


Diagnosis System Conceptual Design Utilizing Structural Methods – Applied on a UAV's Fuel System

Master's thesis performed at:
Division of Vehicular System
Department of Electrical Engineering
Linköpings Universitet

Tobias Axelsson

Reg nr: LiTH-ISY-EX-3552-2004

Supervisors: Lic.Eng. Mattias Krylander, Division of Vehicular Systems,
LiTH
Lic.Eng. Martin Jareland, Saab AB
Examiner: Assistant Prof. Erik Frisk, Department of Electrical
Engineering, LiTH

	Avdelning, Institution Division, Department	Datum Date 2004-08-26
	Institutionen för Systemteknik 581 83 LINKÖPING	

Språk Language Svenska/Swedish X Engelska/English	Rapporttyp Report category Licentiatavhandling X Examensarbete C-uppsats D-uppsats Övrig rapport _____	ISBN ISRN LITH-ISY-EX-3552-2004 Serietitel och serienummer ISSN Title of series, numbering _____
URL för elektronisk version http://www.ep.liu.se/exjobb/isy/2004/3552/		

Titel Title Författare Author	Användande av strukturella metoder vid design av koncept till diagnosystem - Tillämpat på bränslesystemet i en UAV. Diagnosis System Conceptual Design Utilizing Structural Methods – Applied on a UAV's Fuel System Tobias Axelsson
--	---

Abstract <p>To simplify troubleshooting and reliability of a process, a <i>diagnosis system</i> can supervise the process and alarm if any faults are detected. A diagnosis system can also identify one, or several faults, i.e. <i>isolate faults</i>, that may have caused the alarm. If model-based diagnosis is used, tests based on observations from the process are compared to a model of the process to diagnose the process. It can be a hard task to find which tests to be used for maximal fault detection and fault isolation. <i>Structural Methods</i> require not very detailed knowledge of the process to be diagnosed and can be used to find such tests early in the design of new processes. Sensors are used to get observations of a process. Therefore, sensors placed on different positions in the process gives different possibilities for observations. A specific set of sensors are in this work called a <i>sensor configuration</i>.</p> <p>This thesis contributes with a method to predict and examine the fault detection and fault isolation possibility. By using these two diagnosis properties, a suitable sensor configuration is computed and tests to be used in a future diagnosis system are suggested. For this task an algorithm which can be used in the design phase of diagnosis systems, and a Matlab implementation of this algorithm are described.</p> <p>In one part of this work the Matlab implementation and the algorithm are used to study how a model-based diagnosis-system can be used to supervise the fuel system in an Unmanned Aerial Vehicle (UAV).</p>

Nyckelord model-based diagnosis, sensor configurations, structural methods, fuel system, MSS sets

Abstract

To simplify troubleshooting and reliability of a process, a *diagnosis system* can supervise the process and alarm if any faults are detected. A diagnosis system can also identify one, or several faults, i.e. *isolate faults*, that may have caused the alarm. If model-based diagnosis is used, tests based on observations from the process are compared to a model of the process to diagnose the process. It can be a hard task to find which tests to be used for maximal fault detection and fault isolation. *Structural Methods* require not very detailed knowledge of the process to be diagnosed and can be used to find such tests early in the design of new processes. Sensors are used to get observations of a process. Therefore, sensors placed on different positions in the process gives different possibilities for observations. A specific set of sensors are in this work called a *sensor configuration*.

This thesis contributes with a method to predict and examine the fault detection and fault isolation possibility. By using these two diagnosis properties, a suitable sensor configuration and tests to be used in a future diagnosis system are computed. For this task an algorithm which can be used in the design phase of diagnosis systems, and a Matlab implementation of this algorithm are described.

In one part of this work the Matlab implementation and the algorithm are used to study how a model-based diagnosis-system can be used to supervise the fuel system in an Unmanned Aerial Vehicle (UAV).

Acknowledgements

This master's thesis was performed during the spring and the summer 2004 at the Department of Simulation and Thermal Analysis (TDGT), Saab Aerosystems, Saab AB and at the Division of Vehicular System, Linköpings University.

I would like to thank a number of people for supporting me during this work:

My supervisors Martin Jareland (Saab AB) and Mattias Krylander (LiTH), thank you for guidance, help and discussions.

Birgitta Lantto (Saab AB) and Erik Frisk (LiTH) for making this thesis possible.

My colleagues at Saab AB and at Division of Vehicular System for all support and for a great time during and between the coffee breaks.

I would also like to thank my Family and numbers of friends which have encourage and supported me during the work of this thesis and during my years at LiTH.

1	Introduction	1
1.1	Background	1
1.2	Objectives	2
1.3	Outline	2
2	Introduction to Fault Diagnosis	5
2.1	Basic Definitions	5
2.2	The History of Fault Diagnosis	6
2.2.1	Limit Checking	7
2.2.2	Hardware Redundancy	7
2.3	Use of Diagnosis	8
2.3.1	Man and Machine Protection	8
2.3.2	Availability and Cost Reduction	8
2.4	Model-Based Diagnosis	9
2.4.1	Structure of Model-Based Diagnosis-Systems	9
2.4.2	Advantages of Model-Based Diagnosis	10
2.5	Structural Methods	10
2.5.1	Introduction to Structural Methods	10
2.5.2	Product Development Process utilizing Structural Methods	11
3	Modeling Methods	13
3.1	Introduction to the Modeling Methods	13
3.2	Structural Models	14
3.2.1	Structural Model with Analytical Model Available	14
3.2.2	Structural Model Without Analytical Model Available	15
3.3	Study of the Refueling Process in a Conceptual UAV	15
3.3.1	Example Description	16
3.3.2	Included Variables	16
3.3.3	Equations used in the Model	18
3.3.4	Structural Model	23
3.4	Introduction to MSS Sets	24
3.4.1	Structural Singular	25
3.4.2	Minimal Structural Singular (MSS)	25
3.4.3	The Use of MSS Sets	26
4	Algorithm used to find MSS Sets	27
4.1	Differentiate the Model	28
4.1.1	Example of a Differentiated Model	30
4.2	Simplify the Model	32
4.3	Search for MSS sets	33
4.4	Analysis of Isolability	34
4.5	Decouple Faults	36
4.6	Summary of the Structural Algorithm	36

5	Optimizing Sensors Configurations	39
5.1	Fault Classification	39
5.1.1	Properties of Fault Classification	40
5.1.2	Demands for the Fault Classification	41
5.2	Sensor Configurations	41
5.2.1	Sensor Configuration Optimization	42
5.3	Algorithm used to Examine Sensor Configurations	42
5.4	Optimization Strategies using a Fault Isolability Matrix	44
6	Matlab Implementation	45
6.1	Graphic User Interface	45
6.1.1	Definition of Variables	45
6.1.2	Definition of Equations	46
6.2	Objects representing Structural Models and Isolability Matrices	48
6.2.1	SM Objects	48
6.2.2	SMSS Objects	50
6.2.3	FM objects	51
6.3	Functions used in the Matlab Implementation	51
6.3.1	Basic Functions for the MSS Algorithm	51
6.3.2	Functions used to Merge and Change Structural Models	52
6.3.3	Functions for Visualization	52
6.3.4	Functions for Analysis of MSS sets	53
6.4	Utilizing Matlab Implementations for Structural Analysis	53
7	UAV Fuel System Concept	57
7.1	Introduction to Conceptual UAV	57
7.1.1	The Fuel Pump System	58
7.1.2	The Tank Pressurization System	59
7.2	Structural Analysis Strategy	61
7.2.1	Modeling Conditions	61
7.3	Model of the Fuel Pump System	62
7.3.1	Models of the Tanks	63
7.4	Structural Model of the Fuel Pump System	65
7.4.1	Limitations in the Structural Analysis	65
7.4.2	Unknown Variables	66
7.4.3	Sensor Signals	66
7.4.4	Fault Variables	68
7.4.5	Control Signals	69
7.5	System Equations	70
7.5.1	Control Signals Included in the System Equations	71
7.5.2	Faults Included in the System Equations	72
7.5.3	Perfect Matching	74

7.5.4	Sensor Equations	75
7.5.5	Fault Model Equations	77
7.6	Analysis of Sensor Configurations	78
7.6.1	Sensor Classification	79
7.6.2	Fault Classification	80
7.6.3	Evaluation of Sensor Configurations	83
7.6.4	Conclusions related to Normal Flight Mode	87
7.7	Summary of the Structural Analysis	88
8	Discussion and Conclusions	89
8.1	Discussion	89
8.1.1	Discussion Related to the Matlab Implementation	89
8.1.2	Discussion Related to Structural Analysis	90
8.2	Conclusions	90
8.3	Future Work	91
	Bibliography	93
	Appendix A	95
	Appendix B	107

Introduction

This master's thesis has been carried out in cooperation with Saab AB. Saab AB is one of the world's leading high-technology companies, with its main operations focusing on defence, aviation, and space. The company is active both in civil and military industry. This thesis is performed at Saab Aerosystems in Linköping Sweden at the Department of Simulation and Thermal Analysis of General Systems.

1.1 Background

Today many technical processes have one or more *diagnosis systems*. A diagnosis system can supervise a process and alarm if a fault appears. It is also common that diagnosis systems can identify and point out one, or several faults. Modern processes do often have a high complexity and diagnosis systems make troubleshooting easier when a process has failed. It is a very complicated and time demanding task to design a diagnosis system. Obviously it is desirable to construct tools to simplify and automate this assignment.

Mattias Kryanders Licentiate thesis "Design and Analysis of diagnosis Systems Utilizing Structural Methods", which can contribute to this research area,

was presented in 2003 [1]. In his work Krysander describes among others an algorithm to analysis the structure of the processes to be diagnosed. The algorithm is based on graph theory and has also been implemented in Matlab to allow studies and research of large models. The purpose with this method is to find key relations in a process that can be used to derive tests with a high diagnosis capability.

1.2 Objectives

The principal aims with this master thesis are:

- To present a method utilizing structural methods in the early design phase of new products, to simplify and improve design of diagnosis systems.
- To develop a Matlab implementation which can simplify the use of the algorithms and methods used in this work. Since many algorithms already have been implemented in Matlab most of this work aims towards finding a user orientated interface and complement the existing core with new functionality.
- To perform a structural analysis on the fuel system in an *Unmanned Aerial Vehicle* (UAV) concept to show how structural methods can be used to predict the isolability possibilities for a future diagnosis system.

The Expectations on this thesis are that the reader gets a view over how structural analysis can be used to improve the development of new processes.

1.3 Outline

The work in this thesis will be presented as follows:

Chapter 2 is an introduction to the subject diagnosis where also some benefits of Structural Analysis are described.

Chapter 3 is an introduction to the modeling framework which is used in the thesis. There is also an example which shows a part of the process, when a structural analysis is performed.

Chapter 4 briefly describes an algorithm used to find key relations between variables. The algorithm is taken from Mattias Krysanders Lic thesis “Design

and Analysis of diagnosis Systems Utilizing Structural Methods” which can be studied for a full description.

Chapter 5 describes a Method which can be used to evaluate which conditions different sensor configurations gives for a future diagnosis system. As a part of this process a framework which can be used to compare different sensor solutions is introduced.

Chapter 6 is an introduction and description of the Matlab implementations which has been put together to simplify the work with Structural Analysis.

Chapter 7 describes how Structural Analysis can be used to determine the possibilities for a future diagnosis system in the fuel system of a UAV concept.

In **Chapter 8** some conclusions and possibilities for future work are presented.

Introduction

Introduction to Fault Diagnosis

This chapter is an introduction to fault diagnosis topics which are handled in this thesis. It also provides some common definitions which are used later in this work.

2.1 Basic Definitions

To simplify the description of Fault Diagnosis it is necessary to introduce some basic definitions [2]:

- **Fault**
Unpermitted deviation of at least one characteristic property or variable of the system from acceptable/usual/standard/nominal behavior.
- **Failure**
A fault that implies permanent interruption of a systems ability to perform a required function under specified operating conditions.
- **Disturbance**
An unknown and uncontrolled input acting on a system.

- **Fault Detection**
To determine if one or several faults are present in the system and usually also to determine when the present faults have occurred.
- **Fault Isolation**
Determination of the location of a present fault, e.g. which component or components that have failed.
- **Diagnosis**
Diagnosis systems produce *diagnoses*. A diagnosis is a conclusion of what faults that can explain the present process behavior, if the process behavior diverges from the normal behavior.
- **Active Diagnosis**
When the diagnosis is performed by actively exciting the system so that possible faults are revealed.
- **Passive Diagnosis**
When the diagnosis is performed by passively studying the system without affecting its operation.
- **Consistency Relations**
A consistency relation is any relation between known variables that, in the fault free case, always holds.

2.2 The History of Fault Diagnosis

Modern systems often have computers for control, but the computers can also be used to record and evaluate data about running processes. This data can then be used to decide if the process is running normally or if there are any present faults in the process. Such information can be valuable for *safety reasons*, e.g. to avoid or immediately detect faults which can result in serious damages to humans, nature, or equipment. Faults can also be detected before they are serious enough to prevent a process to fulfil a task, e.g. a degraded bearing can be detected before it break down by detecting disturbances in the friction. This can be used to optimize maintenance by replacing components in a system just when it is necessary instead of replacing them according to a maintenance plan.

A support system that gives possible explanations to which fault that has occurred is called a *diagnosis system*. Diagnoses from the diagnosis system can be used to simplify repair by shorten the time for troubleshooting. Figure 2.1 shows the general structure of a diagnosis application. The diagnosis system takes observations of the process to be diagnosed and computes diagnoses by comparing expected behaviors with the expected behavior. The process can

be influenced by control signals, disturbances and faults. If the diagnosis system is correctly designed, it can deliver a diagnosis which tells if any fault has occurred in the process to be diagnosed.

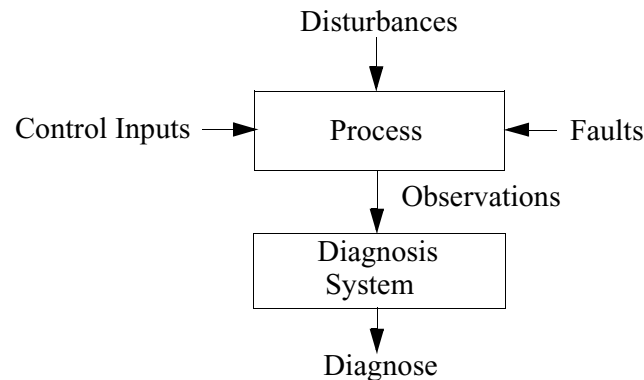


Figure 2.1: Structure of a diagnosis application.

2.2.1 Limit Checking

Traditionally diagnosis of technical systems has been performed by *limit checking*. Limit checking means that an alarm is generated when a signal leaves its normal operating range. The normal range is here predefined and the limits must be chosen according to a worst case scenario or different limits must be used for different operating conditions. This implies to that some faults are not discovered during normal operating conditions. There are also faults which just can be detected as abnormal conditions between different values, e.g. if the temperature in an engine is close to the maximum allowed temperature when it is running at 10% of its capability no alarm is generated from a limit check, despite that probably something is wrong with e.g. the cooling system. Another disadvantage with this method is the lack of knowledge about how different faults affect the system, which makes it hard to isolate a present fault.

2.2.2 Hardware Redundancy

In aircraft *hardware redundancy* is common, hardware redundancy means that some important components are duplicated or even triplicated. For example two or more sensors can be used to measure the same quantity. Hardware redundancy is easy to implement and can also be necessary in some processes

for safety or legal reasons. Three problem areas with hardware redundancy is higher weight, higher space demands and higher costs for hardware. However hardware redundancy contributes with big opportunities to construct a solid diagnosis system since many test quantities are measured.

2.3 Use of Diagnosis

Today diagnosis systems are used in many different areas e.g. vehicles and process facilities. Here follows some applications where diagnosis systems are used:

- Power plants
- Aircraft including all sub-systems
- Industrial robots
- Process facilities

Two main reasons to incorporate diagnosis systems are *Man and Machine protection* and *Availability* which are discussed in the next two sections.

2.3.1 Man and Machine Protection

A fault in a process can sometimes cause damage both to the process and to associated humans and the nature. Man and Machine protection is especially important in safety critical systems like nuclear power plants and aircraft. In this type of systems it is important that faults are detected very quickly. In best cases some faults can be predicted and avoided. For example in automobiles a diagnosis system can detect a fault in the brakes, *Anti Blocking System* (ABS) and alarm. This type of fault is often not detected without a diagnosis system and can then cause or aggravate accidents.

2.3.2 Availability and Cost Reduction

Due to a long startup time it is obvious that some applications like power plants or paper mills must be running continuously. Today it is a common trend that also other systems like for example trucks, aircraft and robots are supposed to run more or less continuously, it is then desirable to have a diagnosis system which can isolate and point out faults that occurs, to simplify troubleshooting. Since processes often have to be stopped during service it is also desirable that the diagnosis system can help to decide what type of maintenance to be done during a planned stop to avoid future failures and unnecessary maintenance. Without this type of system, maintenance must be done

more frequently due to that the maintenance intervals must short to prevent failures and unplanned stops.

2.4 Model-Based Diagnosis

As an alternative to traditional approaches like e.g. limit checking, *model-based diagnosis* have shown to be useful [2]. A model-based diagnosis system compares a process actual behavior with different models of the process like e.g. a model for the normal process and models which includes different faults in the process. The models used can for example be differential equations or logic based models.

2.4.1 Structure of Model-Based Diagnosis-Systems

If the diagnosis system detects that the actual behavior of a process to be diagnosed deviates from the expected behavior estimated from a model of the fault free process, an alarm is generated. By also including information of different fault behaviors in the diagnosis system it is possible to find one or several possible explanations for the actual behavior, which then can be used to explain which fault that caused the alarm.

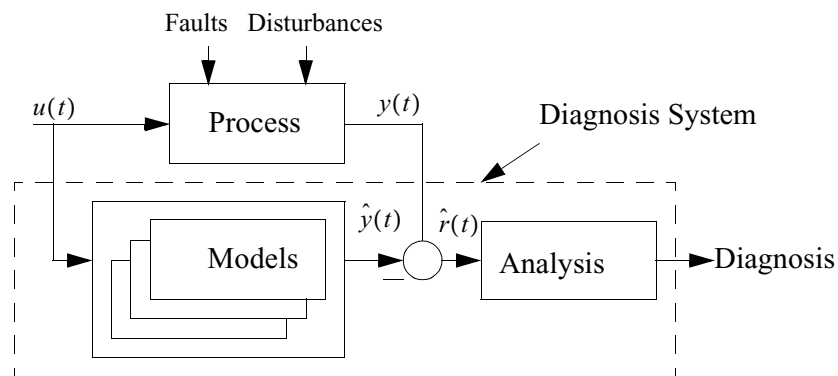


Figure 2.2: Principle of model-based diagnosis.

Figure 2.2 shows a general structure for a model-based diagnosis-system. In Figure 2.2 the process is controlled by a control signal $u(t)$, and the output signal is $y(t)$. The diagnosis system includes models of the process, the fault free model and models for the process with different faults included. The models can be used to predict the output $y(t)$ by using $u(t)$. The predictions of $y(t)$ is a

vector denoted $\hat{y}(t)$. By analyzing deviations $\hat{r}(t)$ between $y(t)$ and $\hat{y}(t)$ from the model of the fault free process, a fault can be detected. As long as the behavior of the process matches the behavior of the model of the fault free process no alarm is generated, but if a fault occurs it can be isolated and announced by finding the model corresponding to the present fault. This since $\hat{y}(t)$ and $y(t)$ are similar when the model corresponding to the actual process behavior is chosen.

2.4.2 Advantages of Model-Based Diagnosis

Model-based diagnosis has advantages compared to traditional methods like e.g. limit checking. Model-based diagnosis can be performing over a large operating range, without defining worst case limits. This improves the diagnosis performance and smaller faults can be detected. Model-based diagnosis needs no extra hardware and can be applied to more kinds of component than hardware redundancy. A disadvantage with model-based diagnosis is the need for reliable models of the process to be diagnosed. The design procedure of the diagnosis system might also be very complicated and time demanding, if model-based diagnosis is to be used.

2.5 Structural Methods

The *Structural Methods* used in this thesis aims to simplify the analysis task, during use of models for diagnosis purposes. Structural methods focus on that there is a relation between variables, instead of examine the analytical properties of the relation.

2.5.1 Introduction to Structural Methods

Structural Methods can be used instead of exact models and simulations during the early design phase of a new product. Structural methods use a special type of model for the process. This type of model is called a *Structural Model* and contains only which variables that are included in each equation, in order to find *elimination schemes*. Elimination schemes are used to eliminate unknown variables to derive *overdetermined equation systems*. These overdetermined equations can then be used to derive *consistency relations* which can be used to implement tests in a diagnosis system, see e.g. [5]. Consistency relations are relations between known and measured variables that in the fault free case, always holds.

