execution. The algorithm searches for partial solutions that are valid for a particular VD. VD

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

d_6 is the first that actually is feasible to monitor and therefore the algorithm adds VG V_1 first as it is the cheapest VG. Then VG V_2 is added as it is the next cheapest thereafter. Through this step, the partial solution becomes valid for VD d_6 and hence no additional VGs have to be added to the solution. The solution is then extended with additional VGs for each VD. Finally, when considering VD d_{12} the algorithm adds VG V_5 which includes VD d_6 one more time, making VG V_2 redundant. The algorithm decides to include V_2 because of its lower cost of 3 units that is lower than the cost of 5 units necessary to include V_5. That V_5 is needed for another VD can not be anticipated beforehand.

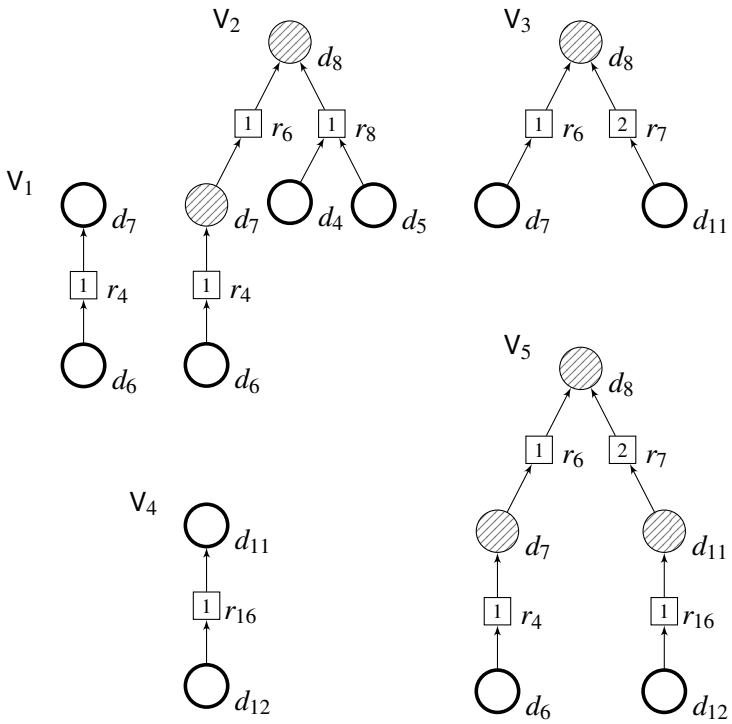

Figure 5.3: VGs of the calculated GS in Figure 5.2

Given these observations, it becomes apparent that the quality of the solution is dependent on the order of the provided VDs. For example, when run in the opposite order the solution would be optimal in the case of the robot model. The right order of the VDs can however not be determined before the execution as it would solve the problem before solving it and ultimately this approach would only improve certain corner-cases. Running on the reverse ordered VDs, the solution is only optimal because the algorithm picks r_6 before r_8 because of the lexicographical sorting in the data-structures. Due to the same cost of the transformation functions, r_8 could be picked as well, leading to a solution with a cost of 9 units, hence also the order of the VRs does influence the quality of the solution. The greedy nature of the algorithm when searching a solution will pick these local optima and is not able to leave them when a better solution would be achievable, by dropping an unnecessary VG.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

This greedy nature has also other negative effects on the quality of the solution, even if the order of the VDs and VRs is optimal by coincidence. For example, in the case of the SD presented in Figure 5.4, the solution returned by the algorithm would incur cost of 41 units. This solution could be optimized by changing the order of the VDs.