2.5.2 Product Development Process utilizing Structural Methods

To be able to start the design of the diagnosis systems early in the design phase of a new process, the design of the diagnosis system cannot be based on a detailed model of the final process concept. Figure 2.3 shows how the total development time can be shortened by starting the design of the diagnosis system early in the product development. If the diagnosis aspects not are considered during the early design phase. It can in a worst case scenario be necessary to redesign the product or parts of the product in which processes are to be diagnosed.

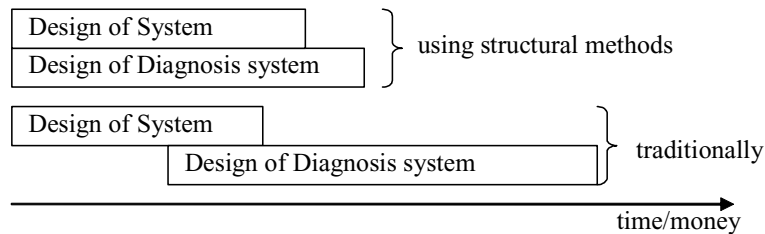


Figure 2.3: Product development utilizing structural methods.

Structural Methods is a solution to these problems since it can be used early in the design phase. Since the product development time then can be shorten, money is to be saved. Figure 2.4 shows how structural methods can be used in the product development process of products which need a diagnosis system. When a concept is obtained a structural analysis can be used to predict the diagnosis possibilities utilizing the suggested concept. This analysis can be used improving the concept, to prevent expensive modifications later in the development process.

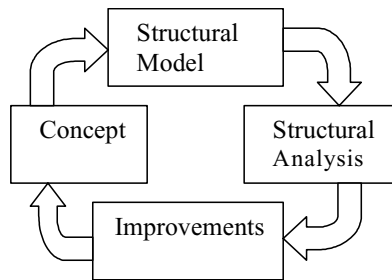


Figure 2.4: Product development utilizing structural methods.

Modeling Methods

Since models fill a main part in this thesis this chapter will briefly describe different aspects of structural and analytical models. There is also a short introduction to a specific type of key relations which can be obtained from structural models.

3.1 Introduction to the Modeling Methods

The behavior of a process depends on in which mode the process is running, e.g. “flying” or “refueling”. For model-based diagnosis it is therefore important to have an accurate model of the process for each mode. If a fault appears it can affect the process in different ways. To each fault a corresponding behavior mode is defined. Examples of behavior modes can be e.g. no-fault mode and sensor fault mode. The behavioral modes and their corresponding behaviors are in this work described with a diagnosis model [1]. This model consists of five different parts $\{M, X, Y, F, B\}$, which are described in Table 3.1.

Table 3.1: Example of a diagnosis model.

Name	Description	Example
M	set of all available equations	$M = \{e_1, e_2, e_3, e_4\} = \dots$ $\{y_1 = a_1x_1 + f_1, x_1 = a_3, \dots$ $y_2 = a_2x_2 + f_2, x_2 = a_4\}$
X	all unknown variables, e.g. internal states	$X = \{x_1, x_2\}$
Y	all known variables, e.g. sensor and control signals	$Y = \{y_1, y_2\}$
F	all fault variables, e.g. leakages or disturbances caused by faults	$F = \{f_1, f_2\}$
B	set of behavioral modes constants	$B = 0$ (no fault) a_1, a_2

3.2 Structural Models

In a structural model the analytical equations are replaced by the knowledge of which variables that are included in each equation. Structural models can then be represented by an *incidence matrix*. An incidence matrix is a matrix where the rows corresponds to the equations and the columns corresponds to the variables in the model. If variable j is included in equation i , position (i,j) in the incidence matrix is marked with an X. In Table 3.2 the incidence matrix corresponding to the model described in Table 3.1 is shown.

Table 3.2: Incidence matrix corresponding to the example in Table 3.1.

	x_1	x_2	y_1	y_2	f_1	f_2
e_1	X		X		X	
e_2	X					
e_3		X		X		X
e_4		X				

3.2.1 Structural Model with Analytical Model Available

It is simple to derive a structural model from an available analytical model. It is just to replace the analytical equations with structural equations. The struc-

tural model obtained can then be used to find consistency relations in order to design a diagnosis system.

3.2.2 Structural Model Without Analytical Model Available

Structural models are far less detailed compared to analytical models, e.g. values of constants are not necessary for a structural model. This implicates that no simulation work is necessary and therefore structural models can be obtained much earlier in the design phase. In diagnosis system design a structural model can be used to perform an early *isolability* analysis, which means an analysis of which faults that can be isolated. This analysis can be performed with only little information about the process available. The structural model used can be obtained using known insights about which variables that have to be included in each equation through physical relations or through previous experiences. If the process to be diagnosed includes several similar components a structural model for one of these components can be used for all of them.

3.3 Study of the Refueling Process in a Conceptual UAV

A concept study describing a part of a UAV during refueling is now used to show how a structural model can be obtained without any analytical model available. This example describes one wing tank during refueling and is an introduction to the full UAV study which is performed in Chapter 7. Figure 3.1 shows a schematic view of the wing tank. The upper unit in Figure 3.1 is the wing tank from its upside and the lower unit is the ventilation system. Only the units used in the refueling process are shown. During refueling, fuel is pressed into the tank through the refueling pipe and the refueling valve, while the air in the tank is ventilated through the ventilation system. Five sensor are used during the refueling. These are two pressure sensors one in the wing tank and one in the ventilation tank, two *fuel probes*, which are sensors that measure the fuel level in the wing tank and one fuel sensor which indicates if it is fuel in the ventilation system.

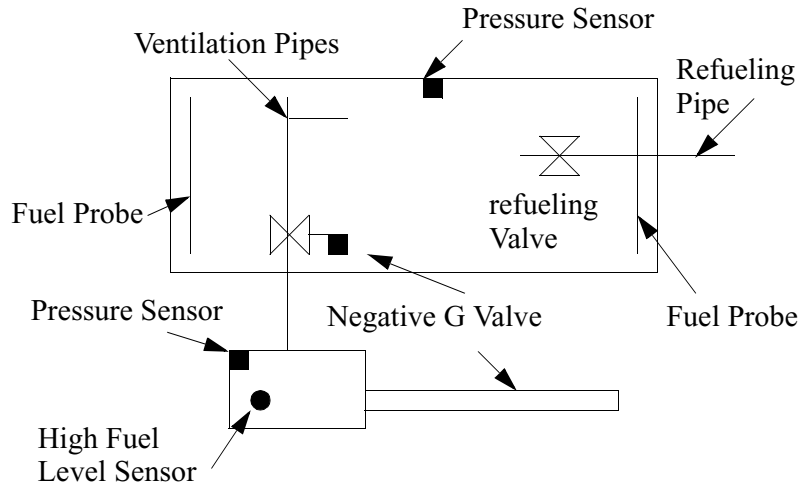


Figure 3.1: An Example of a wing tank in a UAV.

3.3.1 Example Description

Fuel is pressed into the tank through the refueling valve in the right part of Figure 3.1. At the same time air flows out to the ambient air through the ventilation pipes. Two fuel probes are used to measure the fuel level in the tank and the high level sensor indicates if it is fuel in the ventilation system. The fully mechanical negative-g valve in Figure 3.1 is placed in the top if the wing tank and closes if it is exposed to negative-g values, to prevent that fuel flowing into the ventilations pipes e.g. during flight upside-down. In this example there are also two pressure sensors one measures the pressure in the wing tank and one the pressure in the ventilation system.

3.3.2 Included Variables

All variables in X, Y and F in $\{M, X, Y, F, B\}$ that are used to describe the process shown in Figure 3.1 are described in Table 3.3.

Unknown Variables

The unknown variables, X included in this example are the fuel level in the tank, the fuel level in the ventilation system, the air pressure in the tank, the air pressure in the ventilation tank, the air pressure in the ambient air and the

fuel flow into the tank. These variables all represents physical quantities in the model.

Known Variables

All sensor and control signals are known variables, Y in this example.

Fault Variables

Totally 10 different faults typical for this type of processes are included in F . A fault variable is assumed to be zero in absence of the corresponding fault. Notice that some abnormal fuel flows like e.g. f_{OFT} which is a fuel flow through a ventilation pipe, are considered as fault variables instead of unknown variables.

Table 3.3: Variables used to describe the UAV wing tank during refueling.

Label	Description
Unknown Variables	
X_{PT}	air pressure in tank
X_{FT}	fuel level in tank
X_{PV}	air pressure in ventilation system
X_{FV}	fuel level in ventilation system
X_{PA}	air pressure in the ambient air
F_{IN}	fuel flow into the tank
Known Variables	
y_{PST}	pressure sensor in tank
y_{PSV}	pressure sensor in ventilation system
y_{FS1}	fuel probe 1 in tank
y_{FS2}	fuel probe 2 in tank
y_{HFLS}	high-fuel level-sensor in ventilation system
y_{PSA}	ambient air pressure
u_{RV}	control signal for the refueling valve

Fault Variables	
f_{PST}	Fault of pressure sensor in tank
f_{PSV}	Fault of pressure sensor in ventilation system
f_{FS1}	Fault of fuel probe 1
f_{FS2}	Fault of fuel probe 2
f_{HFLS}	Fault of high fuel level sensor in ventilation system
f_{PSA}	Fault of ambient air pressure signal
f_{RV}	Fault in the refueling valve
f_{OFT}	Overfilled tank, e.g. fuel flow into the ventilation system
f_{LT}	leakage from tank
f_{VP}	Clogging in the large ventilation pipe which is connected to the ambient air

3.3.3 Equations used in the Model

When the equations to be used in the model, M of the wing tank are derived, there are two different alternatives which must be examined to find the most appropriate relations to use in the structural analysis.

1. The first alternative is to use equations where the fuel level and the pressure behavior in the tank are connected. This can be done since the pressure build up depends of the total volume of air in the tank.
2. The second alternative is to use that the pressure in the tank system is almost equal to the ambient air pressure in the fault free case.

A small analysis can be performed to examine which alternative to be used. This analysis shows what behavior to expect for the pressure in the tank. Figure 3.2 shows a simple tank model. Fuel is pressed into the tank and air is flowing out from the tank. The total tank volume, V_{tank} is $0,5 \text{ m}^3$.

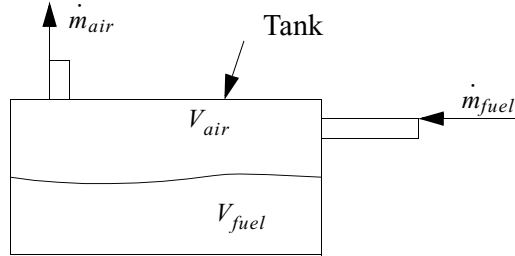


Figure 3.2: Tank model for examination of pressure build up.

The airflow out from the tank \dot{m}_{air} is:

$$P_{ambient} = \sqrt{P_{tank}^2 - \frac{\xi R \cdot 2}{A^2} \dot{m}_{air} T} \Rightarrow \dot{m}_{air} = \sqrt{P_{tank}^2 - P_{ambient}^2} \frac{A^2}{T \xi R} \quad (3.1)$$

Where A is the opening area of the connection between the tank and the ambient air and ξ is the loss coefficient, which depends on what type of orifice there is. P_{tank} is the pressure inside the tank, $P_{ambient}$ is the pressure in the ambient air, R is the ideal gas constant and T is the temperature.

Introducing the efficient opening area as:

$$A_{eff} = \frac{A}{\sqrt{\xi}} \quad (3.2)$$

The air volume in the tank decreases during refueling and since the fuel flow to the tank is constant, the air volume in the tank V_{air} decreases constantly when the fuel volume V_{fuel} increases:

$$V_{air} = V_{tank} - V_{fuel} \quad (3.3)$$

$$\dot{V}_{air} = -\dot{V}_{fuel} = -\frac{\dot{m}_{fuel}}{\rho_{fuel}} \quad (3.4)$$

Inside the tank the pressure is described with the ideal gas law:

$$P_{tank} V_{air} = \dot{m}_{air} R T \quad (3.5)$$

Modeling Methods

The ideal gas law is differentiated and an approximation of \dot{P}_{tank} can be estimated as.

$$\dot{P}_{tank} = -\dot{m}_{air} \frac{RT}{V_{air}} + m_{air} RT (-1) \frac{\dot{V}_{air}}{V_{air}^2} = -\dot{m}_{air} \frac{RT}{V_{air}} - \frac{P_{tank}}{V_{air}} \dot{V}_{air} \quad (3.6)$$

Equations (3.2) and (3.6) imply.

$$\dot{P}_{tank} = -A_{eff} \frac{\sqrt{(P_{tank}^2 - P_{ambient}^2)RT}}{V_{air}} - \frac{P_{tank}}{V_{air}} \dot{V}_{air} \quad (3.7)$$

Figure 3.3 shows the pressure in the tank described in Figure 3.2 during refueling using equations (3.3), (3.4) and (3.7), with an effective area A_{eff} equal to 1 cm^3 and a fuel flow \dot{m}_{fuel} , into the tank constantly equal to 10 kg/s , which is a very high value for this type of application. The tank is empty when the refueling begins and is filled up to 90% in 45 seconds. The tank pressure first increases from the ambient pressure which is set to 101.3 kPa up to a maximum pressure of 101.94 kPa . As seen from Figure 3.3 the pressure increases fast when the refueling begins and decreases back to the ambient air pressure even faster when the refueling ends. This arises from that the total volume of air in the tank is much smaller at the end of the refueling process.

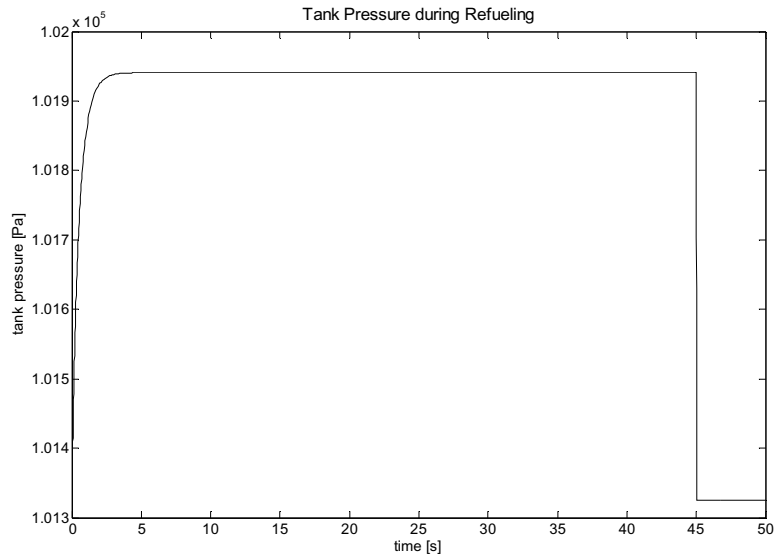


Figure 3.3: Estimated tank pressure in the wing tank during refueling.

Since the pressure differences in Figure 3.3 is very small, it can be very hard to measure and design tests for the pressure changes over time in this type of tank. Therefore the first alternative can not be used to derive a model of the wing tank, and instead the second alternative where the pressure in the wing tank is assumed to be almost equal to the ambient air pressure must be used.

System Equations

Table 3.4 shows the *system equations* used for the structural model of the wing tank during refueling. System equations are equations which are used to describe the process and can be e.g. the *ideal gas law*. Since Figure 3.3 shows that the size of the pressure difference between the tank and the ambient air is very small compared to sensor noise and model uncertainties, the pressure differences in the tank can be considered to be zero.

Table 3.4: System equations in the wing tank model.

EQ	Expression
e_1	$e_1(F_{IN}, \dot{X}_{FT}, f_{OFT}, f_{LT}) = 0$
e_2	$e_2(\dot{X}_{FV}, f_{OFT}) = 0$
e_3	$e_3(X_{PT}, X_{PV}, f_{OFT}) \approx 0$
e_4	$e_4(X_{PV}, X_{PA}, f_{VP}) \approx 0$
e_5	$F_{IN} - u_{RV} + f_{RV} = 0$

Equation e_1 describes the fuel flow to the wing tank, e_2 describes the flow from the wing tank to the ventilation tank if the wing tank is overfilled, e_3 and e_4 describes that the pressure is almost constant in the whole system and the ambient air as long no tank is overfilled or clogging has occurred in the ventilation pipe and e_5 describes the flow to the tank from the refueling valve

Sensors Equations

Sensor equations are used to introduce the sensor signals in the structural model, M . During refueling the UAV is standing on a plain ground. Therefore the fuel level is constant in the tank and can be measured without further knowledge of e.g. the angle of the fuel surface.

Table 3.5: Sensor and signals equations.

EQ	Expression
e_6	$X_{FT} - y_{FS1} + f_{FS1} = 0$
e_7	$X_{FT} - y_{FS2} + f_{FS2} = 0$
e_8	$X_{PT} - y_{PST} + f_{PST} = 0$
e_9	$X_{PV} - y_{PSV} + f_{PSV} = 0$
e_{10}	$X_{PA} - y_{PSA} + f_{PSA} = 0$
e_{11}	$y_{HFLV} = \begin{cases} 0 + f_{HFLS} & \text{if } X_{FV} = 0 \\ 1 - f_{HFLS} & \text{if } X_{FV} > 0 \end{cases}$

Equation e_6 and e_7 describes the fuel level measurements in the wing tank, e_8 and e_9 describes the pressure measurement in the wing tank and in the ventila-

tion tank, e_{10} describes the ambient air pressure signal and e_{11} describes the high fuel level sensor in the ventilation system.

Fault Models

Two equations are introduced to describe the sensor faults in the fuel probes.

Table 3.6: Fault model equations.

EQ	Expression
e_{12}	$\dot{f}_{FS1} = 0$
e_{13}	$\dot{f}_{FS2} = 0$

Equations e_{12} and e_{13} describes the sensor faults for sensors f_{FS1} and f_{FS2} as offset faults.

3.3.4 Structural Model

The set of equations in Table 3.4, Table 3.5 and Table 3.6 can be replaced with a structural model, which is shown in Table 3.7. This type of models will be one input to the analysis presented later in Chapter 4, 5 and 7.

Modeling Methods

Table 3.7: Structural model of wing tank during refueling.

		e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}	e_{13}	
X	X_{PT}			X					X						
	X_{FT}						X	X							
	\dot{X}_{FT}	X													
	X_{PV}			X	X					X					
	X_{FV}		X										X		
	X_{PA}				X						X				
	F_{IN}	X				X									
	<hr/>														
Y	y_{PST}								X						
	y_{PSV}									X					
	y_{FS1}						X								
	y_{FS2}							X							
	y_{HFLS}												X		
	y_{PSA}										X				
	u_{RV}					X									
<hr/>															
F	f_{PST}								X						
	f_{PSV}									X					
	f_{FS1}						X								
	\dot{f}_{FS1}												X		
	f_{FS2}							X							
	\dot{f}_{FS2}													X	
	f_{HFLS}												X		
	f_{PSA}										X				
	f_{RV}					X									
	f_{OFT}	X	X	X											
	f_{LT}	X													
	f_{VP}				X										
			e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}	e_{13}

3.4 Introduction to MSS Sets

Since Structural methods focus on that there is a relation between variables, instead of examine the art of the relation, see section 2.5. A method can be used to find out which relations, that are appropriate to use for a diagnosis sys-

tem. A type of equations sets called *Minimal Structural Singular* (MSS) sets have shown to be useful for design of diagnosis systems [1]. In this work all MSS sets in a structural model, (SM) are used to predict the maximum fault detection and fault isolability which can be obtained from a future diagnosis system.

First some basic definition must be introduced to describe MSS sets. For a more detailed description of MSS sets see [1] or [4].

3.4.1 Structural Singular

A set of equations are *structural singular* if the number of equations are bigger than the number of unknown variables in this set of equations. All structural singular sets of the equations from Table 3.2 are listed in Table 3.8

Table 3.8: Structural singular sets.

Equations	Unknown variables
$\{e_1, e_2\}$	$\{x_1\}$
$\{e_1, e_2, e_3\}$	$\{x_1, x_2\}$
$\{e_1, e_2, e_4\}$	$\{x_1, x_2\}$
$\{e_1, e_2, e_3, e_4\}$	$\{x_1, x_2\}$
$\{e_3, e_4\}$	$\{x_2\}$
$\{e_1, e_3, e_4\}$	$\{x_1, x_2\}$
$\{e_2, e_3, e_4\}$	$\{x_1, x_2\}$

3.4.2 Minimal Structural Singular (MSS)

A structural singular set of equations is a minimal structural singular (MSS) set if none of its proper subset are structural singular. All MSS sets of equations from Table 3.8 are listed in Table 3.9.

Table 3.9: Minimal structural singular sets.

Equations	Unknown variables
$\{e_1, e_2\}$	$\{x_1\}$
$\{e_3, e_4\}$	$\{x_2\}$

3.4.3 The Use of MSS Sets

Since MSS sets are equations without unknown variables these can be used to find tests to implement in a diagnosis system. When MSS sets are used to implement tests each test are sensitive to the fault variables included in the MSS used for that test. Since MSS sets have shown to contribute with high fault detection and fault isolation capability the use of MSS sets has shown to be a good way to find test quantities [1].