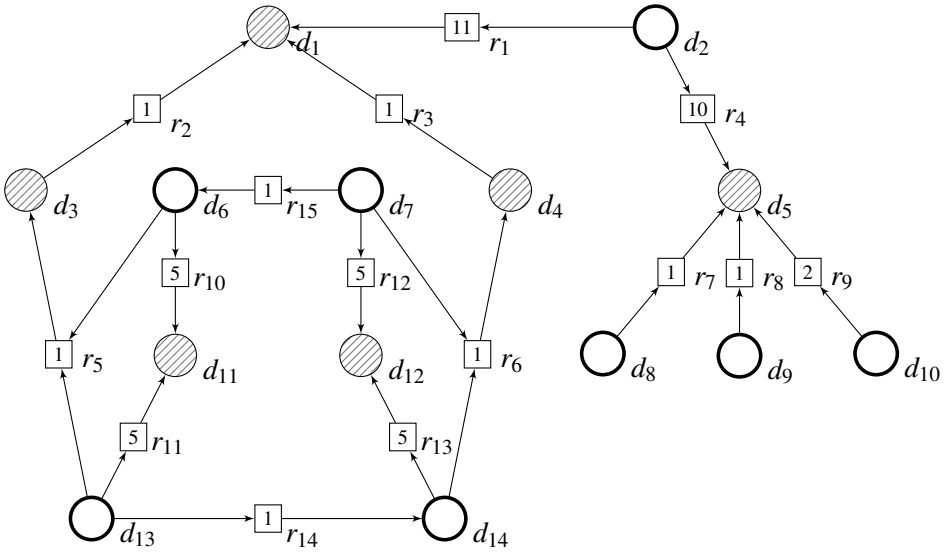

Figure 5.4: SD with a non optimal solution using the implementation. The values in the VRs indicate the cost of the transformations

This becomes apparent when looking at the graph representation of the solution that can be seen in Figure 5.5. The sum of the individual cost is 36, which is the minimal solution that can be achieved with the rearrangement of the VDs. The unnecessary VG compares the measurements of the VDs d_7 and d_{13} at d_{11}, despite of the fact that d_7 and d_{13} are compared to d_6 and d_{14} individually which makes the solution valid already. This unnecessary VG makes it necessary to transform the measurements from d_7 along r_{10} and thereby leads to the additional 5 units of cost.

Despite of the optimization via the change of the order of the VDs the solution is far from optimal as there exists another possible optimization. If the VR r_1 is added to the solution at the beginning of the algorithm, the resulting solution does only incur cost of 21 units. This can be seen in the graphical representation of the solution in Figure 5.6. The inclusion of VR r_1 makes it possible to create a solution without the relative expansive VRs r_{10}, r_{11}, r_{12} and r_{13}. Again, a local optimum, in this case choosing VR r_4 over VR r_1, leads to a solution that is nearly twice as expansive as the optimal solution.

The discussed observation are not surprising in any way, because of the NP-hardness of the problem. The implementation could however be improved to be more effective for certain constrained instances. Such improvements are however outside of the scope of this work and could be investigated as a future work. Further, it remains to determine whether there exists a polynomial time approximation algorithm for the problem and how close such an approximation

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

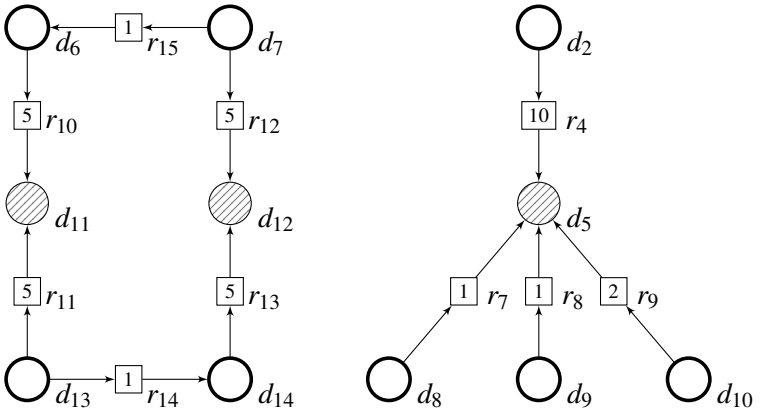

Figure 5.5: Sub-optimal GS of the SD in Figure 5.4 generated by the algorithm with cost 41.

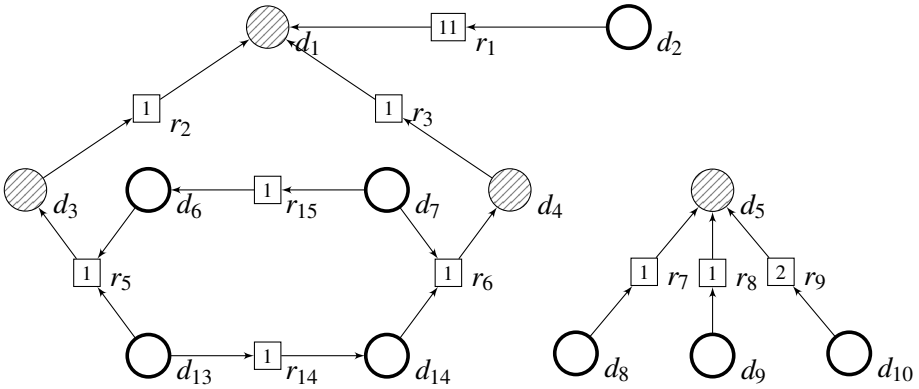

Figure 5.6: Optimal GS of the SD in Figure 5.4 with cost 21. r_1 was added manually at the beginning.

could come to an optimal solution. That our implementation is no such algorithm will be shown in the following section.

5.2 Complexity of the Implementation

In section 3.5, we showed that the problem of selecting sensor redundancies for monitoring is NP-complete. From this result and the definition of NP-hardness it follows that no implementation can compute an optimal solution in polynomial running time unless P=NP. Although there could be algorithms that approximate an optimal solution in polynomial running time, the implemented solution does not have this property. This can be seen in the running time of the example SD presented in Figure 3.2. Most of the 60 seconds running time is spent to find GSs that are valid for VDs d_4 and d_5 as removing d_4 leads to 28 seconds of running time and removing both leads to 0.5 milliseconds of running time. This behavior is caused by the fact that there are no valid GSs

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

for d_4 and d_5, hence the algorithm has to do an extensive search to verify that no solution exists.

In the worst case, what happens during the search is hard to describe using the algorithmic implementation due to its recursive nature, as many branches occur for every VD and VR during the search. The number of branches does depend solely on the number of inputs in the case of VRs and on the number of inputs and outputs in the case of VDs. They are bound in the number of VRs and VDs. As already mentioned, in the worst case the algorithm has to go through all possible MTs, the combination of those in the form of VGs and their combination in the GS until either no valid GS is found or the last GS that is checked is valid. The two cases are the same in terms of the number of constructed MTs and VGs. We will therefore only consider this number as it determines the running time in both cases.

The depth-first nature of the implementation would suggest that the problem can be solved for each VD in running time $\mathbf{T}(|\mathcal{P}|, |\mathcal{R}|) = \Theta(|\mathcal{P}| + |\mathcal{R}|)$ given a SD $\mathsf{S}(\mathcal{P}, \mathcal{R})$. However, in this running time only the root VDs for the VGs can be found. Then for each root VD d_r, VGs can be constructed using pairs of the MT that can be transformed to it. We defined this set of MTs in Equation 3.2 to be $\mathcal{M}(d, S)$. Using the definition, we can determine the number of measurements that can be used in a VD. It is the sum of the measurements that are transformed via its input VRs plus the measurement provided in the VD, if one is provided. For leaf-nodes the number is always one. Formally, given a SD $\mathsf{S}(\mathcal{P}, \mathcal{R})$ and VD $d \in \mathcal{D}$ the number of measurements is

$$\mathbf{M}(d, \mathsf{S}) = |\mathcal{M}(d, \mathsf{S})| = \overline{p}_d + \sum_{r \in \mathcal{I}_d} \mathbf{M}(r, \mathsf{S}). \tag{5.1}$$

The number of measurements that can be produced via the transformation function of a VR $r \in \mathcal{R}$ is the number of possible combination of the measurements in the input VDs i.e.

$$\mathbf{M}(r, \mathsf{S}) = |\mathcal{M}(r, \mathsf{S})| = \prod_{d \in \mathcal{I}_r} \mathbf{M}(d, \mathsf{S}). \tag{5.2}$$