Algorithm used to find MSS Sets

This thesis is partly founded on an algorithm for finding MSS sets. Each MSS set represent a relations between variables and can be used to implement tests in a Diagnosis System. The algorithm, Figure 4.1 can be described in a few steps, *Differentiation*, *Simplification*, *Search for MSS sets*, *Analysis of the diagnosability*, *Decouple of faults* and *Selection MSS sets* of a Structural Model. All steps are briefly described in this chapter. For a full description of the use and further properties of MSS sets it is appropriate to study “*Design and Analysis of diagnosis Systems Utilizing Structural Methods*” [1].

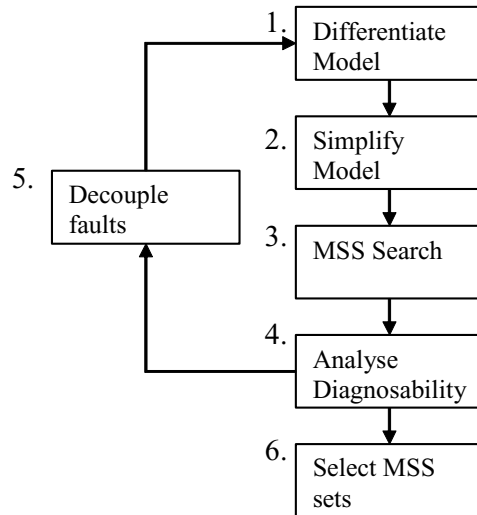


Figure 4.1: Schematic view over the algorithm used to find MSS sets.

4.1 Differentiate the Model

Sometimes it is possible to get more information out from a set of equations if differentiation is considered. First two examples will show why differentiation can contribute to make elimination of unknown variables possible.

Example 1:

Consider the set

$$E = \{e_1, e_2, e_3\} = \{y_1 = x, y_2 = \dot{x}, y_3 = x^2\}$$

of equations. An algorithm that consider derivatives of variables as completely different variables and that is not capable to differentiate equations can obviously not eliminate \dot{x} from e_2 . In general all derivatives of an equation must be considered to achieve the best possible elimination of unknown variables.

Example 2:

Now consider the differentiated set

Algorithm used to find MSS Sets

$$\dot{E} = \{\dot{e}_1, \dot{e}_2, \dot{e}_3\} = \{\dot{y}_1 = \dot{x}, \dot{y}_2 = \ddot{x}, \dot{y}_3 = 2x\dot{x}\}$$

of the equations from example 1. This set of equations shows that variables are handled differently depending on if they are linearly or nonlinearly contained in an equation, notice e.g. how x^2 in e_3 is handled. This implicates that information about which variables that are linear contained and which that are nonlinear contained in each equation must be included in the structural model. This makes it possible to define a structural differentiation that produces a correct structural representation of differentiated equations. This can be defined in the following way:

- If x is linearly contained in an equation e , then \dot{x} is linearly contained in \dot{e} .
- If x is nonlinearly contained in e , then x and \dot{x} are nonlinearly contained in \dot{e} .

Since each differentiation of an equation implies a new equation, it will be infinity many equations if the equations are differentiated infinity many times. A limit which corresponds to the highest order of derivative that can be estimated for each known variable prevents the introduction, of derivatives of a to high order which can not be estimated. Since faults and unknown variables not correspond to signals which must be estimated, they can be differentiated arbitrary many times and therefore no limits are needed for these kind of variables.

It is a complex task to find the differentiated model with the optimal possibilities for elimination of unknown variables. Since the algorithm must prevent introduction of more or equally many unknown variables than introduced equations. For closer view at this step of the algorithm see [1].

4.1.1 Example of a Differentiated Model

A short example will show how the differentiation step works on a small model.

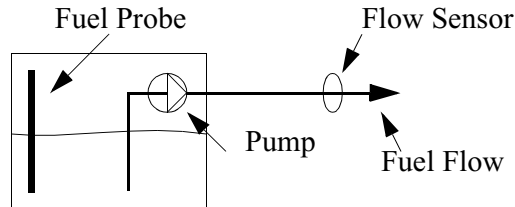


Figure 4.2: Pumping fuel out from a tank.

Figure 4.2 shows a small process where a pump is pumping fuel out from a tank. Inside the tank there is a fuel probe y_{FST} which measures the amount of fuel X_{FT} in the tank constantly. The flow out from the pump F_{FO} is measured with a flow sensor y_{FFS} and the pump is controlled by a control signal u_P . There are also three possible faults, a pump fault f_P and two sensor faults f_{FST} and f_{FFS} in the system. The signals from the known variables y_{FST} , y_{FFS} and u_P are assumed to be possible to derivative one time, meaning that just single derivatives can be used. Derivatives of higher order can be hard to use due to noise.

Table 4.1: Small model over fuel transfer from tank.

EQ	Expression
e_1	$\dot{X}_{FT} + F_{FO} = 0$
e_2	$u_P - F_{FO}^2 + f_P = 0$
e_3	$y_{FST} - X_{FT} + f_{FST} = 0$
e_4	$y_{FFS} - F_{FO} + f_{FFS} = 0$
e_5	$\dot{f}_{FST} = 0$

Table 4.1 shows all equations used to describe the fuel transfer described in Figure 4.2, e_1 describes the fuel quantity, e_2 the pump, e_3 the fuel probe, e_4 the fuel flow sensor and e_5 that the fuel probe just can have a offset fault. Figure 4.3 shows a structural model obtained from the Matlab implementation described in Chapter 6. Note that F_{FO} in e_2 are marked with a cross instead of a dot, which indicates that F_{FO} not is linear.

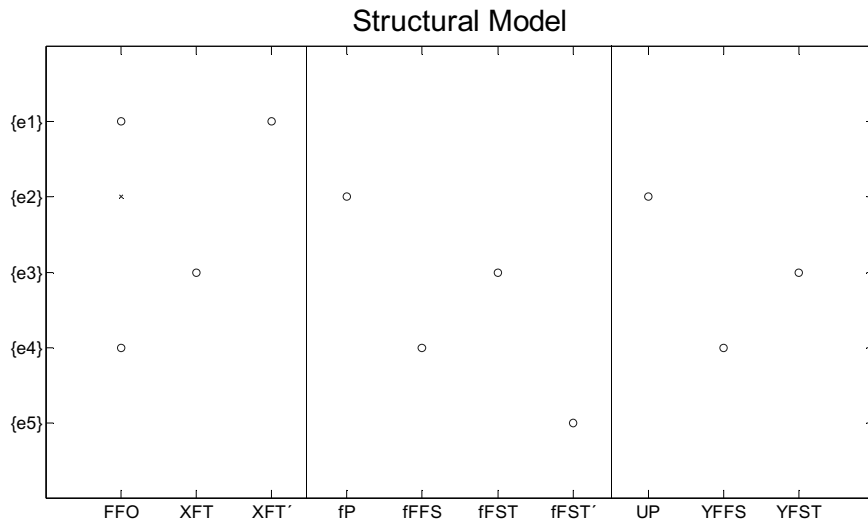


Figure 4.3: Structural model corresponding to Table 4.1

Figure 4.4 shows the differentiated structural model achieved, when the differentiate step of the algorithm operates on the structural model in Figure 4.3. Since X_{FT} is an unknown variable included in e_1 , e_3 must be differentiated if X_{FT} is to be eliminated. Equation e_2 and e_4 are differentiated one time since one new unknown variable, F_{FO} and two new equations \dot{e}_2 and \dot{e}_4 are introduced during that procedure. Note that both F_{FO} and \dot{F}_{FO} are included in \dot{e}_2 since F_{FO} is nonlinear included in e_2 . All steps in this process are handled by the Matlab implementations described in Chapter 6.

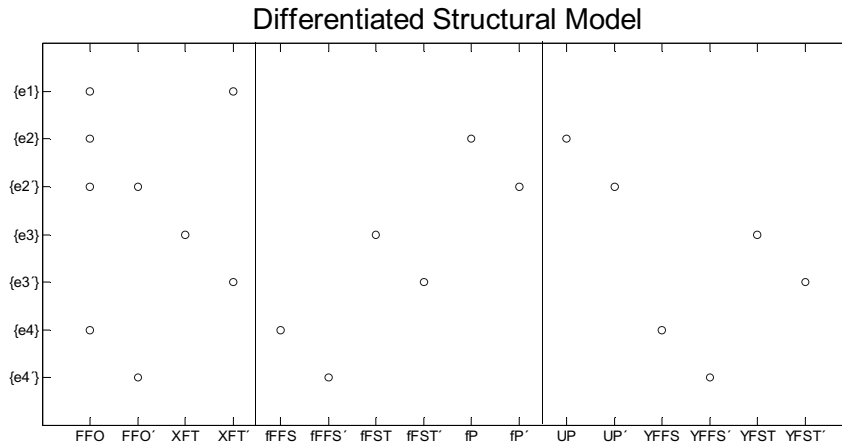


Figure 4.4: Differentiated structural model corresponding to Table 4.1.

4.2 Simplify the Model

To reduce the time for the computations done later in the algorithm, it is desirable to simplify the differentiated model from step 1. This simplification step is computational cheap compared to if the MSS search should operate directly on the differentiated model. Therefore by simplifying the model first the total computational complexity in the algorithm decreases a lot [4]. In the simplification step all equation that includes any variable that are impossible to eliminate are removed from the model, since they cannot be part of any MSS. This can be done with canonical decomposition, see [1].

The equations which must be used together, to eliminate unknown variables they have in common, are merged to reduce the complexity in the following steps. This is done by finding and eliminating subsets of unknown variables that are included in exactly one more equation than the number of the variables. The result after applying the simplification step to the structural model in Figure 4.4 is shown in Figure 4.5.

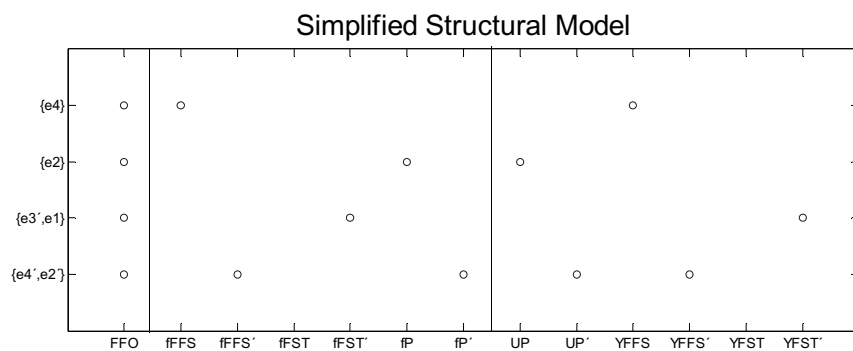


Figure 4.5: Simplified structural model corresponding to Table 4.4.

In Figure 4.5 e_1 and e_3 have been merged since they must be used together if X_{FT} is to be eliminated and for the same reason e_2 and e_4 are merged to eliminate F_{FO} . The only unknown variable left to be eliminated after the simplification step is F_{FO} .

4.3 Search for MSS sets

This step in the algorithm finds all MSS sets in a structural model. For a full description of how this step works see [1]. Figure 4.6 shows all MSS sets which were found in the model described in Figure 4.5. The six MSS sets found represent all different possibilities to eliminate F_{FO} after the simplification step in Figure 4.5. The search for MSS sets can be computational heavy and it is important to first perform the simplification step.

Algorithm used to find MSS Sets

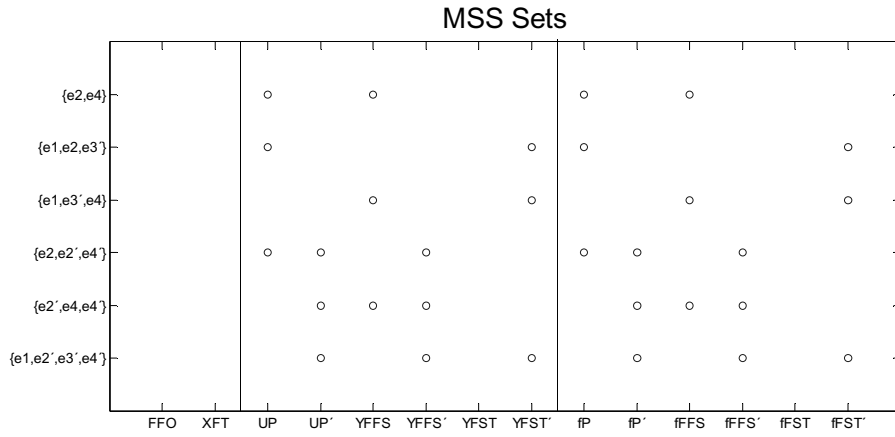


Figure 4.6: MSS sets found in the simplified structural model in Figure 4.5.

4.4 Analysis of Isolability

In this step the isolability for the MSS sets found in step 3 are analysed. Table 4.2 shows which faults that are included in each MSS set.

Table 4.2: Faults included in MSS sets.

MSS Set	Included faults
$\{e_2, e_4\}$	$\{f_P, f_{FFS}\}$
$\{e_1, e_2, e_3\}$	$\{f_P, \dot{f}_{FST}\}$
$\{e_1, \dot{e}_3, e_4\}$	$\{f_{FFS}, \dot{f}_{FST}\}$
$\{e_2, \dot{e}_2, e_4\}$	$\{f_P, \dot{f}_P, \dot{f}_{FFS}\}$
$\{e_2, e_4, \dot{e}_4\}$	$\{\dot{f}_P, f_{FFS}, \dot{f}_{FFS}\}$
$\{e_1, \dot{e}_2, \dot{e}_3, e_4\}$	$\{\dot{f}_P, \dot{f}_{FFS}, \dot{f}_{FST}\}$

Since $\dot{f}_{FST} = 0$, see Table 4.1, Table 4.2 must be modified to achieve the right fault sensitivity of each MSS set. The result after this modification is showed in Table 4.3.

Table 4.3: Faults included in MSS sets after modification.

MSS Set	Included faults
$\{e_2, e_4\}$	$\{f_P, f_{FFS}\}$
$\{e_1, e_2, e_3\}$	$\{f_P\}$
$\{e_1, \dot{e}_3, e_4\}$	$\{f_{FFS}\}$
$\{e_2, \dot{e}_2, e_4\}$	$\{f_P, \dot{f}_P, \dot{f}_{FFS}\}$
$\{e_2, e_4, \dot{e}_4\}$	$\{\dot{f}_P, f_{FFS}, \dot{f}_{FFS}\}$
$\{e_1, \dot{e}_2, \dot{e}_3, e_4\}$	$\{\dot{f}_P, \dot{f}_{FFS}\}$

Table 4.3 shows that f_P and f_{FFS} can be detected and isolated if a diagnosis test is designed by using the second and the third MSS sets, $\{e_1, e_2, e_3\}$ and $\{e_1, \dot{e}_3, e_4\}$. This since a test based on the second MSS only reacts if $\{f_P\}$ affects from zero and a test based on the third MSS only reacts if $\{f_{FFS}\}$ affects from zero. Since f_{FST} not is included in any MSS set f_{FST} can not be detected or isolated. It is therefore possible to run out of fuel without notice, if that sensor fault occurs.

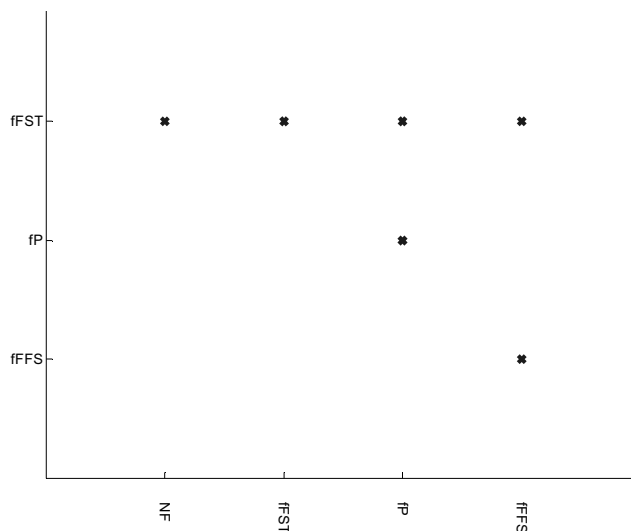


Figure 4.7: Isolability matrix corresponding to Figure 4.6.

Figure 4.7 shows a *isolability matrix* obtained from the Matlab implementations described in Chapter 6, corresponding to the MSS sets in Figure 4.6. A

marking on row i in column j in the isolability matrix means that if the fault corresponding to row i is present it can not be isolated from the fault corresponding to column j . If there is a mark in the first column (NF) of any row, the fault corresponding to that row can not be isolated from the NF mode, i.e. the fault can not be detected.

A quick view at Figure 4.7 shows that f_P and f_{FFS} can be detected and isolated if they occur, while f_{FST} can not be detected.

4.5 Decouple Faults

If the diagnosability of the isolability matrix in Figure 4.7 must be improved it is possible to run the algorithm again, with one or several fault treated as unknown variables, this is called *Fault Decoupling*. If a fault is decoupled this implicates that the MSS sets found not is sensitive to this fault, and can therefore contribute to isolate different faults from each other, see [1] or [2].

4.6 Summary of the Structural Algorithm

Here follows a short summary of all steps in the algorithm used to find the MSS sets:

1. **Differentiate the model:** Sometimes more information and relations can be obtained from a structural model if the structural model is differentiated. If differentiation is to be used it is important to find and differentiate just equations which are meaningful to differentiate for finding MSS sets. Differentiation must not be used, but is always used in this thesis. If differentiation not is used step two in this algorithm will be the first step.
2. **Simplify the model:** Remove all equations which not can be used in any MSS set, from the equations found in step 1. Merge sets of equations that have to be used together in each MSS set. With this simplification step the time used for this step and the third step in the algorithm can be decreased, compared to if a full MSS search is done directly in the differentiated model from step 1.

3. **Search for MSS sets:** Search for MSS sets, this step finds all MSS sets in the model from step 2.
4. **Analysis of diagnosability:** Examine the fault detection and the fault isolation capability of the MSS sets found in step 3. This examination is done by creating a fault isolability matrix from the MSS sets achieved in step 3. In this work all MSS sets found in step 3 are used, but it is also possible to use subsets of them. This might however result in less fault detection and fault isolation compared to if all MSS sets are used. In the Matlab implementations described in Chapter 6 this can be handled.
5. **Decouple faults:** If the diagnosability in step 4 not is enough, faults can be decoupled. To decouple faults, return to step 1 and consider these faults as unknown variables. From this step new MSS sets can be obtained and used together with the MSS sets from step 4. This step can be repeated for all combinations of faults. In this work all single faults are decoupled in all analysis.
6. **Select MSS sets:** Select the MSS sets to be used in the diagnosis system to get the desired diagnosability. In this work are always all MSS sets found after step 5 used. But it can be appropriate to use just a subset of the MSS sets since some of them can be to complex to use, depending on the number of variables included in a MSS set or how hard it is to design a test for the actual MSS.

Algorithm used to find MSS Sets

Optimizing Sensors Configurations

When a process is to be designed, different possible faults in the process can be considered already during the design of the process, to increase the reliability and safety of the process. In some processes one or several sensors are used to control and supervise the process. This chapter shows how different sensor configurations can be examined and optimized to fulfill the requirements on a diagnosis system, using the algorithm described in Chapter 4. This method is later used in Chapter 7.

5.1 Fault Classification

When a diagnosis system for a process is to be designed, it is necessary to decide what fault detection and isolability to require from the diagnosis system.

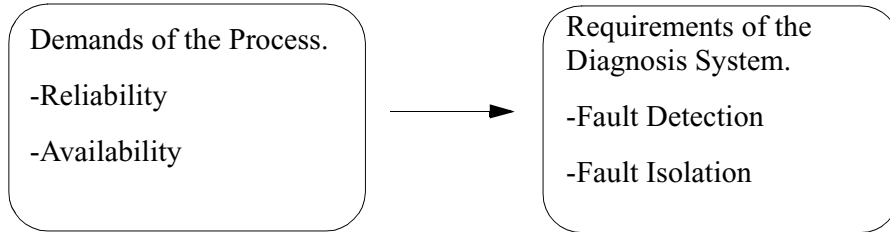


Figure 5.1: Requirements of a diagnosis system.

Figure 5.1 illustrates how the requirements of the process must be transferred to requirements on the diagnosis system. In big processes, this is a big task involving e.g. examinations of necessary process reliability and failure rates of different components in the process.

The approach used in this work is to divide all faults F in a process to be diagnosed into three groups:

1. Faults which have to be uniquely isolated, F_I .
2. Faults which have to be detected, F_D
3. Faults which not have to be detected or isolated, i.e. not prioritized faults F_N .

This can be described like $F = F_I \cup F_D \cup F_N$ where, I stand for Isolated, D for Detected and N for Not Prioritized.