From these two definitions it can be seen again that the number of branches does depend on the in- and out-degree of the VDs and VR of the SD. This number is hard to determine. We can however deduce a lower bound for the worst case. According to the definition the structure of a VG consisting of two MTs. One MT does contain the VD d the algorithm is working on and the other does not. We use the abstraction of only considering the provided VDs used in a MT to describe the MT. This is analogous to the line of reasoning on the number of possible VG in section 3.5. However, due to the fact that multiple MTs can be encoded to the same set of VDs the number of such descriptions represents a lower bound. The family of sets representing all possible measurements is again the power set of the provided VDs. The size of this set is $2^{|\mathcal{P}|}$ and hence $\mathbf{M}(d_{\text{root}}, S) = \Omega(2^{|\mathcal{P}|})$. Due to the two MTs in each VG, this family of sets has to be partitioned into the family of sets containing d and the ones that do not. The two families have the same size. This is trivial, as given the family without d we could construct the one with d by simply adding d to every set in the family. Therefore, the size of each family is $2^{|\mathcal{P}|-1}$ and the number of possible combinations of those MTs is $2^{|\mathcal{P}|-1} * 2^{|\mathcal{P}|-1} = 2^{2|\mathcal{P}|-2}$. In the worst case, a VD that is not feasible to monitor is connected to all VDs in a SD. Then for every VD in the set $\mathcal{D}$ $2^{2|\mathcal{P}|-2}$ VGs have to be checked. From this, it follows that the running time of the algorithm hence is

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

$\Omega(2^{|\mathcal{P}|})$. Due to the fact that each provided VD for which no valid GS exist represents such a worst case, the average case is not as important to the running time of the algorithm. Special restrictions such as a maximum in- or out-degree for VDs and VRs could make the problem more tractable, this is however not in the scope of this work.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 6

Conclusion

6.1 Summary

With this thesis, an approach to select sensor redundancies for monitoring and detection of sensor faults was presented. In the following paragraphs, the most important aspects of this approach are summed up.

First, an ontology and therefore a model of the problem instance was presented in the form of the System Description (SD), which can be used to describe sensors, their Value Domain (VD) and the connection of these via a Value Relation (VR). Based on these basic concepts, a model of a possible solution, a Grouping Solution (GS), was presented which consists of Measurement Transformations (MTs) and Voting Groups (VGs). The necessary properties of such a GS have been given and it was shown that a valid GS can be used to detect sensor faults.

Further, the problem of finding a valid GS for a given SD was defined as the computation theoretical decision problem **select sensors**. This decision problem was analysed and found to be in the complexity class NP via a membership proof. Additionally, a proof of NP-completeness via reduction from **vertex cover** has been given.

Next, an implementation in the form of an algorithm was presented and it was shown that the algorithm will produce valid GSs. In the evaluation of the algorithm, it was shown that the algorithm will generate sub-optimal GS in terms of cost. Further, it was shown that the worst case running time of the algorithm is exponential and that any not sufficiently connected sensor can represent such a worst case. For small SDs in the size of a few tens of VDs, the problem remains tractable however.

6.2 Future work

Due to the presented problems in the implementation the approach offers a basis for future research on the topic. This section gives an overview over future work.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

One topic that remains open is the inclusion of additional cost metrics into the cost function. Many sensors demand high amounts of memory or even computational resources to produce a measurement. If such a sensor can be turned off, the cost of operation of a CPS with a capped energy budget can be lowered.

As another topic which is related it could be determined how the frequency of measurements can be incorporated into the model of selecting sensor fault. If for example a sensor does only provide measurements of a certain VD in a frequency that is too low for an application, then it cannot be used as a monitoring method or phrased otherwise the influence of the frequency of the measurements on the monitoring could be researched. Analogously the variance and other stochastic properties of the measurements could be studied in relation to the quality of monitoring.

Another possible extension of this approach would be the addition of virtual sensors that provide measurements according to some prediction according to the view of the system. And further it would be interesting to determine whether such a predictive model could be generated simply from the knowledge in a SD.