5.1.1 Properties of Fault Classification

Table 5.1 shows a typical isolability matrix that satisfies the given classification that is defined above. The isolability matrix in Table 5.1 shows that all faults in F_I , (f_{I1} , f_{I2} and f_{I3}) can be isolated and that all faults in F_D , (f_{D1} , f_{D2} and f_{D3}) can be detected. All faults in F_N , (f_{N1} , f_{N2} and f_{N3}) can not be isolated from NF and it is therefore not sure that they are detected. However since these faults belong to F_N they are subjects to no isolability requirements.

Table 5.1: Isolability matrix where the included faults are classified.

		F_I			F_D			F_N		
		f_{I1}	f_{I2}	f_{I3}	f_{D1}	f_{D2}	f_{D3}	f_{N1}	f_{N2}	f_{N3}
F_I	f_{I1}	X								
	f_{I2}		X							
	f_{I3}			X						
F_D	f_{D1}				X					
	f_{D2}				X	X				
	f_{D3}					X	X			
F_N	f_{N1}	X	X	X	X	X	X	X	X	X
	f_{N2}								X	
	f_{N3}	X	X	X	X	X	X	X	X	X

5.1.2 Demands for the Fault Classification

This classification is important and must be well substantiated to prevent time demanding modifications later. Since this analysis has a great impact on the design of the diagnosis system. It is important to consider many different aspects like e.g. how hard it is to troubleshoot and find a fault, if the fault can cause damage to man or machine. For this work it is appropriate to find or develop a reliable method. Some inputs to such work can be found in Method for Diagnosis System Requirement's Prioritization [6].

5.2 Sensor Configurations

The choice of sensor configuration used in a process can be optimized in different ways, in this work to minimize the number of sensors that are needed to fulfill the isolability requirements. It is possible to focus on other properties e.g. the price or the quality of different sensors, to minimize a special type of sensors or to minimize the total costs for the sensors. Note that introduction of extra sensors can result in decreasing fault isolability. This because of that adding a new sensor often also implicates adding a new sensor fault.

5.2.1 Sensor Configuration Optimization

To simplify the work of finding sensor configurations which are possible to use in the diagnosis system, all sensors Y_S can be divided into two groups:

1. Sensors which have to be included e.g. for control or legal reasons, Y_{SR} .
2. Sensors which not have to be included, Y_{SO} .

This can be described like: $Y_S = Y_{SR} \cup Y_{SO}$ where, S stand for Sensor, R for Required and O for Optional.

All possible sensor configurations must contain all sensors i Y_{SR} , this can reduce the number of possible sensor configurations heavily. The set $\{Y_i | Y_{SR} \subseteq Y_i \subseteq Y_{SR} \cup Y_{SO}\}$ is the set of all possible sensor configurations Y_i such that (1) and (2) is fulfilled for the classification of Y_S .

A full analysis can then be applied on the structural model for each remaining sensor configuration Y_i , to exam which configurations that have enough isolation and detection capability, to fulfill the demands.

5.3 Algorithm used to Examine Sensor Configurations

The algorithm used to find sensor configurations in this work is described in Figure 5.2. The objectives with this algorithm are to find the sensor configuration or sensor configurations, with least number of sensors, which can fulfill the requirements, put on a diagnosis system. The input to the algorithm are a structural model SM with all possible sensors categorized $Y_S = Y_{SR} \cup Y_{SO}$ and a fault classification $F = F_I \cup F_D \cup F_N$. The result after the algorithm is a structural model, all MSS sets found in the structural model and a isolability matrix for each sensor configuration which fulfills the diagnosis task.

Since the total number of sensor configurations is growing exponential to the number of optional sensors this analysis can be time demanding, e.g. if there are five optional sensors to be tested the algorithm must be called $2^5 = 32$ times. Since the time required for this analysis grows exponential, first all single combination of sensors can be studied. If one or several of them fulfils the demands it is not necessary to study configurations with more sensors.

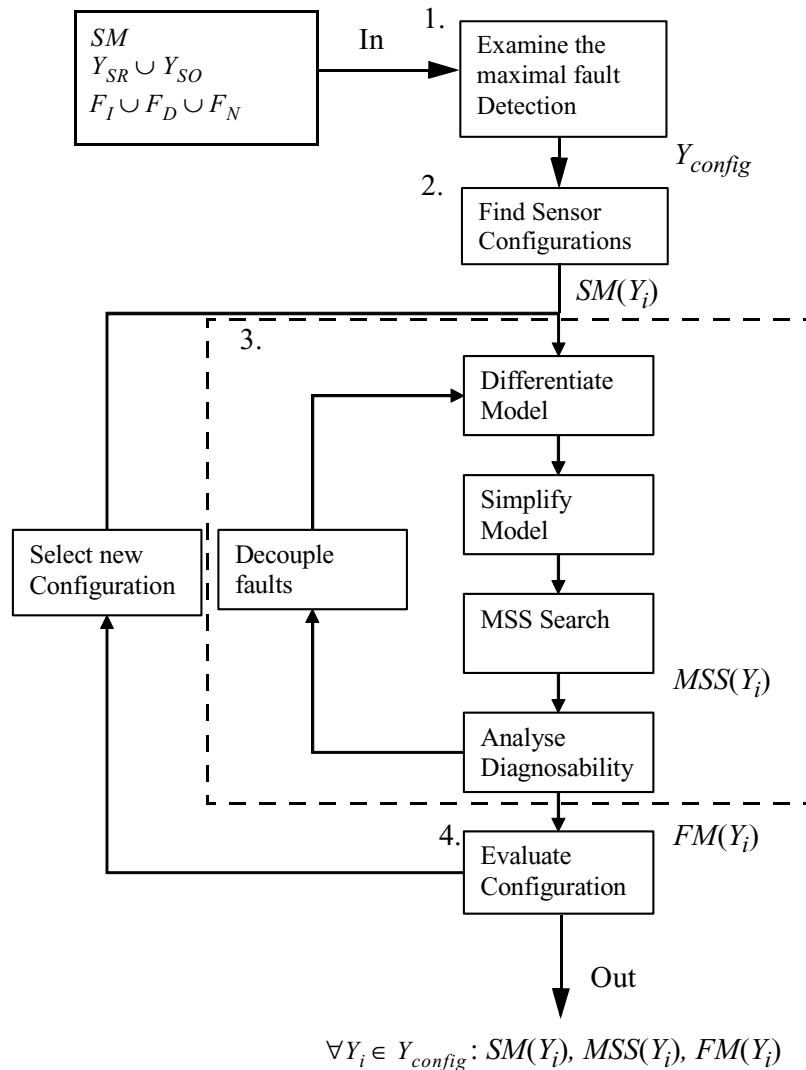


Figure 5.2: A schematic view of the examination of different sensor configurations.

1. **Examine the maximal fault detection:** First a structural analysis is performed with all sensors Y_S included, to determine the maximal fault detection which can be obtained in the process. For this analysis a full structural model of the process including all sensors Y_S is used. If this maximal sensor configuration can fulfill the fault detection

requirements put on the diagnosis system, it is possible to examine if the requirements can be fulfilled also with other sensor configurations including fewer sensors.

2. **Find all possible sensor configurations:** Since all sensors of the type Y_{SR} must be included in each sensor configuration the configurations which must be further examined is the configuration which only includes all sensors of type Y_{SR} and all uniquely configurations which include all sensors of type Y_{SR} and one or more sensors of type Y_{SO} .
3. **Perform the full MSS algorithm:** To evaluate all possible sensor configurations the full MSS algorithm described in chapter 4 is performed for each sensor configuration from step 2.
4. **Examine if the MSS sets found can fulfill the diagnosis task:** Examine if the present sensor configuration can fulfil all detection and isolability demands by using an isolability matrix where the included faults are classified, see Table 5.1. This examination can be simplified by using functions in the Matlab implementations, which are described in Chapter 6.

5.4 Optimization Strategies using a Fault Isolability Matrix

The fault isolability matrix described in Table 5.1 can be used in different ways to optimize, evaluate, and examine fault isolability matrices. Since fault isolability matrices can be achieved from the Matlab implementations described in Chapter 6 this analysis can be powerful and flexible. In “Method for Diagnosis System Requirement’s Prioritization”[6] further inputs to this optimization can be found.

6

Matlab Implementation

A Matlab implementation of the algorithms in Chapter 4 and in Chapter 5 is described in this Chapter. The implementation consists of several independent functions which perform the different steps of the algorithm used in this work. The functions can then be used in e.g. a Matlab m-file to perform structural analysis using the different parts from the algorithms.

6.1 Graphic User Interface

A graphic user interface (GUI) is used to simplify the implementation of structural models in Matlab. The GUI consist of two parts one for defining included variables, Figure 6.1 and another for defining included equations, Figure 6.2.

6.1.1 Definition of Variables

To define all variables in a structural model, the GUI shown in Figure 6.1 is used. In Matlab the GUI is called with the command:

```
define_variables('savefile')
```

It is then easy to define each variable with name and type. The name is typed in the input field for variables name and the type is chosen by marking the right box. Finally the variables can be confirmed and saved by clicking “Add Variable”.

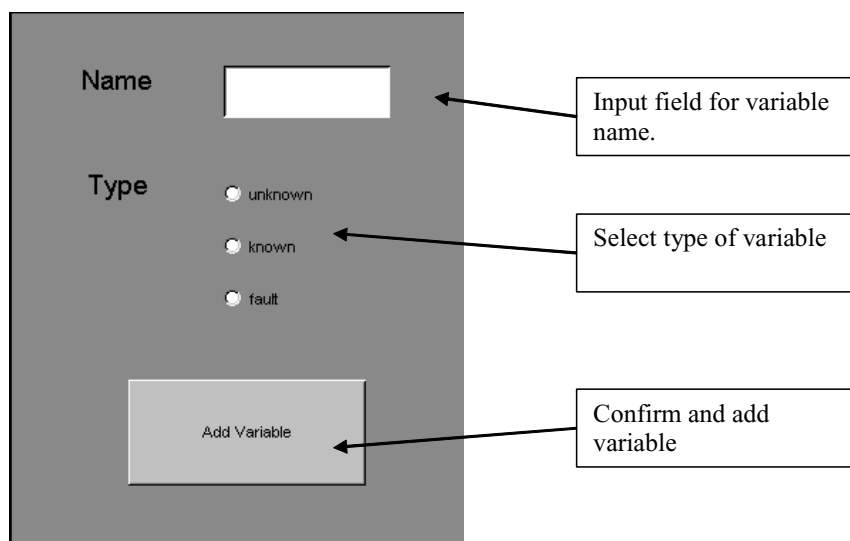


Figure 6.1: GUI for definition of variables.

To redefine a variable, the same variable name can be used again. The variable file is saved in the present working directory.

6.1.2 Definition of Equations

To define all equations in a structural model, the GUI in Figure 6.2 is used. In Matlab the GUI is called with the command:

```
define_equations('variablefile','savefile')
```

The first argument “variablefile” is a predefined file, for example generated by the command `define_variables`, with all variables which are to be used in the model. In this GUI it is easy to name and define all equations in the model.

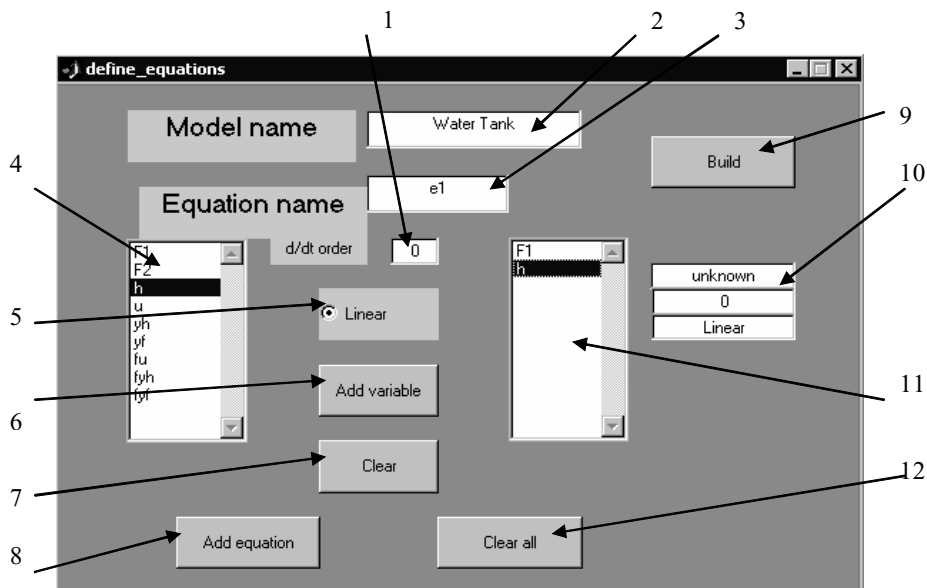


Figure 6.2: GUI for definition of equations in a structural model.

The different objects in the GUI described in Figure 6.2 are:

1. **Derivative Field:** This field is used for input of derivative order when a variable is to be added into 11. Zero represent a non differentiated variable.
2. **Name Field:** This field is used to name the structural model.
3. **Equation Name Field:** This field is used to name the present equation.
4. **Variable Field:** This field shows all variables which can be used.
5. **Linear Box:** This box is marked if a variable is to be linearly included into the equation.
6. **Add Variable Button:** Add the selected variable in 4 to the present equation in 11.
7. **Clear Button:** Clear field 11 from all variables.
8. **Add Equation Button:** Add the present equation in 11 to the model.

9. **Build Button:** Builds an SM object and saves it to a file in the present Matlab working directory.
10. **Info Field:** This field shows the properties for the selected variables in 11, first line shows the variable type, second line the derivative order, and third line shows if the variable is linearly or not linearly included in the equation.
11. **Equation Field:** This field shows all variables included in the present equation. Variables can be added to this field from 4 by using 6 and removed from the field with 7.
12. **Clear all Button:** This button clears all equations from the model.

By pressing “Build” the model is saved as an SM object in a file named after the second input argument, 'savefile'.

6.2 Objects representing Structural Models and Isolability Matrices

Different objects are used to represent structural models and isolability matrices.

6.2.1 SM Objects

In this implementation structural models are represented as SM objects. Table 6.1 shows an SM object as it is shown in Matlab. Most functions in the implementation takes SM objects as arguments.

Table 6.1: The structure of an SM object.

```

SM =
m:      [15x28 double]
e:      {1x15 cell}
v:      {1x28 cell}
type:   'Original Structural Model'
name:   'demo UAV'
mode:   {}
x:      {'PA' 'PVU' 'PVT' 'PngT' 'PT3R' 'PT3L' 'PT2' 'PT1'}
y:      {'Pamb' 'PsVU' 'PsVT' 'PsT3R' 'PsT3L' 'PsT2' 'PsT1'}
f:      {1x13 cell}
ylimit: [1 1 1 1 1 1 1 1]
flimit: 0
xlimit: 0
    
```

- **SM.m**
SM.m is a matrix where the rows represent the structural equation and the columns the variables in a structural model. If variable j is included in equation i element (i,j) in the matrix is set to one if the variable is linear included or two if the variable is nonlinear included in the equation. If variable j is not included element (i,j) is zero. Differentiated variables are treated equal to normal variables.
- **SM.e**
SM.e consists of cells with the names and the order of derivative for all relations included in SM, like e.g. $\{\{ 'e1', [1] \}, \{ 'e2', [0] \} \}$. The cell on position i in SM.e consist of the name and order of derivative for the relation represented on row i in SM.m.
- **SM.v**
SM.v consists of cells with the names and the derivative order of all variables included in SM, like e.g. $\{\{ 'PA', [1] \}, \{ 'PVU', [0] \} \dots \}$. The cell on position j in SM.v consists of the name and order of derivative for the variable represented in column j in SM.m.
- **SM.type**
SM.type shows information of what type of structural model it is, e.g. Differentiated Structural Model or Simplified Structural Model. This information depends on which functions that previous has been perform on the model.

- **SM.name**
SM.name is the name of the model.
- **SM.mode**
SM.mode defines the fault mode of the model by a set of fault variables that are considered to be unknown variables.
- **SM.x**
SM.x is a list with names of all unknown variables in the model.
- **SM.y**
SM.y is a list with names of all known variables in the model.
- **SM.f**
SM.f is a list of all fault variables in the model.
- **SM.ylimit**
SM.ylimit is a list with information about the highest allowed derivative for each known variable in the model. Each element in the matrix corresponds to a known variable in SM.y.
- **SM.flimit**
SM.flimit is not used in this version.
- **SM.xlimit**
SM.xlimit is not used in this version.

6.2.2 SMSS Objects

For some functions in the Matlab implementation *Sortable MSS* (SMSS) objects are used. SMSS objects are transformations of SM objects with another structure.

Table 6.2: The structure of an SMSS object.

	Equations			Variables			
Equations/ Variables	A=	2	4	9	B=	3	7
derivative		0	1	2		0	2

SMSS objects consists of two matrices A and B for each SM Object. The matrices represents the equations and the variables included in an MSS set. Table 6.2 illustrates an SMSS object. Matrix A shows that equation 2, equation 4 and equation 9 in SM.e are used. Matrix A also shows that equation 4 is differentiated one time and equation 9 differentiated 2 times. Matrix B shows that variable 3 and variable 7 in SM.v are used and that variable 7 is differen-

tiated 2 times. Since SMSS objects just contains the positions of the equations and the variables, elements in SM.e and elements in SM.v. It is necessary to have access to the same SM object which were used to transform the MSS set to an SMSS, when an SMSS is to be transformed back to an SM object.

6.2.3 FM objects

Fault isolability matrices are saved as FM objects, Table 6.3 in the Matlab implementations. These contains of a isolability matrix, FM.m and the names of all faults, FM.f.

Table 6.3: The structure of a FM object.

```
FM =  
m:    [15x15 double]  
f:    {1x15 cell}
```

6.3 Functions used in the Matlab Implementation

Here follows a description of some functions used in the Matlab implementation. A full description of more functions in the Matlab implementation can be found in Appendix A:

6.3.1 Basic Functions for the MSS Algorithm

This functions performs the different steps in the algorithm described in Chapter 4 which is used to find MSS sets.

- **GetSMDecoupling()**
Performs the decoupling step of the algorithm.
- **Differentiate()**
Performs the differentiation step of the algorithm described in Chapter 4.
- **OverDetSM()**
Finds the over determined part of a structural model.
- **SimplifiedSM()**
Performs the simplification step of the algorithm.

- **FindMSSsets()**
Performs the find MSS step in the algorithm.

6.3.2 Functions used to Merge and Change Structural Models

This functions can be used for modifications of structural models to examine different sensor configurations and to merge MSS sets.

- **MakePowerSet()**
Takes one or two arguments, one argument generates all possible fault modes, a second arguments sets a limit for how big fault modes that must be included.
- **mergeMSS()**
Merges two MSS sets to one.
- **Eliminatevar()**
Eliminates a variable and all equations which are dependent to that variable in an SM object.
- **removeeqSM()**
Removes a equation from an SM object.
- **removevarSM()**
Removes a variable from an SM object without removing the equations which are dependent of the variable, e.g. to eliminate a fault.
- **makeSMSS()**
Transforms an MSS set to an SMSS object.
- **getbackMSS()**
Transforms an SMSS model to an MSS model.

6.3.3 Functions for Visualization

These functions can be used to plot structural models and fault isolability matrices.

- **PlotSM()**
Plots an SM object
- **PlotFM()**
Plots a fault matrix
- **PlotFMCat()**
Plots a fault classification isolability matrix as it is described in Chapter 5.

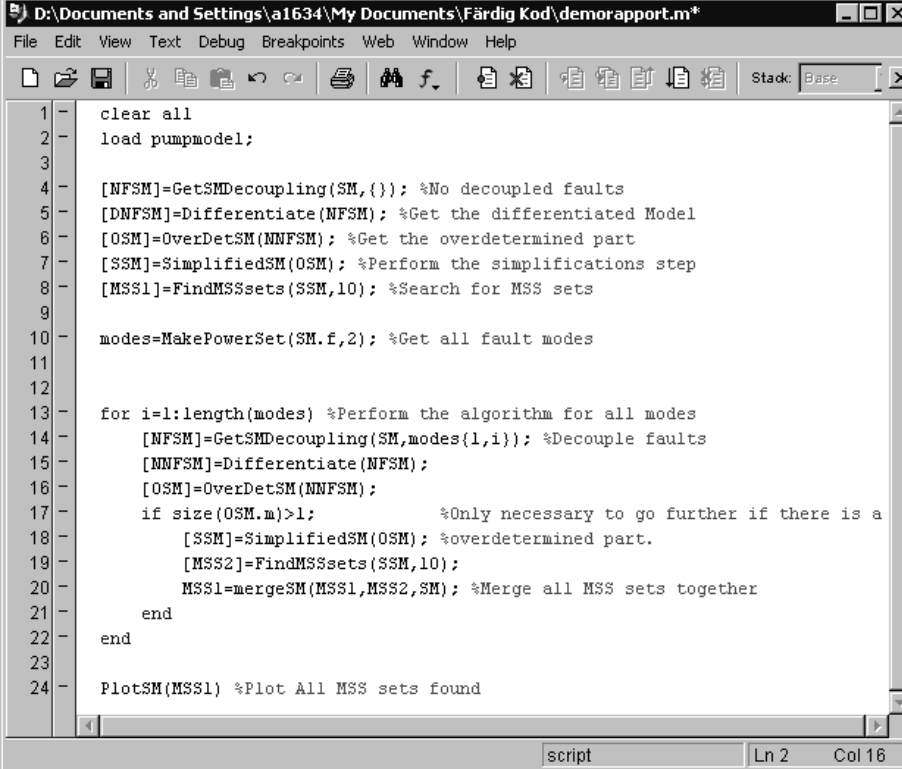
6.3.4 Functions for Analysis of MSS sets

- **getFaultmatrix()**
Generates a fault incidence matrix from an SM object with MSS sets.

6.4 Utilizing Matlab Implementations for Structural Analysis

Since the Matlab implementation, consists of several independent functions the use of the implementation can be flexible. This also means that a user must have some basic knowledge about the algorithms used.

Matlab Implementation



```
D:\Documents and Settings\al1634\My Documents\Färdig Kod\demorapport.m*
File Edit View Text Debug Breakpoints Web Window Help
Stack: Base
1 - clear all
2 - load pumpmodel;
3
4 - [NFSM]=GetSMDecoupling(SM,{}); %No decoupled faults
5 - [DNFSM]=Differentiate(NFSM); %Get the differentiated Model
6 - [OSM]=OverDetSM(NNFSM); %Get the overdetermined part
7 - [SSM]=SimplifiedSM(OSM); %Perform the simplifications step
8 - [MSS1]=FindMSSsets(SSM,10); %Search for MSS sets
9
10 - modes=MakePowerSet(SM.f,2); %Get all fault modes
11
12
13 - for i=1:length(modes) %Perform the algorithm for all modes
14 -     [NFSM]=GetSMDecoupling(SM,modes{1,i}); %Decouple faults
15 -     [NNFSM]=Differentiate(NFSM);
16 -     [OSM]=OverDetSM(NNFSM);
17 -     if size(OSM.m)>1; %Only necessary to go further if there is a
18 -         [SSM]=SimplifiedSM(OSM); %overdetermined part.
19 -         [MSS2]=FindMSSsets(SSM,10);
20 -         MSS1=mergeSM(MSS1,MSS2,SM); %Merge all MSS sets together
21 -     end
22 - end
23
24 - PlotSM(MSS1) %Plot All MSS sets found
script Ln 2 Col 16
```

Figure 6.3: Example of how the Matlab implementation can be used.

Figure 6.3 shows how a Matlab m-file and some functions of the Matlab implementation can be used to perform the algorithm described in Chapter 4 and Figure 4.1.

Row 1 in Figure 6.3 clear all variables in Matlab. On row 2 a structural model (SM-object) is loaded from a file named “pumpmodel”. On the rows 4-9 in Figure 6.3 the steps (1)-(4) of the algorithm in Figure 4.1 are performed. On row 10 all fault modes which are to be decoupled are defined, using the MakePowerSet function which is described in Appendix A. On the rows 13-22 in Figure 6.3 the decoupling of all fault modes are performed. Notice how all the present MSS sets are merged together with all previously found MSS sets on row 20.

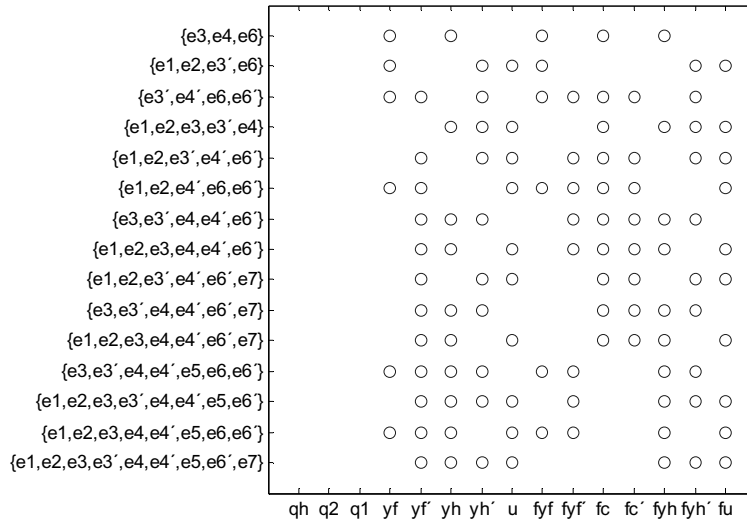


Figure 6.4: Example of found MSS sets.

Figure 6.4 shows the Matlab Plot which is done on row 24 in Figure 6.3. Obviously no unknown variables (qh,q2 and q1) can be included in the MSS sets since they have to be eliminated. In Figure 6.4 the structural properties of all MSS sets found in the structural model (SM) can be seen.

Further examples of how the Matlab implementation can be used are found in Chapter 7.

UAV Fuel System Concept

Most aircraft's fuel systems consist of several tanks due to e.g. center of gravity management, safety, space and slosh reasons. It is also easier to measure the fuel level in a small tank compared to a large. A general layout consists of one or more boost pumps that feed the engine from a tank close to the center of gravity [3]. However, the fuel system has two main tasks, in a safe way provide the engine with fuel under all flight conditions and to keep the center of gravity at a constant optimal position.

7.1 Introduction to Conceptual UAV

In this chapter the fuel system in a UAV concept is studied. This UAV, Figure 7.1 has stealth capabilities on the upper side and must therefore be able to fly long distances upside-down to avoid radar detection. Therefore the fuel system must include a *negative-g compartment*, which is a tank from which fuel can be taken during inverted flight. The fuel system includes two subsystems, a *Fuel Pump System* and a *Tank Pressurization System*.

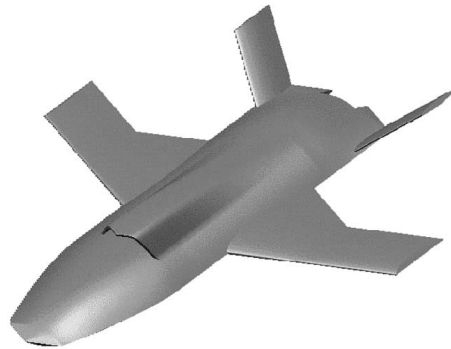


Figure 7.1: UAV with stealth capabilities at one side.

7.1.1 The Fuel Pump System

Figure 7.2 shows the wing tanks, tank 1 and tank 2 in the conceptual UAV fuel system. Fuel to feed the engine is always taken from the negative g compartment in tank 1. During normal flight, fuel is flowing into the negative-g compartment from the upper part of tank 1, but fuel can not flow back from the negative-g compartment to the upper part in tank 1, if the UAV is flying upside-down, since the transfer pipes ends over the fuel level.

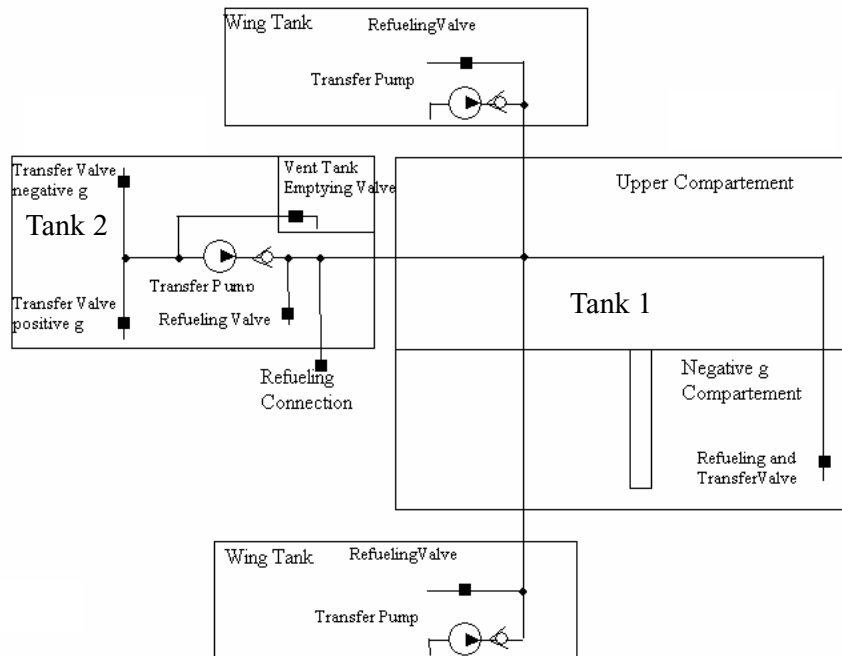


Figure 7.2: Schematic view of UAV fuel system concept.

Fuel pumps in the fuel system make it possible to transfer fuel between different tanks and to the engine. The system shown in Figure 7.2 consists of three transfer pumps, one in each wing tank and one in tank 2. There is also a double ended boost pump not shown in Figure 7.2 to feed the engine. All pumps are controlled by control signals. Fuel can be transferred to tank 1 from the wing tanks and from tank 2, but during flight upside-down fuel can only be transferred from tank 2 to tank 1. Fuel transferred to tank 1 is always placed in the negative g compartment to maximize the amount of fuel that can be used during long time flight upside-down.

7.1.2 The Tank Pressurization System

There are mainly two reasons for tank pressurizations. It prevents cavitations at high altitudes and damages to the tanks caused by high pressure differences between the tanks and the ambient air. The tank pressurization system must expel air during climb or refueling and add pressure during dive or fuel transfer. All tanks stand normally under some extra effective pressure by adding compressed air from the engine compressor. All tanks can be pressurized from

the bottom or from the top depending on if the UAV is flying normal or upside-down.

The Combined Over and Under Pressure Valve

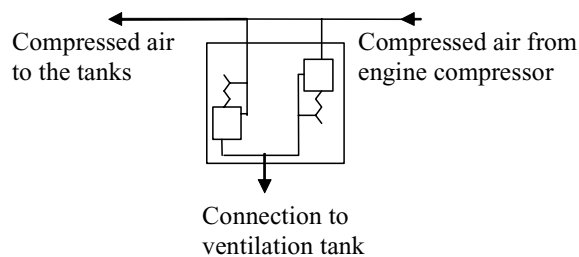


Figure 7.3: Schematic outline of combined over and under pressure valve

The tank pressure is controlled by a combined over and under pressure valve which is connected to the tanks and to the ambient air through the ventilation tank, see Figure 7.3. The over pressure valve opens if the tank pressure exceeds the ambient air pressure with a certain value. If the tank pressure instead sinks below the ambient air pressure the under pressure valve opens. This process is totally mechanical and no control signals are used. Instead the over pressure valve opens when the pressure difference between the tank system and the ambient air overcomes the pressure added from a spring. The amount of compressed air from the engine compressors is normally large enough to hold the over pressure valve open during flight. The over and under pressure valve is also dimensioned to take care of surplus fuel if a refueling valve fails to close. This to secure that the refueling pressure not damages the tanks if one or several tanks are overfilled.

Negative-G Valves

All ventilation pipes are equipped with valves to prevent fuel finding its way into them. These valves are called *negative-g valves* and closes when they are exposed to negative loads, e.g. during flight upside down. The valves work fully mechanically and are not controlled by any signal or supervised with any sensor. Despite the valves some fuel might find its way into the pipes and end up in the ventilation tank or in some other tank. Since all tanks are directly connected through the pressurization system it is not possible to control the tank pressure individually for any tank. This also implicates that the pressure is supposed to be constant in the whole system as long no fault has occurred.

The Pressure Regulator

A pressure regulator, controls the bleed air into the tank pressurization system, to prevent to high pressure and damages in the tanks. The pressure regulator can also shut off the tank pressurization.

7.2 Structural Analysis Strategy

To find relations between the different variables in the fuel system it is appropriate to analyse three types of equations, physical, signal, and boundary equations. The physical represents e.g. fluxes and pressure equations. Signal equations have to do with for example signals from sensors and control signals to different parts of the system. The boundary relations set the limits for the problem and can be both analytical and logical formulas e.g. that a sensor fault appears as a constant offset value.

7.2.1 Modeling Conditions

First an attempt to find solid relations between the fuel level and the tank pressurization was performed. Several interviews with experts at Saab Aerosystems in Linköping Sweden were performed to examine the possibilities for a relation between pressure and fuel levels in the tanks. A model over a typical UAV tank was implemented in EASY5, which is a modeling tool that can be used for e.g. pressure and flow simulations. The EASY5 model and the wing tank analysis in Chapter 3 showed together with the interviews that it was not appropriate to use relations between the fuel level and the pressure in the tanks, like e.g. the ideal gas law, in the structural analysis of the fuel system. This since these type of relations probably can not be used in a future diagnosis system and therefore can give a to optimistic view at the fault isolation possibilities of a future diagnosis system.

The Pressurization System

Today Saab Aerosystems uses only *pressure switches*, which indicates if the pressure in the fuel system exceeds or is below a certain reference value. However, in the future more advanced pressure sensors which measure pressure continuously and present a value can be used instead. These sensors are more expensive to use since more hardware and software is needed and since the sensors are more expensive to buy compared to pressure switches.

Since all tanks in the fuel system used in the conceptual UAV, studied in this work, are connected to each other and to the ambient air, one pressure sensor can be placed in the ventilation tank showed in Figure 7.2. This sensor can then supervise the pressure in the whole fuel system as long as no negative-g valve is stuck closed or clogging appears in a ventilation pipe. This together with that no relation between the pressure and the fuel level can be used leads to that the pressurization system is not more examined in this work. A similar fuel system design is used in Saab 105/SK 60 which has been flying without any of these problem since 1963.

The Fuel Pump System

In the fuel pump system there are opportunities for a model-based diagnosis system which can supervise the fuel transfers in the system. Such a system can contribute with information about the system both during flight and during maintenance. The case studied in this thesis is if the pump sensors after the transfer pumps can be eliminated and replaced with a model of how the fuel level changes in the different tanks. Since a normal transfer-pump sensor just is a switch which indicates if the pump works or not there is no possibilities to detect if the flow from the pump is just little lower than normal. Such information can be used to optimize maintenance and to replace pumps with small defects. However the fuel levels in the tanks can not be measured very exact, during steady flight the measure accuracy is at least 5% in a tank with a geometry which is easy to measure in, but the fault can be much larger if the tank is almost empty and fuel just splaches around. In other tanks e.g. wing tanks the measure fault can be very large also when the tanks are filled since they often have a complex geometry. Since the measurement is so uncertain it is maybe impossible to expel the transfer-pump sensors due to security reasons, but in this thesis it is assumed that the fuel measurements can be used if the fuel level is studied over a long time.

7.3 Model of the Fuel Pump System

The first task was to decide how detailed the structural model could be to prevent introduction of connections impossible to use. A first objective was to exam if the ideal gas law could be used to study the relation between the fuel level and the pressure in the tanks. One idea was to examine if the pressure variations over time was significant dependent to the air volume inside the tanks. But since this not was possible, see 7.2.1 no such relation are introduced.

7.3.1 Models of the Tanks

During normal flight, fuel is transferred from the wing tanks and tank 2, into tank 1. The diagnosis system is supposed to be active only when the UAV is flying steady without any exceptional maneuvers. The purpose of the structural analysis for this mode is to examine the possibilities for a diagnosis system to supervise the fuel pump system. All tanks in the fuel system are described separately to simplify the analysis. Help variables like fluxes between the tanks and states like e.g. fuel levels in the tanks and pressure in the tanks are described for each tank. Notice that all variables which are considered as fault variables are represented with f_{index} and normal flows are represented as unknown variables F_{index} .

Wing Tanks

In Figure 7.4 and in Table 7.2 a wing tank is described schematic with, fuel probes (FP), a refueling valve (RV) a pump, a fuel pipe and different fuel flows like e.g. F_{PWL} . Since the two wing tanks are similar, just one figure is showed. Note that the indexes in Table 7.1 introduces notation for variables describing both the left and the right wing tank.

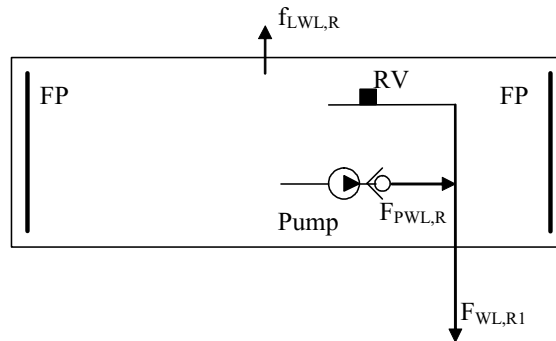


Figure 7.4: Schematic view of the Wing Tanks.

The refueling valve in left/right wing tank can have a leakage, $f_{RVWL,R}$. The transfer pump in left/right wing tank has one possible fault $f_{PWL,R}$ and is controlled by a signal $u_{PWL,R}$. The pump is supervised with a sensor, $y_{PPWL,R}$ which can have a sensor fault $f_{PPSWL,R}$. Fuel level sensors in the tank measure the fuel quantity and are represent with a sensor signal $y_{FPWL,R1,2}$, and a sensor fault $f_{FPWL,R1,2}$.

Tank 1

Figure 7.5 shows a schematic view over tank 1 with fuel probes (FP), a negative g valve (NGV) and a refueling valve (RV). Fuel is transferred from tank 2, F_{21} and from the wing tanks, F_{V1} to tank 1 through fuel pipes. During flight is fuel transferred from tank 1 to the engine, F_{1E} with the boost pump. The boost pump is controlled by a sensor signal u_{BP} and is shown in the lower left corner in Figure 7.5. If the boost pump has a failure, fuel is transferred directly to the engine, f_{EE} but in this analysis of the process during normal flight this fault is not considered. The boost pump is supervised with a pressure sensor, y_{BPS} and six other sensors $y_{FPI,1-6}$ are used to measure the fuel level in the tank.

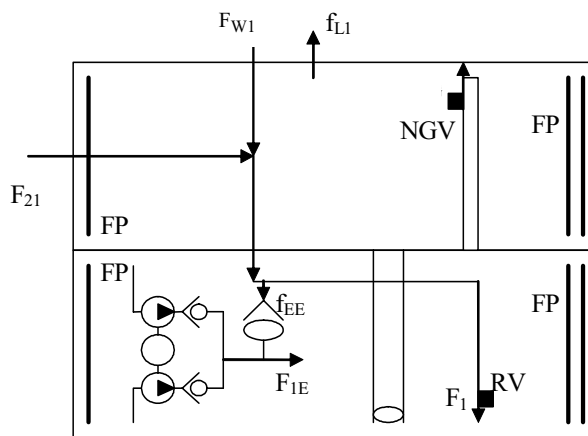


Figure 7.5: Schematic view of Tank 1.

The lower part of tank 1 is the negative-g compartment which makes it possible to fly upside-down, since the fuel stops in this part of the tank and can be transferred to the engine with the upper end of the boost pump in Figure 7.5. The two parts of tank 1 are connected with a big pipe so that the fuel can flow into the negative-g compartment during normal flight. The ventilation pipe, which can not be seen in Figure 7.2 is used to allow air to stream out, when the fuel flows in, to the negative g compartment.

Tank 2

Figure 7.6 shows a schematic view over tank 2, fuel probes (FP), a refueling valve (RV), the refueling connection (RC), a pump and fuel pipes. Fuel can be transferred from tank 2 to the negative-g compartment in tank 1 during flight upside-down. Therefore, the transfer pump, which is controlled by u_{P2} , must have inlet pipes both in the bottom and in the top of the tank. The transfer pump is supervised with a pressure sensor, y_{PS2} and two other sensors $y_{FP21,2}$ are used to measure the fuel level in the tank. Since it is necessary to prevent the pump from sucking air, the two pipes are equipped with negative-g valves.

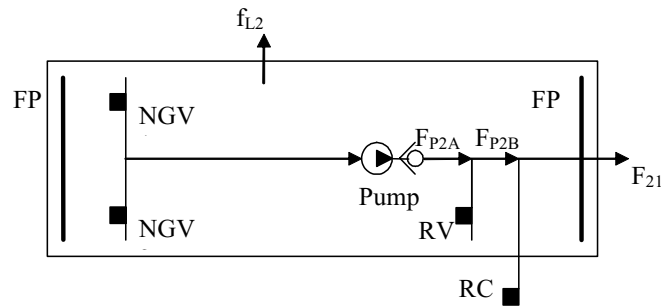


Figure 7.6: Schematic view of Tank 2.

7.4 Structural Model of the Fuel Pump System

All models of the different part of the fuel pump system can now be put together. It is also necessary to include fault variables, control signals and sensor signals used by a the diagnosis system concept.

7.4.1 Limitations in the Structural Analysis

The structural analysis in this Chapter includes some limitations:

- All relations used in the structural model are considered to be linear. This is motivated since the speed of the fuel flows in the system are slow.
- Only single faults are examined, e.g. just one fault at the time occurs in the process to be diagnosed.

7.4.2 Unknown Variables

All unknown variables which are included in the structural model of the fuel pump system are described in Table 7.1. The unknown variables are here fuel levels and flows.

Table 7.1: Unknown variables used to describe the fuel system.

Variable	Description
F_{WL1}	Fuel flow from left wing tank to tank 1
F_{WR1}	Fuel flow from right wing tank to tank 1
F_{21}	Fuel flow from tank 2 to tank 1
F_{W1}	Total fuel flow from the wing tanks
F_1	Total fuel flow from the wing tanks and from tank 2 to tank 1
F_{IT}	Fuel flow into tank 1
F_{IE}	Fuel flow to the engine
F_{PWL}	Fuel flow from the pump in left wing tank
F_{PWR}	Fuel flow from the pump in right wing tank
F_{P2A}	Fuel flow from the pump in tank 2
F_{P2B}	Fuel flow from the pump in tank 2, after the refueling valve
X_{FWL}	Fuel level in left wing tank
X_{FWR}	Fuel level in right wing tank
X_{F1}	Fuel level in tank 1
X_{F2}	Fuel level in tank 2

7.4.3 Sensor Signals

All sensor signals used in the structural model of the fuel pump system during normal flight are shown in Table 7.2. Only derivatives of first order are allowed for the known variables during analysis of the structural model, since derivatives of higher order are assumed to be impossible to estimate.

Table 7.2: Sensor Signals used in the fuel pump system.

Variable	Description
y_{PPSWL}	Pump pressure sensor in left wing tank
y_{PPSWR}	Pump pressure sensor in right wing tank
y_{PPS2}	Pump pressure sensor in tank 2
y_{BPS}	Pump pressure sensor after the boost pump
y_{FP1WL}	Fuel Probe 1 in left wing tank
y_{FP2WL}	Fuel Probe 2 in left wing tank
y_{FP1WR}	Fuel Probe 1 in right wing tank
y_{FP2WR}	Fuel Probe 2 in right wing tank
y_{FP11}	Fuel Probe 1 in tank 1
y_{FP12}	Fuel Probe 2 in tank 1
y_{FP13}	Fuel Probe 3 in tank 1
y_{FP14}	Fuel Probe 4 in tank 1
y_{FP15}	Fuel Probe 5 in tank 1
y_{FP16}	Fuel Probe 6 in tank 1
y_{FP21}	Fuel Probe 1 in tank 2
y_{FP22}	Fuel Probe 2 in tank 2
$y_{V(x,y,z)}$	Load in x,y and z direction during flight

Note that the load in x,y,z direction is merged to one variable, since no exact model is used for the fuel level in relation to the load in this work. All fuel probes are described with just a sensor signal. This is schematic correct but in reality many different disturbances must be considered. The fuel probes used consist of a small cylinder inside of a bigger cylinder which are placed in the tank, see Figure 7.7. The capacitance between the two cylinders is related to how high the fuel level is and can be measured.

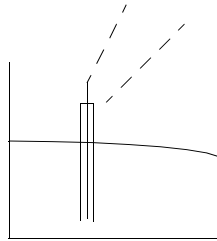


Figure 7.7: A Schematic view of a Fuel Probe in a tank.

7.4.4 Fault Variables

All fault variables used in the structural model are shown in Table 7.3. A fault variable is zero as long as the corresponding fault not has occurred.

Table 7.3: Fault Variables used in the structural model.

Fault	Description
f_{PWL}	Pump failure in left wing tank
f_{PWR}	Pump failure in right wing tank
f_{P2}	Pump failure in tank 2
f_{BP}	Boost pump failure
f_{RVWL}	Leakage in the refueling valve in the left wing tank
f_{RVWR}	Leakage in the refueling valve in the right wing tank
f_{RV2}	Leakage in the refueling valve in tank 2
f_{RC}	Leakage in the refueling connection
f_{LWL}	Leakage from left wing tank
f_{LWR}	Leakage from right wing tank
f_{L1}	Leakage from tank 1
f_{L2}	Leakage from tank 2
f_{BPS}	Sensor fault of boost pump pressure sensor in tank 1
f_{FP1WL}	Sensor fault of fuel probe 1 in left wing tank
f_{FP2WL}	Sensor fault of fuel probe 2 in left wing tank
f_{FP1WR}	Sensor fault of fuel probe 1 in right wing tank
f_{FP2WR}	Sensor fault of fuel probe 2 in right wing tank

f_{FP11}	Sensor fault of fuel probe 1 in tank 1
f_{FP12}	Sensor fault of fuel probe 2 in tank 1
f_{FP13}	Sensor fault of fuel probe 3 in tank 1
f_{FP14}	Sensor fault of fuel probe 4 in tank 1
f_{FP15}	Sensor fault of fuel probe 5 in tank 1
f_{FP16}	Sensor fault of fuel probe 6 in tank 1
f_{FP21}	Sensor fault of fuel probe 1 in tank 2
f_{FP22}	Sensor fault of fuel probe 2 in tank 2
f_{PPSWL}	Sensor fault of pump pressure sensor in left wing tank
f_{PPSWR}	Sensor fault of pump pressure sensor in right wing tank
f_{PPS2}	Sensor fault of pump pressure sensor in tank 2

7.4.5 Control Signals

The fuel pumps are represented with the control variables in Table 7.4, these variables are known variables in the structural analysis. All pumps are described with just a control signal and the flow delivered from the pump. In reality the flow from a pump is depending on the pressure and the pump characteristics.

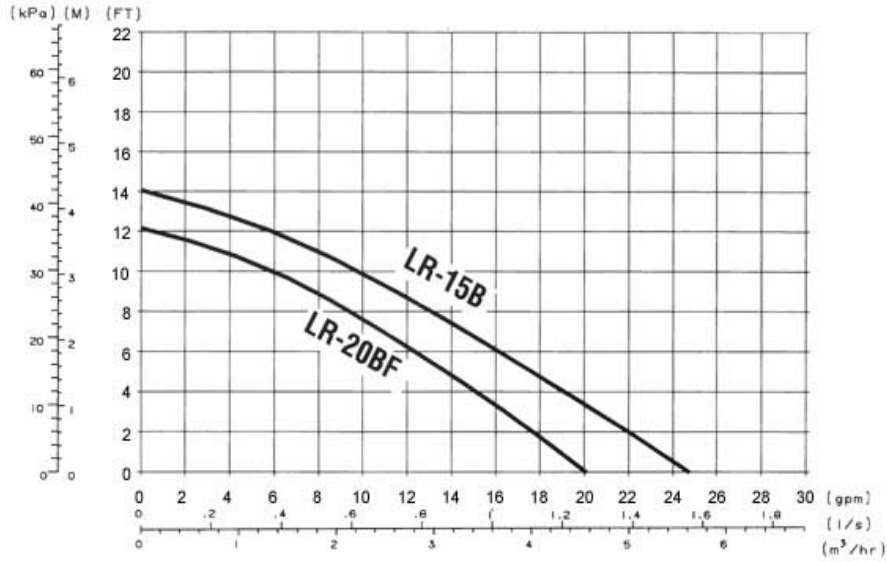


Figure 7.8: Pump Characteristic Curves for two different pumps.

Figure 7.8 shows the pump characteristic curves for two different pumps, with the pump pressure on the y-axis and flow on the x-axis. However a relation between the control signal and the flow from the pump can be estimated, by include a model if the pump characteristic in a future diagnosis system, which is enough to use the relation in a structural model.

Table 7.4: Control signals used to supervise the fuel pump system.

Variable	Description
u_{PWL}	Control signal to the pump in left wing tank
u_{PWR}	Control signal to the pump in right wing tank
u_{P2}	Control signal to the pump in tank 2
u_{BP}	Control signal to the boost pump in tank 1

7.5 System Equations

The relations between the variables that describes the fuel system are considered as *system equations*. In this UAV fuel system, the system equations represent the relations between the fuel flows and the fuel levels in the system.

Table 7.6 shows a structural model including all system equations used for the model of the fuel pump system. All flows that are used in the structural model are represented by just a variable, e.g. F_{2I} . In a future diagnosis system, pipe friction, orifices and pressure differences must be considered.

7.5.1 Control Signals Included in the System Equations

The control signals in Table 7.4 and the unknown variables in Table 7.1 are used to obtain structural system equations, Table 7.5.

Table 7.5: Basic process equations used in the structural model.

Name	Equation	
e_1	$e_1(u_{PWL}, F_{PWL}) = 0$	Left Wing Tank
e_2	$e_2(F_{PWL}, F_{WLI}) = 0$	
e_3	$e_3(F_{WLI}, \dot{x}_{FWL}) = 0$	
e_4	$e_4(u_{PWR}, F_{PWR}) = 0$	Right Wing Tank
e_5	$e_5(F_{PWR}, F_{WRI}) = 0$	
e_6	$e_6(F_{WRI}, \dot{x}_{FWR}) = 0$	
e_7	$e_7(u_{P2}, F_{P2A}) = 0$	Tank 2
e_8	$e_8(F_{P2A}, F_{P2B}) = 0$	
e_9	$e_9(F_{P2B}, F_{2I}) = 0$	
e_{10}	$e_{10}(F_{2I}, \dot{x}_{F2}) = 0$	
e_{11}	$e_{11}(F_{WI}, F_{WRI}, F_{WLI}) = 0$	Transfer Flows
e_{12}	$e_{12}(F_{WI}, F_{2I}, F_I) = 0$	
e_{13}	$e_{13}(F_I, F_{IT}) = 0$	
e_{14}	$e_{14}(F_{IF}, F_{IE}, \dot{x}_{FI}) = 0$	Tank 1
e_{15}	$e_{15}(u_{BB}, F_{IE}) = 0$	

The equations in Table 7.5 describe the fuel pump system, without sensors and faults, and can be considered as a model of the basic process. The first six equations, e_1 to e_6 , in Table 7.5 describe the fuel flows in the left and in the right wing tank, the equations e_7 to e_{10} describe the fuel flows in tank 2 and equations e_{14} and e_{15} the fuel flows in tank 1. Equation e_{11} , e_{12} and e_{13} in Table 7.5 describe the transfer flows between the tanks in the fuel pump system. Note that equations e_3 , e_6 and e_{10} describe the fuel flow out from each tank. An analytical model of the pump equations, e.g. e_1 can be implemented as linear equations:

$$u_{PWL} - F_{PWL} = 0 \quad (7.1)$$

if no fault is present. The fuel flow equations like, e.g. e_2 and e_3 can also be linear modeled, like e.g.

$$F_{PWL} - F_{WLI} = 0 \quad (7.2)$$

and

$$F_{WLI} - \dot{x}_{FWL} = 0 \quad (7.3)$$

if no faults are considered. However, if structural analysis are to be used this modeling must be well founded to prevent introduction of relation which not holds in the process to be analysed.

7.5.2 Faults Included in the System Equations

Different fault variables are now introduced in the system equations to describe different fault behaviors of the fuel pump system.

Table 7.6: Equations including faults used in the structural model.

Name	Equation	
e_1	$e_1(u_{PWL}, F_{PWL}, f_{PWL}) = 0$	Left Wing Tank
e_2	$e_2(F_{PWL}, F_{WLI}, f_{RVWL}) = 0$	
e_3	$e_3(F_{WLI}, \dot{x}_{FWL}, f_{LWL}) = 0$	
e_4	$e_4(u_{PWR}, F_{PWR}, f_{PWR}) = 0$	Right Wing Tank
e_5	$e_5(F_{PWR}, F_{WRI}, f_{RVWR}) = 0$	
e_6	$e_6(F_{WRI}, \dot{x}_{FWR}, f_{LWR}) = 0$	
e_7	$e_7(u_{P2}, F_{P2A}, f_{P2}) = 0$	Tank 2
e_8	$e_8(F_{P2A}, F_{P2B}, f_{RV2}) = 0$	
e_9	$e_9(F_{P2B}, F_{21}, f_{RC}) = 0$	
e_{10}	$e_{10}(F_{21}, \dot{x}_{F2}, f_{L2}, f_{RC}) = 0$	
e_{11}	$e_{11}(F_{W1}, F_{WRI}, F_{WLI}) = 0$	Transfer Flows
e_{12}	$e_{12}(F_{W1}, F_{21}, F_1) = 0$	
e_{13}	$e_{13}(F_1, F_{1T}) = 0$	
e_{14}	$e_{14}(F_{1T}, F_{1E}, \dot{x}_{F1}, f_{L1}) = 0$	Tank 1
e_{15}	$e_{15}(u_{BP}, F_{1E}, f_{BP}) = 0$	

The first six equations, e_1 to e_6 , in Table 7.6 describe the fuel flows in the left and in the right wing tank including two pump faults f_{PWL}, f_{PWR} , two refueling valve faults f_{RVWL}, f_{LWR} and leakages in each wing tank f_{LWL} and f_{LWR} . Equations e_7 to e_{10} describe the fuel flows in tank 2 with a pump fault f_{P2} , a refueling valve fault f_{RV2} , a leakage from the refueling connection f_{RC} and a leakage from the tank f_{L2} . Equations e_{14} and e_{15} describe the fuel flows in tank 1, including a boost pump fault, f_{BP} . Equation e_{11} , e_{12} and e_{13} in Table 7.6 describe the transfer flows between the tanks in the fuel pump system. Note that equations e_3 , e_6 and e_{10} describe the fuel flow out from each tank.

7.5.3 Perfect Matching

The unknown variables in the system equations can be examined to see if the fault free model has equally many equations as unknown variables.

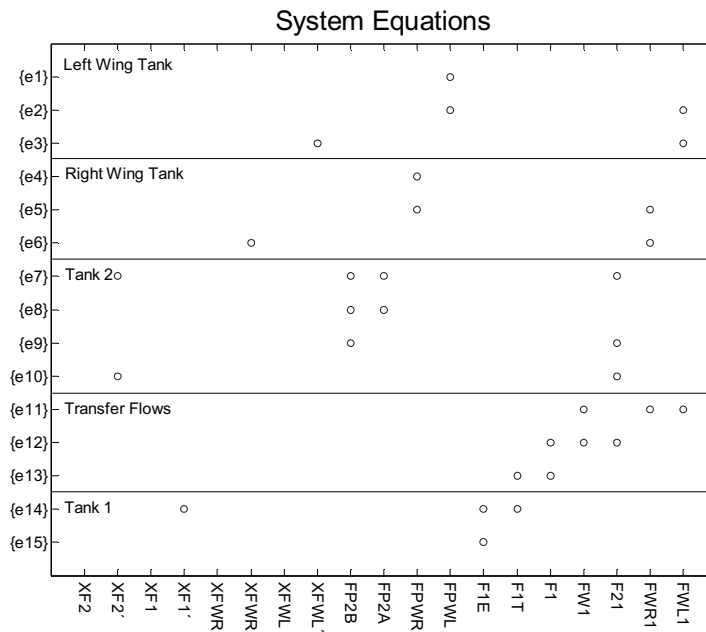


Figure 7.9: Structural model of the unknown variables in the system equations.

The equations in Figure 7.9 are similar to the equations used for the structural model of the system equations in Table 7.5, but only the unknown variables are included. This implies that only relations between the fuel flows and the fuel levels are studied.

A Dulmage-Mendelsohn (DM) permutation is used to return a row permutation p so that if matrix A has full column rank, $A(p)$ is square with non-zero diagonal. This is also called a *maximum matching*.

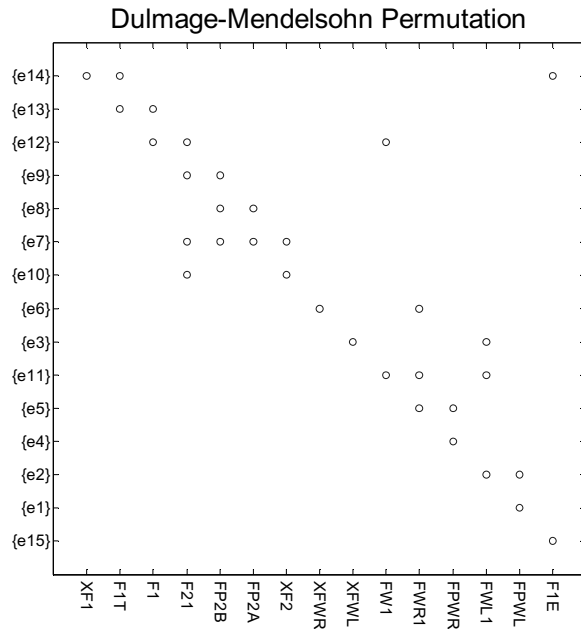


Figure 7.10: Dulmage-Mendelsohn permutation of system equations.

Figure 7.10 shows the structural model from Figure 7.9 of the system equations after the DM permutation. No consideration is taken to derivatives since derivatives of variables can be eliminated just like normal variables. The DM permutation is used to determine that all equations in the physical model are related and the non-zero diagonal shows that the structure of the model has a perfect matching.

7.5.4 Sensor Equations

In this analysis the sensor signals are assumed to represent the physical value of the measured quantity, e.g. if it is 80 kg fuel in a wing tank the value of the sensor signal from the fuel probe is 80 kg. This means the sensor signals can be linearly included in the structural analysis.

Table 7.7: Sensor equations for the sensors used to supervise the pumps.

Name	Equation
e_{16}	$e_{16}(y_{PPSWL}, F_{PWL}, f_{PPSWL}) = 0$
e_{17}	$e_{17}(y_{PPSWR}, F_{PWR}, f_{PPSWR}) = 0$
e_{18}	$e_{18}(y_{PPS2}, F_{P2}, f_{PPS2}) = 0$
e_{19}	$e_{19}(y_{BPS}, F_{IE}, f_{BPS}) = 0$

Equations e_{16} to e_{19} in Table 7.7 describe the sensors used to supervise the pumps. Each equation is a relation between the sensor signal, e.g. y_{PWL} the actual flow, F_{PWL} and a sensor fault f_{PPSWL} . An analytical model of a sensor equation is

$$y_{PPSWL} - F_{PWL} - f_{PPSWL} = 0 \quad (7.4)$$

which means that the sensor signal has the same value as the flow if no sensor fault is present. The pressure after the pump is measured with a pressure sensor. The sensor value and the pump characteristic is used to estimate the flow from the pump.

Table 7.8: Sensor equations for the fuel probes used to measure the fuel levels.

Name	Equation
e_{20}	$e_{20}(y_{FP1WL}, X_{FWL}, f_{FP1WL}, y_{V(x,y,z)}) = 0$
e_{21}	$e_{21}(y_{FP2WL}, X_{FWL}, f_{FP2WL}, y_{V(x,y,z)}) = 0$
e_{22}	$e_{22}(y_{FP1WR}, X_{FWR}, f_{FP1WR}, y_{V(x,y,z)}) = 0$
e_{23}	$e_{23}(y_{FP2WR}, X_{FWR}, f_{FP2WR}, y_{V(x,y,z)}) = 0$
e_{24}	$e_{24}(y_{FP11}, X_{F1}, f_{FP11}, y_{V(x,y,z)}) = 0$
e_{25}	$e_{25}(y_{FP12}, X_{F1}, f_{FP12}, y_{V(x,y,z)}) = 0$
e_{26}	$e_{26}(y_{FP13}, X_{F1}, f_{FP13}, y_{V(x,y,z)}) = 0$
e_{27}	$e_{27}(y_{FP14}, X_{F1}, f_{FP14}, y_{V(x,y,z)}) = 0$
e_{28}	$e_{28}(y_{FP15}, X_{F1}, f_{FP15}, y_{V(x,y,z)}) = 0$
e_{29}	$e_{29}(y_{FP16}, X_{F1}, f_{FP16}, y_{V(x,y,z)}) = 0$
e_{30}	$e_{30}(y_{FP21}, X_{F2}, f_{FP21}, y_{V(x,y,z)}) = 0$
e_{31}	$e_{31}(y_{FP22}, X_{F2}, f_{FP22}, y_{V(x,y,z)}) = 0$

The equations, e_{20} to e_{31} , in Table 7.8 describe the fuel measurements in the tanks for each fuel probe. Note that a sensor fault and the load on the aircraft, $y_{V(x,y,z)}$ is included in each equation. All equations are relations between the sensor signal from the fuel probe, the fuel level, a sensor fault, and the load in x,y and z direction, $y_{V(x,y,z)}$.

7.5.5 Fault Model Equations

A sensor fault can be both static e.g. an offset fault and dynamic e.g. a time delay. In this example the fuel level can not be measured with a very high accuracy. This means that the measurement must take place during a long time and therefore are just offset faults interesting. It is therefore appropriate that the derivative of all sensors faults are set to zero, see Table 7.9.

Table 7.9: Fault model equations of sensor faults.

Name	Equation
e_{32}	$\dot{f}_{FP1WL} = 0$
e_{33}	$\dot{f}_{FP2WL} = 0$
e_{34}	$\dot{f}_{FP1WR} = 0$
e_{35}	$\dot{f}_{FP2WR} = 0$
e_{36}	$\dot{f}_{FP11} = 0$
e_{37}	$\dot{f}_{FP12} = 0$
e_{38}	$\dot{f}_{FP13} = 0$
e_{39}	$\dot{f}_{FP14} = 0$
e_{40}	$\dot{f}_{FP15} = 0$
e_{41}	$\dot{f}_{FP16} = 0$
e_{42}	$\dot{f}_{FP21} = 0$
e_{43}	$\dot{f}_{FP22} = 0$
e_{44}	$\dot{f}_{PPSWL} = 0$
e_{45}	$\dot{f}_{PPSWR} = 0$
e_{46}	$\dot{f}_{PPS2} = 0$
e_{47}	$\dot{f}_{BPS} = 0$

7.6 Analysis of Sensor Configurations

To examine the fault detection and fault isolation possible to achieve by using a diagnosis system to supervise the fuel pump system a structural model over the full process, including all sensors is implemented in Matlab, see Figure 7.11.

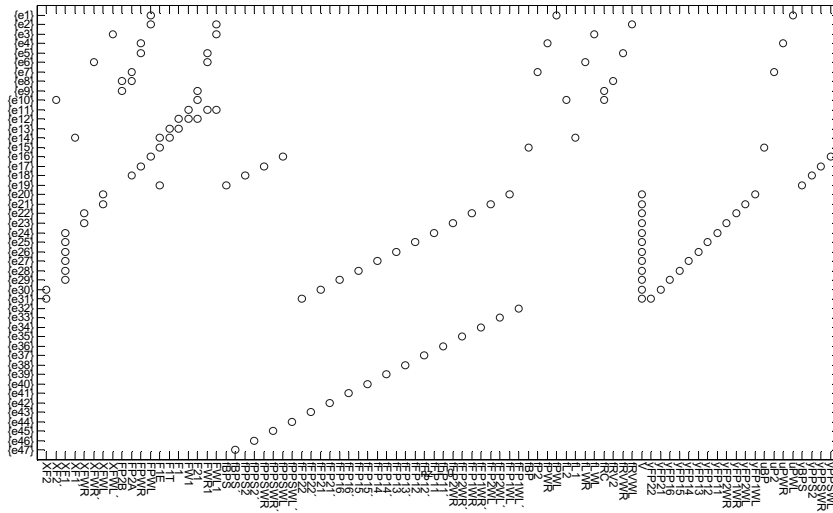


Figure 7.11: Structural model of the fuel pump system, including all sensors.

7.6.1 Sensor Classification

For analysis of different sensor configurations, all sensors are divided into two sets according to the algorithm in Chapter 5, required sensors Y_{SR} and optional sensors Y_{SO} .

Required Sensors

In the structural analysis all fuel probes are considered to be required sensors, because they must be used to measure the fuel level. The boost pump pressure sensor is also required since a boost pump failure is critical and must be detected immediately. This means that Y_{SR} is the set of sensors:

$$\{Y_{FP1WL}, Y_{FP2WL}, Y_{FP1WR}, Y_{FP2WR}, Y_{FP11}, Y_{FP12}, Y_{FP13}, Y_{FP14}, Y_{FP15}, Y_{FP16}, Y_{FP21}, Y_{FP22}, Y_{BPS}\}$$

Optional Sensors

Since a transfer pump failure in tank 2 or in the wing tanks not immediately causes a critical situation, the transfer pump sensors are considered as optional sensors for the diagnosis system. This means that Y_{SO} is the set of sensors:

$$\{Y_{PPSWL}, Y_{PPSWR}, Y_{PPS2}\}$$

```

1 %Fault Classification
2 %FI
3 Fi={'fPWL'},{'fPWR'},{'fP2'},{'fBP'};
4
5 %FD
6 Fd={'fRVWL'},{'fRVWR'},{'fRV2'},{'fRC'},{'fLWL'},{'fLWR'},{'fL1'},{'fL2'}...
7     {'fBPS'},{'fFP1WL'},{'fFP2WL'},{'fFP1WR'},{'fFP2WR'},...
8     {'fFP11'},{'fFP12'},{'fFP13'},{'fFP14'},{'fFP15'},{'fFP16'},...
9     {'fFP21'},{'fFP22'};
10
11 %FN
12 Fn={'fPPSWL'},{'fPPSWR'},{'fPPS2'};
13
14 %Fc
15 Fc={Fi Fd Fn};

```

Figure 7.12: Implementation of the sensor configurations in a Matlab m-file.

Rows 9 to 14 in Figure 7.12 shows how the sensor configurations and the associated sensor faults are implemented in Matlab. The **MakePowerSet** function returns all subsets of optional sensors, Y_O .

7.6.2 Fault Classification

First the algorithm described in Chapter 5 is used to get a incidence matrix for a sensor configuration with all required and optional sensors included. Figure 7.11 shows the structural model of the fuel pump system, including this sensor configuration, i.e. $Y_{SR} \cup Y_{SO}$.

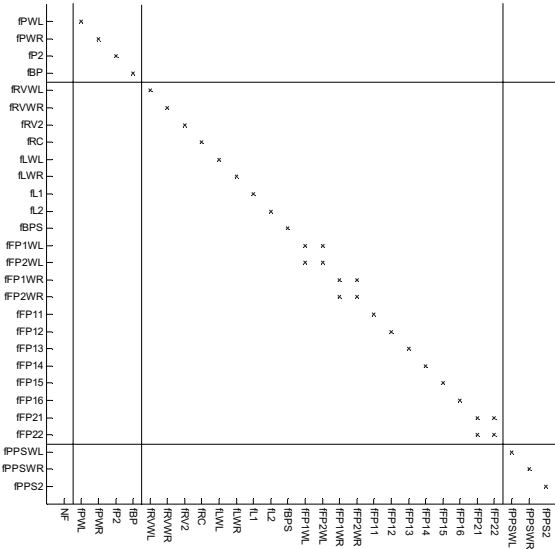


Figure 7.13: The isolability matrix related to the structural model of the fuel pump system during normal flight with all possible sensors included.

Figure 7.13 shows the fault isolability matrix obtained from the structural model of the fuel pump system with all sensors included. In this model there are 3564 MSS sets, when all sensors are included. The isolability matrix shows that all faults can be detected since the first column representing the NF mode is empty. However some sensor faults can not be fully isolated, like e.g. f_{FP1WL} and f_{FP2WL} which can not be isolated from each other.

Faults to be Isolated

It is important to know if a transfer pump in any tank breaks down since no fuel can be transferred from the actual tank if that happens. Therefore a pump fault must be isolated if it occurs. Hence pump faults are prioritized and placed in F_I during the structural analysis, i.e.

$$F_I = \{f_{PWL}, f_{PWR}, f_{P2}, f_{BP}\}.$$

The faults to be isolated are placed first in the isolability matrix showed in Figure 7.13 like in the fault classification described in Chapter 5.

Faults to be Detected

A fault in a refueling valve or a leakage can lead to loss of fuel and must therefore be detected. But since a fault of this kind compared to a pump fault, not must imply a direct loss of all fuel in a tank, it is enough to detect the fault. Therefore all faults connected to valves and all leakages are included in F_D , i.e.:

$$F_D = \{f_{RVWL}, f_{RVWR}, f_{RV2}, f_{RC}, f_{LWL}, f_{LWR}, f_{L1}, f_{L2}, f_{BPS}, f_{FP1WL}, f_{FP2WL}, f_{FP1WR}, f_{FP2WR}, f_{FP11}, f_{FP12}, f_{FP13}, f_{FP14}, f_{FP15}, f_{FP16}, f_{FP21}, f_{FP22}\}$$

The faults to be detected are placed in the middle section of the isolability matrix showed in Figure 7.13.

Not Prioritized Faults

Sensor faults of optional sensors must not be prioritized since these sensors fill no function except to be a part of the diagnosis system. If an not prioritized sensor fault can not be detected the corresponding sensor is unnecessary, since the sensor is not used in any test. Therefore all sensor faults corresponding to optional sensors are handled like not prioritized fault in the structural analysis, i.e.:

$$F_N = \{f_{PPSWL}, f_{PPSWR}, f_{PPS2}\}$$

The not prioritised faults are placed last in the isolability matrix showed in Figure 7.13.

It is important that the corresponding sensor faults are removed from a structural model if a sensor is removed, e.g. if sensor y_{PPSWL} is removed from a structural model f_{PPSWL} must also be removed. This can be done using the function **Eliminatevar** in the matlab implementation.

```

%Fault Classification
%FI
- Fi={{'fPWL'}, {'fPWR'}, {'fP2'}, {'fBP'}};

%FD
- Fd={{'fRVWL'}, {'fRVWR'}, {'fRV2'}, {'fRC'}, {'fLWL'}, {'fLWR'}, {'fL1'}, {'fL2'}...
      {'fBPS'}, {'fFP1WL'}, {'fFP2WL'}, {'fFP1WR'}, {'fFP2WR'}, ...
      {'fFP11'}, {'fFP12'}, {'fFP13'}, {'fFP14'}, {'fFP15'}, {'fFP16'}, ...
      {'fFP21'}, {'fFP22'}};

%FU
- Fu={{'fPPSWL'}, {'fPPSWR'}, {'fPPS2'}};

%Fc
- Fc={Fi Fd Fu};
    
```

Figure 7.14: A fault classification implemented in Matlab.

Figure 7.14 shows how the fault classification can be implemented in Matlab and later this .m file is used to obtain fault isolability matrices with the **PlotFMCat** function.

7.6.3 Evaluation of Sensor Configurations

The full algorithm described in Chapter 5 was used to analyse how different sensor configuration affected the maximal fault detection and fault isolation which could be obtained in the fuel pump system.

```

16 %For each sensorconfiguration
17 for configuration=1:length(configurations)
18     for i=1:length(configurations{1,conf})
19         SM=Eliminatevar(OriginalSM,configurations{1,conf}(i));
20         SM=Eliminatevar(OriginalSM,faultconfigurations{1,conf}(i));
21     end
22     [NFSM]=GetSMDecoupling(SM,{});
23     [NNFSM]=Differentiate(NFSM);
24     [OSM]=OverDetSM(NNFSM);
25     [SSM]=SimplifiedSM(OSM);
26     [MSS_SM]=FindMSSsets(SSM,10)
27     MSS1=MSS_SM;
28     %Fault modes to be decoupled
29     modes={'fBPS','fPPS2','fPPSWR','fPPSUL','fFP22','fFP21','fFP16','fFP15','fFP14',...
30           {'fFP13','fFP12','fFP11','fFP2WR','fFP1WR','fFP2WL','fFP1WL','fEP',...
31           {'fP2','fPWR','fPWL','fL2','fL1','fLWR','fLWL','fERC','fRV2','fRVWR','fRVWL'}};
32     %For each fault mode to be decoupled
33     for i=1:length(modes)
34         [NFSM]=GetSMDecoupling(SM,modes{1,i});
35         NFSM.ylimit=ones(1,length(SM.y));
36         [NNFSM]=Differentiate(NFSM);
37         [OSM]=OverDetSM(NNFSM);
38         if size(OSM,m) ~= 1
39             [SSM]=SimplifiedSM(OSM);
40             [MSS_SM]=FindMSSsets(SSM,10);
41             MSS2=MSS_SM;
42             MSS=mergeSM(MSS1,MSS2,OriginalSM);
43         end
44     end
45     %FM and MSS sets for each sensor configuration
46     FM=getFaultmatrix(MSS);
47     allFM(FMindex)=FM;
48     allMSS(FMindex)=MSS
49     FMindex=FMindex+1;
50     clear MSS
51     save workFM0610 allFM
52     save tobbeMSS0810 allMSS
53 end
    
```

Figure 7.15: Implementation of the algorithm from Chapter 5.

Figure 7.15 shows how the different functions in the Matlab implementation described in Chapter 6 are used to perform the algorithms from Chapter 4 and Chapter 5. All possible sensor configurations Y_{config} are examined and all single fault, modes which can be seen on row 29, are decoupled in each sensor configuration. The result is a fault matrix, row 47 and the corresponding MSS sets, row 48, to each sensor configuration. The fault isolability matrices related to each sensor configuration is to be found in Appendix B. Figure 7.16 shows e.g. that f_{PWL} on row 1 can be isolated from all other fault since only the column representing f_{PWL} is marked in row 1, but f_{P2} on row 3 can not be

isolated from f_{RV2} since column 4, f_{P2} and column 8, f_{RV2} both are marked on row 3.

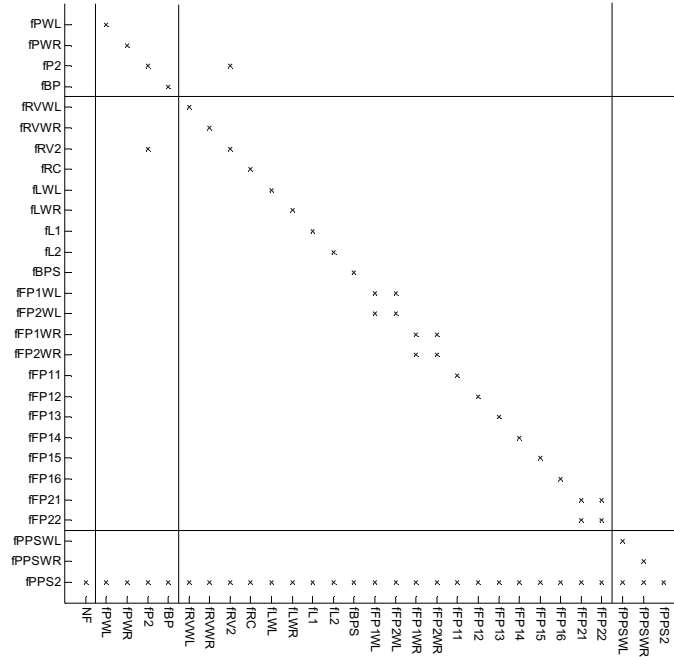


Figure 7.16: Isolability matrix with the optional sensors y_{PPSWL} and y_{PPSWR} included.

The isolability matrix in Figure 7.13 shows that all requirements of the diagnosis system can be fulfilled if all pump pressure sensors, Y_O are included. All faults in F_I in Figure 7.13 are isolated since the isolability matrix only have crosses in the diagonal at the first 4 rows which corresponds to the faults in F_I . All faults in F_D are detected since they can be isolated from the NF mode.

The Isolability matrix in Figure 7.16 shows that the requirements can not be fulfilled if the pump pressure sensor in tank 2, f_{PPS2} is not included, since f_{P2} on row 3 can not be isolated from f_{RV2} . However all other requirements are fulfilled since f_{PWL} , f_{PWR} and f_{BP} in F_I can be isolated and all faults in F_D can be detected.

UAV Fuel System Concept

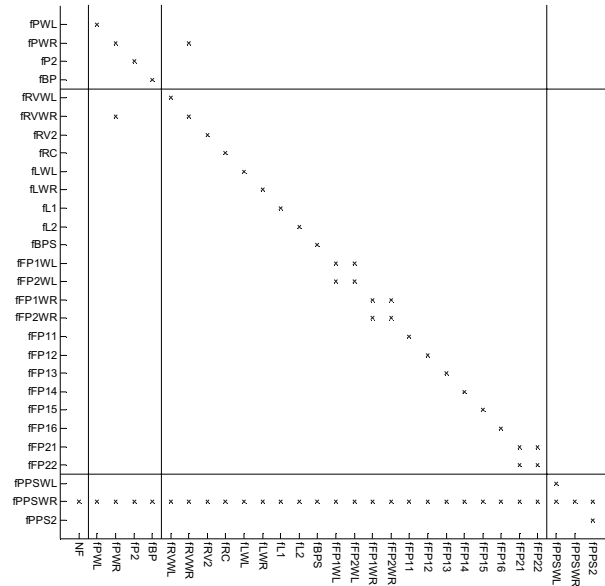


Figure 7.17: Isolability matrix with the optional sensors y_{PPSWL} and y_{PPS2} included.

The isolability matrix in Figure 7.17 shows that the requirements can not be fulfilled if the pump pressure sensor in the right wing tank, f_{PPWR} is not included, since f_{PWR} on row 2 can not be isolated from f_{RVWR} . However all other requirements are fulfilled since f_{PWL} , f_{PW2} and f_{BP} in F_I can be isolated and all faults in F_D can be detected.

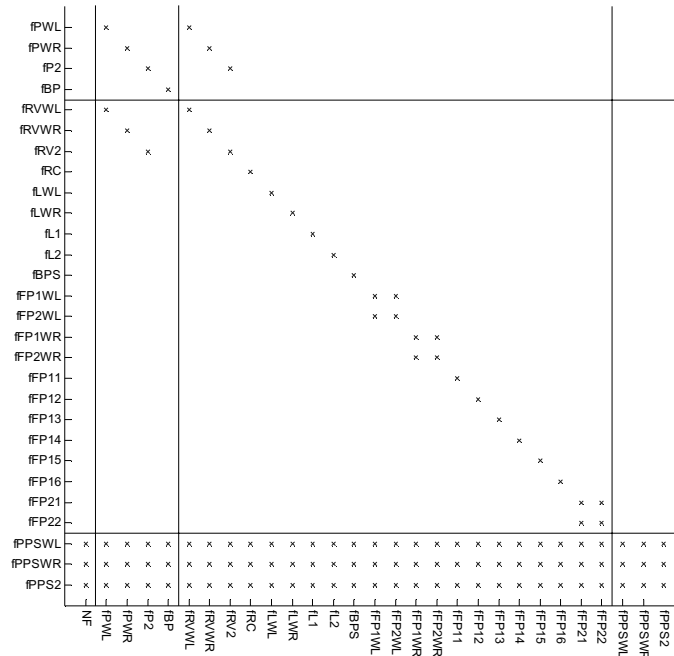


Figure 7.18: Isolability matrix with no optional sensor included.

The isolability matrix in Figure 7.18 shows that the requirements can not be fulfilled if no optional sensors are included. But f_{BP} is isolated from all other faults and all faults in F_D are detected. The only faults which can not be isolated from each other are the transfer pump faults, f_{PWL} , f_{PWR} and f_{P2} which can not be isolated from the transfer valve faults in the same tank.

7.6.4 Conclusions related to Normal Flight Mode

Since the analysis of the different sensor configurations showed that the transfer pump faults and the refueling valve faults in each tank not could be isolated there are three different possibilities:

- Include all pump pressure sensors to get desired fault isolation. This is the only option which fulfills the diagnosis system requirements.
- Accept that transfer pump faults and refueling valve leakages can not be isolated. This alternative can be acceptable here since the two types of faults result in the same type of problem, i.e. disturbances in the fuel

transfer from the actual tank.

- Chose another design of the system which separates the pumps and the refueling valves. This option connects to Figure 2.4 and illustrates how structural analysis can be used to improve the possibilities to design diagnosis systems in new product.

7.7 Summary of the Structural Analysis

This chapter shows how a rough structural analysis contributes with knowledge about which possibilities there are for model-based diagnosis when a new process is to be constructed, already early in the design phase. One of the most important thing handled in the chapter is the choice of relations to be included in the structural analysis, which must secure that no relations impossible to use in a future diagnosis system are introduced.

In this chapter the fault models are included like fault variables in the structural models. The analysis in this chapter shows that simple fault models can contribute to determine isolability possibilities. However if structural analysis is to be used in large scale product development it is probably necessary to obtained one sub model for each fault, since fault sometimes can affect the process in different ways.

Discussion and Conclusions

The objectives with this thesis are to present ideas for use of structural analysis in the early design phase of a new product, to develop a Matlab implementation for structural analysis and to perform a structural analysis of the fuel system in a UAV concept.

8.1 Discussion

During the work with the different parts of this master's thesis several decisions were to be made. In this section some of these questions are discussed.

8.1.1 Discussion Related to the Matlab Implementation

During the work with the Matlab implementations some questions to be answered were:

- How can structural models be implemented in Matlab?
- Which functionality is to be required from the Matlab implementation?
- How can the Matlab implementation be put together?

Since many algorithms already were implemented in Matlab most of the work with the Matlab implementation focused towards completing the existing code with new functionality. A GUI was made to simplify the implementation of Structural Models in the implementation. This GUI fits the previously implemented code. The functions in the implementation are made as independent Matlab functions and can be used for different parts of the algorithms described in this thesis and in [1]. This means that the user can combine functions to perform structural analysis. It is also easy to complete the implementation with new functions if that is required. However this also requires that the user has some knowledge about the algorithms.

8.1.2 Discussion Related to Structural Analysis

During the work with the structural analysis of the fuel system, the most important questions were:

- How detailed can the structural model of the fuel system be?
- Which faults are to be detected in the process?

It is important that no relations impossible to use in a future diagnosis system are introduced during the structural analysis. If this happens the structural analysis can give a too optimistic view of the diagnosis possibilities and harm the design of the diagnosis system.

Another hard task is to identify all faults that must be considered in a future product already in an early structural model. This probably means that some basic knowledge about similar products is required if structural analysis is to be used successfully.

8.2 Conclusions

This master's thesis gives an introduction and hopefully further ideas to how structural analysis can be used in the design phase of a model-based diagnosis system. A framework which is described in Chapter 5 is created to specify the requirements of a diagnosis system and to examine which sensors that are necessary to fulfill these requirements. The Matlab implementation described in Chapter 6 contributes with tools for all steps in the algorithms. By utilizing the Matlab implementation structural analysis can easily be used. Chapter 7 summarizes how structural analysis, the framework in Chapter 5 and the Matlab implementations in Chapter 6 can be used in a large industrial example, a UAV concept. By utilizing methods from the thesis, different sensor configura-

rations in the fuel system are examined, to decide if any sensors can be removed. This decision is handled with the fault isolability matrix where all faults are classified, described in Chapter 5. The analysis shows that it is possible to use the methods presented in this thesis for examination of sensor configurations, and gives a unambiguous answer to which faults that can be detected and isolated by use of a specific sensor configuration. The result in this analysis was that no sensors can be removed if the original requirements put on the diagnosis system must be fulfilled. However, the structural analysis also gives ideas to how the design of the UAV's fuel system can be modified to allow reduction of sensors.

8.3 Future Work

Here follows some ideas to future work:

- To find a good method to decide which MSS sets to be used in a diagnosis system. In this thesis all MSS sets in a model are used to examine the fault isolation and the fault detection. But since it can be thousands of MSS sets, it might not be possible to implement them all in a diagnosis system.
- The fault classification isolability matrix used in this work can be replaced with a matrix which contains "points" at each position in the isolability matrix, to rank the different faults. The matrix can then be used to find a sensor configuration with the highest score.
- The algorithms and the Matlab implementation used in this thesis might have to be improved, if the method is to be used in future large scale projects.

Discussion and Conclusions

Bibliography

- [1] Krysander, M.: *Design and Analysis of diagnosis Systems Utilizing Structural Methods*, ISBN 91-7373-733-X, 2003.
- [2] Nyberg, M, Frisk, E: *Model Based diagnosis of Technical Processes*, Linköping University, 2003.
- [3] Gavel, H,: *Aircraft Fuel System Conceptual Design*, LiTH-IKP-R1330, 2004.
- [4] Krysander, M. and Nyberg, M.: *Structural Analysis for Fault Diagnosis of DAE Systems Utilizing Graph Theory and MSS Sets*, LiTH-ISY-R-2410, Linköping University, Linköping, 2002.
- [5] Kensing, V: *Black-Box Model Development of the JAS 39 Gripen Fuel Tank Pressurization System*, LiTH-ISY-EX-3294-2002, 2002.

Bibliography

- [6] Åman, M: *Method for Diagnosis System Requirement's Prioritization-A Case Study on the Fuel System of JAS 39 Gripen*, LiTH-IKP-Ex-2102, 2003.

Appendix A

This appendix contains a description of the functions and data types implemented in this thesis, some of this information is taken from Chapter 6. All functions are implemented in Matlab and can be used for all steps in the algorithm described in Chapter 4 and Chapter 5. Further information of the Matlab implementation can also be found in Chapter 6 and Chapter 7.

A.1 Objects

Here follows some types of objects used in the matlab implementation.

SM Objects

Structural Models and MSS sets are saved as SM Objects in the Matlab implementation.

```

SM =
m:      [15x28 double]
e:      {1x15 cell}
v:      {1x28 cell}
type:   'Original Structural Model'
name:   'demo UAV'
mode:   {}
x:      {'PA' 'PVU' 'PVT' 'PngT' 'PT3R' 'PT3L' 'PT2' 'PT1'}
y:      {'Pamb' 'PsVU' 'PsVT' 'PsT3R' 'PsT3L' 'PsT2' 'PsT1'}
f:      {1x13 cell}
ylimit: [1 1 1 1 1 1 1 1]
flimit: 0
xlimit: 0

```

Figure A.1: The structure of an SM object.

- **SM.m**
SM.m is a matrix where the rows represent the structural equation and the columns the variables in a structural model. If variable j is included in equation i element (i,j) in the matrix is set to one if the variable is linear included or two if the variable is nonlinear included in the equation. If variable j is not included element (i,j) is zero. Differentiated variables are treated equal to normal variables.
- **SM.e**
SM.e consists of cells with the names and the order of derivative for all relations included in SM, like e.g. $\{\{ 'e1', [1] \}, \{ 'e2', [0] \} \}$. The cell on position i in SM.e consist of the name and order of derivative for the relation represented on row i in SM.m.
- **SM.v**
SM.v consists of cells with the names and the derivative order of all variables included in SM, like e.g. $\{\{ 'PA', [1] \}, \{ 'PVU', [0] \} \dots \}$. The cell on position j in SM.v consists of the name and order of derivative for the variable represented in column j in SM.m.
- **SM.type**
SM.type shows information of what type of structural model it is, e.g. Differentiated Structural Model or Simplified Structural Model. This

information depends on which functions that previously has been perform on the model.information depends on which functions that previous has been perform on the model.

- **SM.name**
SM.name is the name of the model.
- **SM.mode**
SM.mode defines the fault mode of the model by a set of fault variables that are considered to be unknown variables.
- **SM.x**
SM.x is a list with names of all unknown variables in the model.
- **SM.y**
SM.y is a list with names of all known variables in the model.
- **SM.f**
SM.f is a list of all fault variables in the model.
- **SM.ylimit**
SM.ylimit is a list with information about the highest allowed derivative for each known variable in the model. Each element in the matrix corresponds to a known variable in SM.y.
- **SM.flimit**
SM.flimit is not used in this version.
- **SM.xlimit**
SM.xlimit is not used in this version.

SMSS Objects

For some functions in the Matlab implementation *Sortable MSS* (SMSS) objects are used. SMSS objects are transformations of SM objects with another structure.

	Equations			Variables			
Equations/ Variables	A=	2	4	9	B=	3	7
derivative		0	1	2		0	2

Figure A.2: The structure of an SMSS object.

SMSS objects consists of two matrices A and B for each SM Object. The matrices represents the equations and the variables included in an MSS set.

Figure A.2 illustrates an SMSS object. Matrix A shows that equation 2, equation 4 and equation 9 in SM.e are used. Matrix A also shows that equation 4 is differentiated one time and equation 9 differentiated 2 times. Matrix B shows that variable 3 and variable 7 in SM.v are used and that variable 7 is differentiated 2 times. Since SMSS objects just contains the positions of the equations and the variables, elements in SM.e and elements in SM.v. It is necessary to have access to the same SM object which were used to transform the MSS set to an SMSS, when an SMSS is to be transformed back to an SM object.

FM Objects

Fault isolability matrices are saved as FM Objects in the Matlab implementations. These contains of an isolability matrix, FM.m and the names of all faults, FM.f.

```

FM =
    m:    [15x15 double]
    f:    {1x15 cell}

```

Figure A.3: The structure of a FM object.

A.2 Functions

Here follows a description of the functions in the Matlab implementation. The functions can be merged together in a Matlab m-file.

name	<i>outarg1</i> = addMSS (<i>inarg1</i> , <i>inarg2</i>)
input arguments	<i>inarg1</i> : SMSS object, <i>inarg2</i> : SMSS object
output arguments	<i>outarg1</i> : SMSS object
description	The function merges two SMSS objects, <i>inarg1</i> and <i>inarg2</i> and returns an SMSS object, <i>outarg1</i> which contains the set of all MSS sets in the two input arguments without doublets. This function is called by mergeSM .

name `[outarg1, outarg2]=checker(inarg1, inarg2)`
input arguments *inarg1*: SMSS object, *inarg2*: double
output arguments *outarg1*: list of strings, *outarg2*: list of doubles
description The function can be used to examine which variables that are included in an equation. The function Plots all variable names and derivatives, *outarg1* and *outarg2*, included in an equation (*inarg2*) of an SM object (*inarg1*) in the Matlab window.

name name `define_equations(inarg1, inarg2)`
input arguments *inarg1*: Matlab file (string), *inarg2*: string
output arguments
description The function opens a GUI and makes it possible to define a structural model. *Inarg1* is a matlab .mat-file from the function `define_variables`. The variables are stored to a file named after *inarg2*.

name name `define_variables(inarg1)`
input arguments *inarg1*: string
output arguments
description The function opens a GUI and makes it possible to define variables to be used in `define_equations`. The variables are stored to the file named after *inarg1*.

name `outarg1=Differentiate(inarg1)`
input arguments *inarg1*: SM object
output arguments *outarg1*: SM object
description The function performs the differentiation step described in Chapter 4 on a structural model, *inarg1* and returns a differentiated structural model, *outarg1*.

name *outarg1=dmpermSM(inarg1)*
input arguments *inarg1*: SM object
output arguments *outarg1*: SM object
description The function performs a DM permutation, described in Chapter 7 to a structural model, *inarg1* and returns a structural model under the permutation, *outarg1*.

name *outarg1=eliminatevar(inarg1, inarg2)*
input arguments *inarg1*: SM object, *inarg2*: string
output arguments *outarg1*: SM object
description The function eliminates a variable, *inarg2* and all equations which contains the variable from an SM object, *inarg1* and returns an SM object, *outarg1*.

name [*outarg1, outarg2*]=**ExamineIsolability**(*inarg1, inarg2, inarg3*)
input arguments *inarg1*: SM object, *inarg2*: double, *inarg3*: double
output arguments *outarg1*: SM object, *outarg2*: FM object
description The function perform the full MSS algorithm described in Chapter 4 to a structural model, *inarg1*, with *inarg2* as power sets to be decoupled and *inarg3* as search depth in the **findMSSsets** function, e.g.
ExamineIsolability(*SM, 2, 10*) returns a all MSS sets and the isolability matrix from a structural model SM, with all double and single faults decoupled and the search depth set to 10 in the MSS search.

name *outarg1=FindMSSsets(inarg1, inarg2)*
input arguments *inarg1*: SM object, *inarg2*: double
output arguments *outarg1*: SM object
description The function performs the search for MSS step described in chapter 4 and returns a differentiated Structural Model. *inarg2* is the maximal number of equations included in an MSS set to be merged with another MSS set, this reduces the possible MSS sets and not all are found.

name *outarg1=getbackMSS(inarg1, inarg2)*
input arguments *inarg1*: SMSS object, *inarg2*: SM object
output arguments *outarg1*: SM object
description The function converts an SMSS, *inarg1* object to an SM object *outarg1*. an SM object, *inarg2* is used as reference to get the variable and equations names. Obviously the same SM object, *inarg2* which was used when the SMSS object was created with the **makeSMSS** function must be used in **getbackMSS**.

name *outarg1=getFaultmatrix(inarg1)*
input arguments *inarg1*: SM object (MSS set)
output arguments *outarg1*: FM object
description The function returns a fault isolability matrix, *outarg1* without the NF mode from an MSS set, *inarg1*.

name *outarg1=***getFaultmatrix2**(*inarg1*, *inarg2*)
input arguments *inarg1*: SM object (MSS set), *inarg2*: double
output arguments *outarg1*: FM object
description The function returns a fault isolability matrix, *outarg1* without NF mode from a subset of the MSS sets, *inarg1* related to first “*inarg2*” number of MSS sets.

name *outarg1=***getFaultmatrix3**(*inarg1*)
input arguments *inarg1*: SM object (MSS set)
output arguments *outarg1*: FM object
description The function returns a fault isolability matrix, *outarg1* including the NF mode from an MSS sets, *inarg1*.

name *outarg1=***GetNondiffSM**(*inarg1*)
input arguments *inarg1*: SM object
output arguments *outarg1*: SM object
description The function takes an SM object (*inarg1*) and returns an SM object (*outarg1*) where no difference between different derivatives of the same variable is made.

name *outarg1=***GetSMDecoupling**(*inarg1*, *inarg2*)
input arguments *inarg1*: SM object, *inarg2*: list of strings
output arguments *outarg1*: SM object
description The function performs the decoupling step described in Chapter 4 to an SM object, *inarg1* and returns an SM object, *outarg1* with the decoupled faults, *inarg2* transferred from SM.f to SM.x.

name *outarg1*=**MakePowerSet**(*inarg1*, *inarg2*)
input arguments *inarg1*: cell, *inarg2*: double (optional)
output arguments *outarg1*: cell
description The function takes a cell including strings, *inarg1* and returns a cell, *outarg1* including the power set of strings in *inarg1*. If a second argument, *inarg2* is used the *outarg1* only contains the power set, with strings of length *inarg2*. The function is used to find all sensor configurations.

name *outarg1*=**makeSMSS**(*inarg1*, *inarg2*)
input arguments *inarg1*: SM object, *inarg2*: SM object
output arguments *outarg1*: SMSS object
description The function transforms SM objects, *inarg1* to SMSS objects, *outarg1*, using an SM object (*inarg2*) as reference, see also **getbackMSS**.

name *outarg1*=**mergeSM**(*inarg1*, *inarg2*, *inarg3*)
input arguments *inarg1*: SM object, *inarg2*: SM object, *inarg3*: SM object
output arguments *outarg1*: SM object
description The function merges two SM objects (*inarg1* and *inarg2*) to one SM object (*outarg1*). The function uses an SM object (*inarg3*) which must contain all variables used in the two SM objects to be merged as reference. The function uses the functions **makeSMSS** and **getbackMSS**.

name **outarg1=OverDetSM(inarg1)**
input arguments *inarg1*: SM object
output arguments *outarg1*: SM object
description The function returns the part of the SM object (*inarg1*)
that is overdetermined.

name **PlotFM(inarg1)**
input arguments *inarg1*: FM object (without NF mode)
output arguments
description This function plots a fault isolability matrix in a new
window.

name **PlotFMCAT(inarg1,inarg2)**
input arguments *inarg1*: FM object (from **getFaultmatrix**), *inarg2*: fault
classification
output arguments
description This function plots a fault isolability matrix with the
faults classified after the pattern described in Chapter 5.

name **PlotFMNF(inarg1)**
input arguments *inarg1*: FM object (with NF mode)
output arguments
description This function is used to plot FM objects from the func-
tion **getFaultmatrix3**. The fault isolability matrix
includes the NF mode.

name **PlotSM**(*inarg1*)
input arguments *inarg1*: SM object
output arguments
description This function plots an SM object, *inarg1*.

name **PlotSM2**(*inarg1*, *inarg2*)
input arguments *inarg1*: SM object *inarg2*: int
output arguments
description This function plots the first *inarg2* equations in an SM object, *inarg1*.

name **PlotSMV**(*inarg1*)
input arguments *inarg1*: SM object
output arguments
description This function plots an SM object in a new window, with the names of the variables (x-axis) vertical.

name *outarg1*=**removeeqSM**(*inarg1*, *inarg2*)
input arguments *inarg1*: SM object, *inarg2*: 'string'
output arguments SM object
description This function removes an equation from an SM object, *inarg1*. The name of the equation to be removed is taken as *inarg2*.

name	<i>outarg</i> = removevarSM (<i>inarg1</i> , <i>inarg2</i>)
input arguments	<i>inarg1</i> : SM object, <i>inarg2</i> : 'string'
output arguments	SM object
description	This function removes a variable from an SM object without removing any equations. This can be used to e.g. remove a fault.

Appendix B

This appendix contains fault isolability matrices for all possible sensor configurations examined in Chapter 7. The faults are categorized to fit the algorithm in Chapter 5. There is also information about how many MSS sets that were used to get each isolability matrix. Since differentiating is used in the algorithm to find MSS sets new derivatives of equation can be introduced in the differentiating step. This means that the differentiated structural model of a sensor configurations with one or more sensors are removed, can not be seen as subsets of the differentiated model for the sensor configuration with all sensors included. This means that it is possible that more MSS sets can be found in a smaller model, a model with one ore more sensors removed.

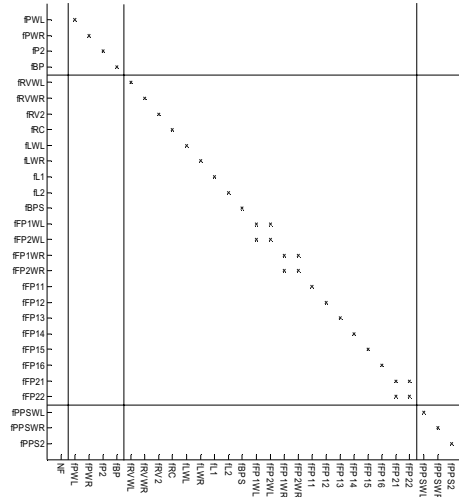


Figure B.1: Isolability matrix with all optional sensors included, totally 3564 MSS sets were found for this sensor configuration.

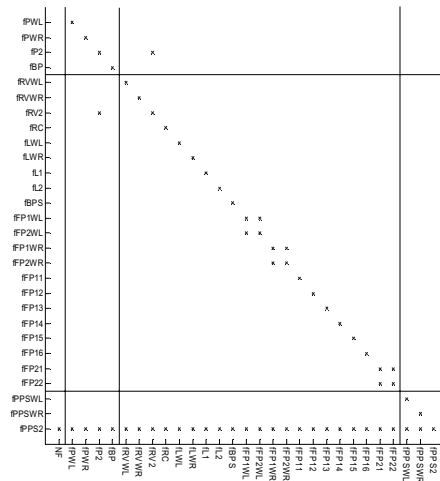


Figure B.2: Isolability matrix with optional sensors y_{PPSWL} and y_{PPSWR} included, totally 2597 MSS sets were found for this sensor configuration.

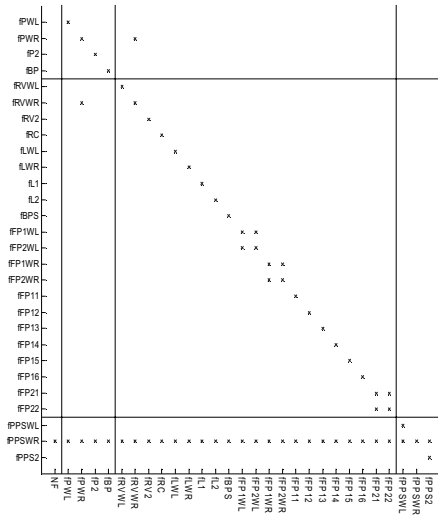


Figure B.3: Isolability matrix with optional sensors y_{PPS2} and y_{PPSWL} included, totally 2597 MSS sets were found for this sensor configuration.

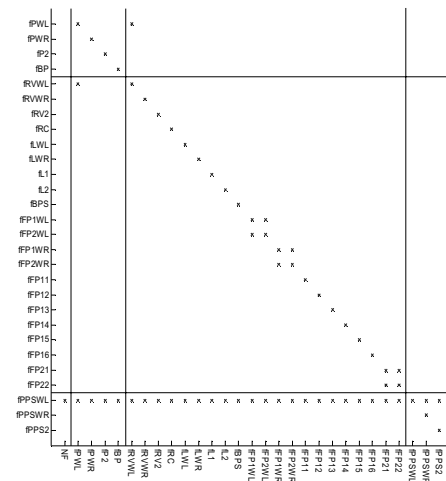


Figure B.4: Isolability matrix with optional sensors y_{PPS2} and y_{PPSWR} included, totally 2597 MSS sets were found for this sensor configuration.

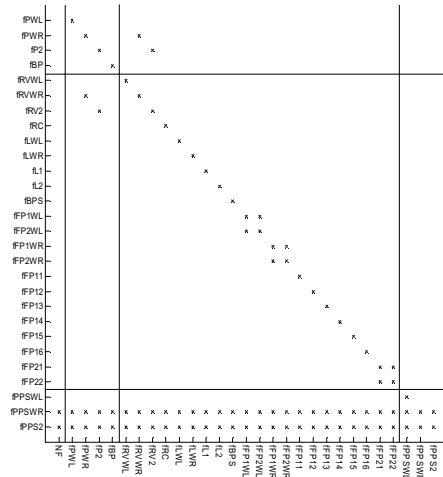


Figure B.5: Isolability matrix with optional sensor y_{PPSWL} included, totally 17206 MSS sets were found for this sensor configuration.

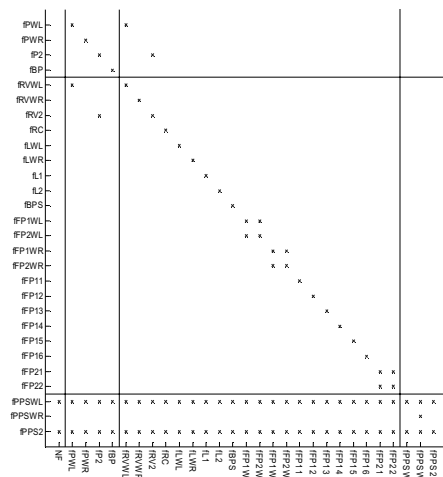


Figure B.6: Isolability matrix with optional sensor y_{PPSWR} included, totally 17206 MSS sets were found for this sensor configuration.



The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances. The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility. According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement. For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: <http://www.ep.liu.se/>

© Tobias Axelsson