Finally, an extended study on the complexity of the problem would be another topic. It could be determined if the problem could be restricted in some way such that it is in the class P and therefore maybe usable in real-time. Further, an approximate algorithm that is in the class P could be researched or whether such an algorithm exists.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Tables

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Algorithms

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acronyms

AI Artificial Intelligence. 7

AuR Autonomous Robotics Subgroup. 14

CPS Cyber Physical System. ix, xi, 1, 2, 5, 6, 14, 17–19, 21, 45, 56, 68

DAG directed acyclic graph. 24–27, 42

GS Grouping Solution. ix, xi, 17, 30–56, 60–62, 64–67, 69, 71, 73

MT Measurement Transformation. 24, 26–33, 37–40, 42, 44–56, 61, 65, 67, 69, 71, 73

MTTF Mean Time To Failure. 6

OWL Web Ontology Language. 7

RDF Resource Description Framework. 7

RDFS Resource Description Framework Schema. 7

SD System Description. 17, 20–28, 30, 31, 33, 39, 40, 42–53, 56, 59–61, 63–65, 67–69

SLAM Simultaneos localization and mapping. 20

VD Value Domain. 18–27, 29–53, 57, 60–69, 71, 73

VG Voting Group. 17, 27, 28, 30–44, 46, 48–54, 56, 57, 60–63, 65, 67, 69, 71

VOWL Visual Notation for OWL Ontologies. 7

VR Value Relation. 18–25, 27, 39, 42–45, 47–53, 55, 56, 59, 60, 62, 63, 65–67, 69, 73

XML eXtended Markup Language. 7

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

[1] Rolf Isermann. *Fault-Diagnosis Systems*. Springer Berlin Heidelberg, 2006.

[2] Jean-Claude Laprie. *Dependability: Basic Concepts and Terminology*. Springer Vienna, 1992, p. 3.

[3] Debanjan Ghosh et al. „Self-Healing Systems Survey and Synthesis“. In: *Decision Support Systems* 42.4 (2007), pp. 2164–2185.

[4] Rudi Studer, Richard Benjamins, and Dieter Fensel. „Knowledge Engineering: Principles and Methods“. In: *Data & Knowledge Engineering* 25.1-2 (1998), pp. 161–197.

[5] Simon Blackburn. *Ontology*. In: *The Oxford dictionary of philosophy*. Oxford University Press, 1996, p. 270.

[6] Enrique Martí, Jesús García, and José M. Molina. „Adaptive sensor fusion architecture through ontology modeling and automatic reasoning“. In: *2015 18th International Conference on Information Fusion (Fusion)*. 2015, pp. 1144–1151.

[7] Franz Baader, Ian Horrocks, and Ulrike Sattler. „Description Logics“. In: *Handbook on Ontologies*. Springer Berlin Heidelberg, 2009, pp. 21–43.

[8] Grigoris Antoniou and Frank van Harmelen. „Web Ontology Language: OWL“. In: *Handbook on Ontologies*. Springer Berlin Heidelberg, 2009, pp. 91–110.

[9] Steffen Lohmann et al. „VOWL 2: User-Oriented Visualization of Ontologies“. In: *Proceedings of the 19th International Conference on Knowledge Engineering and Knowledge Management (EKAW '14)*. Vol. 8876. LNAI. Springer, 2014, pp. 266–281.

[10] Jérôme Euzenat and Pavel Shvaiko. *Ontology Matching*. Springer Berlin Heidelberg, 2013.

[11] Karel Hrbacek and Thomas Jech. *Introduction to Set Theory, Third Edition, Revised and Expanded*. CRC Press, Marcel Dekker, 1999.

[12] Richard A. Brualdi. *Introductory Combinatorics, 5th Edition*. Pearson, 2009.

[13] Reinhard Diestel. *Graph theory*. Springer, 2017.

[14] Adrian Bondy and M. Ram Murty. *Graph Theory*. Springer, 2008.

[15] Aydın Buluç et al. „Recent Advances in Graph Partitioning“. In: *Algorithm Engineering: Selected Results and Surveys*. Ed. by Lasse Kliemann and Peter Sanders. Springer International Publishing, 2016, pp. 117–158.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[16] Charles-Edmond Bichot and Patrick Siarry. *Graph Partitioning*. John Wiley & Sons, Inc., 2013.

[17] Jørgen Bang-Jensen and Gregory Z. Gutin. *Digraphs*. Springer Monographs in Mathematics. Springer London, 2009, pp. 32–34.

[18] Sanjeev Arora. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[19] Thomas Cormen. *Introduction to algorithms*. MIT Press, 2009.

[20] Robert Sedgewick and Kevin Wayne. *Algorithms Part 1*. Addison-Wesley, 2014.

[21] Christos Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[22] Robert Sedgewick and Kevin Wayne. *Algorithms Part 2*. Addison-Wesley, 2014.

[23] Donald Knuth. *The Art of Computer Programming, Volume Three: Sorting and Sarching*. 2nd ed. Addison-Wesley, 1997.

[24] Python Software Foundation. *CPython Set Implementation*. 2019. URL: `https://github.com/python/cpython/blob/v3.7.4/Objects/setobject.c` (visited on 08/27/2019).

[25] Yury Selivanov. *PEP 492 – Coroutines with async and await syntax*. 2015. URL: `https://www.python.org/dev/peps/pep-0492/` (visited on 08/27/2019).

[26] Donald Knuth. *The Art of Computer Programming, Volume One: Fundamental Algorithms*. 3rd ed. Addison-Wesley, 1997.

[27] Neil Schemenauer, Tim Peters, and Magnus Lie Hetland. *PEP 255 – Simple Generators*. 2001. URL: `https://www.python.org/dev/peps/pep-0255/` (visited on 08/27/2019).

[28] Barbara Liskov. „A History of CLU“. In: *History of Programming languages—II*. Ed. by Thomas J. Bergin Jr. and Richard G. Gibson Jr. ACM, 1996, pp. 471–510.

[29] Stephan Murer et al. „Iteration Abstraction in Sather“. In: *ACM Transactions on Programming Languages and Systems* 18.1 (1996), pp. 1–15.

[30] Behzad Bayat et al. „Requirements for Building an Ontology for Autonomous Robots“. In: *Industrial Robot: An International Journal* 43.5 (2016), pp. 469–480.

[31] Oliver Höftberger and Roman Obermaisser. „Ontology-based runtime reconfiguration of distributed embedded real-time systems“. In: *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)*. 2013.

[32] Denise Ratasich et al. „A Self-Healing Framework for Building Resilient Cyber-Physical Systems“. In: *IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC)*. 2017.

[33] Oliver Höftberger. „Knowledge-based dynamic reconfiguration for embedded real-time systems“. PhD thesis. 2015.

[34] Denise Ratasich et al. „Self-healing by property-guided structural adaptation“. In: *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*. 2018.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[35] Abhishek B. Sharma, Leana Golubchik, and Ramesh Govindan. „Sensor Faults: Detection Methods and Prevalence in Real-World Datasets“. In: *ACM Transactions on Sensor Networks (TOSN)* 6.3 (2010), p. 23.

[36] Vandi Verma et al. „Real-Time Fault Diagnosis“. In: *IEEE Robotics & Automation Magazine* 11.2 (2004), pp. 56–66.

[37] Iskander Boulaabi, Anis Sellami, and Fayçal Ben Hmida. „Robust Delay-Derivative-Dependent Sliding Mode Observer for Fault Reconstruction : A Diesel Engine System Application“. In: *Circuits, Systems, and Signal Processing* 35.7 (2015), pp. 2351–2372.

[38] Alejandra Ferreira de Loza et al. „Sensor Fault Diagnosis Using a Non-Homogeneous High-Order Sliding Mode Observer With Application To a Transport Aircraft“. In: *IET Control Theory & Applications* 9.4 (2015), pp. 598–607.

[39] Ofir Cohen and Yael Edan. „A Sensor Fusion Framework for Online Sensor and Algorithm Selection“. In: *Robotics and Autonomous Systems* 56.9 (2008), pp. 762–776.

[40] Puneet Goel et al. „Fault detection and identification in a mobile robot using multiple model estimation and neural network“. In: *Robotics and Automation, 2000. Proceedings. ICRA '00.* Vol. 3. 2000, pp. 2302–2309.

[41] M. Hashimoto, H. Kawashima, and F. Oba. „A multi-model based fault detection and diagnosis of internal sensors for mobile robot“. In: *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings.* Vol. 4. 2003, pp. 3787–3792.

[42] Kurt Mehlhorn and Peter Sanders. *Algorithms and Data Structures*. Springer Berlin Heidelberg, 2008, p. 52.

[43] Jampala Madhusudana Rao. „Products of Sets: Ordered and Unordered“. In: *Resonance* 21.6 (2016), pp. 557–564.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN