Fault Detection and Isolation for a Supermarket Refrigeration System

- Master Thesis -

June 3^{rd} 2009

Project Title:
Fault Detection and Isolation for a Supermarket Refrigeration System

Project Period:
2^{nd} of september 2008 - 3^{rd} of june 2009
(9^{th} and 10^{th} semester of IRS master)

AAUE Supervisor:
Zhenyu Yang

Danfoss Supervisor:
Roozbeh Izadi-Zamanabadi

Group Members:
Anh Tuan Kieu
Karsten Bolsmand Rasmussen

Abstract:
This report is the documentation for the work done by two master students at the IRS (Intelligent Reliable Systems) line at Aalborg University Esbjerg. The project is developed based on a project-proposal from Danfoss A/S, which also supplied a base-model of a refrigeration system. The focus of the report is the development and testing of different methods for detecting and isolating faults in a supermarket display case. The Kalman Filter, Extended Kalman Filter and Unknown Input Observer have all been tested and compared to each other. A simple online parameter estimation is also tested with good initial results, but due to time constraints it has not been completed. Based on the comparisons, a suggestion for a fault detection and isolation system is given, which will be able to both detect and isolate the most often occurring faults in a display case.

Karsten Bolsmand Rasmussen

Anh Tuan Kieu

Preface

This project has been created as documentation for the work done by Karsten Rasmussen and Tuan Kieu, studying at Aalborg University Esbjerg. The project has been developed during 9^{th} and 10^{th} semesters of the IRS (Intelligent Reliable Systems) masters, and as such it is a master thesis.

The project has been made in cooperation with Danfoss - which also offered the original project proposal. During the project we have visited Danfoss to learn the system and to keep the project on track. Our guidance and help has come from Roozbeh Izadi-Zamanabadi, Danfoss and Zhenyu Yang, Aalborg University Esbjerg. We would like to express our thanks for their interest in the project, and their help along the way.

The language in the report is kept as non-technical as possible to appeal to a wide range of readers. With this said, it should be noted that some sections and chapters may require previous knowledge of modeling and fault detection terms. The report should require no previous knowledge of refrigeration-systems, but the subject can have a steep learning curve. As this is the case, anyone who wants to learn more about refrigeration should read the book [Din03] or if a shorter more precise description is sufficient, one could visit the website [WRE].

The report has 3 main chapters, namely Chapter 2 (Thermodynamic Principles), 3 (Modeling) and 4 (Fault Detection and Isolation). These main chapters have their own table of contents, chapter overview and introduction. Obviously all chapters will be based on the previous ones, but the chapter overview and table of contents will help any reader who wishes to read specific chapters instead of the report as a whole. The remaining chapters are structural elements of the report.

References are used where specific comments or formulas need to be documented. In these cases, a square bracket box will appear in the text, holding a key, which is a reference to an element in the bibliography in the back of the report (as can be seen above in the reference to refrigeration material).

Accompanying the report is a CD - or in the case of the online digital version, a zip-file with the CD's contents. This CD or file holds all data, which has been found relevant to the project hand-in. Mainly this is MATLAB, Simulink and Dymola files, a LaTeX-version of the report, and finally the report as a PDF and some of the referenced material. When something is referred to on the CD, square brackets are used to indicate the folder where it can be found, i.e.: [CD/Data/M-Files].

Contents

Appendices

1 Introduction

This project was selected based on a proposal from Danfoss. The focus of the project proposal was to develop, test and implement methods to detect and isolate faults in a supermarket refrigeration system. Furthermore it needs to fit to the different scenarios that such a system will go through in normal operation - primarily the operation in, and shift between night- and daytime.

As many display cases in stores hold very expensive goods (meat especially), it is evident that fault detection in refrigeration systems can save a store from a large economical loss in case of a failure. If no system exists to detect faults, the only way to verify that the system is running correctly, is by manually checking temperature gauges (assuming they show the correct temperature). As most stores have many display cases and the employees are occupied elsewhere, a fault can go unseen for a long time. When detected, the goods may very well be permanently damaged as the low temperatures are used for conservation of food items.

By implementing a method to detect faults in the refrigeration system, an early warning can be given so that the goods may be moved to a cold-storage or another display case before they are damaged. If the fault is not only detected, but also isolated, a repair-man will know what to bring - or the store themselves could perhaps solve the problem. In this way the loss will be minimal, and the problem can be solved quickly.

As a basis for the project, is a model which is created by Danfoss for the purpose of simulating a refrigeration plant. The model, which is described in Section 3.1, is universal and should be adaptable to most store-based refrigeration systems by adjusting the parameters. Because of the simplicity of some elements of the model (this will be described later) and to limit the size of the project, the scope has been narrowed down to focus on creating a fault detection and isolation system for a single display case. The different versions of the display case model which have been used for this project is described in the subsections of Section 3.2. The developed method can be fitted to all

display cases in a store to check for local faults, or perhaps implemented in a larger FDI (Fault Detection and Isolation) setup.

To test the FDI methods, Danfoss have provided a model of a normal setup for use with Dymola, a piece of simulation software. The development- and simulation software used throughout this project is the Simulink-Toolbox within MATLAB, and MATLAB itself for specific calculations. As the interface between Dymola and Simulink, is a Simulink-block which enables communication between the two programs. The main reason for not reproducing the model in Simulink is the fact that the model uses some refrigerant equations, released under the name RefEqns. These have been implemented into, and included with the Dymola model from Danfoss, and will require implementation into MATLAB/Simulink in case the model was reproduced here. Unfortunately RefEqns has only been developed for a prior version of MATLAB/Simulink and as such they do not work with the used version of MATLAB/Simulink. Based on this, it is a requirement either to install Dymola or develop a MATLAB/Simulink model to be able to replicate some of the experiments in this report. To allow the reader some freedom for testing, a series of models which use logged measurements instead of a direct Dymola link has been supplied on the CD, in the subfolders of [CD/Models/Simulink], and un-edited data is available as well [CD/Data/From Dymola].

As Danfoss already have controllers for refrigeration systems this project will not focus on anything within control. The plant model in Dymola needs some kind of control, so Danfoss has provided a simple hysteresis controller for the display case valve, and a similar hysteresis controller has been developed for the compressor. Appendix B gives a description of the compressor controller, and the changes to the original Dymola model in general.

The success criteria for the developed FDI method will be the ability to detect normally occurring faults within a display case and adaptability to change between night- and daytime operation. The faults which are considered in this project are described in Section 4.1 on page 35, and the difference between night- and daytime will be described in the modeling chapter, and tested in Section 4.6.2. Obviously, as this project is developed in cooperation with Danfoss, it will be a bonus if the FDI solution is easily adaptable to a real display case.

2 Thermodynamic Principles

Chapter Overview:

This chapter consists of three main sections. The first section goes over the fundamental terms and the first and second law within thermodynamics which are essential for understanding the rest of the report. Concepts such as heat, enthalpy and heat capacity is introduced and described. The second section gives a description of how refrigeration systems in general work with a focus on the Vapor-Compression cycle which is the most used method for refrigeration. The Pressure-Enthalpy diagram is described followed by the third section. The final section describes the use of refrigeration systems in supermarkets and gives a description of the most frequently used display cases.

Before building a fault detection and isolation system, or even starting to model, it is necessary to have the basics of thermodynamics in place. This chapter is written to give a short overview of the basics on which the refrigeration system is based on. Depending on the readers prior knowledge, the section might be either surplus or insufficient. The reason for not covering every piece of knowledge used is that the field of thermodynamics is huge and generally very complex.

2.1 Fundamental Knowledge

Refrigeration systems can be quite complex in their dynamics, especially as they are primarily based on thermodynamical laws. Thereby some introduction to the primary terms used in the report is required, and so is an introduction to the basic laws on which the system is based.

2.1.1 Thermodynamic Terms

There are some common terms used in refrigeration system, such as latent heat (L), heat transfer ($\dot{Q}$), specific enthalpy (h), and specific heat capacity (C_p). These terms are important in design and performance of vapor-compression refrigeration systems, and is described briefly in this section.

Latent Heat

Latent heat is the amount of heat content of a refrigerant, which is used for absorbing or discharging heat during a phase change from one state to another. During the change, the temperature and pressure do not change. The phase change of the refrigerant in the refrigeration system takes place during evaporation or condensation process and thereby gives or takes heat to/from the refrigerant. By definition the latent heat is energy divided by mass as in the following equation:

$$L = \frac{Q}{m}$$

where L is latent heat for a particular substance in $[\frac{J}{kg}]$. Q is the amount of energy released or absorbed during the change of phase of the substance in joules and m is the mass of the substance.

Enthalpy

Enthalpy is defined as the total energy available within a refrigerant, that is used for conversion into heat at the refrigerants current pressure and temperature. The enthalpy is used to determine the difference in energy between two process states. The calculation of the enthalpy (H) of the system is defined as

$$H = U + P \cdot V$$

where H is total heat content, U is internal energy, P is the pressure, and V is volume.

In order to determine the specific enthalpy (h) for a particular refrigerant, the enthalpy is divided by mass.

$$h = \frac{H}{m}$$

Specific Heat Capacity

Specific heat capacity (C_p) of a substance is a measurement of how much energy required for increase the temperature of the substance by 1 degree. The subscript p in C_p indicates that this is the specific heat capacity measured under constant pressure.

$$C_p = \frac{\Delta Q}{m \cdot \Delta T}$$

where ΔQ is heat added to the substance, m is the mass of the substance, C_p is the specific heat capacity measured in $\left[\frac{J}{kg \cdot K}\right]$ and ΔT is the temperature difference.

Heat Transfer

As discussed above, the two types of latent heat in a refrigeration system are: Absorption of latent heat during evaporation $\dot{Q}_e$ and discharge of latent heat during condensation $\dot{Q}_c$. Refrigerant R-134a is most often used, and serves well as a heat transfer medium in a refrigeration system. It absorbs heat in the evaporator and transfers that heat to the condenser.

Refrigerant R-134a

As mentioned the refrigerant is the heat transfer medium, which allows heat transfer from a hot medium to a cold medium. The main use of the refrigerant is in refrigerators/freezers and air-conditioning systems. R-134a is the refrigerant used in most supermarket refrigeration system. It has been selected for use in refrigeration systems because of inherent advantages that we will notice below:

- R-134a is less ozone depleting, while the other refrigerant such as R-12 contributes to the ozone depletion.
- R-134a is noncorrosive, that is fit for the components used for compressor, piping, evaporator and condenser.
- the refrigerant has a relative high boiling point (-26.6^oC) at a pressure of 1 bar.

In order to analyze the vapor-compression refrigeration cycle, lets begin with some basic concepts of thermodynamics laws. The first-law (conservation of mass and energy) and the second-law of thermodynamics or entropy. These laws are discussed below.

2.1.2 First Law of Thermodynamics

The first law of thermodynamics is applied to each of the components of the vapor-compression cycle (the description of the cycle will be given later in this chapter). It

should be noted that each component in the vapor-compression cycle is considered as a steady-state, steady-flow process. For an ideal vapor-compression cycle, the following relationships are derived:

- Compression: $\dot{W} = \dot{m}_{ref} \cdot (h_2 - h_1)$
- Condensation: $\dot{Q}_c = \dot{m}_{ref} \cdot (h_2 - h_3)$
- Expansion: $h_3 = h_4$
- Evaporation: $\dot{Q}_e = \dot{m}_{ref} \cdot (h_1 - h_4)$

The mass flow rate in the system can be calculated by the refrigerant capacity divided by the specific enthalpy. The mass flow rate is calculated as

$$\dot{m}_{ref} = \frac{\dot{Q}_e}{h_1 - h_4}$$

2.1.3 Second Law of Thermodynamics

The second-law states that there exists a state variable entropy (S). The entropy is the measurement of the dispersal of energy at a specific temperature, i.e. the change of entropy (dS) between two states of the vapor-compression process is given by the heat transfered (dQ_{rev}) divided by temperature (T). The change of entropy is defined by equation

$$dS = \frac{dQ_{rev}}{T}$$

where T is temperature and the subscript *rev* indicates to the system where process are reversible.

The change of entropy is indicated as heat absorbed in the evaporator when dQ_{rev} is positive, tends to the entropy (S) increases, or heat discharged in condenser when dQ_{rev} is negative, tends to the entropy (S) decreases.

2.2 Refrigeration Principles

The basic principles of a refrigeration system is to exchange heat between a cold and a hot zone. Generally, it can be stated that refrigeration systems hold both an evaporator, which is cooled down and a condenser which is heated up. As the term refrigeration is used, it is obvious that the main interest is to cool down an area, even though the same theory can be used for heat pumps, where the hot zone is of interest. The heat exchange is created by using a refrigerant (liquid, gas or both) as a medium for transporting the heat-energy around a closed system.

One method which is frequently used is known as the vapor-compression cycle. By moving the refrigerant into different pressure-zones, the temperatures for the bubble point and dew point are altered, and a change in state of the refrigerant either consumes heat-energy from the environment, or disspells it. Evaporation occurs at a low pressure and temperature, and consumes energy in the form of heat from the environment, where

condensation occurs at a high temperature and high pressure. Consequently it is possible to transfer heat from a cold display case to the surroundings.

2.2.1 Vapor-Compression Cycle

The Vapor-compression system consists of four main components:

- Compressor
- Condenser
- Expansion valve
- Evaporator

Figure 2.1 shows a diagram of a general vapor-compression cycle. The vapor-compression cycle is divided up into two pressure zones, a low pressure evaporating side (blue), and a high pressure condensing side (red). After the expansion valve, mainly inside the evaporator, the low-pressure liquid expands, absorbs heat, and evaporates, changing to a low-pressure vapor at the outlet of the evaporator.

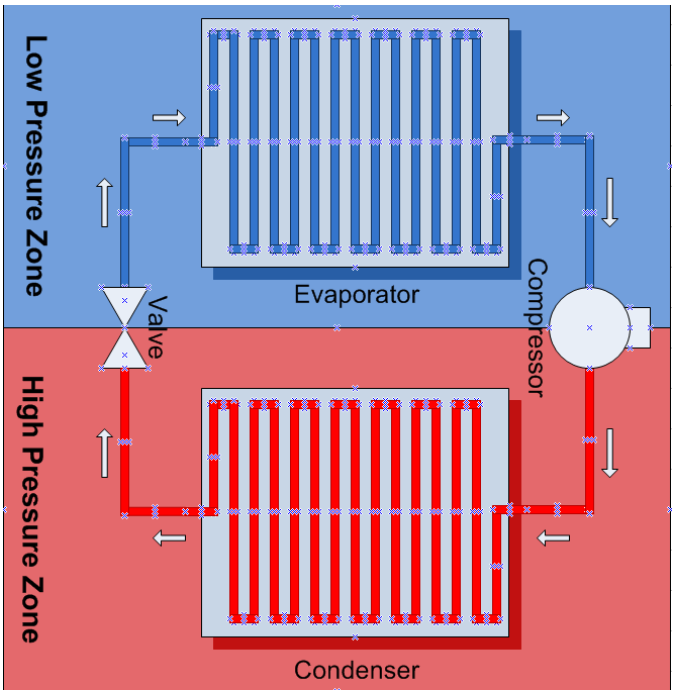


Figure 2.1: Diagram of a simple vapor-compression refrigeration cycle

The compressor takes this gas from the evaporator and raises its pressure and thereby its temperature and discharges it to the condenser. In the condenser, heat from the discharge is transfered to ambient air or to cooling water, causing the refrigerant to change back into a liquid state. The high-pressure liquid refrigerant then passes through an expansion valve to the evaporation side, where the pressure, and therefore the temperature is lower.

The cycle restarts, and the low-pressure liquid refrigerant enters the evaporator, where it again absorbs heat from the refrigerated zone, cooling it further.

To control the process, two actions can be taken. The first method is to control the compressor (on/off or variable), and the other method is to open and close the expansion valve - again either by on/off-control or by using a variable expansion valve.

2.2.2 Pressure-Enthalpy Diagram

Figure 2.2 on the facing page illustrates a vapor-compression cycle on a pressure-enthalpy diagram. The diagram shows the state of the refrigerant at any combination of pressure and enthalpy. The horizontal axis of the diagram has units of specific enthalpy and the vertical axis in units of absolute pressure.

The area underneath the curve, is an area where the refrigerant is a mixture of liquid and vapor. In the area above the curve, and to the left of the critical point (the absolute top of the diagram), the refrigerant is in liquid form, and saturated, and as such there will be no vapor in this area also called Sub-Cooled (SC). In the area to the right of the critical point (still outside the curve), the refrigerant is entirely in a saturated gas-state, also referred to as SuperHeat (SH). The line represents the values of pressure and enthalpy for the states designated the "Bubble point" (left of the critical point) and "Dew point" (to the right of the critical point) respectively.

In the mixture region (underneath the curve) there are horizontal lines indicating that change of phase takes place under constant pressure at constant temperature. Likewise, expansion of the vapor takes place at constant enthalpy. Starting at the compressor suction line, each phase of the refrigeration cycle will be described with reference to the following P-h diagram.

Compression (Going from state point 1 to 2)

The vaporized refrigerant comming from the evaporator is slightly superheated and at a low pressure. When it enters the compressor, the refrigerant is compressed, and discharged into the high-pressure zone. The compression is a reversible adiabatic compression process. The work done by the compressor is required for performing an isentropic compression process and is given by:

$$\dot{W} = \dot{m}_{ref} \cdot (h_2 - h_1) \tag{2.1}$$

where $\dot{m}_{ref}$ is refrigerant massflow rate, and h_1 and h_2 is the specific enthalpy at points 1 and 2.

Condensation (Goint from state point 2 to 3)

Assuming the refrigerant comming into the condenser is hotter that the condenser itself, which in term is hotter than the ambient air, the refrigerant is cooled down. Energy $\dot{Q}_c$ is released to the condenser and thereby the air, and the refrigerant condenses.

$$\dot{Q}_c = \dot{m}_{ref} \cdot (h_2 - h_3) \tag{2.2}$$

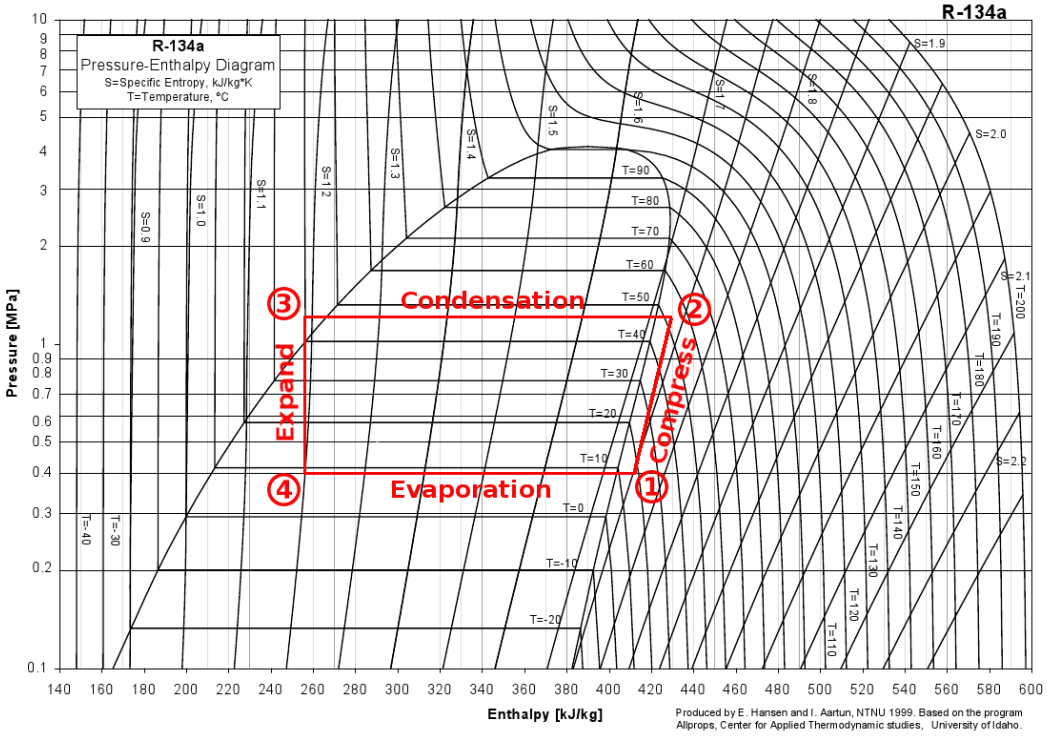


Figure 2.2: Pressure enthalpy diagram for refrigerant R-134a

To ensure no refrigerant passes through the valve in gaseous form, point 3 must be outside the vapor-mix area. In other words the liquid is cooled below its bubble point temperature. If vaporized refrigerant enters the expansion valve, the pressure difference, and thereby flow rate, will suffer significantly.

Expansion (Going from state point 3 to 4)

The refrigerant is now a liquid at high pressure. While it flows through the expansion valve, the pressure is dropped from the condensing pressure to the evaporation pressure, which also entails a drop in refrigerant temperature. The pressure drop, and relatively higher temperature causes the refrigerant to instantly start evaporating. The refrigerant exits the valve at the state 4 as a two-phase liquid-vapor mixture.

Evaporator (Going from state point 4 back to 1)

After the expansion, the refrigerant enters the evaporator. Here, it is assumed that the refrigerant is colder than the evaporator, which in term is colder than the surrounding air. Based on this, the refrigerant will absorb heat from the evaporator and thereby the air. The evaporation itself also consumes energy in the form of heat, cooling the evaporator further.

$$\dot{Q}_e = \dot{m}_{ref} \cdot (h_1 - h_4) \tag{2.3}$$

Point 1 is needs to be superheated, that is, heated above the dew point temperature. The superheating ensures no liquid will enter the compressor, which would cause damage. Unfortunately, the heat transfer coefficient is smaller for gas than for liquid. Therefore efficiency decreases the more superheated the refrigerant becomes. A balance has to be found between protecting the compressor (high superheat) and system efficiency (no superheat).

After evaporation, the low pressure, superheated gas is ready for compression again and the cycle repeats itself.

2.3 Supermarket Refrigeration

There are many different equipment technologies offered in the supermarket today, such as, multiplex direct expansion (DX), distributed, secondary-loop, and advanced self-contained refrigeration systems. The multiplex direct expansion has become the most common technology employed for the supermarket refrigeration. The multiplex DX system will be briefly described in this section.

2.3.1 Multiplex Direct Expansion

As seen in Figure 2.3, the layout of the multiplex system consists of three main components, display case, compressor rack and condenser unit. In this design each display case in the sale area is connected to a suction- and liquid manifold located remotely near the compressor rack.

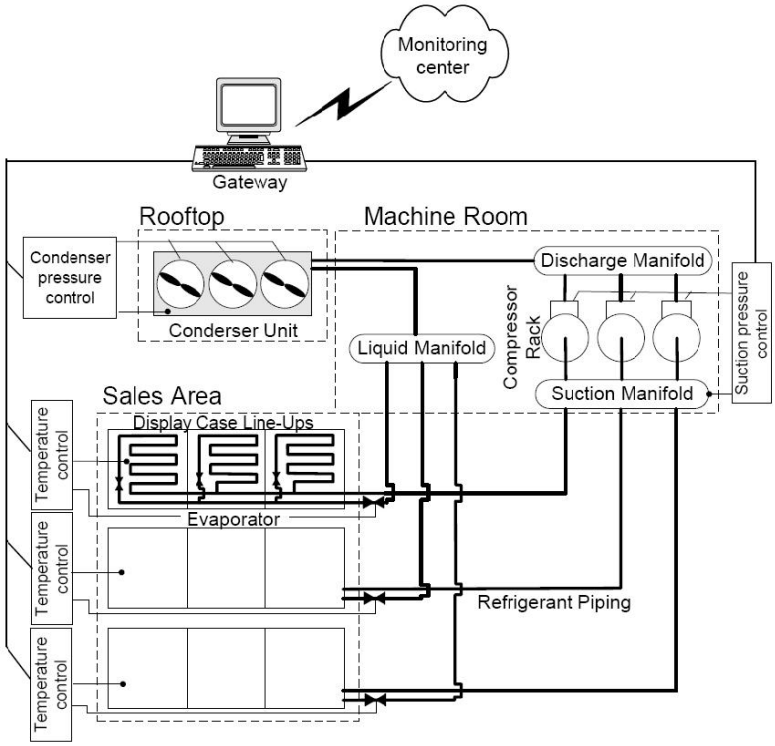


Figure 2.3: Multiplex direct expansion refrigeration system [Lar07]

The condenser unit is mounted outside the building, for example on the rooftop. The condenser unit for the multiplex system is typically air-cooled, where heat is dispelled to the atmosphere by using a dry coil and a rotating fan.

By placing the condenser unit and compressor rack outside the sales area allows for easy to maintance, help to reduce noise level, and better heat rejection.

2.3.2 Supermarket Display Cases

There are many different types of supermarket display cases, the two most common types of display cases used for store goods in the supermarket are single-deck refrigerator for frozen goods, and multi-deck dairy refrigerator for non-frozen goods. Figure 2.4 illustrates these types.

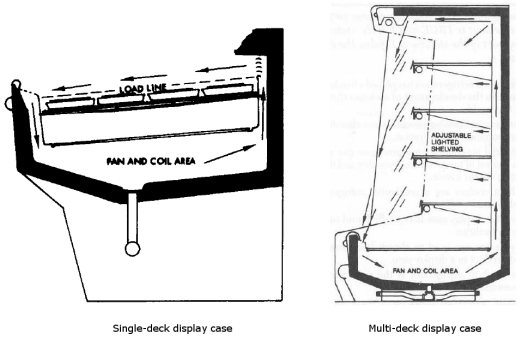


Figure 2.4: Cross view of a single-deck and a multi-deck display case [DH01]

A general overview block diagram of a display case is shown in Figure 2.5, where the evaporator is located below the display case. As the fan rotates, the air is being circulated from the open display case and moved into the evaporator. Inside the display case, a colder air leaves the coil area and creates a flow of chilled air over the goods inside the case.

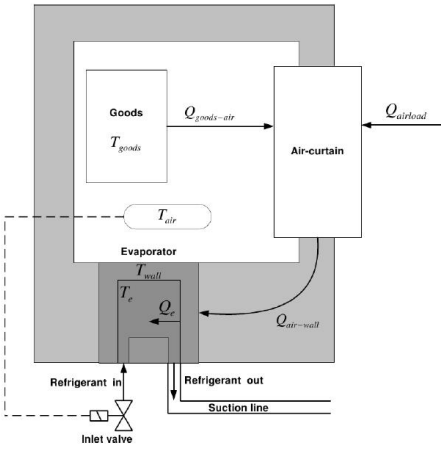


Figure 2.5: Cross section of a display case [LIZW]

3 Modeling

Chapter Overview:

The modeling chapter gives a description of the entire refrigeration plant based on the model given by Danfoss. There is also a description of some concerns related to using the model for FDI and possible improvements. Finally there is a more detailed description of the display case, including the specific models used for the different FDI methods.

To be able to detect errors and/or faults, it is necessary to either work directly with the plant or have a model of the system. Using a model and not the system has a lot of advantages; no need to be physically near the system, no need to wait for real time execution, no possibility of unrecoverable damage and so forth. As Danfoss has supplied a complete model of a refrigeration system within Dymola, there is no need to replicate the model in Simulink which could also give some problems in relations to the RefEqns software. With this said, it is still necessary to create a model in Simulink for the display case as many of the available FDI schemes are model-based and the tool used throughout this project is Simulink.

3.1 The Complete Danfoss model

As the development team at Danfoss also use models, it was possible to acquire a model from them. Unfortunately some elements are not ideally suited for fault detection and isolation. The concerns about the model and their effect are described in section 3.1.5.

The model of the plant can be seen as a group of sub-models based on the separate elements of the system. This gives separate sub-models for the compressor, condenser, evaporator and expansion valve. As the focus of the Danfoss model is the display case the evaporator and the expansion valve has been combined into a single unit called the display case. The expansion valve acts as an actuator and affects the evaporator directly, while none of the other elements have a direct impact on the display case.

As most supermarket refrigeration systems accommodate several compressors and display cases, a suction manifold is included in the Danfoss model. This acts as a combined container for all the refrigerant coming from the display cases, and as a source of refrigerant for all active compressors. In other words, it can be perceived as a reservoir between the display cases and compressors.

An illustration of a supermarket refrigeration system can be seen in Figure 3.1 on the facing page.

3.1.1 The Display Case

The dynamics of the display case are affected by multiple items. As mentioned it holds the evaporator and expansion valve, but it also contains both air and goods inside the case, and the walls between the evaporator and display-area. The display case can be described by four states, three temperatures and one mass-flow. It should be noticed that a temperature sensor is normally placed in a display case, but obviously not in the goods. The sensor measures the air-temperature inside the case, but the temperature of the goods will affect the air temperature which is why it is included in the model.

The description of the variables used for the model equations are as can be seen in Table 3.1 on the next page. Notice that all used units are SI-units, and are taken from [LIZW].

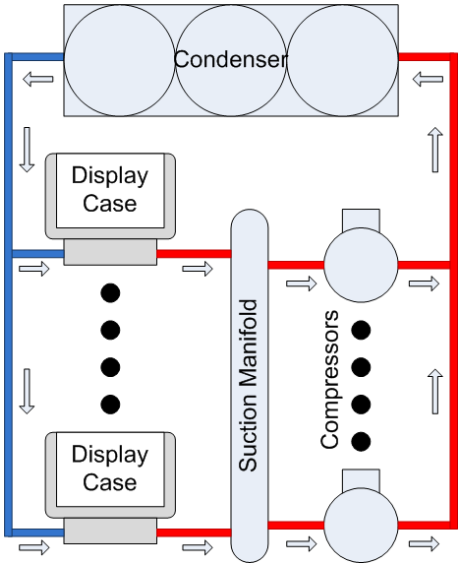


Figure 3.1: Illustration of Refrigeration-Setup

Variable	Unit	Description
T_{sub}	C	The temperature of the goods, air or wall determined by the subscript *sub*.
M_{refrig}	kg	Mass of refrigerant in the evaporator.
P_{suc}	bar	Pressure reading the suction manifold.
V_p	N/A	Position of expansion valve, 0=closed, 1=open.
$Q_{airload}$	J	Heat Flow from ambient air.
M_{sub}	kg	The mass of the element denoted by the subscript *sub*.
$C_{P_{sub}}$	$\frac{J}{kg \cdot K}$	The specific heat capacity of the element denoted by the subscript *sub*.
$UA_{sub_a \to sub_b}$	$\frac{J}{s \cdot K}$	The heat transfer coefficient from element sub_a to sub_b.
$\dot{Q}_{sub_a \to sub_b}$	$\frac{J}{s}$	Power/Watt/Heat flow rate from sub_a to sub_b.

Table 3.1: Model Equations Legend

T$_{goods}$

It is assumed the primary cooling of the goods is done by the air inside the case, and that any direct heat transfer between the display case and goods can be neglected. Based on this assumption, the following equality can be stated for the goods:

$$\dot{T}_{goods} = \frac{-\dot{Q}_{goods \to air}}{M_{goods} C_{P_{goods}}} \tag{3.1}$$

where $\dot{Q}_{goods \to air}$ can be described by:

$$\dot{Q}_{goods \to air} = UA_{goods \to air} \left(T_{goods} - T_{air} \right)$$

$\mathbf{T}_{air}$

The temperature of the air inside the display case is affected by the goods, the wall and obviously the air inside the store, which can be written as the equation

$$\dot{T}_{air} = \frac{\dot{Q}_{goods \to air} + \dot{Q}_{Airload} - \dot{Q}_{air \to wall}}{M_{air} C_{P_{air}}} \tag{3.2}$$

where $\dot{Q}_{Airload}$ should be seen as a value representing the heat flow rate from the ambient air to the air in the display case. This value changes during normal operation, primarily as a function of the temperature of the ambient air. The change is most dramatic when the display cases are covered up for the night and when the store is closed. $\dot{Q}_{air \to wall}$ is described by the following formula:

$$\dot{Q}_{air \to wall} = UA_{air \to wall} \left(T_{air} - T_{wall}\right)$$

$\mathbf{T}_{wall}$

The wall obviously transfers heat between the air and the evaporator. If we name the evaporator-energy $\dot{Q}_e$ the equation will be:

$$\dot{T}_{wall} = \frac{\dot{Q}_{air \to wall} - \dot{Q}_e}{M_{wall} C_{P_{wall}}} \tag{3.3}$$

and $\dot{Q}_e$ is described by the following formula:

$$\dot{Q}_e = UA_{wall \to refrig} \left(M_{refrig}\right) \left(T_{wall} - T_e\right)$$

Here, T_e is the evaporation temperature, which is a function of suction pressure (P_{suc}) in relation to which refrigerant is used. This value can, given the pressure, be calculated by using RefEqns [REQ], or estimated by the function:

$$T_e = -4.3544 \cdot P_{suc}^2 + 29.2240 \cdot P_{suc} - 51.2005$$

$UA_{wall \to refrig} \left(M_{refrig}\right)$ is not a constant as the other values with the UA prefix. In this case, it is a function of the mass of the remaining refrigerant inside the evaporator, and can be described by the formula:

$$UA_{wall \to refrig} \left(M_{refrig}\right) = UA_{wall \to refrig_{max}} \frac{M_{refrig}}{M_{refrig_{max}}}$$

where $UA_{wall \to refrig_{max}}$ is the heat transfer coefficient with the evaporator filled up, and $M_{refrig_{max}}$ is the total mass of the full evaporator.

$\mathbf{M}_{refrig}$

The final state was as mentioned the mass flow of refrigerant through the evaporator. As the state is dependant on the position of the valve, several equations need to be introduced. The formula from [LIZW] is

$$\dot{M}_{refrig} = \begin{cases} \frac{M_{refrig_{max}} - M_{refrig}}{\tau_{fill}}, & \text{if } valve = 1 \\ \frac{-\dot{Q}_e}{\Delta h_{lg}}, & \text{if } valve = 0 \text{ and } M_{refrig} > 0 \\ 0, & \text{if } valve = 0 \text{ and } M_{refrig} = 0 \end{cases} \tag{3.4}$$

but Danfoss has later corrected the discription to

$$\dot{M}_{refrig} = \begin{cases} \frac{M_{refrig_{max}} - M_{refrig}}{\tau_{fill}} - \frac{\dot{Q}_e}{\Delta h_{lg}}, & \text{if } valve = 1 \\ \frac{-\dot{Q}_e}{\Delta h_{lg}}, & \text{if } valve = 0 \text{ and } M_{refrig} > 0 \\ 0, & \text{if } valve = 0 \text{ and } M_{refrig} = 0 \end{cases} \tag{3.5}$$

where τ_{fill} is a constant describing the filling time of the evaporator with the valve open and Δh_{lg} is the specific latent heat of the remaining liquified refrigerant in the evaporator, which can be calculated using the RefEqns [REQ] again, or described as a nonlinear function of the evaporation pressure by the function:

$$\Delta h_{lg} = \left(0.0217 \cdot P_{suc}^2 - 0.1704 \cdot P_{suc} + 2.2988\right) \cdot 10^5$$

Unfortunately, a lot of development was completed before the corrected matrix was presented to us. Thereby it would require a lot of work to be re-done if the change was implemented in the Dymola model. Based on this, it was decided to move along with the old description. As equation 3.4 is not suited for use in modelling (because it has multiple equations), it needs to be rewritten in the correct form. The equation will then be:

$$\dot{M}_{refrig} = V_p \cdot \left(\frac{M_{refrig_{max}} - M_{refrig}}{\tau_{fill}}\right) + (V_p - 1)\left(\frac{\dot{Q}_e}{\Delta h_{lg}}\right) \tag{3.6}$$

where V_p is the position of the valve which will be "1" for open, and "0" for closed.

3.1.2 The Suction Manifold

The suction manifold dynamics are relatively simple, and can be modelled by a single state. The interesting element of the suction manifold is the pressure, which is dependant on the flow in from the display cases, the flow out through the compressor and the volume of the suction manifold. The mass balance can be written as:

$$\dot{P}_{suc} = \frac{\dot{M}_{in-suc} + \dot{M}_{ref,const} - \dot{V}_{comp} \cdot \rho_{suc}}{V_{suc} \cdot \frac{d\rho_{suc}}{dP_{suc}}}$$

where

- $\dot{M}_{in-suc}$ is the total mass flow rate in from the display cases.
- $\dot{M}_{ref,const}$ is a constant flow into the suction manifold originating from unmodeled refrigeration units (ie. cold storages).
- $\dot{V}_{comp}$ is the volume flow out produced by the compressor.
- ρ_{suc} is the density of the refrigerant inside the manifold and can be approximated by $\rho_{suc} = 4.6073 \cdot P_{suc} + 0.3798$.
- V_{suc} is the volume of the suction manifold
- and $\frac{d\rho_{suc}}{dP_{suc}}$ is the pressure derivative of the density which can be approximated by $\frac{d\rho_{suc}}{dP_{suc}} = -0.0329 \cdot P_{suc}^3 + 0.2161 \cdot P_{suc}^2 - 0.4742 \cdot P_{suc} + 5.4817$.

As previously mentioned, SI-units have been used.

3.1.3 The Compressor

The compressor is, like the suction manifold, represented by a single function. Here, the flow through the compressor is the interesting part, and is described by:

$$\dot{V}_{comp,i} = Comp_i \cdot \frac{1}{100} \cdot V_{sl} \cdot \eta_{vol}$$

where i denotes the compressor-number, $Comp_i$ is the i'th compressors capacity, V_{sl} is the total displacement volume and η_{vol} is the constant volumetric efficiency. As the equation is only for a single compressor, the flow from all compressors must be summed up to give V_{comp}.

$$\dot{V}_{comp} = \sum V_{comp,i} \quad i = 1, 2 \cdots, n$$

where n is the total amount of compressors.

3.1.4 The Condenser

The condenser has no dynamics in the Danfoss model. It is assumed to be ideal, and is modelled to give a fixed output (temperature, pressure, flow). This is one of the concerns mentioned in the following section.

3.1.5 Concerns About The Model

Before mentioning the concerns related to the model, it should be underlined that it was built to simulate a working system at the operating point, while keeping it reasonably linear. For this purpose the model is ideal and it cannot be faulted. Unfortunately, the operating-terms for this project are not ideal, as it deals with faults. This leads to some problems that should be known, or even better, solved.

Even though the given model represents the systems behavior quite well in the defined work-area, it has some issues which make it inconvenient for more precise modeling, especially outside the desired operating-point. This is evident when modeling system errors, as these will usually make the system deviate from the designated work-area.

The impacts of any and all of the used simplifications are not researched in relation to this project. If one wishes to use the model to simulate normal operation, Danfoss have verified the dynamics to resemble a real refrigeration plant. The concerns in this project are purely related to the lack of detail in faulty conditions.

An extreme example of the problems with this models ability to be used for error-detection is a compressor failure, which is described in Table 3.2. This is due to many issues of which the majority is described in this section. The erroneous behavior of this has been confirmed by fixing the compressor speed to zero in the model, and then running the simulation.

General issues

Often, the real system has both temperature and pressure sensors in a lot of places. This is because both temperature and pressure changes all the way through the system.

	Real System	**Modeled System**
Pressure	Pressure at each side of the expansion valve equalizes given enough time.	Pressure fixed at high pressure side, pressure rising to extremes at low pressure side.
Flow	Will fall until both sides are at equal pressure, then stop.	No change in flow.
Temperature	Will stabilize at ambient temperature over time.	High pressure side constant in temperature, low pressure side including goods stabilize at over 100 degrees.

Table 3.2: Example of total compressor failure - Expansion valve assumed open.

The model only calculates these values once for each side of the refrigeration system (high- and low pressure zones) and then feeds those forwards, which does not give a precise interpretation of the system. Another general issue, is the fact that enthalpy is not calculated - and therefore not used - anywhere in the model. Instead temperature is used, which holds no value relating to the amount of heat-energy in the refrigerant.

The Compressor

The model of the compressor is not based on any real compressor dynamics, but instead an efficiency parameter, a theoretical maximum and a value representing at which speed the compressor is running. The values are multiplied and the result is a mass flow rate which is linear, but appears not to be specific for any type of compressor.

The Condenser

One of the four main elements of the refrigeration system, namely the condenser, is actually not a dynamic model at all. Its functionality is just assumed to be ideal and in the operating point at all times. It has no dependencies on the other elements of the system, and thereby actually breaks the system loop. Given a known condensation temperature, and a desired sub-cooling degree, the pressure is calculated using refrigerant equations. Any input to the condenser is ignored, and the output will be the same at any given time.

The Display Case

The display case is the most extensive part of the model. It holds two of the main elements, and is obviously important when modeling the goods temperature, which the model was made for. The display case is base for several concerns. One concern is the fact that the ambient airs effect on the air inside the display case is a fixed load, and not temperature-related. Furthermore the flow through the expansion valve is not pressure-related, but again just a fixed value depending on how much the valve is opened.

As a final note to the display case, it should be noticed that it is assumed that the goods have no heat transferred from the walls themselves, and that the flow of refrigerant

into the display case is created as an intuitive filling-time formula, and not by using thermodynamic laws.

3.1.6 Possible Improvements

To improve the model for use in FDI, several modifications could be made, but it should be noted that each improvement will likely make the model increasingly non-linear. Before solutions are suggested, it should be emphasized that the suggested solutions are exactly that - suggestions. The impacts of the concerns have as mentioned not been verified in relation to this project, and the solutions have not been tested or implemented in any way. Some may not affect the model sufficiently to justify implementation and other problems may have been overseen completely.

A way to improve the model would be to calculate the enthalpy. This should ideally be the main variable of the model, as it should influence the system dynamics. By including this, it is possible to ”input” energy into the refrigerant and ”remove” it again.

To solve the problems with the condenser it obviously is necessary to create a model that is dynamic and not static. Assuming the above mentioned enthalpy improvement has also been implemented, the condenser should lower the enthalpy in the refrigerant, while dynamically calculating pressure based on the flow in and out of the condenser, and temperature based on ambient air temperature and -flow and consumed heat-energy from the refrigerant.

The compressor model should be verified to resemble a real compressor, or better yet, be modeled with the applicable thermodynamic laws and formulas to resemble a real system compressor more precisely. This will also enable the option of simulating specific errors in the compressor - i.e. a broken impeller-blade.

Finally, the display case could be improved regarding the airload- and flow-values. The airload should not be a fixed value, but again based on thermodynamic laws, and give a dynamic load depending on the temperature of the ambient air. Another important fix, is changing the flow through the expansion valve to be dynamic, and dependant on the pressure in the evaporator and the condenser, or more precisely, before and after the valve.

On the included CD the Dymola model from Danfoss can be found in two versions [CD/Models/Dymola]. The original one from Danfoss, and the final version used in this project with small adjustments to enable fault-introduction. For an overview of the changes in the model, see Appendix B.

3.2 The Display Case Model

As there is a working interface between Dymola and MATLAB and the focus is on the display case, there is no reason to build a model of the complete system in Simulink. Instead a connection is made between the two programs, and then the Dymola-model will play the role of the plant/system. With this approach chosen, only models for fault detection and isolation are needed, as many schemes are model-based. The Linear Kalman Filter needs a linear model, the Extended Kalman Filter uses the non-linear

Variable	Value	Unit	Variable	Value	Unit
M_{goods}	200	kg	$C_{P_{goods}}$	1000	$\frac{J}{kg \cdot K}$
M_{air}	50	kg	$C_{P_{air}}$	1000	$\frac{J}{kg \cdot K}$
M_{wall}	260	kg	$C_{P_{wall}}$	385	$\frac{J}{kg \cdot K}$
$M_{refrig_{max}}$	1	kg	τ_{fill}	40	s
$UA_{goods \rightarrow air}$	300	$\frac{J}{s \cdot K}$	$UA_{air \rightarrow wall}$	500	$\frac{J}{s \cdot K}$
$UA_{wall \rightarrow refrig_{max}}$	4000	$\frac{J}{s \cdot K}$			

Table 3.3: Established values for the used refrigeration plant [LIZW]

model, and the Unknown Input Observer uses a reduced order (T_{air}, T_{wall} and M_{goods}) linear model.

For the display case model simulated in Dymola, there are several variables. These range from the mass of the goods to the heat transfer coefficient between the air and the evaporator wall, and can (for this specific setup) be seen in Table 3.3.

3.2.1 Continuous Non-linear Model

The non-linear model, is essentially just the difference equations described in section 3.1.1. It is used for the Extended Kalman Filter, as this is based on a non-linear model. For good measure, the equations will be reformatted into a structure where the states are isolated where possible.

First, T_{goods}, based on equation 3.1:

$$\begin{aligned}
\dot{T}_{goods} &= \frac{-UA_{goods \rightarrow air}\left(T_{goods} - T_{air}\right)}{M_{goods}C_{P_{goods}}} \\
\dot{T}_{goods} &= -\frac{UA_{goods \rightarrow air}}{M_{goods}C_{P_{goods}}}T_{goods} + \frac{UA_{goods \rightarrow air}}{M_{goods}C_{P_{goods}}}T_{air}
\end{aligned}$$

Now T_{air} which is based on equation 3.2:

$$\begin{aligned}
\dot{T}_{air} =& \frac{UA_{goods \rightarrow air}\left(T_{goods} - T_{air}\right) + \dot{Q}_{airload} - UA_{air \rightarrow wall}\left(T_{air} - T_{wall}\right)}{M_{air}C_{P_{air}}} \\
\dot{T}_{air} =& \frac{UA_{goods \rightarrow air}}{M_{air}C_{P_{air}}}T_{goods} - \frac{UA_{goods \rightarrow air} + UA_{air \rightarrow wall}}{M_{air}C_{P_{air}}}T_{air} \\
&+ \frac{UA_{air \rightarrow wall}}{M_{air}C_{P_{air}}}T_{wall} + \frac{\dot{Q}_{airload}}{M_{air}C_{P_{air}}}
\end{aligned}$$

T_{wall}, based on equation 3.3:

$$\dot{T}_{wall} = \frac{UA_{air\rightarrow wall}\left(T_{air} - T_{wall}\right) - UA_{wall\rightarrow refrig_{max}}\frac{M_{refrig}}{M_{refrig_{max}}}\left(T_{wall} - T_e\right)}{M_{wall}C_{P_{wall}}}$$

$$\dot{T}_{wall} = \frac{UA_{air\rightarrow wall}}{M_{wall}C_{P_{wall}}}T_{air} - \frac{UA_{air\rightarrow wall} + UA_{wall\rightarrow refrig_{max}}\frac{M_{refrig}}{M_{refrig_{max}}}}{M_{wall}C_{P_{wall}}}T_{wall} + \frac{UA_{wall\rightarrow refrig_{max}}\frac{M_{refrig}}{M_{refrig_{max}}}}{M_{wall}C_{P_{wall}}}T_e$$

The value T_e can be inserted to give:

$$\dot{T}_{wall} = \frac{UA_{air\rightarrow wall}}{M_{wall}C_{P_{wall}}}T_{air} - \frac{UA_{air\rightarrow wall} + UA_{wall\rightarrow refrig_{max}}\frac{M_{refrig}}{M_{refrig_{max}}}}{M_{wall}C_{P_{wall}}}T_{wall} + \frac{UA_{wall\rightarrow refrig_{max}}\left(-4.3544P^2{}_{suc} + 29.2240P_{suc} - 51.2005\right)}{M_{wall}C_{P_{wall}}M_{refrig_{max}}}M_{refrig}$$

M_{refrig}, from equation 3.6:
Here the unlinearities block the possibility of uniquely isolating the states. Therefore it is not modified much, but Δh_{lg}, which as mentioned can be described by $\left(0.0217P^2{}_{suc} - 0.1704P_{suc} + 2.2988\right) \cdot 10^5$, can be inserted which results in:

$$\dot{M}_{refrig} = V_p\left(\frac{M_{refrig_{max}} - M_{refrig}}{\tau_{fill}}\right) + (V_p - 1)\left(\frac{UA_{wall\rightarrow refrig_{max}}\frac{M_{refrig}}{M_{refrig_{max}}}\left(T_{wall} - T_e\right)}{\left(0.0217P^2{}_{suc} - 0.1704P_{suc} + 2.2988\right)\cdot 10^5}\right)$$

Finally, T_e can be inserted to give:

$$\dot{M}_{refrig} = V_p\left(\frac{M_{refrig_{max}} - M_{refrig}}{\tau_{fill}}\right) + (V_p - 1) \cdot \left(\frac{UA_{wall\rightarrow refrig_{max}}\frac{M_{refrig}}{M_{refrig_{max}}}\left(T_{wall} + 4.3544P^2_{suc} - 29.2240P_{suc} + 51.2005\right)}{\left(0.0217P^2_{suc} - 0.1704P_{suc} + 2.2988\right)\cdot 10^5}\right)$$

With the continuous non-linear model presented, it can now be linearized.

3.2.2 Continuous Linear Model

Normally, to create a linear model, one would just linearize an existing non-linear model around a working point, assuming the nonlinearities close to the working point are small enough. In case the nonlinearities are a driving force around the working point, it might be necessary to create several models to cover the entire operational range of the system. The problem with creating several models is that the complexity increases exponentially.

Assuming two models are required to describe a system, all model-based methods will require twice the amount of filtering/fault detection systems. An example of this could be a bank of discrete linear Kalman filters, where one would usually create a Kalman filter for each measurement. In case there are two base-models and two measurements, a total of four Kalman filters and a selection logic is required, as opposed to two in case the system could be described sufficiently using only one model. Fortunately, the display case dynamics can be described using a single model, even though some changes are necessary.

The Kalman filter and Unknown Input Observer are both based on linear models in a state-space form. The Kalman filter uses the discrete version, which will be described next, and the UIO uses a reduced order version of the continuous linear model. Just as a note, the linearization is obviously based on the continuous non-linear model.

The general structure of the state space representation is described by

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) + Ed(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned}$$

where x is the states, u is the input and d is the disturbance, which can be written as

$$x = \begin{bmatrix} T_{goods} \\ T_{air} \\ T_{wall} \\ M_{refrig} \end{bmatrix} \quad u = \begin{bmatrix} V_p \\ P_{suc} \end{bmatrix} \quad d = \begin{bmatrix} Q_{airload} \end{bmatrix}$$

Knowing that the only measurable states are T_{air} and T_{wall}, the total setup will be

$$\begin{bmatrix} \dot{T}_{goods} \\ \dot{T}_{air} \\ \dot{T}_{wall} \\ \dot{M}_{refrig} \end{bmatrix} = A \begin{bmatrix} T_{goods} \\ T_{air} \\ T_{wall} \\ M_{refrig} \end{bmatrix} + B \begin{bmatrix} V_p \\ P_{suc} \end{bmatrix} + E \begin{bmatrix} Q_{airload} \end{bmatrix}$$

$$\begin{bmatrix} T_{air} \\ T_{wall} \end{bmatrix} = C \begin{bmatrix} T_{goods} \\ T_{air} \\ T_{wall} \\ M_{refrig} \end{bmatrix} + D \begin{bmatrix} V_p \\ P_{suc} \end{bmatrix}$$

Starting backwards, the output will not be directly affected by the inputs, so the D-matrix is a 2x2 and holds only zeros, while the C-matrix is a 2x4 matrix which feeds the two measurable states right through.

$$C = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

To fill out the elements of the A- and B-matrix, each difference equation is differentiated by the system states and the results are put into the A-matrix at their respective places. The first two difference equations, T_{goods} and T_{air} are both linear, and therefore require no linearization.

T_{goods}:

$$\frac{d\dot{T}_{goods}}{dT_{goods}} = -\frac{UA_{goods\to air}}{M_{goods}C_{P_{goods}}}$$
$$\frac{d\dot{T}_{goods}}{dT_{air}} = \frac{UA_{goods\to air}}{M_{goods}C_{P_{goods}}}$$
$$\frac{d\dot{T}_{goods}}{dT_{wall}} = \frac{d\dot{T}_{goods}}{dM_{refrig}} = \frac{d\dot{T}_{goods}}{dV_p} = \frac{d\dot{T}_{goods}}{dP_{suc}} = \frac{d\dot{T}_{goods}}{dQ_{airload}} = 0$$

T_{air}:

$$\frac{d\dot{T}_{air}}{dT_{goods}} = \frac{UA_{goods\to air}}{M_{air}C_{P_{air}}}$$
$$\frac{d\dot{T}_{air}}{dT_{air}} = -\frac{UA_{goods\to air} + UA_{air\to wall}}{M_{air}C_{P_{air}}}$$
$$\frac{d\dot{T}_{air}}{dT_{wall}} = \frac{UA_{air\to wall}}{M_{air}C_{P_{air}}}$$
$$\frac{d\dot{T}_{air}}{dQ_{airload}} = \frac{1}{M_{air}C_{P_{air}}}$$
$$\frac{d\dot{T}_{air}}{dM_{refrig}} = \frac{d\dot{T}_{air}}{dV_p} = \frac{d\dot{T}_{air}}{dP_{suc}} = 0$$

The third difference equation, T_{wall}, has several non-linearities, and must therefore be linearized in several places and fixed work-points are introduced. The fixed values and their origin are described later in this section.

T_{wall}:

$$\frac{d\dot{T}_{wall}}{dT_{air}} = \frac{UA_{air\to wall}}{M_{wall}C_{P_{wall}}}$$
$$\frac{d\dot{T}_{wall}}{dT_{wall}} = -\frac{UA_{air\to wall}}{M_{wall}C_{P_{wall}}} - \frac{UA_{wall\to refrig_{max}} \cdot \frac{M_{refrig,0}}{M_{refrig_{max}}}}{M_{wall}C_{P_{air}}}$$
$$\frac{d\dot{T}_{wall}}{dM_{refrig}} = -\frac{UA_{wall\to refrig_{max}} \cdot \left(T_{wall,0} + 4.3544P_{suc,0}^2 - 29.2240P_{suc,0} + 51.2005\right)}{M_{wall}C_{P_{wall}}M_{refrig_{max}}}$$
$$\frac{d\dot{T}_{wall}}{dP_{suc}} = \frac{UA_{wall\to refrig_{max}\cdot M_{refrig,0}}\left(-8.7088P_{suc,0} + 29.2240\right)}{M_{wall}C_{P_{wall}}M_{refrig_{max}}}$$
$$\frac{d\dot{T}_{wall}}{dT_{goods}} = \frac{d\dot{T}_{wall}}{dV_p} = \frac{d\dot{T}_{wall}}{dQ_{airload}} = 0$$

where all variables with a subscript ending with ", 0" is a fixed workpoint value for that variable.

The final difference equation that needs to be linearized is M_{refrig}, but the result of such a linearization would have some problems. The non-linear state has relations to T_{wall}, M_{refrig}, P_{suc} and V_p, but needs to be able to stabilize at zero when the valve is closed. If a non-zero relation exists with T_{wall} and/or P_{suc}, the system will never stabilize as these

values change all the time. Given a negative relation to itself, the system will always stabilize when no other parameters affect the state. As it is necessary to be able to fill up the evaporator as well, a positive relation to V_p will ensure that whenever the valve is open ($V_p = 1$), the evaporator will fill up, and when the valve is closed ($V_p = 0$) there will be no other elements affecting the massflow, than the massflow itself.

Based on this, an intuitive linearization has been made, using initial guesses for the values. This could easily seem un-scientific, but as the model will go though a parameter estimation, it is not necessary to know the exact values, as these will be discovered by the estimation process. All the found values, including the guesses from M_{refrig} can be inserted into the A, B and E matrices

$$A = \begin{bmatrix} -\frac{UA_{goods\to air}}{M_{goods}C_{P_{goods}}} & \frac{UA_{goods\to air}}{M_{goods}C_{P_{goods}}} & 0 & 0 \\ \frac{UA_{goods\to air}}{M_{air}C_{P_{air}}} & -\frac{UA_{goods\to air}+UA_{air\to wall}}{M_{air}C_{P_{air}}} & \frac{UA_{air\to wall}}{M_{air}C_{P_{air}}} & 0 \\ 0 & \frac{UA_{air\to wall}}{M_{wall}C_{P_{wall}}} & a_{33} & a_{34} \\ 0 & 0 & 0 & -0.05 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & b_{32} \\ 0.1 & 0 \end{bmatrix} \qquad E = \begin{bmatrix} 0 \\ \frac{1}{M_{air}C_{P_{air}}} \\ 0 \\ 0 \end{bmatrix}$$

where

$$a_{33} = -\frac{UA_{air\to wall}}{M_{wall}C_{P_{wall}}} - \frac{UA_{wall\to refrig_{max}} \cdot \frac{M_{refrig,0}}{M_{refrig_{max}}}}{M_{wall}C_{P_{wall}}}$$

$$a_{34} = -\frac{UA_{wall\to refrig_{max}} \cdot \left(T_{wall,0} + 4.3544P_{suc,0}^2 - 29.2240P_{suc,0} + 51.2005\right)}{M_{wall}C_{P_{wall}}M_{refrig_{max}}}$$

$$b_{32} = \frac{UA_{wall\to refrig_{max}\cdot M_{refrig,0}} \left(-8.7088P_{suc,0} + 29.2240\right)}{M_{wall}C_{P_{wall}}M_{refrig_{max}}}$$

The estimated values are found by calculating the average during normal operation. The specific parameters have been logged for 36000 samples (settled system), summed up and then divided by the total amount of samples. The values can be seen in Table 3.4.

Variable	**Value**
$T_{wall,0}$	-4.2442
$M_{refrig,0}$	0.0669
$P_{suc,0}$	1.2999

Table 3.4: Calculated averages

By inserting the values used in Dymola, which are described in the introduction to section 3.2 combined with estimates for the ",0"-values, an initial guess for the state space representation can be found. To make the model work for the linear Kalman filter and parameter estimation, B and E need to be combined (as the KF does not support

an E matrix), which will make $Q_{airload}$ be seen as an input instead. Calculating the matrices, they will be

$$A = \begin{bmatrix} -0.0015 & 0.0015 & 0 & 0 \\ 0.0060 & -0.0160 & 0.0100 & 0 \\ 0 & 0.0050 & -0.0077 & -0.6524 \\ 0 & 0 & 0 & -0.05 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 2 \cdot 10^{-5} \\ 0 & 0.0479 & 0 \\ 0.1 & 0 & 0 \end{bmatrix}$$

The values that are inserted into matrices above should only be seen as initial guesses, as especially the linearization might have given bad estimations of the correct value. To get the model to fit to the plant dynamics better, a grey-box parameter identification has been applied in MATLAB. The method uses the ”PEM” function with two parameters, namely a datamodel built on real data from the plant and the initial guesses given above. The C and D matrices have been fixed to the initial values, while in the A- and B-matrices, all zeros are locked, and the structure of the model is therefore locked. It should be noted that all four states have been used to find the best fitting model, and thereby a 4x4 identity matrix in the place of the C-matrix. The complete state space model including the results of the parameter estimation is

$$A = \begin{bmatrix} -0.0015 & 0.0015 & 0 & 0 \\ 0.0059 & -0.0161 & 0.0100 & 0 \\ 0 & 0.0085 & -0.0084 & -0.4634 \\ 0 & 0 & 0 & -0.0677 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 2.0074 \cdot 10^{-5} \\ 0 & -0.0129 & 0 \\ 0.0454 & 0 & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad D = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

As the D-matrix will not change to anything other than zeros, it will not be mentioned any further.

The final model has been compared to measured data in MATLAB by using the compare function and some newly generated data from the plant (with T_{air} and T_{wall} measurements, starting at different temperatures). As the plant has two built-in controllers, the inputs (P_{suc} and V_p) are controlled automatically, and as such, the comparison is only to illustrate the correctness of the linear model. The result, which can be seen in Figure 3.2 on the facing page, is very good with a fit of more than 90% for the three temperatures, and a fit of 80% of the mass flow. Even though the changes made by the parameter estimation are generally quite small, as a comparison, if ”compare” is called with the model before parameter estimation, all four states will have a negative fit.

3.2.3 Sampling Time

To determine the required sampling speed for the discrete versions of the model, the just established linear continuous model can be used to determine the system bandwidth. By

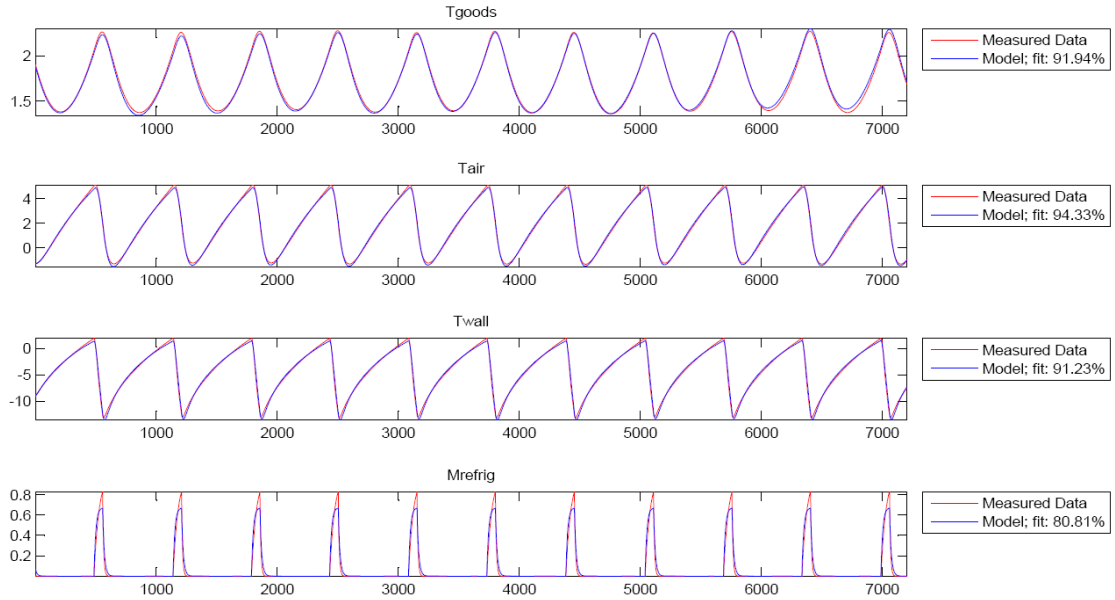


Figure 3.2: Comparison between Linear model and measured data

using the m-file supplied on the cd [CD/Data/M-Files] named bodeplot (also illustrated in Appendix D), the plots can be reproduced. The bodeplot illustrating the fastest dynamics is illustrated in Figure 3.3, while all the bodeplots from each input to each output have also been plotted and can be found in Appendix A, Figure A.1.

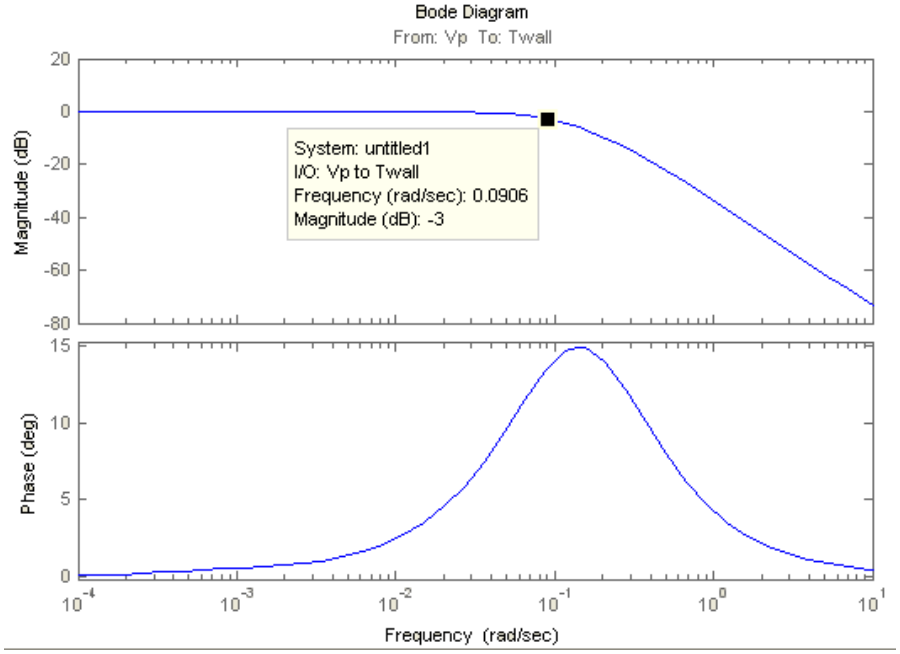


Figure 3.3: Bodeplot - V_p to T_{wall}

In the figure, the -3 dB point is marked which is the base for determining the system bandwidth. The highest frequenzy (which is the one illustrated in Figure 3.3) is 0.0906.

By dividing this with 2π, it is translated into HZ.

$$\frac{0.0906}{2\pi} = 0.0144$$

As can be seen from the system bandwidth, the system has quite slow dynamics. Generally it is said that sampling must be at least 10 times faster than the system bandwidth, and preferrably at least 20-30 times faster. By multplying the discovered system bandwidth with 30, the minimum sampling rate is 0.4326. By taking the inverse of the sampling rate, the sampling time is disovered, which is the time (in seconds) between each sample.

$$\frac{1}{0.4326} = 2.3117$$

As can be seen, a sample every 2 seconds is more than sufficient to get the full dynamics of the plant. Even though this is the case, to be sure the sampling is fast enough, and to make the following calculations easier, a sample time of 1 second is used.

3.2.4 Discrete Non-linear Model

The discrete version is used for the Extended Kalman Filter and discretization is done by using Euler's method [FPW98] (page 59, equation 3.2), which is

$$\dot{x}(k) \cong \frac{x(k+1) - x(k)}{T}$$

where $x(k)$ is the value of x at time t_k, k is an integer indicating the step-number, and T is the sampling time (the inverse of the sampling rate). As the sampling time is set to 1 for this project, the function can be reduced to

$$\dot{x}(k) \cong x(k+1) - x(k) \tag{3.7}$$

By combining equation 3.7 with the difference equations for the four states, the discrete non-linear version will appear.

T_{goods}:

$$T_{goods}(k+1) - T_{goods}(k) = -\frac{UA_{goods\to air}}{M_{goods}C_{P_{goods}}}T_{goods}(k) + \frac{UA_{goods\to air}}{M_{goods}C_{P_{goods}}}T_{air}(k)$$

$$T_{goods}(k+1) = \left(1 - \frac{UA_{goods\to air}}{M_{goods}C_{P_{goods}}}\right)T_{goods}(k) + \frac{UA_{goods\to air}}{M_{goods}C_{P_{goods}}}T_{air}(k)$$

T_{air}:

$$\begin{aligned} T_{air}(k+1) - T_{air}(k) = {} & \frac{UA_{goods\to air}}{M_{air}C_{P_{air}}}T_{goods}(k) - \frac{UA_{goods\to air} + UA_{air\to wall}}{M_{air}C_{P_{air}}}T_{air}(k) \\ & + \frac{UA_{air\to wall}}{M_{air}C_{P_{air}}}T_{wall}(k) + \frac{\dot{Q}_{airload}(k)}{M_{air}C_{P_{air}}} \end{aligned}$$

$$T_{air}(k+1) = \frac{UA_{goods\to air}}{M_{air}C_{P_{air}}}T_{goods}(k) + \left(1 - \frac{UA_{goods\to air} + UA_{air\to wall}}{M_{air}C_{P_{air}}}\right)T_{air}(k) + \frac{UA_{air\to wall}}{M_{air}C_{P_{air}}}T_{wall}(k) + \frac{\dot{Q}_{airload}(k)}{M_{air}C_{P_{air}}}$$

T_{wall}:

$$T_{wall}(k+1) - T_{wall}(k) = \frac{UA_{air\to wall}}{M_{wall}C_{P_{wall}}}T_{air}(k) - \frac{UA_{air\to wall} + UA_{wall\to refrig_{max}}\frac{M_{refrig}}{M_{refrig_{max}}}}{M_{wall}C_{P_{wall}}}T_{wall}(k) + \frac{UA_{wall\to refrig_{max}}\left(-4.3544P^2{}_{suc} + 29.2240P_{suc} - 51.2005\right)}{M_{wall}C_{P_{wall}}M_{refrig_{max}}}M_{refrig}(k)$$

$$T_{wall}(k+1) = \frac{UA_{air\to wall}}{M_{wall}C_{P_{wall}}}T_{air}(k) + \left(1 - \frac{UA_{air\to wall} + UA_{wall\to refrig_{max}}\frac{M_{refrig}}{M_{refrig_{max}}}}{M_{wall}C_{P_{wall}}}\right)T_{wall}(k) + \frac{UA_{wall\to refrig_{max}}\left(-4.3544P^2{}_{suc} + 29.2240P_{suc} - 51.2005\right)}{M_{wall}C_{P_{wall}}M_{refrig_{max}}}M_{refrig}(k)$$

M_{refrig}:

$$M_{refrig}(k+1) - M_{refrig}(k) = V_p(k)\cdot\left(\frac{M_{refrig_{max}} - M_{refrig}(k)}{\tau_{fill}}\right) + (V_p - 1)\left(\frac{UA_{wall\to refrig_{max}}\frac{M_{refrig}(k)}{M_{refrig_{max}}}\left(T_{wall}(k) - T_e(k)\right)}{\left(0.0217P^2{}_{suc}(k) - 0.1704P_{suc}(k) + 2.2988\right)\cdot 10^5}\right)$$

$$M_{refrig}(k+1) = M_{refrig}(k) + V_p\cdot\left(\frac{M_{refrig_{max}} - M_{refrig}(k)}{\tau_{fill}}\right) + (V_p - 1)\left(\frac{UA_{wall\to refrig_{max}}\frac{M_{refrig}(k)}{M_{refrig_{max}}}\left(T_{wall}(k) - T_e(k)\right)}{\left(0.0217P^2{}_{suc}(k) - 0.1704P_{suc}(k) + 2.2988\right)\cdot 10^5}\right)$$

3.2.5 Discrete Linear Model

To discretizise the linear model is, for this specific setup, very easy. As was seen from the discretization of the continuous model, the only real change, was the addition of each state to its own equation. For a liniear system, the approach will be the same, and can thereby intuitively be solved by

$$A(discrete) = I + A(continuous)$$

This method can be used on all stages of the A-matrix, but will in this report only be done for the final version described in the previous section. The result will be

$$A = \begin{bmatrix} 0.9985 & 0.0015 & 0 & 0 \\ 0.0059 & 0.9839 & 0.0100 & 0 \\ 0 & 0.0085 & 0.9916 & -0.4634 \\ 0 & 0 & 0 & 0.9323 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 2.0074 \cdot 10^{-5} \\ 0 & -0.0129 & 0 \\ 0.0454 & 0 & 0 \end{bmatrix} \quad C = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Faulty Discrete Linear Models

To isolate faults, models describing other scenarios are also needed (for more information, see the Kalman Filter section in Chapter 4). They have been found in the same way as the model (using parameter estimation) which was just described, so the process will not be re-written here. The models only change for the A and B matrix, and consist of three scenarios. The matrices are presented with only three decimals to be able to keep them within the width of the page, but in Simulink and MATLAB all decimals from the calculations are used.

Scenario 1: Non-full display case ($M_{goods} = 25$ kg), $UA_{air \to wall}$ normal:

$$A_1 = \begin{bmatrix} 0.994 & 0.006 & 0 & 0 \\ 0.005 & 0.987 & 0.008 & 0 \\ 0 & 0.009 & 0.992 & -0.430 \\ 0 & 0 & 0 & 0.942 \end{bmatrix} B_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1.66 \cdot 10^{-5} \\ 0 & -0.012 & 0 \\ 0.042 & 0 & 0 \end{bmatrix}$$

Scenario 2: Full display case ($M_{goods} = 200$ kg), $UA_{air \to wall}$ at 250 (faulty):

$$A_2 = \begin{bmatrix} 0.998 & 0.002 & 0 & 0 \\ 0.006 & 0.989 & 0.005 & 0 \\ 0 & 0.004 & 0.994 & -0.216 \\ 0 & 0 & 0 & 0.970 \end{bmatrix} B_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 2.01 \cdot 10^{-5} \\ 0 & -0.024 & 0 \\ 0.028 & 0 & 0 \end{bmatrix}$$

Scenario 3: Non-full display case ($M_{goods} = 25$ kg), $UA_{air \to wall}$ at 250 (faulty):

$$A_3 = \begin{bmatrix} 0.994 & 0.006 & 0 & 0 \\ 0.005 & 0.991 & 0.004 & 0 \\ 0 & 0.010 & 0.990 & -0.223 \\ 0 & 0 & 0 & 0.972 \end{bmatrix} B_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1.66 \cdot 10^{-5} \\ 0 & -0.058 & 0 \\ 0.027 & 0 & 0 \end{bmatrix}$$

3.2.6 Reduced Order Models

The Unknown Input Observer needs a reduced order version of the continuous linear model. The T_{goods} state is taken out of the model, and will instead be a disturbance to

the system. The reason for the change will be discussed in the relevant section, namely where it is used, in Section 4.4.2.

Basically the model is almost identical to the full order version except that the some rows and columns must be removed. In the A-matrix the first row and column must be removed, and likewise the first column of C and the first row of B. The resulting matrices can be seen below, with one difference. The element a_{11} in the A matrix includes a parameter related to the T_{goods} state, which must be eliminated as well. Furthermore, the B matix is again divided into two seperate, the B and E matrix.

$$A = \begin{bmatrix} a_{11} & 0.0100 & 0 \\ 0.0085 & -0.0084 & -0.4634 \\ 0 & 0 & -0.0677 \end{bmatrix} B = \begin{bmatrix} 0 & 0 \\ 0 & -0.0129 \\ 0.0454 & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad E = \begin{bmatrix} 2.0074 \cdot 10^{-5} \\ 0 \\ 0 \end{bmatrix}$$

where a_{11} is

$$a_{11} \neq -\frac{UA_{goods \to air} + UA_{air \to wall}}{M_{air} C_{P_{air}}}$$
$$a_{11} = -\frac{UA_{air \to wall}}{M_{air} C_{P_{air}}} = 0.01$$

In the same way as for the full order model, a parameter estimation is run to make sure the model fits well. The value in E must now represent both T_{goods} and $Q_{airload}$, but as the dynamics of the disturbance are assumed unknown for the UIO, the combined result can be changed to 1. After running the pem function, the complete model can be described by

$$A = \begin{bmatrix} -0.0139 & 0.0090 & 0 \\ 0.0110 & -0.0090 & -0.6004 \\ 0 & 0 & -0.1049 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 \\ 0 & -0.0168 \\ 0.0570 & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad E = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Faulty reduced order models

Faulty models are required in the UIO section to isolate faults. The faulty versions are found in the same way as the above, with the difference that the data which is used for estimation comes from simulations where the faults have been introduced. The fault in this case, is a change of the $UA_{air \to wall}$ value from 500 to 450, 375 and 300 which is made before the simulation is started. The reason for using the specified values is discussed in the UIO section of the FDI chapter. Again the A and B matrices change, and they

become

$$A_{450} = \begin{bmatrix} -0.0143 & 0.0086 & 0 \\ 0.0054 & -0.0050 & -0.3962 \\ 0 & 0 & -0.1784 \end{bmatrix} \quad B_{450} = \begin{bmatrix} 0 & 0 \\ 0 & -0.0025 \\ 0.1229 & 0 \end{bmatrix}$$

$$A_{375} = \begin{bmatrix} -0.0129 & 0.0073 & 0 \\ 0.0045 & -0.0042 & -0.4841 \\ 0 & 0 & -0.1776 \end{bmatrix} \quad B_{375} = \begin{bmatrix} 0 & 0 \\ 0 & -0.0027 \\ 0.0925 & 0 \end{bmatrix}$$

$$A_{300} = \begin{bmatrix} -0.0096 & 0.0051 & 0 \\ 0.0765 & -0.0389 & -0.6789 \\ 0 & 0 & -0.0353 \end{bmatrix} \quad B_{300} = \begin{bmatrix} 0 & 0 \\ 0 & -0.2935 \\ 0.0305 & 0 \end{bmatrix}$$

4

Fault Detection and Isolation

Chapter Overview:

The goal of this chapter is to develop a method to detect and isolate faults. The chapter defines the faults and then continues on to the Kalman Filter and Extended Kalman Filter. The Unknown Input Observer follows and then the Parametric Estimation, which is included as a proof-of-concept but not completed. The final two sections enable the reader to compare the developed methods and present a method where several methods are combined into a fully operational FDI system.

Fault detection schemes can in general be divided into two different categories; Model based and Data/Signal based. In this project it has been decided to use model based methods, as a model of the system is available. The general structure of a model based FDI system is seen in Figure 4.1. The basic idea of the model based FDI is to use a model to estimate how the system will evolve, and then compare it to measurements from the plant. The difference between the two is referred to as the residual, and can be used (either directly, or transformed into a fault indicator) to detect and isolate faults, depending on how the models are set up. In a non-faulty system, assuming

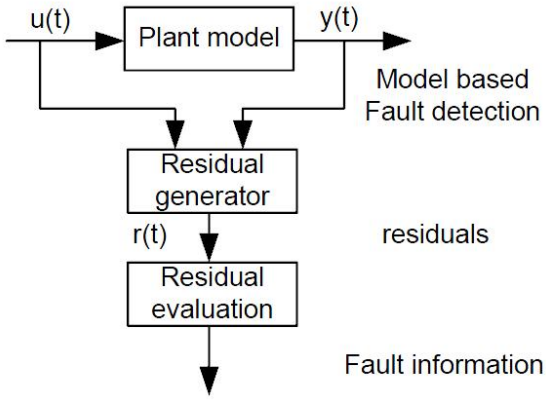


Figure 4.1: General structure of model based FDI

ideal conditions, the residual will be zero when no fault has occurred. Under realistic operating conditions, the residual should be a zero-mean normal distributed signal, which is due to the noise in the system and (especially) sensors. There is a lot of terminology being used by different literature within fault detection and isolation. Not all of the terms are yet well defined, which can make the area somewhat confusing. To eliminate confusion the terms used throughout this project are described here, and are primarily taken from [TRM] which is a row of suggestions for standard terms within *Supervision, Fault Detection and Safety for Technical Processes* based on the paper [IB97].

- Fault
  An unpermitted deviation of at least one characteristic property or parameter of the system from the acceptable, usual or standard condition.

- Failure
  A permanent interruption of a systems ability to perform a required function under specified operating conditions

- Fault detection
  Can be seen as a binary decision. Either the system works properly or it does not work properly

- Fault isolation
  This is in some texts referred to as identification. The purpose is to determine the location of the fault. It could for example be a malfunctioning sensor or actuator

In this project, it is assumed that the supermarkets have no employees with sufficient knowledge to determine if a fault-warning is a false alert, early warning of a pending error or a full failure. Based on this assumption, it is important that false alarms are nonexistent, or at least extremely infrequent. With this in mind, the objective of the fault detection and isolation scheme is to be able to tell the employees at the supermarket when a fault has occurred, how critical the problem is, and what actions should be taken.

4.1 Refrigeration System Faults

As the focus of the report is to detect and identify faults within the refrigeration system, it is necessary to establish both the type and nature of the faults for which detection is desired. For the purpose of this project, in cooperation with Danfoss, two types of faults have been chosen:

- Sensor Faults
  - T_{air} - Drift, offset, freeze and hard-over.
  - T_{wall} - Drift, offset, freeze and hard-over.
- Plant (parametric) faults.
  - $UA_{air \rightarrow wall}$ drop (related to ice/dirt buildup on the evaporator).

The system will not only have faults, but also disturbances which will also change the dynamics of the system but should not generate a fault. The main disturbance to the system is the value $Q_{airload}$, but the mass and heat capacity of the goods, and the heat transfer coefficient between goods and air can also be seen as disturbances. Any change in these values should be ignored, and considered a disturbance.

The two types of faults are described in the two following subsections. As these two types of errors (sensor and plant/parametric) will themselves provide a challenge to detect, it has been decided not to look for the third common type of faults, namely actuator faults. Furthermore, as the project has not dealt with control for the refrigeration system, it would seem out of place to consider FTC (Fault Tolerant Control). Therefore the focus has been kept on detecting faults, and if possible isolating them. Solving any arisen fault will then be up to the supervisor or repairman.

Sensor Faults

The different types of sensor faults may not seem obvious from a single word, so here is a description of each fault:

1. Sensor Drift.
   A drifting sensor will often over time converge to a specific value (i.e. zero or a sensor minimum or maximum). An example is a pressure sensor where the sensor has a small leak which lets the pressure on one side slip into the other side. The drift fault can be hard to detect as the dynamics only change marginally. The

slope of the drift determines how hard it is to detect and isolate. In Simulink it has been simulated as a ramp with a small slope which is added to the measured signal.

2. Sensor Offset.
   The offset error adds or subtracts a fixed value or percentage of the real value to all measurements. The offset fault can be hard to detect, just like the drift fault, because the dynamics are similar to the one of a non-faulty system. In this project it is simulated as a constant which is added to - or subtracted from the measurement. The fault is introduced abruptly in Simulink, but may also be an effect of bad calibration or the above mentioned drift.

3. Sensor Freeze.
   When a sensor freezes it stops making new measurements at any given point, often because of mechanical restriction. An example could be a sensor which is mechanically hindered from moving, and therefore gives the same measurement at all samples. Some types of sensors are more prone to freeze than others, and even though temperature sensors are often non-mechanical sensors and therefore seldom affected by this type of error, it can still occur. It differs from hard-over in that it happens within the working range, and there is no abrupt change in the value of the measurement. As the dynamics change drastically when a sensor freezes, it should be quite easy to detect. On the other hand, the lack of abrupt change could fool some methods. In Simulink, the sensor freeze is simulated by repeating the last given signal (unit delay loop).

4. Sensor Hard-Over.
   The hard-over fault has some resemblance to the freeze fault, but instead of stopping within the working range, it jumps to an extreme value. This will usually be the lower or upper boundary of the measurable range of the sensor or zero. The fault could come from a sensor dropping out, a short circuit, sensor power-loss and many similar problems. The abrupt and large change in dynamics should make this fault easy to detect. The fault is simulated by exchanging the measured signals by a given constant.

After the signal has been introduced to its fault (if any) noise is added. The noise comes from a white noise block with a power of 0.25. The noise makes the faults harder to detect, and as the faults are introduced as constant faults (not intermittent), the test scenarios are more comparable. Intermittent faults are harder to distinguish - as they may be a result of a low-budget sensor or related to a disturbance. Other times they may be an early warning of a defect sensor.

Parametric Fault

The $UA_{air \rightarrow wall}$ parameter governs the heat transfer between the air inside the display case, and the evaporator wall. If this parameter changes, it will be caused by a fault. Two regularly occurring faults that will change the $UA_{air \rightarrow wall}$ parameter are:

1. Freeze-over.
   This happens when the refrigeration system has been running for some time. The

surface of the evaporator is prone to condensation, which combined with the low temperatures, will build up ice on the surface of the evaporator. When the surface of the evaporator is full of ice, the surface area gets smaller and thereby less efficient. Furthermore, there will be a delay in transferring the heat from the refrigerant to the air, as the heat transfer now needs to go through another medium. Freeze-over is usually solved by holding the valve shut while heating the evaporator with electrical wires until the ice has melted. This is done manually in some systems, while it happens regularly during the normal operating routine of other refrigeration plants.

2. Dirt build-up.
   As a lot of air from the shop passes through the evaporator, given enough time, it will be covered in dust and dirt. As the surface is covered more and more, the airflow through the evaporator will be increasingly limited. The combination of less air flowing through the condenser and the extra medium (the dirt) which needs to be overcome, results in a reduced heat transfer coefficient. The normal way to solve this is by taking the display case out of operation and then cleaning the evaporator.

In the same way as the sensor fault, the parametric fault is only simulated as a constant error, as this will be the case for true faults. Under normal operating conditions it will grow larger over a long time. In Simulink it has been simulated simpler by adding a slope to the parameter.

4.2 The Kalman Filter

Originally, the Kalman Filter was developed as exactly that - a filter. The filter required prior knowledge of how a given system would develop over time. The more precise the knowledge (model) was, the better the filter would work. The filter is extremely usefull for isolating system behaviour in noisy conditions, but requires that the system can be described reasonably well by a linear model. When the Kalman Filter was developed, the estimated states were the interesting part, as they could be used as a reliable estimate of the states instead of the noise-filled measurements. Furhtermore the Kalman Filter could be used when some states of a system were not measureable. For use in Fault Detection and Isolation the states them selves are not that interesting - but the residuals (the difference between the measured and estimated states) are.

4.2.1 Kalman Theory

The Kalman Filter recursively calculates the estimated states for the system. All state estimates $\hat{x}(k)$ (where k indicated the current sample) are based on a weighting of the measured state, and an initial estimation which is based on a linear description of the state development with the previous input $u(k-1)$ and state $\hat{x}(k-1)$ inserted. The estimated states are then used recursively when calculating the next state estimate.

By using knowledge of how the system is supposed to behave in the form of a model, it is possible to filter out a large amount of noise, disturbances and small faults. As the faults are the focus for this project the difference between the measured states and the

estimated state, also known as the residual, is not only used to correct the estimate, but also used to determine if a fault has occurred.

The Kalman filter model assumes the true state at time k is evolved from the state at $(k-1)$ according to

$$x(k) = A_k \cdot x(k-1) + B_k \cdot u(k-1) + w_k$$
$$y(k) = C_k \cdot x(k-1) + v_k$$

where w_k is the process noise and v_k is the observation noise. They are assumed to be independant (of each other), white and with normal probability distributions

$$p(w_k) \approx N(0, Q_k)$$
$$p(v_k) \approx N(0, R_k)$$

The term Q_k is hard to determine, as there is no specific way to determine how much the estimated states deviates from the measured states. For this specific project, the covariance of the observation noise R_k is defined in Simulink, and has been set to 0.25. As the noise dynamics are assumed identical for all samples, the subscript k is unnecessary, and can be removed. For real systems, the R-matrix can be found by looking at datasheets for the sensors, or by testing the sensor at a fixed temperature, where R will be the variance for the measurements.

The Kalman filtering can in general be divided into two separate actions; the prediction/estimation and the correction/update. Figure 4.2 on the facing page shows the two different processes with the update procedure divided into two sections. Other works using the Kalman Filter may give alternative descriptions for the equations in the two processes, especially in relation to the equations in the Pre-Update phase, but basically they will all be the same. The specific equations here, come from [WB06] and [WKF]. The equations will be described in the following sections, but an important thing to notice here, is the distinction between $\hat{x}$ and $\hat{x}^-$, like P and P^- which are not the same. The superscript $-$ indicates an *a priori* estimate, while the others are *a posteriori* estimates. The meaning of these terms are introduced when relevant.

Estimation

The estimation process consists of two equations. The first step is estimating what all states should be for the current sample. The result is written $\hat{x}^-(k)$, where $\hat{x}$ is the estimated state-vector and k is the sample, while the minus (as mentioned) denote that it is the *a priori* estimate. The *a priori* term means that it is calculated with knowledge up to, but not including, the current sample. In practice this means that measurements of the current state of the system are not used to predict the current state estimate. Instead the previous estimate $\hat{x}(k-1)$ and input $u(k-1)$ are used to estimate where the system states will be at the current sample by using the model. This can be illustrated

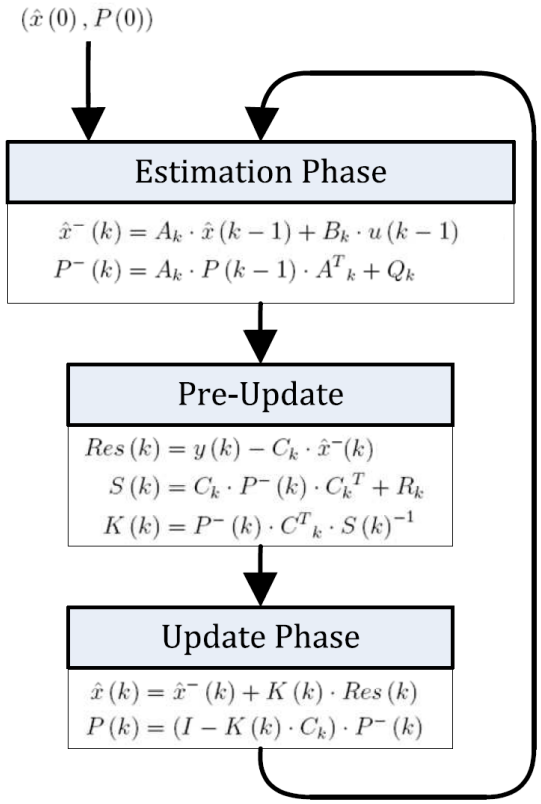


Figure 4.2: Kalman filter operation

by the first equation of the Kalman filter:

$$\hat{x}^{-}(k) = A_k \cdot \hat{x}(k-1) + B_k \cdot u(k-1)$$

where A_k and B_k are the system matrices. In case the model cannot be described linearly, the Kalman filter can use several system and noise models, with different variables for each sample k. This is illustrated by the use of subscript k. As the refrigeration system can be described sufficiently with a single model, the subscript k will no longer be included for this equation. A final thing to note is, that for the first iteration, it is necessary to have an initial guess for $\hat{x}(k-1) = \hat{x}(0)$.

The second equation within the Kalman estimation is the prediction uncertainty. This is referred to as the predicted estimate covariance $P^{-}(k)$. The covariance is an estimated accuracy of the results of the first equation. The formula for calculating $P^{-}(k)$ is:

$$P^{-}(k) = A_k \cdot P(k-1) \cdot A^T{}_k + Q_k$$

Q is, as mentioned previously, the covariance of the process noise. As before (due to the assumption that the noise features will not change) the subscript k is unnecessary so it will be left out from here on.

In the same way as the first equation, it is necessary to have an initial guess of $P^{-}(k)$. The value should be in the form of a diagonal matrix, with the values on the diagonal representing the uncertainty of the initial guesses supplied to the first equation. In case

the initial value $\hat{x}(0)$ is known to be 100% precise, the values on the diagonal of $P^{-}(0)$ should all be 0. The more uncertain one is of the initial condition the higher the values of the diagonal should be. A high value diagonal will allow the model to quickly adjust itself to the measured states.

Update

The update process consists of five equations of which three are described as pre-update in this report. Some sources do not calculate these equations explicitely, but rather as a part of the update equations. As the residual and the residual covariance are needed for further calculations, the equations are split up. The pre-update equations are: the residual (also known as the innovation)$Res(k)$, the residual (or innovation) covariance $S(k)$ and the Kalman gain $K(k)$. The true update equations are the *a posteriori* state estimate $\hat{x}(k)$ and finally the updated estimate covariance $P(k)$. To understand the update process, a few terms need to be understood. First of all, there is the term *a posteriori*. Opposed to the *a priori* estimate, the *a posteriori* state estimate uses all knowledge, including the current measurements to make its estimate.

The first equation in the update process calculates the residual, which as mentioned is used to detect faults. The result of the equation is the difference between the measured and estimated output.

$$Res(k) = y(k) - C_k \cdot \hat{x}^{-}(k)$$

The second equation is the residual covariance. Note the R_k value, which is a main part of the noise and fault filtering.

$$S(k) = C_k \cdot P^{-}(k) \cdot {C_k}^T + R_k$$

After the residual covariance has been found, the optimal Kalman gain is calculated, which is used to correct the *a priori* state estimate. The larger the Kalman gain is, the more the residual will affect the *a posteriori* state estimate. If $K(k)$ is zero, the *a posteriori* state estimate will be equal to the *a priori* state estimate. $K(k)$ can be calculated by:

$$K(k) = P^{-}(k) \cdot {C^T}_k \cdot S(k)^{-1}$$

The optimal Kalman gain is then multiplied with the residual, and added to the *a priori* state estimate, which results in the *a posteriori* estimate:

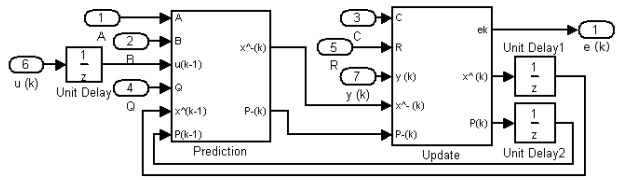
$$\hat{x}(k) = \hat{x}^{-}(k) + K(k) \cdot Res(k)$$

The final calculation in the update process is the estimate covariance, which needs to be updated. This is done with the following equation:

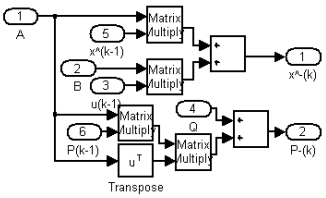
$$P(k) = (I - K(k) \cdot C_k) \cdot P^{-}(k)$$

where I is the identity matrix. After the update has been completed, the process is repeated as another sample is taken. As $Res(k)$ will be very noise-filled, a fault indicator based on the residual can be used to detect faults instead. This can be found by using the equation:

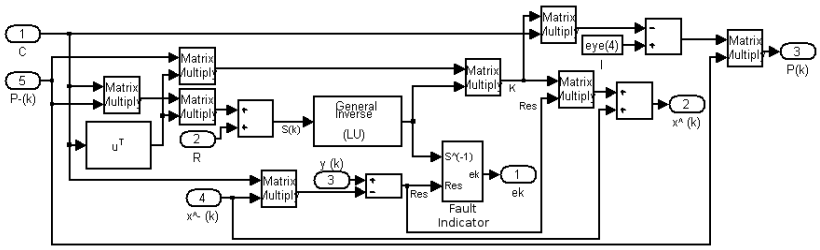
$$e_k = Res^T(k) \cdot S^{-1}(k) \cdot Res(k)$$



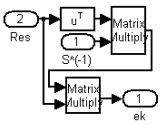
(a) Kalman Filter (Main Model)



(b) Prediction (Submodel)



(c) Update (Submodel)



(d) Fault I.

Figure 4.3: Kalman filter including sub-models

4.2.2 Implementation

As fault detection and identification requires different methods, this section describes how the Kalman filter is implemented generally. In the detection and isolation sections, some elements are added and/or changed within the kalman filter, which will be described in the relevant documentation. For a complete description of the Simulink implementation, check Appendix C. The Simulink model can be seen in Figure 4.3, where subfigure 4.3a is the Kalman Filter itself, and subfigure 4.3b and 4.3c are the submodels within. Subfigure 4.3d is the fault indicator, which is a submodel of the update process. By checking the model, it is possible to verify that the results are in agreement with the equations described in Section 4.2.1. The places where initial conditions are required can be identified by the unit delay. The unit delay is used three places, to feed back the *a posteriori* state estimate and prediction error covariance and finally to delay the input signal. The input signal and state estimate are initialized at zero, as it is not possible to know exactly what these values will be at any given time, without making a measurement. As the initial state is unknown, the initial value for the prediction error covariance is set to be the identity matrix. This should allow the estimation to correct itself quickly.

The A, B and C matrices are taken from the discrete linear model derived in section 3.2.5. The R matrix is as mentioned the observation noise covariance (for this scenario 0.25 for both measurements) and is therefore $I \cdot 0.25$ as the measurement noise is assumed uncorrelated. The Q matrix is initially set to be identical to R, and then tweaked in the following sections to give the desired result. Generally Q is not a value to tweak, but as it can be hard to determine, some adjustment is allowed. The y and u signals are taken from the Dymola interface.

4.2.3 Fault Detection

Detecting a fault in a system using the Kalman Filter, is as mentioned done by checking the size of the residual. In case an error occurs during normal plant operation, it should be evident by the residual, depending on the magnitude and nature of the fault. Unfortunately, as measurements can be very noisy, it can be hard to detect changes in the model, and even harder to establish a way for automatic detection. A way to improve the detectability is to use a fault indicator based on the residual instead. This should improve chances of detecting a fault.

In practice, detection is done by comparing the residual (or the fault indicator) to a pre-determined border. In the ideal situation, where no noise exists, and the model is a perfect description of the system the residual (and obviously the fault indicator) will, at all times, be zero. When a fault is introduced to the measurements of the plant or the plant itself, the residual will be non-zero. As an example, if the temperature sensor of Tair suddenly has an offset of plus one degree, the residual for Tair will jump to 1. The residual for the next iteration will then be somewhere between 0 and 1, depending on the size of Q in relation to R. This is because the Kalman Filter seeks to adjust itself so the residual becomes zero. In case the fault is constant, the residual will never return to zero, but will always have a small steady state error. This will also be the case for the Twall measurement (as one affects the other, an error in one state will propagate itself to the others).

The problem with the mentioned method is that the measurements are not noise-free in the real world, and the total plant dynamics are rarely completely described in the model and are subject to small changes due to disturbances. This implies that the residual will never stabilize at zero, but instead be a product of model uncertainty and noisy measurements. In case the model is reasonably well described, the mean of the residual will be close to zero with the main part of the white noise propagated directly into the residual.

In case faults are hard to detect it might indicate that Q is too small. By making Q smaller the model is trusted more as less of the residual is assumed to be due to a bad model. With the correct values for R and Q, the residual will hopefully show a spike when a fault occurs, and the fault indicator should indicate the fault clearly.

The main problem with Q becoming too small is that model errors will also be much more visible in the residual. This will cause problems for systems where the plant is poorly modeled. It is visible that the Kalman Filter methods works best for situations where a good plant model exists, or where faults change system dynamics drastically.

The simplest way to detect a fault is to compare the fault indicator of a faulty signal to one with no errors and then determine a ”threshold” where a fault is claimed. By selecting a border which is less than the fault indicator in case of fault, but larger (and preferrably with a margin) when no fault is introduced, a fault can be claimed. A more reliable method, is using ”CUSUM” (Cumulative Sum), which will be described later.

The Q matrix is, as mentioned, initially defined as being a diagonal matrix with 0.25 in

all places. By using trial and error (to make faults more visible), Q has been corrected.

$$R = \begin{bmatrix} 0.25 & 0 \\ 0 & 0.25 \end{bmatrix} \quad Q = \begin{bmatrix} 0.25 \cdot 10^{-3} & 0 & 0 & 0 \\ 0 & 0.25 \cdot 10^{-3} & 0 & 0 \\ 0 & 0 & 0.25 \cdot 10^{-3} & 0 \\ 0 & 0 & 0 & 0.25 \cdot 10^{-3} \end{bmatrix}$$

The above mentioned Q and R matrices give a clearer result. The reason for R being a 2x2 matrix, and Q a 4x4, is that R is related to the measurements (T_{air} and T_{wall}) while Q is related to the states. Both Q and R should always fit to the system (depending on noise levels, model uncertainty, faulty scenarios etc.). The used values for Q and R indicate that the state estimates are trusted to be very reliable while the measurements are less reliable. A problem that can arise from trusting the model to much, is that disturbances and model uncertainty will also be claimed as faults, and the system will take a long time to stabilize.

Now that Q and R have been determined, the faults can be applied to the system. Section 4.1 described the faults, which for the Kalman Filter are implemented in the following way:

- Sensor Drift
  Implementation: Add a ramp with a slope of +0.001 °C per second/sample.
- Sensor Offset
  Implementation: Add a fixed value of 2.5 °C to each measurement.
- Sensor Freeze
  Implementation: Repeat the previous signal.
- Sensor Hard-Over
  Implementation: Replace measured signal by a fixed value of 25 °C.
- Dirt/Ice buildup ($UA_{air \rightarrow wall}$ drop)
  Implementation: Steady drop in $UA_{air \rightarrow wall}$ from 500 at time t=6000 to 250 at time t=16000.

In Figure 4.4 and 4.5, the fault indicators can be seen for faults introduced to the T_{air}- and T_{wall}-sensor respectively. Each figure consists of four plots, while Figure 4.6 is the dirt/ice buildup fault. Each figure shows a five hour simulation (18000 samples - sampling at 1 Hz), and after 2.5 hours (9000 samples) the faults are introduced to their respective simulation.

As can be seen from the three figures, some of the faults generated are almost impossible to detect by using a simple detection border. It is especially hard to distinguish between faults and normal operation for drift and offset faults. To detect these faults, a CUSUM (Cumulative Sum) method can be used. CUSUM for a signal can be calculated as:

$$Z(k) = Z(k-1) + (e(k) - mean) \tag{4.1}$$

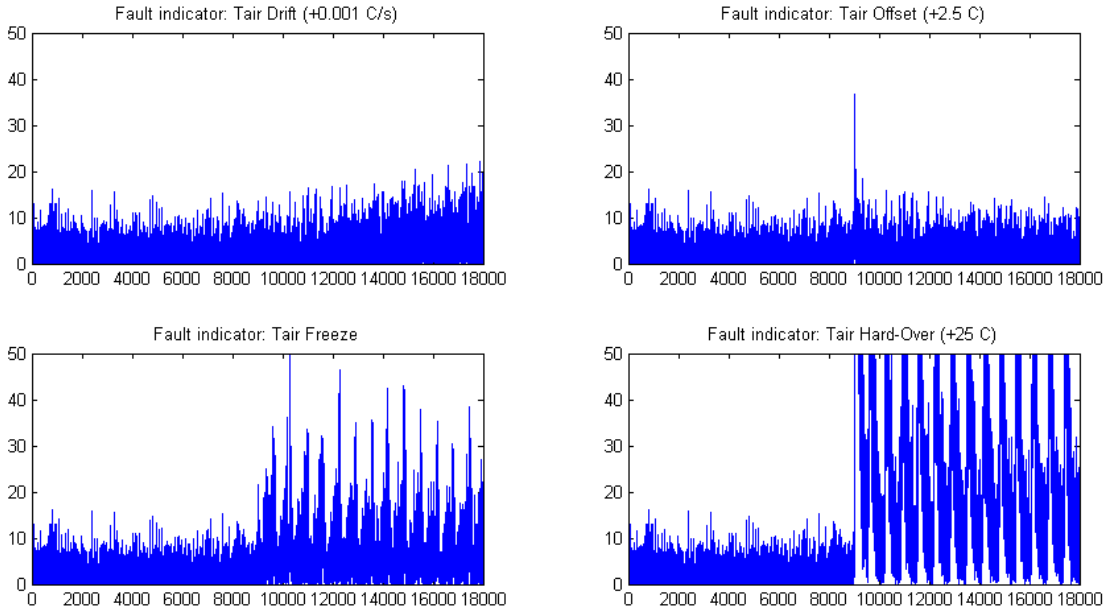


Figure 4.4: T_{air} sensor faults

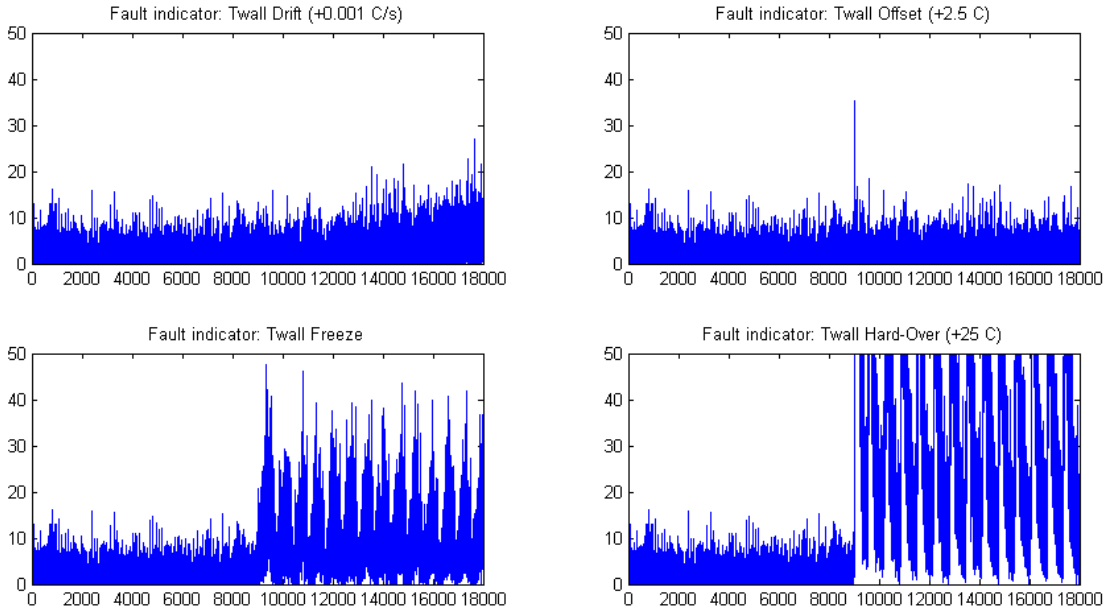


Figure 4.5: T_{wall} sensor faults

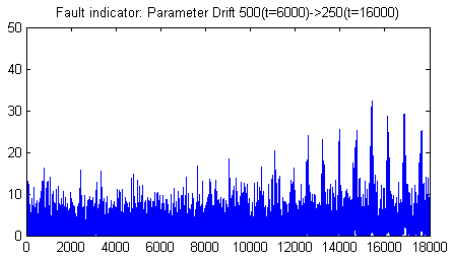


Figure 4.6: $UA_{air \rightarrow wall}$ drop (Dirt/Ice)

where $mean$ is the mean value of the signal in nominal operating conditions and $Z(0) = 0$. The problem with using this method, is that if the dynamics change even marginally, the cumulative sum will given enough time rise above/below any given detection border. Another method which can be found at [WCU] is

$$Z(k) = max(0, Z(k-1) + (e(k) - \omega_n)) \tag{4.2}$$

where ω_n is weights, and $Z(0) = 0$ again. This method will only detect positive changes, and will therefore require a similar function where "max" is exchanged with "min" if negative changes are also important to detect. For this case, as $e(k)$ is a normalized signal, negative changes are not important, as they will mean the model is more correct.

Comparing equation 4.1 with 4.2, they will be identical for positive faults if ω_n is assumed to be the mean. To solve the problem of small dynamical changes being detected as faults, the weight ω_n can be set to the mean plus a deadzone. Using this approach, the signal has to overcome the deadzone for Z to grow. The deadzone should be small enough not to hinder detection, but large enough to filter out small allowable changes in the dynamics.

The Dymola simulation has been run with a non-faulty scenario and the mean of the $e(k)$ signal has been found to be 1.9661. The deadzone has been set to 0.1 by trial and error. The results can be seen in Figure 4.7 and 4.8 on the next page for sensor faults, while a non-faulty scenario and the parametric fault is shown in Figure 4.9. As can be

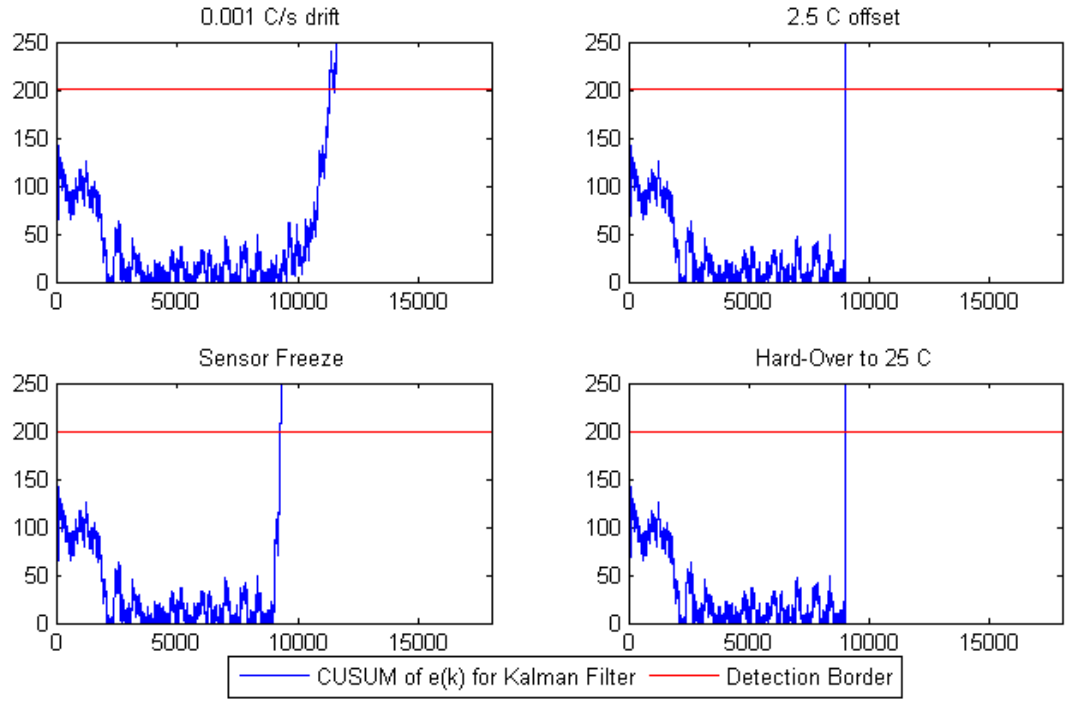


Figure 4.7: CUSUM for T_{air} sensor faults

seen from the figures, faults are easily detectable using this method. A detection border is needed for this method as well. By checking the graphs it seems a border of 200 should

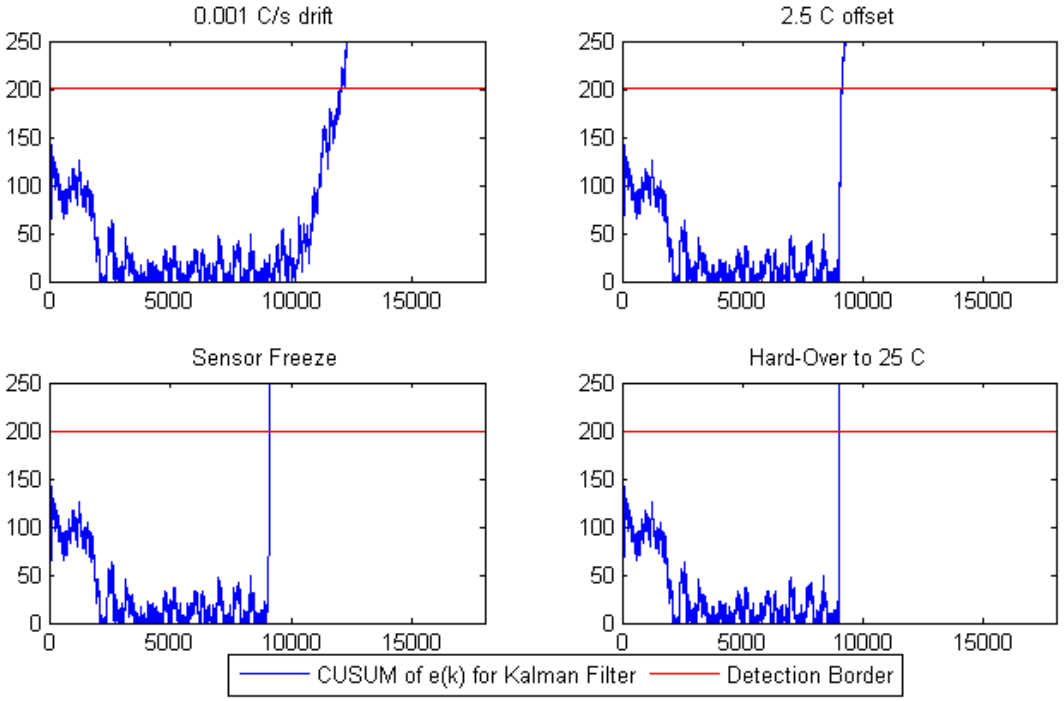


Figure 4.8: CUSUM for T_{wall} sensor faults

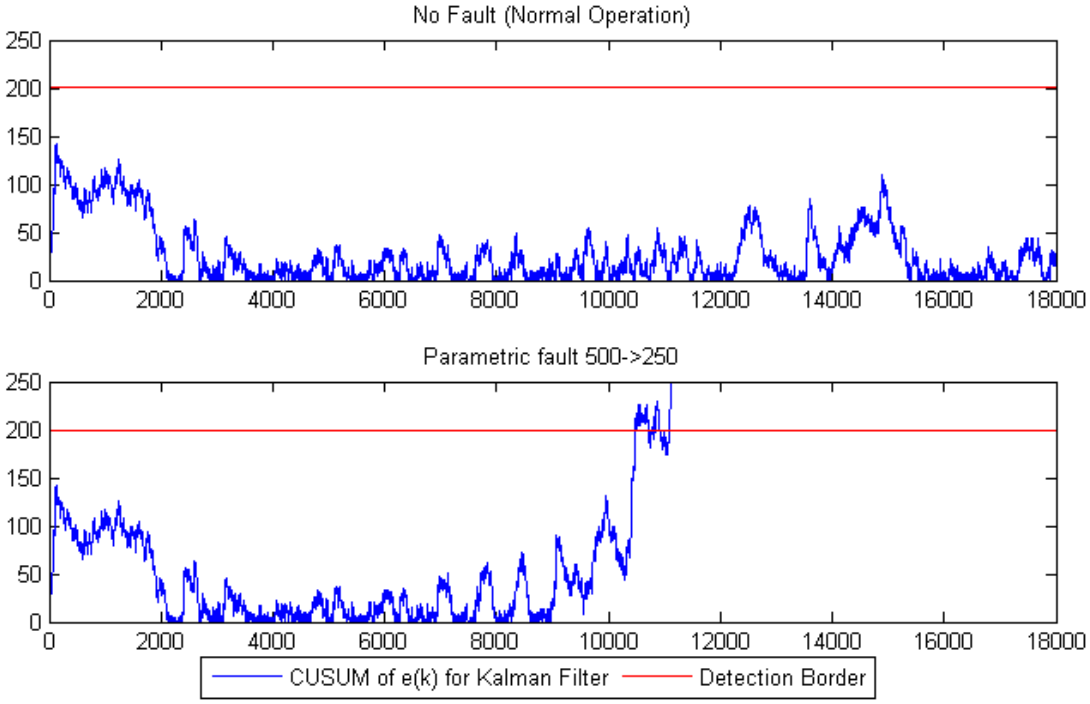


Figure 4.9: CUSUM for nominal and $UA_{air \rightarrow wall}$ drop (Dirt/Ice)

be ideal. It should be noted that the mean, deadzone and detection (border) values need to be recalculated for other systems.

4.2.4 Fault Isolation

When a fault has been detected, isolating the fault will be the next task. To be able to localize where any given fault has occurred, there are a few methods available using the Kalman Filter. The general approach is to expand the amount of available knowledge of the fault, often by creating a bank of Kalman filters. In other words, several Kalman filters are run in parallel with different conditions. One method is based on splitting the measured plant outputs up, and feeding them to identical Kalman Filters with C-matrices matching the used sensor. This approach is ideal for spotting sensor faults, and will be described further in the next subsection.

Another method is based on generating multiple system models - usually one for each of the faulty scenarios and one for nominal operation. The models are then used in parallel Kalman Filters, and the residual of each filter is used to determine which model fits the current scenario best. This method will allow detection of several types of problems, especially parametric faults, but unfortunately it will in most cases require a large amount of models as there are often many theoretical scenarios. This method will be described in the subsection *Isolation of parametric faults* on page 49.

Isolation of sensor faults

Usually when splitting the measured signals into smaller groups, one of two methods are used:

- using only one sensor per Kalman Filter, or
- using all except one sensor for each Kalman Filter.

Figure 4.10 illustrates the two above mentioned methods. In the example the plant has three outputs (The red, green and blue line). If one sensor is used per KF, then the faulty sensor will trigger a fault on the KF attached to that sensor. The advantage is, that two or more sensors can fail, and they can still be isolated by this method. On the other hand, using all except one sensor for every kalman filter will only have one KF with the correct result in case a sensor fails. The advantage here is robustness to plant devations (harder to generate a sensor fault) but the disadvantage is the limitation to only a single sensor fault, as multiple sensor faults will make all KF's claim a fault. As an example, if the red line in Figure 4.10 represents a faulty sensor, the "One sensor" approach will claim a fault on Kalman Filter 1, while the other method will claim a fault on all KF's, except Kalman Filter 3.

For this project, the two methods will be identical though, as there are only two measurements. The Kalman filter bank will therefore consist of two Kalman filters, one for each of the measurements T_{air} and T_{wall}. By using several Kalman Filters with different signals, if any sensor is faulty or completely fails, the Kalman filter using that sensor as

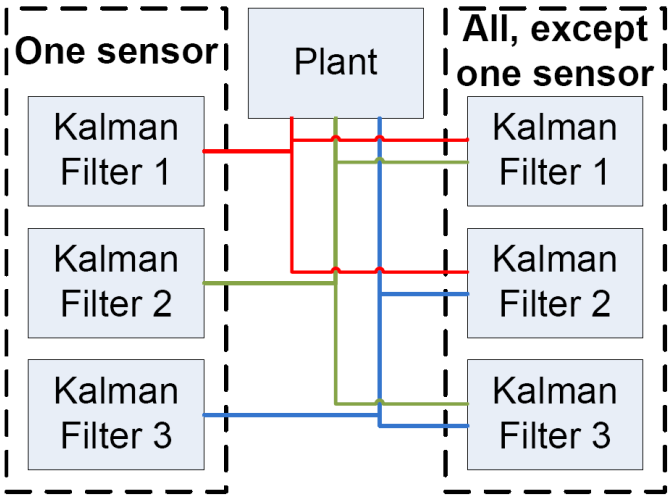


Figure 4.10: The two split-measurements KF Bank methods

input will generate larger residuals, while the filter using the healthy sensor will continously have small residuals. If only a single Kalman filter is used, it can be hard (or even impossible) to identify which sensor is defect in the fault-scenarios shown in Figure 4.7 and 4.8 on page 46.

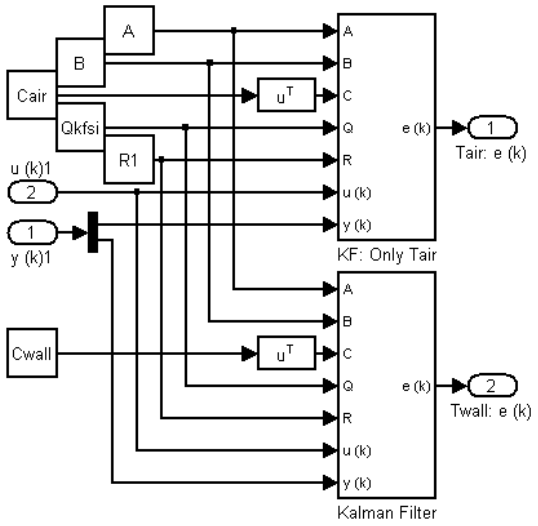


Figure 4.11: Simulink Implementation: KF Bank - Split Sensors

As the Kalman filters are identical to the one described in section 4.2.3, the implementation will look like seen in Figure 4.11. The matrix Q and the detection border established for the CUSUM (Cumulative Sum) in section 4.2.3 are re-used, as they provided good results for detection. The R matrix is reduced to a scalar ($R = 0.25$), as each model only uses one plant output in this setup. To check isolation properties, all faults have been introduced to the system again, and the CUSUM results can be seen in Figures 4.12 to 4.14 on pages 49–50. As $e(k)$ has changed, new mean values must be calculated. They are 0.9821 and 0.9957 for the Kalman Filter using the T_{air} sensor and T_{wall} sensor

respectively. The first figure illustrates faults on the T_{air} sensor, the second illustrates faults for T_{wall} and the final figure shows plots for a non-faulty situation and the parameter fault again. In all figures the green line is the residual from the first Kalman filter (using T_{air} as input), and the blue line is from the second Kalman filter (using T_{wall} as input).

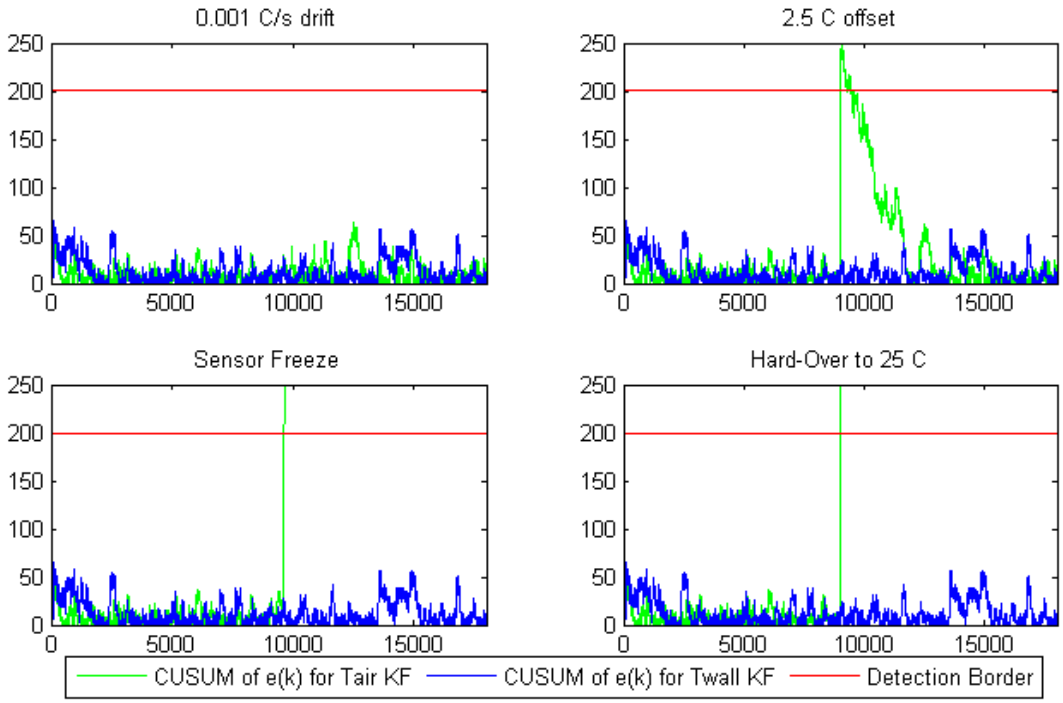


Figure 4.12: CUSUM for T_{air} sensor faults

As can be seen from Figure 4.12 and 4.13 on the next page, sensor freeze and hard-over are quite easy to detect. Unfortunately, the sensor offset only passes the detection border briefly for one of the scenarios, and not at all for the other. Sensor drift isolation is on the other hand close to impossible, at least for small changes in temperature. The drop in $UA_{air \rightarrow wall}$ will also trigger a detection with the assumption it is based on sensor fault. Based on the above results, the standard KF is not good enough to use for sensor fault isolation.

Isolation of parametric faults

As mentioned earlier, the main problem with using multiple models is the amount of models that are required to cover all scenarios. If some scenarios are not covered, their occurance may result in a non-faulty conditions to be mistaken for a faulty one, and vice versa. As the amount of goods are allowed to change during operation, a change in M_{goods} must not be seen as a fault. The remaining parameters should not change under normal operating conditions, especially not the $UA_{air \rightarrow wall}$ parameter. Based on this, a total of four models are needed.

Initially, two models can be created using the nominal heat transfer between air and wall

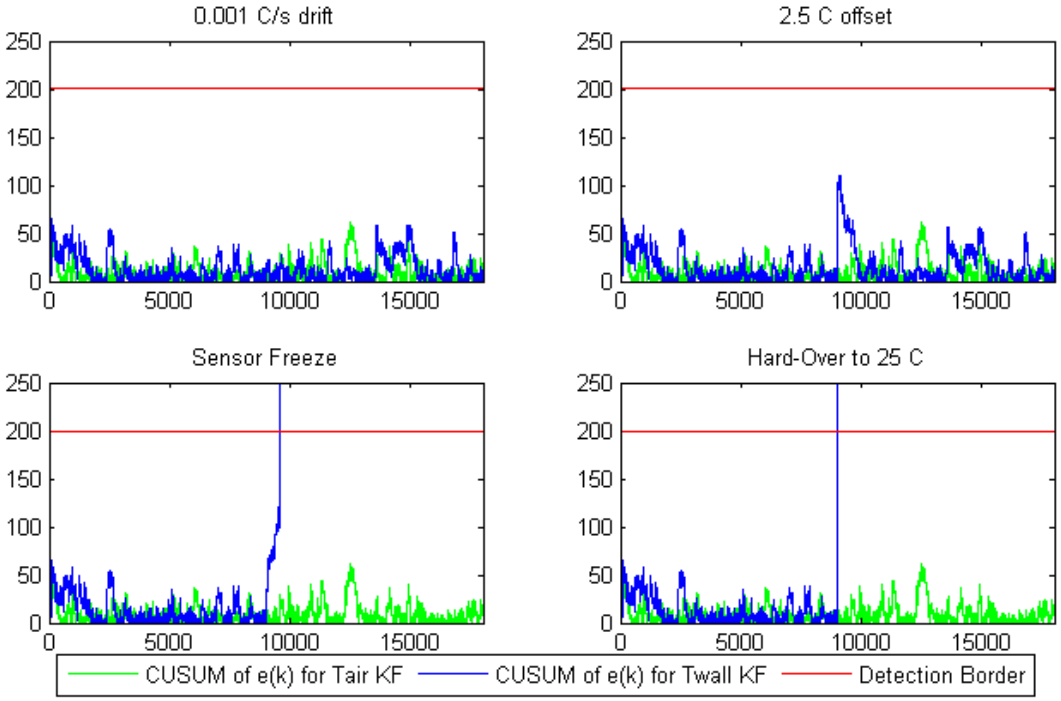


Figure 4.13: CUSUM for T_{wall} sensor faults

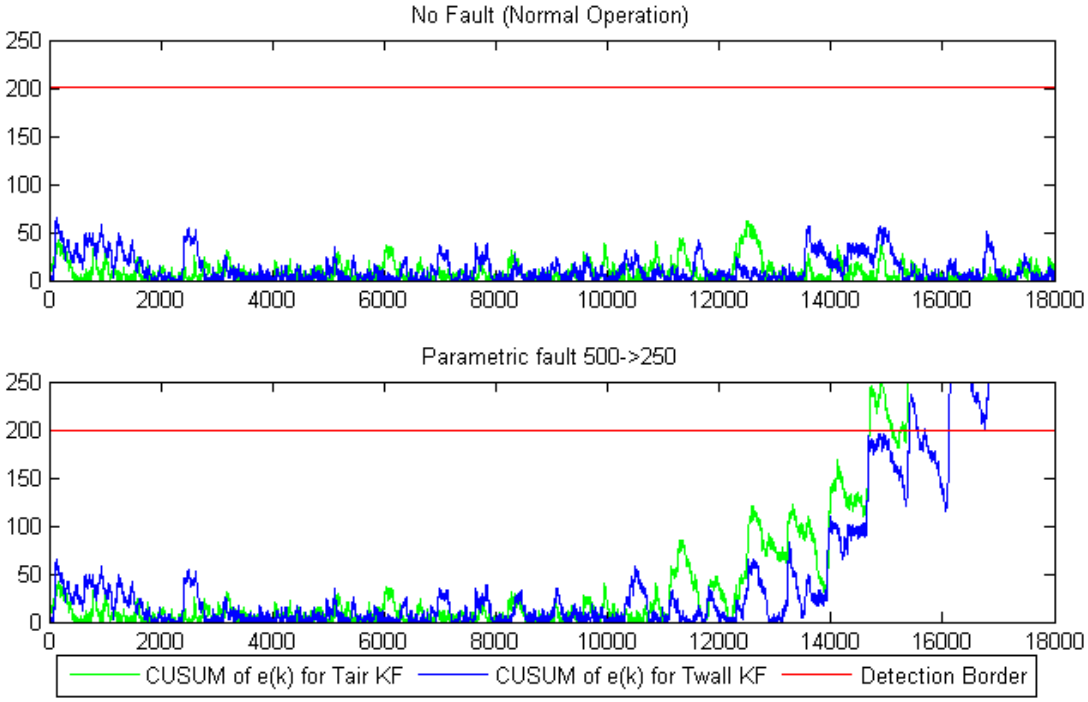


Figure 4.14: CUSUM for nominal situation and $UA_{air \rightarrow wall}$ drop (Dirt/Ice)

($UA_{air \rightarrow wall} = 500$), one for a full display case (200 Kg goods), and one which is close to empty (25 Kg). For each of the two models representing different amounts of goods within the case, another model needs to be created where the heat transfer coefficient is reduced to 250. This gives a total of four models, which are:

1. Nominal model ($M_{goods} = 200$ and $UA_{air \rightarrow wall} = 500$).
2. Non-full case, no fault ($M_{goods} = 25$ and $UA_{air \rightarrow wall} = 500$).
3. Full case, dirt/ice-over ($M_{goods} = 200$ and $UA_{air \rightarrow wall} = 250$).
4. Non-full case, dirt/ice-over ($M_{goods} = 25$ and $UA_{air \rightarrow wall} = 250$).

The models have all been found in the modeling chapter, and have for identification purposes been named *xyyyzzzz*, where x identifies the matrix (x=A for the A-matrix etc.), *yyy* is the $UA_{air \rightarrow wall}$ parameter and *zzzz* is either "Half" or "Full", where "Half" is used as "Non-Full". The nominal model matrices have been named by the same standard as well. The setup can be seen in Figure 4.15.

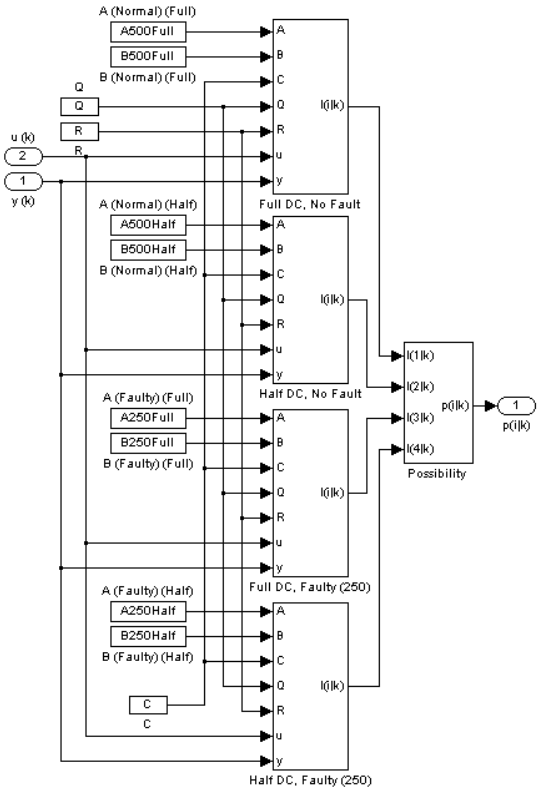


Figure 4.15: KF: Multiple Model Adaptive Estimator

To be able to reliably select the the most likely scenario, a method referred to as MMAE (Multiple Model Adaptive Estimation) has been used. This method calculates the probability of each model being true, based on the residual ($Res\,(k)$) and the residual covari-

ance ($S(k)$). The equation for calculating the probability is:

$$p(i|k) = \frac{l(i|k) \cdot p(i|k-1)}{\sum_{j=1}^{N} l(i|k) \cdot p(i|k-1)}$$

where N is the total amount of models, and $l(i|k)$ is the likelihood that the i'th model is correct at sample k, described by:

$$l(i|k) = \frac{1}{|2\pi S_i(k)|^{\frac{1}{2}}} \cdot \exp\left[-\frac{1}{2} \cdot {Res_i}^T(k) \cdot {S_i}^{-1}(k) \cdot Res_i(k)\right]$$

where it should be noted that $|2\pi S_i(k)|^{\frac{1}{2}}$ is the square root of the determinant of $2\pi S_i(k)$.

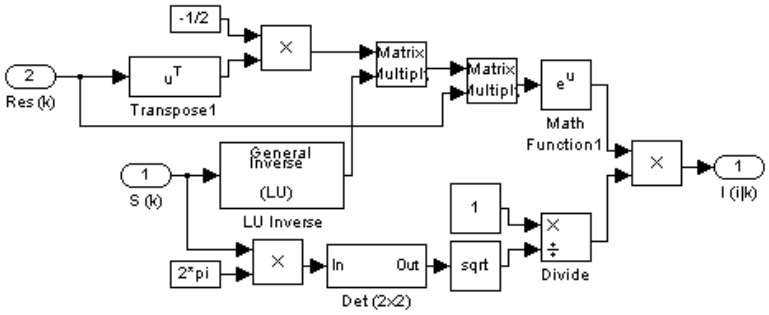


Figure 4.16: Likelihood Function in Simulink

Implementationwise, the likelihood is calculated locally within each Kalman filter in a submodel which is illustrated in Figure 4.16. As can be seen from the figure, $Res(k)$ and $S(k)$ are the only inputs, and the block gives the likelihood ($l(i|k)$) as output. A block was created for the purpose of finding the determinant of a 2x2 matrix. As with the remaining Simulink setup, Appendix C has a description of the determinant block. When

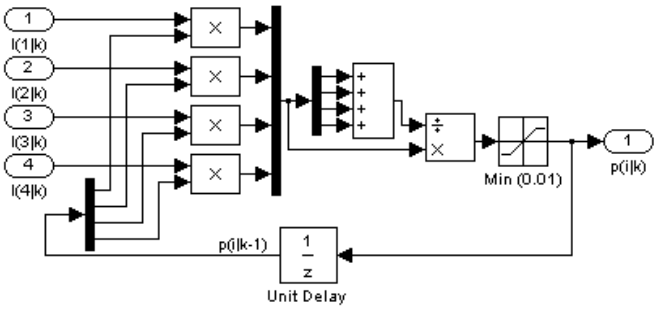


Figure 4.17: Possibility Function in Simulink

all likelihoods have been calculated, they are combined in another submodel to form the probability. This submodel can be seen in Figure 4.17. The model holds a saturation-block, that is necessary to make sure the probabilities do not lock themselves. In case any single probability (e.g. $p(s|k)$ for s) reaches zero, then it is locked to a probability of zero, as $p(s|k-1) = 0$ is multiplied with the new likelihood in the numerator. By

setting a lower border of 0.01 and no upper limit for all probabilities, the freedom for the probability calculator is still very high.

To verify that the MMAE approach can actually detect when a fault occurs without giving fake warnings, two scenarios have been tested. For the following three figures, it should be noted, that the first few hundred samples show an unsettled system and Kalman Filters. Both scenarios have a drift in $UA_{air \to wall}$ which is introduced at time t=6000 with a downward slope of -250/10000. At time t=16000 the slope is removed, and the value is stable at 250 for the remaining time.

The difference between the two scenarios, is the amount of goods inside the display case. For the first scenario, M_{goods} is fixed at 200 kg, while it is fixed at 25 for the other scenario.

The plant and Kalman filter should have more than enought time to stabilize at the beginning - and to adjust the prediction to the new scenario at the end of the change in mass. The probability outputs can be seen in Figure 4.18 and 4.19 on the following page, where

1. Blue Line: Nominal model ($M_{goods} = 200$ and $UA_{air \to wall} = 500$).
2. Green Line: Non-full case, no fault ($M_{goods} = 25$ and $UA_{air \to wall} = 500$).
3. Red Line: Full case, dirt/ice-over ($M_{goods} = 200$ and $UA_{air \to wall} = 250$).
4. Cyan Line: Non-full case, dirt/ice-over ($M_{goods} = 25$ and $UA_{air \to wall} = 250$).

The mathematical description of the development in the two figures is

1. $UA_{air \to wall} = \begin{cases} 500 & for\ t < 6000 \\ 500 - (t - 6000) \cdot 0.025 & for\ 6000 \leq t \leq 16000 \\ 250 & for\ t > 16000 \end{cases}$
   $M_{goods} = 200$ kg
   For Figure 4.18 on the next page
2. $UA_{air \to wall} = \begin{cases} 500 & for\ t < 6000 \\ 500 - (t - 6000) \cdot 0.025 & for\ 6000 \leq t \leq 16000 \\ 250 & for\ t > 16000 \end{cases}$
   $M_{goods} = 25$ kg
   For Figure 4.19 on the following page

As can be seen, the MMAE method does not give a clear result, but it gives a good estimate at the correct model. Based on the results of the above tests, it can be seen in Figures 4.18 and 4.19 on the next page that the system can detect when the evaporator is either iced over or built up with dirt given normal daytime operational conditions. As the $Q_{airload}$ value is an input to the system, the model should fit well under night-time operation as well, assuming the input is changed when the display case is covered. This will be checked for in Section 4.6.

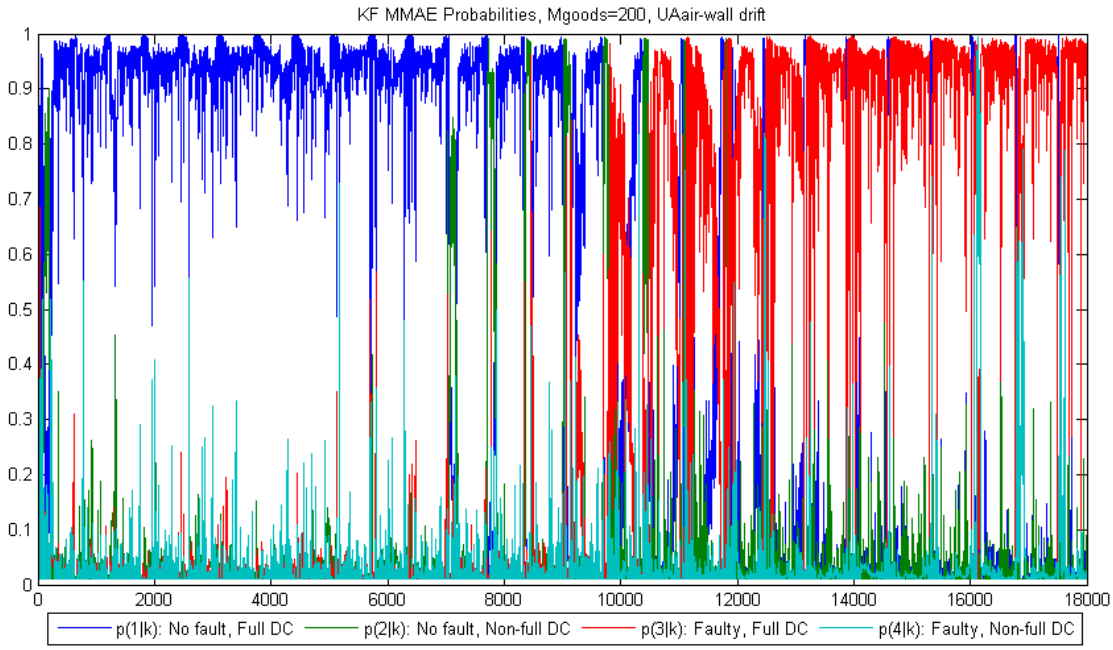


Figure 4.18: KF-MMAE: M_{goods}=200, Drift in $UA_{air \rightarrow wall}$

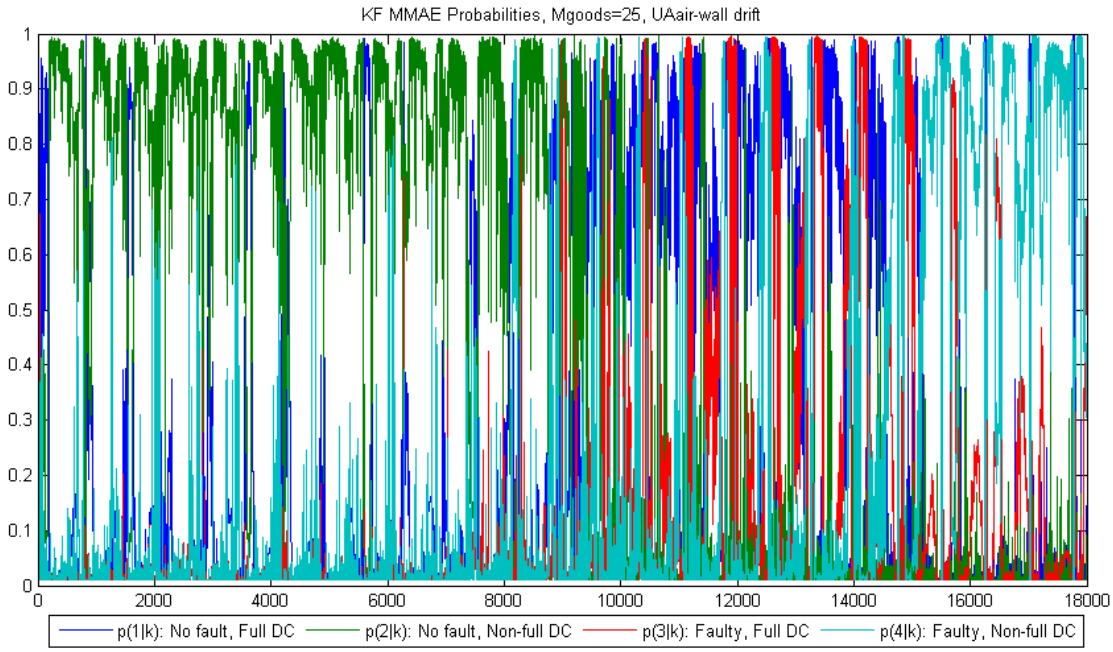


Figure 4.19: KF-MMAE: M_{goods}=25, Drift in $UA_{air \rightarrow wall}$

Another method referred to as IMM (Interactive Multiple Model), could theoretically give better results for fault isolation, but as the KF has some shortcommings, it has been decided not to spend time developing this method.

4.2.5 Complete FDI System

As a FDI system consists of several types of isolation used in conjunction with a detection method, it is necessary to determine how they should interact. If processing power is an issue, it is an advantage only to have the detection system active when running in a nominal situation, and then only activating the isolation algorithms in case of a fault. Unfortunately some faults are detectable but not possible to isolate using the Kalman Filter.

Based on this, it is not possible to create a complete FDI system using the Kalman Filter that can actually isolate all faults. As was visible from the split measurements approach, faults that follow the systems dynamics (drift and offset) are impossible to isolate. Complete sensor malfunction (freeze and hard-over) can be detected, and the sensor isolated - and so can parametric changes with the MMAE approach. Given the relative good results here, there is a chance the MMAE could also detect the problematic sensor faults. Unfortunately this would require several added models, as both models for a positive and a negative offset are required for each of the two sensors. If it is assumed that no faults occur simultaneously, and that the dynamics related to the goods inside the display case will not affect the model, four added models are sufficient. With that said, there is a good chance that false alerts will be generated, or that faults are not detected based on the uncertainty related to the linearization, model uncertainty in general, and the fact that the changes are relatively small. Finally, as changes related to the goods *will* probably affect the model, and the fact that the airload can change during the day, the chance of correctly isolating a small deviation in temperature is marginal.

The MMAE method for isolation of parametric faults on the other hand seems relatively good. Unfortunately, if the sensors are faulty in any way, the probability calculation is no longer reliable. Figure 4.20 illustrates the probabilities in an example where the T_{air} sensor freezes at time 9000. As can be seen from the figure, the sensor freeze fault makes the MMAE isolate the fault as being a parametric fault related to the $UA_{air \rightarrow wall}$ value for periods of time. Based on the mentioned problems, it has been decided to move on to other FDI methods, and stop development on the Linear Discrete Kalman Filter.

4.2.6 Conclusion

Detection based on the linear Kalman Filter is possible, but the results are not impressive. The model is trusted a factor 1000 more than the measurements, which in the worst case could result in model uncertainty triggering a false detection. Unlike fault detection, isolation was a failure. The implementation using split measurements to a bank of Kalman Filters can only detect complete failure of the sensors, and no fault where the dynamics are intact to some degree. The MMAE approach works quite well when no sensor faults are present, and could theoretically be used to detect and isolate ice-over and dirt-buildup in cases where the sensors are known to be non-faulty. By adding sensor fault charectaristics to additional models, the MMAE (or even IMM) approach could

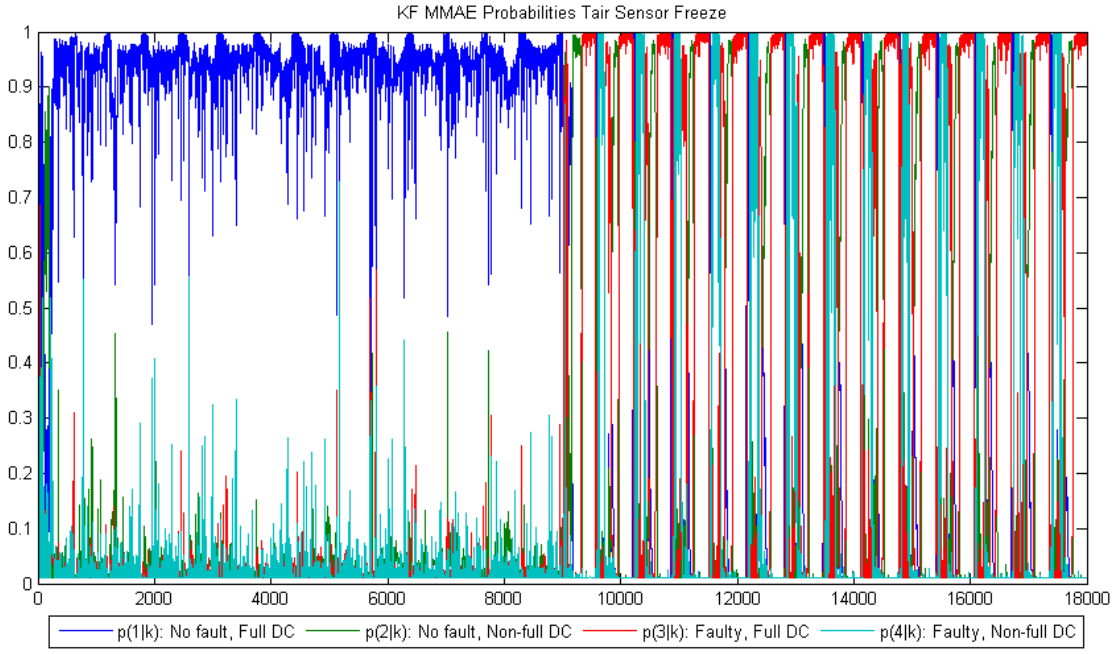


Figure 4.20: KF-MMAE: T_{air} Sensor Freeze

theoretically be used exclusively for fault isolation, and thereby eliminate the problems with the split measurements KF bank. Unfortunately this would require a large amount of models, and an equal amount of Kalman Filters in a bank. This approach has not been tested, but it should be noted that it could require substantional computational power and the models may be to identical for the MMAE method to actually determine the most reliable.

4.3 Extended Kalman Filter

The linearized KF (Kalman Filter) described in the previous section can be used to detect most faults but not always isolation. By using a more precise model, it should be possible to get better results for detecting and especially isolating faults. A way to gain a more precise model, is by using the Extended Kalman Filter.

As the EKF (Extended Kalman Filter) is an extension of the KF, the two methods share a lot of theory. For some theory, the description lies in the Kalman Filter section.

4.3.1 EKF Theory

In the same way as the KF, an initial (*a priori*) estimate is created, which combined with the prediction covariance creates the base for an update/correction procedure. The main difference is, that the EKF can overcome the uncertainty related to the linearization, by using a non-linear *a priori* estimation and a prediction covariance based on running local linearization.

Estimation

The basic theory of the estimation phase is the same for the KF and the EKF. The first step is to create an *a priori* state-estimate, but as mentioned the EKF uses a nonlinear model, described by

$$x(k) = f(x(k-1), u(k-1), w_{k-1}) \tag{4.3}$$
$$y(k) = h(k(k), v_k) \tag{4.4}$$

The estimation can then be described by

$$\hat{x}^-(k) = f(\hat{x}(k-1), u(k-1), 0) \tag{4.5}$$
$$\hat{y}(k) = c(k(k), 0) \tag{4.6}$$

where $\hat{x}(k-1)$ is obviously the *a posteriori* state estimate from the previous sample, and $u(k-1)$ is the input to the same sample. By using this method, the state estimation does not get affected by linearization problems, and a more precise estimate can be generated, not only within the bounds of a working-point.

The second step of the estimation, is calculating the predicted estimation covariance, which is described by:

$$P^-(k) = A_k \cdot P(k-1) \cdot {A^T}_k + W_k \cdot Q \cdot {W_k}^T$$

The equation is nearly identical to the one used in the KF, but two things have changed. The KF uses a pre-generated linearized version of the A matrix, where the EKF linearizes the matrix recursively around the previous sample as a Jacobian matrix. The other thing, is the addition of the W_k matrix. This is also an Jacobian matrix of partial derivatives of f with respect to w (the process noise, as described in the KF section). We can write a set of equations that linearize an estimate around equations 4.5 and 4.6.

$$x(k) \approx \hat{x}^-(k) + A(x(k-1) - \hat{x}(k-1)) + Ww_{k-1}$$
$$y(k) \approx \hat{y}(k) + H\left(x(k) - \hat{x}^-(k)\right) + Vv_k$$

where

$$A_{[i,j]} = \frac{\delta f_{[i]}}{\delta x_{[j]}}(\hat{x}(k-1), u(k-1), 0)$$
$$W_{[i,j]} = \frac{\delta f_{[i]}}{\delta w_{[j]}}(\hat{x}(k-1), u(k-1), 0)$$
$$C_{[i,j]} = \frac{\delta h_{[i]}}{\delta x_{[j]}}\left(\hat{x}^-(k-1), 0\right)$$
$$V_{[i,j]} = \frac{\delta h_{[i]}}{\delta v_{[j]}}\left(\hat{x}^-(k-1), 0\right)$$

As it can be hard to determine the specific dynamics of the noise in any given system, the W_k and V_k matrices are assumed to be the identity matrix for all samples, and the C matrix is linear and therefore fixed as well. Thereby the only dynamic part left is the A-matrix. The equations 4.3 and 4.3 can thereby be seen as

$$x(k) = f(x(k-1), u(k-1)) + w_{k-1}$$
$$y(k) = h(k(k)) + v_k)$$

Update

Theoretically the update phase of the EKF is identical to the KF with two exceptions. The first exception is that the C matrix is calculated recursively for each sample. As the C matrix for this model has no nonlinear elements it will never change and therefore will be identical for all samples, and be identical to the KF. The second exception is the addition of a V_k matrix in the calculation of the residual covariance, $S(k)$. This is the Jacobian matrix of partial derivatives of C with respect to v (the observation noise, as described in the KF section). In the same way as W_k, this is also assumed to be the identity matrix. The equations are:

$$\begin{aligned} Res(k) &= y(k) - C \cdot \hat{x}^-(k) \\ S(k) &= C \cdot P^-(k) \cdot C^T + V_k \cdot R \cdot {V_k}^T \\ K(k) &= P^-(k) \cdot C^T \cdot S(k)^{-1} \\ \hat{x}(k) &= \hat{x}^-(k) + K(k) \cdot Res(k) \\ P(k) &= (I - K(k) \cdot C) \cdot P^-(k) \end{aligned}$$

The final equation which must be mentioned, is the fault indicator, which is identical to the one used for the Kalman Filter.

$$e_k = Res^T(k) \cdot S^{-1}(k) \cdot Res(k)$$

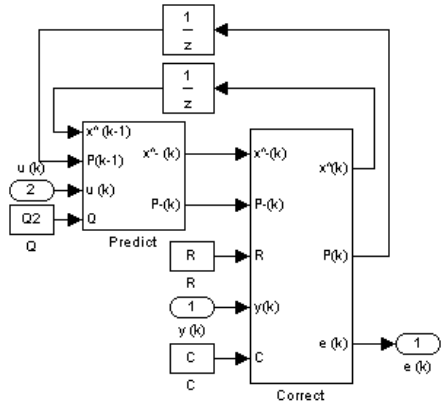
Now that the theory is in place, the implementation is described as it is a little different.

4.3.2 Implementation

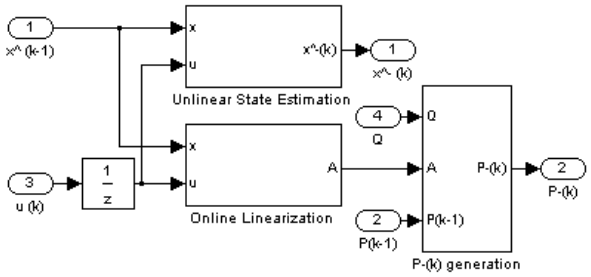
The implementation in Simulink can be seen in Figure 4.21, and is in general quite similar to the implementation of the KF. There are again two seperate submodels containing the estimation (Prediction) and update (Correct) procedures respectively. As there are several levels of models for the estimation phase, it is not described here. To find a description, see Appendix C. The update submodel only differs from the KF version in that the V_k matrix is added. Theoretically it is not necessary as long as it assumed to be the identity matrix, but is included in case V_k needs to be changed.

4.3.3 Fault Detection

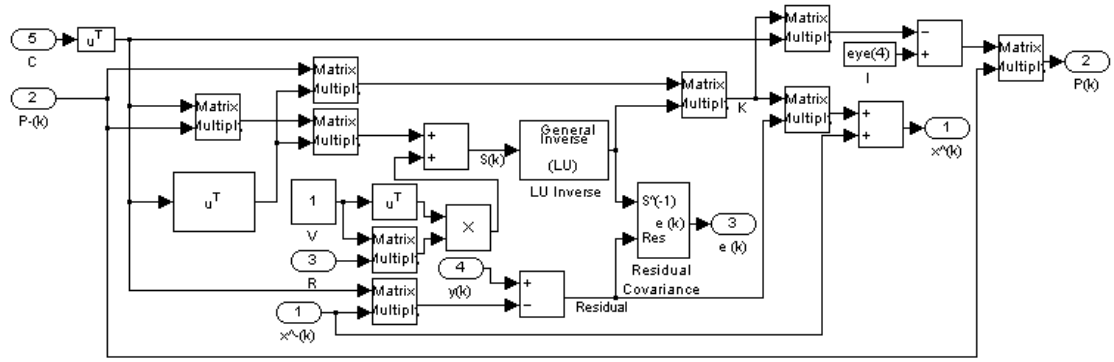
The fault detection procedure uses one instance of the EKF. By checking the size of the fault indicator ($e(k)$), or better yet the CUSUM based on the fault indicator, it is possible to detect if the measured signal dynamics fit with the model-based estimation. This filters out the main part of the noise, but if a fault occurs, the signal will change based on the dynamics of the fault. As the model is more precise when using an EKF for a nonlinear system, it is more reasonable to assume deviations are due to noise or faults as linearization errors are eliminated. By checking the magnitude of the fault indicator



(a) EKF (Main Model)



(b) Prediction (Submodel)



(c) Correct (Submodel)

Figure 4.21: Extended Kalman filter including primary sub-models

or CUSUM, it is possible to detect if a fault has occurred. By setting a maximum border for the signal, a fault can be claimed if the signal becomes larger than the border. As for the KF, this method is not robust enough

Similar to the KF, it is necessary to use a Q and R matrix. R is the same as before, as the manually added sensor noise is identical to the one which is used for the KF. Q was initially assumed to hold the same values, but it has been adjusted to give more reliable results.

$$R = \begin{bmatrix} 0.25 & 0 \\ 0 & 0.25 \end{bmatrix} \quad Q = \begin{bmatrix} 0.01 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0.01 \end{bmatrix}$$

With the given Q and R implemented, the different faults can now be introduced to find the fault-detection border.

- Sensor Drift
  Implementation: Add a ramp with a slope of $+0.001\,^{\circ}\mathrm{C}$ per second/sample.
- Sensor Offset
  Implementation: Add a fixed value of $2.5\,^{\circ}\mathrm{C}$ to each measurement.

- Sensor Freeze
  Implementation: Repeat the previous signal.
- Sensor Hard-Over
  Implementation: Replace measured signal by a fixed value of 25 °C.
- Dirt/Ice buildup ($UA_{air \to wall}$ drop)
  Implementation: Steady drop in $UA_{air \to wall}$ from 500 at time t=6000 to 250 at time t=16000.

Figure 4.22 and 4.22 shows the CUSUM of $e(k)$ (the fault indicator) for T_{air} and T_{wall}. Each figure consists of four sub-figures, plotting a five hour simulation. Figure 4.24 on the next page shows the signal for the nominal (non-faulty) condition and a parametric fault on $UA_{air \to wall}$, also for a five hour period. All scenarios are non-faulty until sample

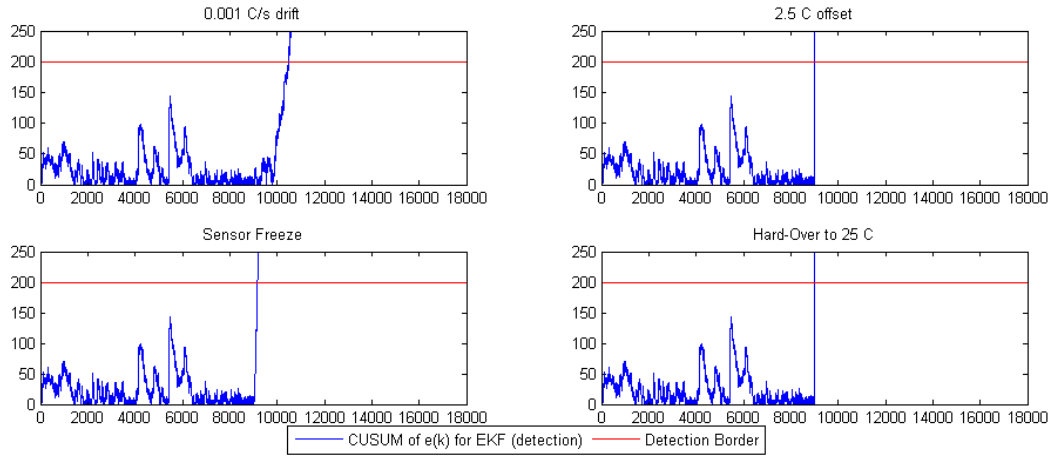


Figure 4.22: CUSUM for T_{air} sensor faults

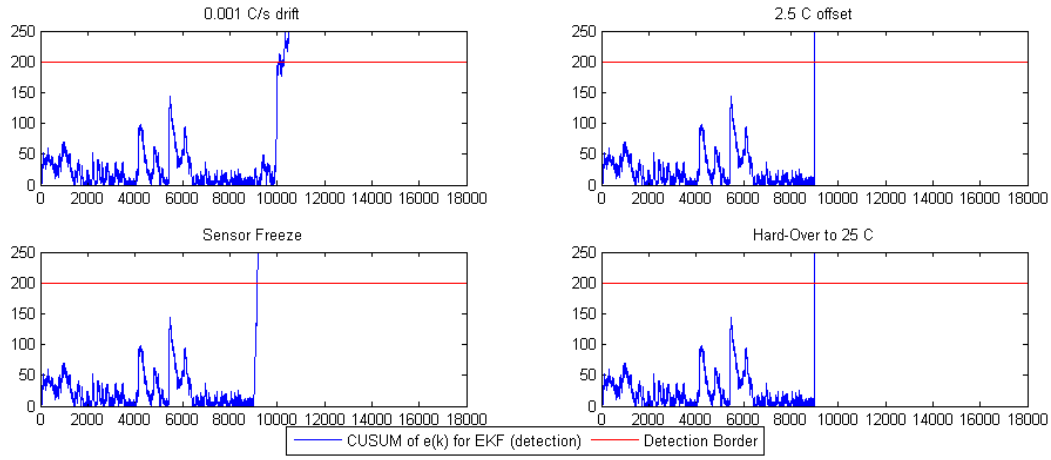


Figure 4.23: CUSUM for T_{wall} sensor faults

9000, where each fault is introduced respectively. The figures clearly show that all faults are easy to detect with the EKF. All types of faults generate large values using the

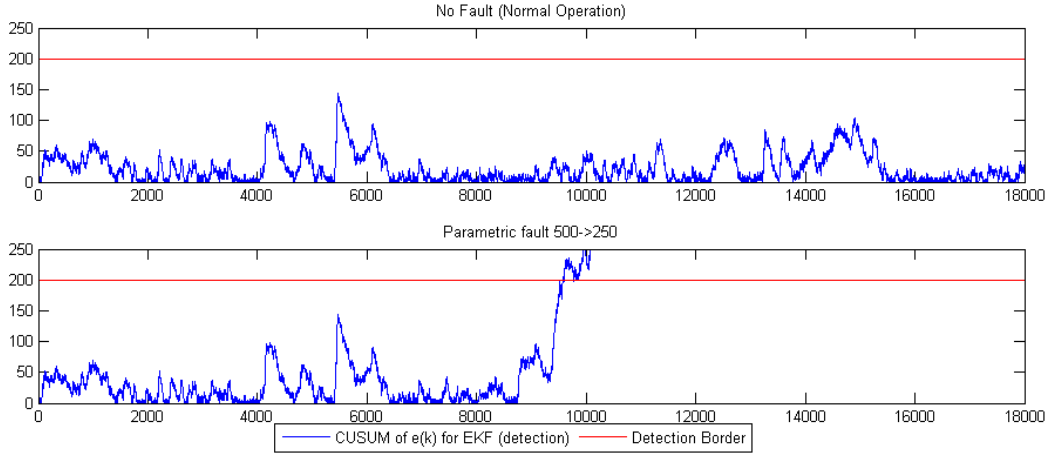


Figure 4.24: CUSUM for nominal and $UA_{air \rightarrow wall}$ drop (Dirt/Ice)

CUSUM method. By using a detection border of 200 again, all faults will be identified, without giving false alerts (based on the test-scenarios). To illustrate when a fault will be claimed, the border has been drawn as a red line on the previous figures.

4.3.4 Fault Isolation

When a fault has been detected, it needs to be isolated. Just like the KF, the isolation procedure can be divided into two sections. The first is isolation of sensor faults, the other is isolation of parametric faults.

Isolation of sensor faults

To isolate a sensor fault the measured outputs from the plant are split up, and fed to two different EKF's. By using this approach, if one sensor fails, the fault indicator signal (and thereby the CUSUM) for that sensor will claim a fault while the fault indicator for EKF with the good sensor as input will be fine. In this way, it is possible to see that one sensor is faulty, and the system itself is not. The faulty sensor is then isolated, and a warning is given to the operator, so the sensor can be changed or repaired as fast as possible. The described setup can be seen in Figure 4.25 on the following page. The basic setup is identical to the one used for the KF, and in the same way the R matrix is reduced to a single scalar of 0.25. The Q matrix is identical to the EKF used for detection. Figure 4.26 and 4.27 on the next page show the CUSUM output for all four sensor faults for T_{air} and T_{wall} respectively. Figure 4.28 shows the nominal non-faulty scenario and a scenario with a drop in $UA_{air \rightarrow wall}$ going from 500 up until time 6000 down to 250 at time 16000 and on.

From the three figures it can be seen that all faults, both sensor and parametric, are easily detected by using a border of 200, which is again marked by a red line. If the signal goes above this border, a fault can be claimed.

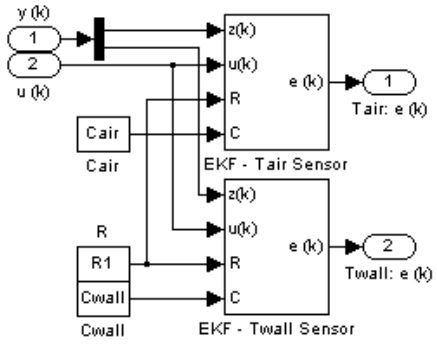


Figure 4.25: split measurement implementation for EKF

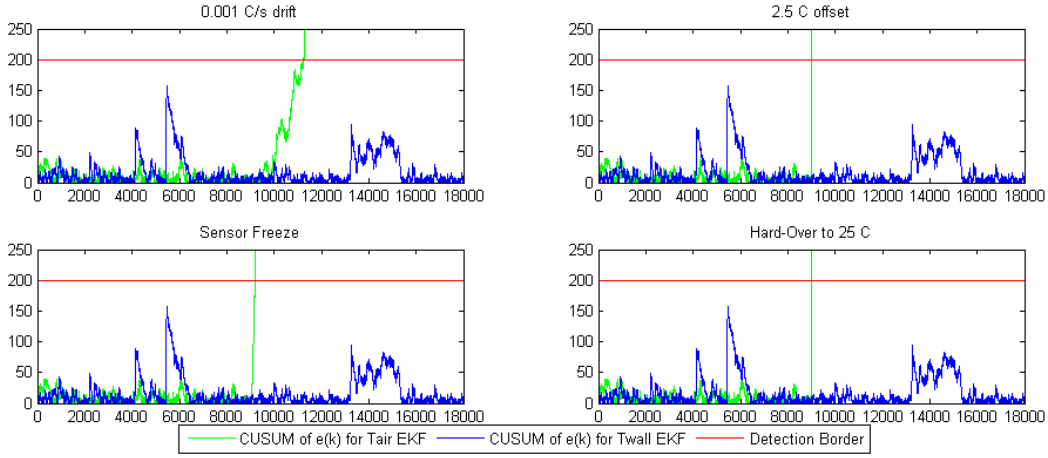


Figure 4.26: CUSUM for T_{air} sensor faults

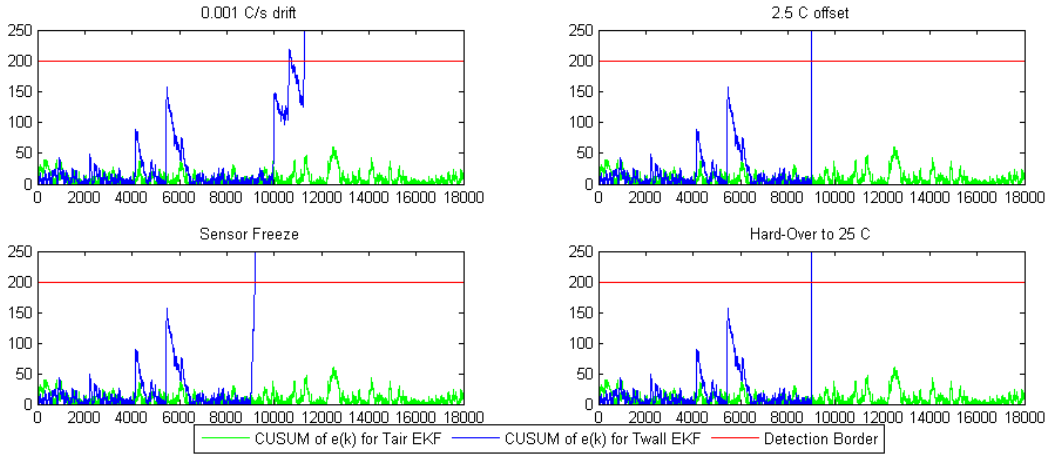


Figure 4.27: CUSUM for T_{wall} sensor faults

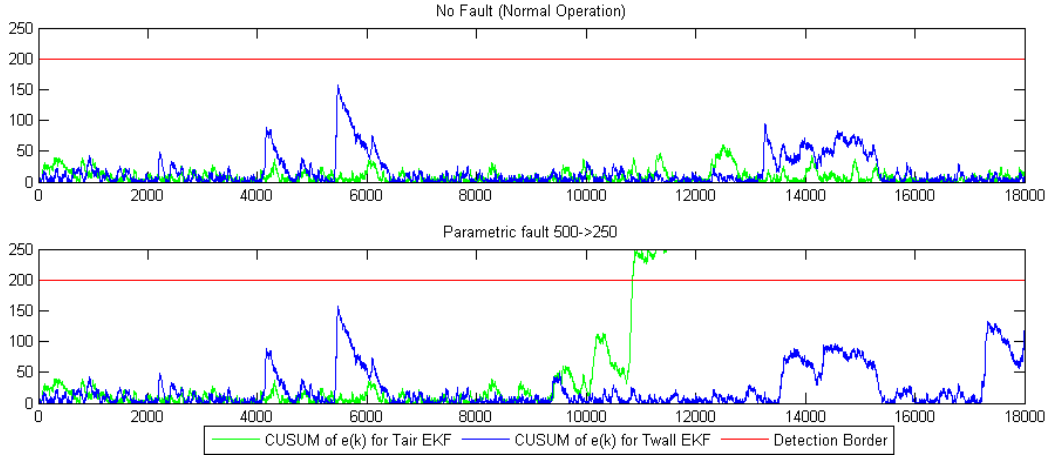


Figure 4.28: CUSUM for nominal and $UA_{air \rightarrow wall}$ drop (Dirt/Ice)

Isolation of Parametric Fault

Isolating a parametric fault is similar to the linear Kalman Filter based on Multiple Model Adaptive Estimation (MMAE). Again the parameters related to the goods are allowed to change, but not the remaining parameters, especially $UA_{air \rightarrow wall}$. A change in this value indicates a buildup of dirt or ice on the evaporator. To differ between allowable changes and faulty scenarios, a total of four models have been used. They are:

1. Nominal model ($M_{goods} = 200$ kg and $UA_{air \rightarrow wall} = 500$).
2. Non-full case, no fault ($M_{goods} = 25$ kg and $UA_{air \rightarrow wall} = 500$).
3. Full case, dirt/ice-over ($M_{goods} = 200$ kg and $UA_{air \rightarrow wall} = 250$).
4. Non-full case, dirt/ice-over ($M_{goods} = 25$ kg and $UA_{air \rightarrow wall} = 250$).

The same likelihood and probability submodels which were used for the KF has been re-used for the EKF. These are described in the KF section, and in Appendix C, so only the overall setup will be illustrated here, and can be seen in Figure 4.29 on the following page. The implementation of faulty models was done just by changing the values in the equations. Q and R are re-used from the detection section. The same two scenarios that were tested on the KF are tested here as well:

1. $$UA_{air \rightarrow wall} = \begin{cases} 500 & for\ t < 6000 \\ 500 - (t - 6000) \cdot 0.025 & for\ 6000 \leq t \leq 16000 \\ 250 & for\ t > 16000 \end{cases}$$
   $M_{goods} = 200$ kg
   For Figure 4.18 on page 54

2. $$UA_{air \rightarrow wall} = \begin{cases} 500 & for\ t < 6000 \\ 500 - (t - 6000) \cdot 0.025 & for\ 6000 \leq t \leq 16000 \\ 250 & for\ t > 16000 \end{cases}$$
   $M_{goods} = 25$ kg
   For Figure 4.19 on page 54

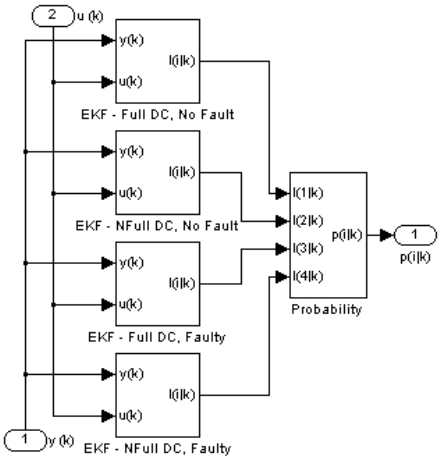


Figure 4.29: MMAE implementation for EKF

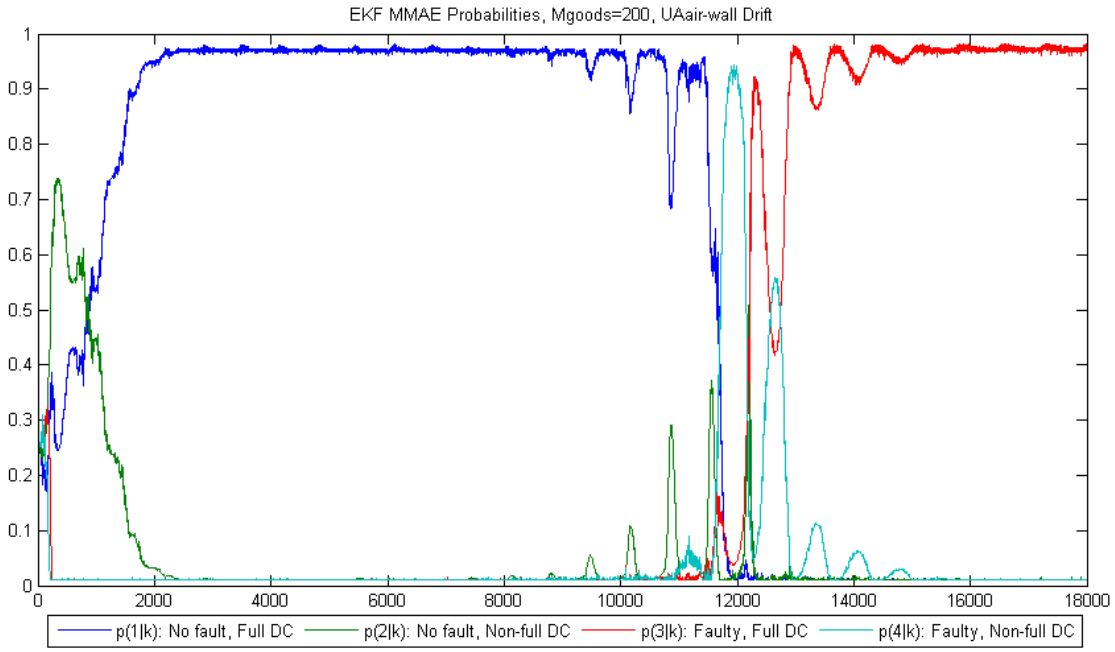


Figure 4.30: EKF-MMAE: M_{goods}=200, Drift in $UA_{air \rightarrow wall}$

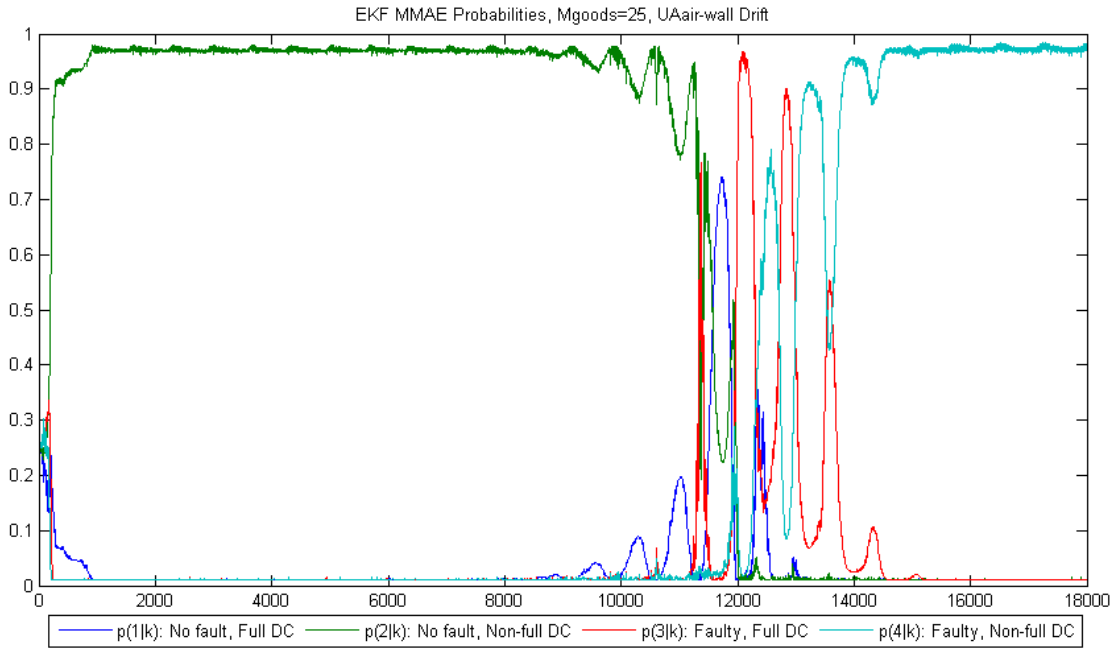


Figure 4.31: EKF-MMAE: M_{goods}=25, Drift in $UA_{air \rightarrow wall}$

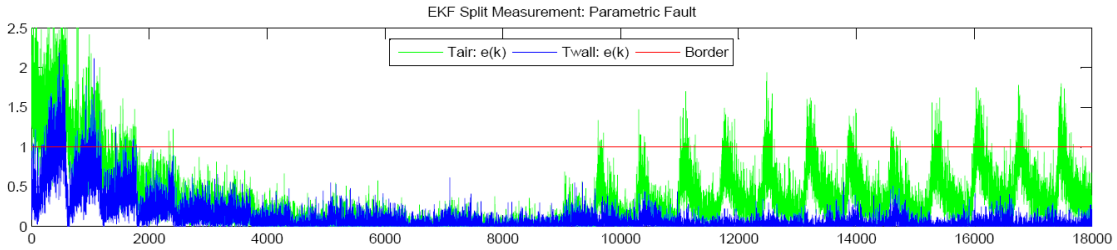
The MMAE for the EKF detects the true scenario in all cases with a much better result than the KF, even though it some times takes some time to adjust itself to the right scenario. The MMAE for the EKF is thereby successful.

4.3.5 Complete FDI System

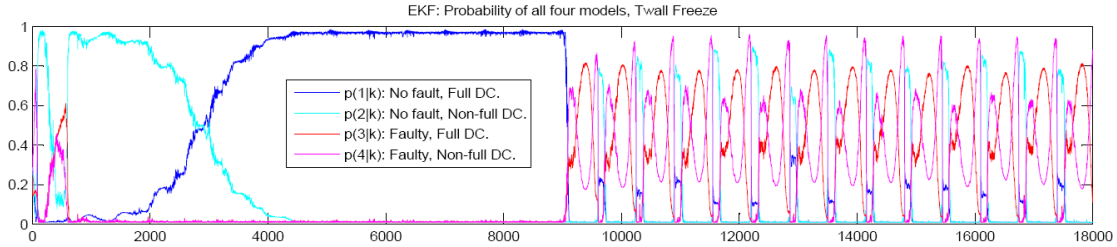
To use the EKF as a complete FDI system, it must be able to detect and isolate faults. As was illustrated, detecting faults is no problem for the EKF, and fault isolation has been successful as well, but there is still a problem. When introducing a sensor fault, the MMAE approach will in most cases claim a faulty scenario as the most likely. In other words, $p(i|k)$ will regularly be higher for a faulty model, even though the parameter has not changed.

The problem also exists when a parametric fault has occured where the split signal method will also claim a fault. This problem makes it impossible to distinguish between the two types of fault.

To illustrate the problem, Figure 4.32a on the next page shows the CUSUM of the fault indicator for the Split Signal approach with a parametric fault introduced, while Figure 4.32b on the following page illustrates the MMAE output when a sensor fault occurs. The parametric fault is identical to the one used elsewhere in the report. At time t=6000 the parameter $UA_{air \rightarrow wall}$ starts to drop until time t=16000 where it stabilizes at 250. The sensor fault is the standard offset fault, where a fixed value of 2.5 is added to the measurements. The figures clearly illustrate that it is not possible to determine what kind of fault has happened. Developing a method using the EKF exclusively has based on this problem been abandoned.



(a) Split measurement residual output, parametric fault



(b) MMAE output, sensor freeze

Figure 4.32: False fault isolation problems

4.3.6 Conclusion

Detection using the EKF gives good results, and so does the isolation. The difference between nominal condition and faulty is larger, and even relatively small faults can be detected, and isolated as well. Unfortunately, the EKF has problems isolating the type of fault, as it cannot determine of a fault is parametric or a sensor fault. If one of the two scenarios could somehow be ruled out, a complete FDI system could be made using only the EKF.

4.4 Unknown Input Observer

The Unknown Input Observer (UIO) is primarily used when a dominating unknown input exists, for instance a disturbance. The general thought behind this method, is that states that are affected by disturbances are decoupled from the unknown input (disturbance). By defining $Q_{airload}$ as an unknown input, its effect does not need to be modeled, and it can change freely without affecting the model. Futhermore, to use the UIO for the refrigeration plant model used in this project, it is necessary to detach T_{goods} (an explanation of this will be given later in this section). T_{goods} will thereby also become a disturbance. This is by all means an advantage, as any change in the goods dynamics will not make the system give false alerts.

4.4.1 UIO Theory

The formal definition of an Unknown Input Observer is

An observer is defined as an UIO for the system described by

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) + Ed(t) \\ y(t) &= Cx(t) + f_s \end{aligned} \tag{4.7}$$

if its state estimation error vector $e(t)$ approaches zero asymptotically; regardless of the presence of the unknown input (disturbance) in the system.

Equation 4.7 is the standard state space representation of a dynamic system without direct relationship between in- and output. The final state space variables in each line ($Ed(t)$ and f_s) are not always used and are thereby introduced; $d(t)$ is the unknown disturbance input vector goverened by the matrix E and f_s is an additive bias signal from the sensor which can be considered as a sensor fault. It should be noted that the term $Ed(t)$ is described as an additive disturbance and can relate to different kinds of modelling uncertainties (e.g. noise, non-linear terms, linearization, approximation errors and parameter variations). Another thing to notice about the definition, is the term "unknown input", which is obviously a main part of the Unknown Input Observer.

A full-order UIO structure is described by

$$\begin{aligned} \dot{z}(t) &= Fz(t) + TBu(t) + Ky(t) \\ \hat{x}(t) &= z(t) + Hy(t) \end{aligned} \tag{4.8}$$

where $z(t)$ is the state of the UIO, $\hat{x}(t)$ is the estimate of the state vector $x(t)$, and F, T, H and K are matrices to be designed. The UIO can be seen visualized as a block-diagram in Figure 4.33. A method to design H, T, F and K matrices has been proposed

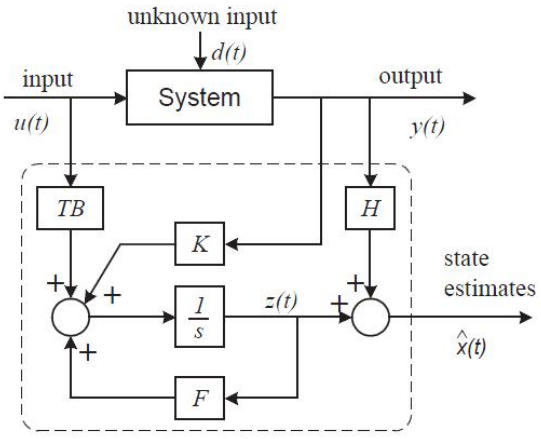


Figure 4.33: The structure of UIO [CPZ96]

in [CPZ96]. The method starts by looking at the state estimation error, which is defined as $e(t) = x(t) - \hat{x}(t)$. By inserting 4.8 and 4.7 the result is:

$$\begin{aligned} \dot{e}(t) = [A - HCA - K_1C]\, e(t) + [F - (A - HCA - K_1C)]\, z(t) \\ + [K_2 - (A - HCA - K_1C)]\, y(t) + [T - (I - HC)]\, Bu(t) \\ + (HC - I)\, Ed(t) \end{aligned}$$

where

$$K = K_1 + K_2 \tag{4.9}$$

If the following relations hold true

$$(HC - I)\,E = 0$$
$$T = I - HC \tag{4.10}$$
$$F = A - HCA - K_1 C \tag{4.11}$$
$$K_2 = FH \tag{4.12}$$

then the state estimation error will be $\dot{e}\,(t) = Fe\,(t)$. This means that if the eigenvalues of the F matrix are stable, $\dot{e}\,(t)$ will approach zero asymptotically, $\hat{x} \rightarrow x$. Hence, the observer described by equation 4.8 on the preceding page is an UIO for the system in equation 4.7 according to the definition for an UIO.

Two conditions that must be met before an UIO can exist are

1. $rank\,(CE) = rank\,(E)$
2. (A_1, C) must be a detectable pair

where

$$A_1 = A - HCA \tag{4.13}$$
$$H = E\left[(CE)^T CE\right]^{-1} (CE)^T \tag{4.14}$$

If these conditions are met, then there exists a solution for the matrix H illustrated above.

4.4.2 Implementation

This section provides an overview of the implementation of the UIO, where the technique and algorithms are based on the UIO theory in the previous section. The UIO is implemented according to the block diagram in Figure 4.34, which can be confirmed by comparing it to the implementation illustrated in Figure 4.33 on the previous page.

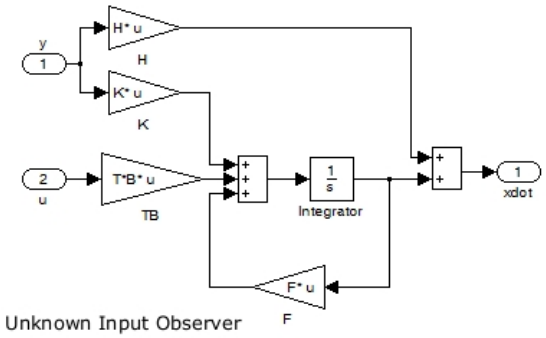


Figure 4.34: Block diagram for UIO implementation

As can be seen from the figures, y and u are taken as input and the state estimation is the only output. To create the F, T, K and H matrices, a model of the system is

necessary. This is established in the modeling chapter, and is reduced by a single state to become a three state model. The reason for a reduction in states, is the fact that the unknown input ($Q_{airload}$) is related to the T_{air} state, and when calculating A_1 for the four state setup, the T_{goods} state is disconnected from the remaining states, and is thereby unobservable. As the T_{goods} state only affects T_{air}, and the goods are unimportant in relation to fault detection in the display case, the dynamics related to the goods can be eliminated from the model and will become a part of the dynamics of the disturbance affecting T_{air}. In this way, there is a single disturbance to the system which only affects T_{air}, and is actually a combination of T_{goods} and $Q_{airload}$. The relevant model matrices from the modeling chapter are reintroduced below.

$$A = \begin{bmatrix} -0.0139 & 0.0090 & 0 \\ 0.0110 & -0.0090 & -0.6004 \\ 0 & 0 & -0.1049 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 \\ 0 & -0.0168 \\ 0.0570 & 0 \end{bmatrix} \tag{4.15}$$

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad E = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{4.16}$$

The first thing to do when creating an Unknown Input Observer, is verifying that one actually exists for the system. The first condition is as mentioned, that $rank(C * E) = rank(E)$. The other condition is that (A_1, C) is a detectable pair.

The rank of E is obviously 1, and the rank of CE is

$$rank\left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}\right) = rank\left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}\right) = 1$$

Thereby the first condition is met. To verify that the second condition is also met, the observer canonical form of A_1 and C must have full column rank. First, by using equations 4.14 and 4.13 H and A_1 are calculated.

$$H = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \quad A_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0.0110 & -0.0090 & -0.6004 \\ 0 & 0 & -0.1049 \end{bmatrix}$$

Now that A_1 has been calculated, the observability matrix for (A_1, C) can be found

$$\begin{bmatrix} C \\ CA_1 \\ CA_1{}^2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0.0105 & -0.0090 & -0.4941 \\ 0 & 0 & 0 \\ -0.0001 & 0.0001 & 0.0480 \end{bmatrix}$$

The observability matrix already suggest that it is observable, but to be sure, the function rref in MATLAB can be called with the above result as an argument, which will give the

reduced row echelon form of the observable matrix (A1,C)

$$rref\,(obs) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{4.17}$$

As can be seen the rank of the observable matrix (A1,C) is 3, so the system is fully observable.

Now that is has been verified that an UIO exists, it can be designed. The matrix F must be able to stablilize the fault, which can be achieved by selecting the values of $K1$. The poles can be any randomly selected poles which keep F stable (poles in the left half plane). The poles for $A1$ can be found by using the function $eig(A1)$, which results in

$$eig(A1) = \begin{bmatrix} -0.0090 & 0 & -0.1049 \end{bmatrix} \tag{4.18}$$

The system is marginally stable, but as it needs to be completely stable, three new stable poles are selected. By selecting poles that are relatively close to the ones of $A1$ the gains of $K1$ are kept relatively small. The selected poles are

$$poles = \begin{bmatrix} -0.001 & -0.01 & -0.1 \end{bmatrix} \tag{4.19}$$

With MATLAB, the gain matrix $K1$ is calculated by using the function $K1 = (place(A1', C', poles))'$, whereafter T, F, $K2$ and finally K can be calculated by using equations 4.10, 4.11, 4.12 and 4.9 respectively.

$$K1 = \begin{bmatrix} 0.0025 & 0.0005 \\ 0.0511 & -0.0054 \\ 0.0034 & -0.0008 \end{bmatrix} K2 = \begin{bmatrix} -0.0025 & 0 \\ -0.0401 & 0 \\ -0.0034 & 0 \end{bmatrix} K = \begin{bmatrix} 0 & 0.0005 \\ 0.0110 & -0.0054 \\ 0 & -0.0008 \end{bmatrix}$$

$$F = \begin{bmatrix} -0.0025 & -0.0005 & 0 \\ -0.0401 & -0.0035 & -0.6004 \\ -0.0034 & 0.0008 & -0.1049 \end{bmatrix} T = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

With the matrices determined, the UIO can be implemented.

4.4.3 Fault Detection

The main advantage of the UIO-setup, is that both $Q_{airload}$ and T_{goods} are now considered as disturbances, and a change in their dynamics (which can happen during normal operating conditions) will not trigger a false fault detection. Only the fixed part of the dynamics is modeled, which makes the setup much more robust against false detection.

To detect faults, a single UIO is used, where a fault indicator ($r(k)$) is compared to an upper bound (a threshold). The fault indicator is actually just a normalization, and is calculated recursively by the following formula

$$r\,(k) = e\,(k)^T \cdot e\,(k) \qquad where \quad e(k) = x(k) - \hat{x}(k)$$

To determine the bound, faults have to be introduced to the system, and the size of the residual will determine when a fault should be claimed. The faults are described in Section 4.1, and will not be discussed further in this section. The effect of the faults can be seen in Figure 4.35 and 4.36 on the following page for T_{air} and T_{wall} sensor faults respectively, and finally Figure 4.37 for a fault in $UA_{air \to wall}$. All faults are introduced after 9000 samples, except for the parametric fault which is nominal up to time t=6000 where a downward slope is added until time t=16000 where $UA_{air \to wall}$ stabilizes at 250.

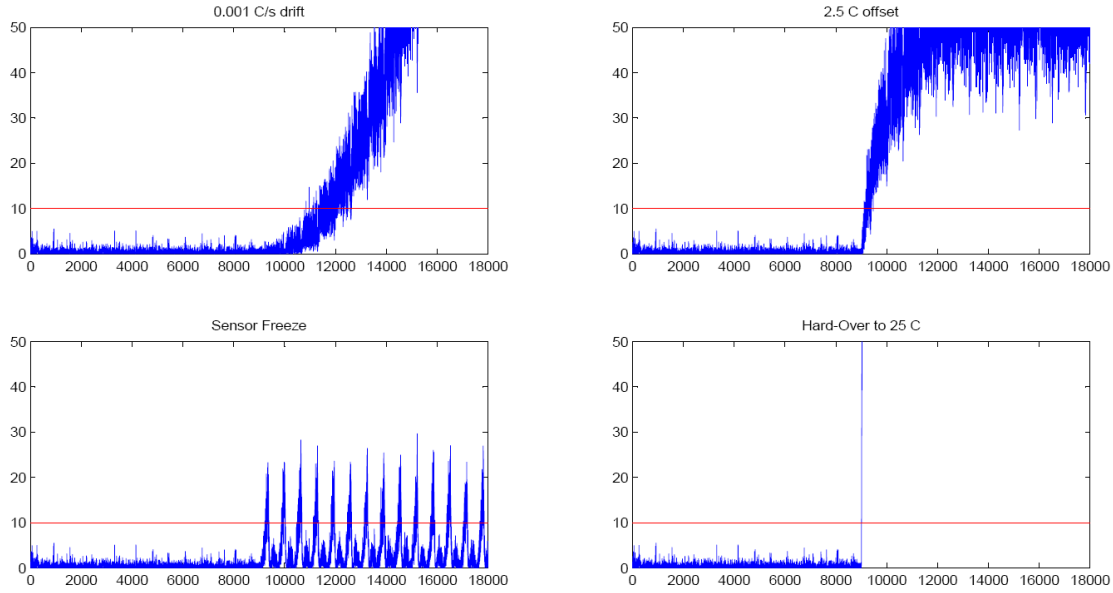


Figure 4.35: T_{air} sensor faults

As can be seen from Figures 4.35 to 4.37 on pages 71–72, all faults are easily detectable. Based on these results, the CUSUM (Cumulative Sum) method is not necessary. Even with small faults, the residual changes dramatically, so a suggested border/threshold of 10 is included in Figures 4.35 to 4.37 to illustrate when a detection would occur.

4.4.4 Fault Isolation

In the same way as the Kalman filter and EKF, it is possible to use a bank of UIO's. The two previously used methods were split sensor, and a method based on multiple models. The split sensor method will not work for the UIO, as the T_{air} measurement is required for the UIO to work. Furthermore, if the T_{wall} measurement is ignored, the residual could easily get quite large, as T_{air} is allowed to change due to disturbances.

Using multiple models will allow detection of parametric faults (ice-over and dirt buildup), but not detection of sensor faults. Obviously it is necessary to create new models for each UIO, and thereby theoretically also new values for H, K, T and F. In reality, only F and K change, so the remaining two can be re-used. Three new models are created with values of 450, 375 and 300 for $UA_{air \to wall}$. The model creation is specified in the

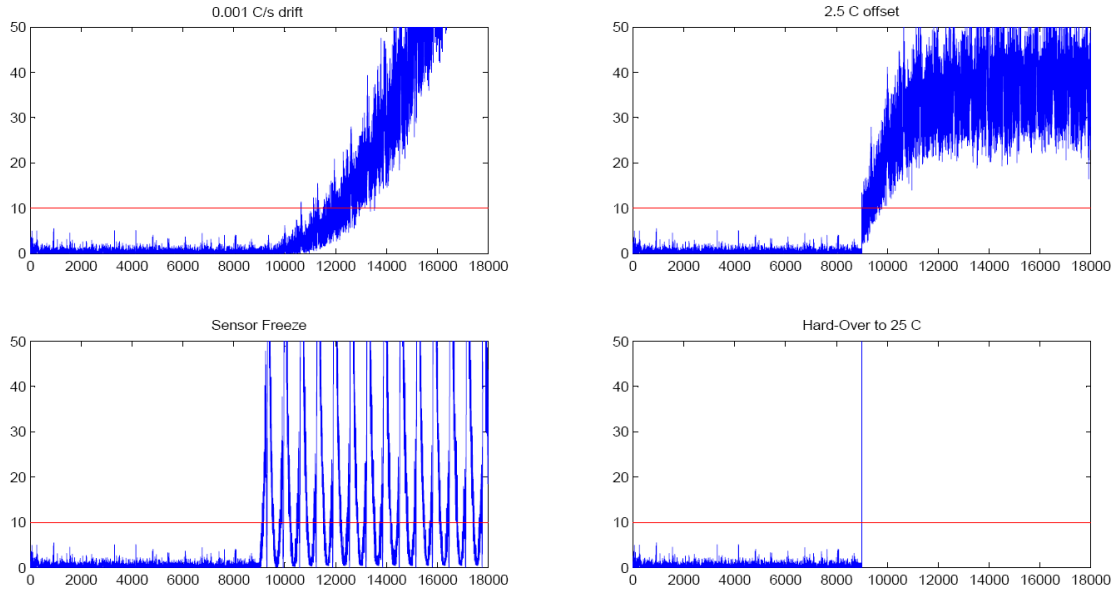


Figure 4.36: T_{wall} sensor faults

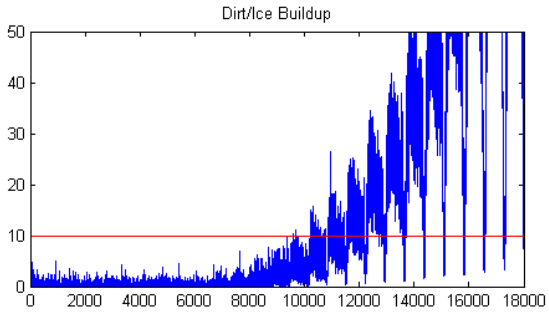


Figure 4.37: Dirt/Ice buildup ($UA_{air \to wall}$ drift)

modeling chapter, and the matrices are as follows

$$F_{f1} = \begin{bmatrix} -0.0100 & 5.8 \cdot 10^{-5} & 0 \\ 2.3 \cdot 10^{-4} & 0.0774 & -0.3962 \\ 6.9 \cdot 10^{-5} & 0.0351 & -0.1784 \end{bmatrix} \quad K_{f1} = \begin{bmatrix} 0 & -5.8 \cdot 10^{-5} \\ 0.0054 & -0.0823 \\ 0 & -0.0351 \end{bmatrix}$$

$$F_{f2} = \begin{bmatrix} -0.0099 & -1.5 \cdot 10^{-4} & 0 \\ -0.0084 & 0.0765 & -0.4841 \\ -0.0021 & 0.0283 & -0.1776 \end{bmatrix} \quad K_{f2} = \begin{bmatrix} 0 & 1.5 \cdot 10^{-4} \\ 0.0045 & -0.0807 \\ 0 & -0.0283 \end{bmatrix}$$

$$F_{f3} = \begin{bmatrix} -0.0085 & 4.6 \cdot 10^{-4} & 0 \\ 0.0486 & -0.0672 & -0.6789 \\ -0.0014 & -0.0031 & -0.0353 \end{bmatrix} \quad K_{f3} = \begin{bmatrix} 0 & -4.6 \cdot 10^{-4} \\ 0.0765 & 0.0283 \\ 0 & 0.0031 \end{bmatrix}$$

where the subscript denotes the faults

- $f1$: $UA_{air \to wall} = 450$
- $f2$: $UA_{air \to wall} = 375$

- $f3$: $UA_{air \rightarrow wall} = 300$

By comparing the three fault indicators at each sample and selecting the smallest value, it can be used for isolation. With an isolation border of 50, it can be assumed that when at least one of the models have a fault indicator below 50, the fault is related to the $UA_{air \rightarrow wall}$ parameter, and thereby the fault can be isolated. It has been decided that values for $UA_{air \rightarrow wall}$ below approximately 250 should no longer be thought of as parametric fault, but instead something more severe. This is based on a test with an $UA_{air \rightarrow wall}$ value of 200 which made the dynamics of the plant change drastically. Figure 4.38 illustrates a test where $UA_{air \rightarrow wall}$ is gradually taken down from 500 to 250 in steps of 50, lasting an hour (3600 seconds) each after a 3600 second stabilizing period at the beginning. The isolation border of 50 has also been illustrated, where a parametric fault on $UA_{air \rightarrow wall}$ will be claimed if the fault indicator is below. If all the fault indicators go above the border, the fault is no longer assumed to be due to ice/dirt buildup and can be passed on to some other isolation procedure.

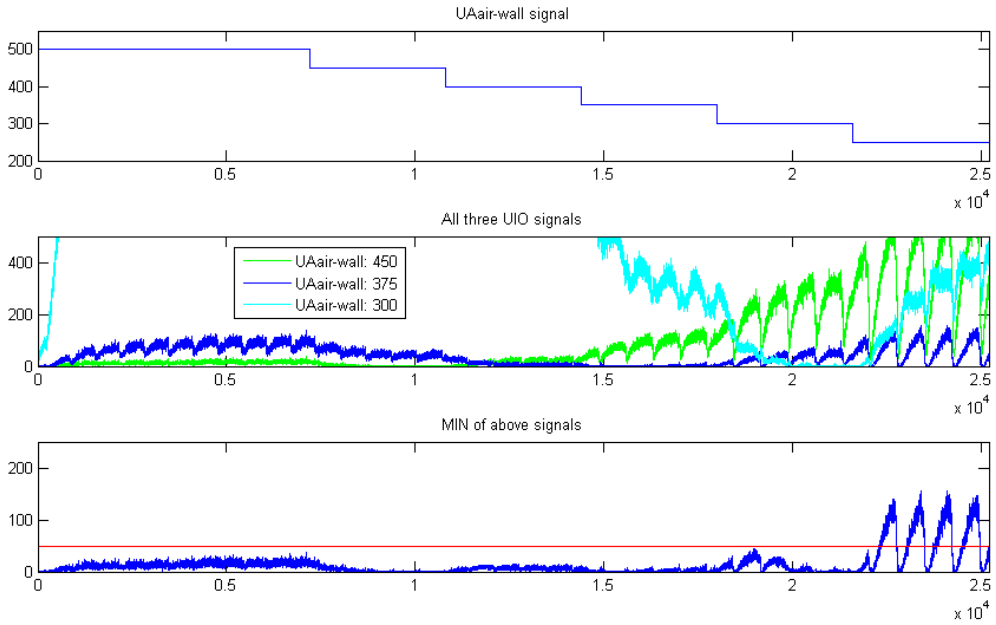


Figure 4.38: UIO: Dirt/Ice buildup fault isolation

From Figure 4.38 it can be seen that if $UA_{air \rightarrow wall}$ drops to 250 it is isolated as not being a parametric fault. From 300 up to 500 (both inclusive), the fault is assumed to be due to a parametric fault in $UA_{air \rightarrow wall}$. To verify that a sensor fault will not be claimed as a parametric fault, two tests have been made with the system in nominal operating conditions. The first is a drift fault on T_{wall} with a slope of 0.001, while the other is an offset fault on T_{air} of +2.5 degrees, both beginning at time t=9000. These faults have been confirmed to trigger a detection, and from Figure 4.39 on the next page it is evident that these faults are not isolated as parametric faults.

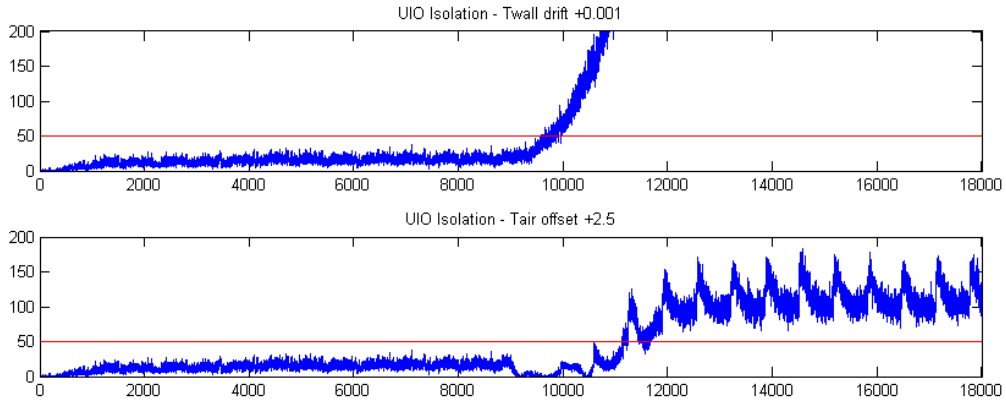


Figure 4.39: UIO: Sensor Faults in Parametric isolation

4.4.5 Complete FDI System

As the UIO cannot isolate sensor faults, it is not possible to create a complete FDI solution using only UIO. The UIO method can be used for fault detection but not for fault isolation, unless it is in cooperation with another method.

4.4.6 Conclusion

In comparison with the Kalman filter and EKF, the UIO performs equally good or even better at detecting faults. The ratio between faulty residual and non-faulty residual is very high, so even small faults are easily detectable using the UIO. Another advantage is that T_{goods} and $Q_{airload}$ are seen as disturbances, which means that any change in type, amount and heat parameters for the goods will not affect the model. Likewise, it is not necessary to know when the display cases are during their day- or night cycle or room temperature, as it is part of the disturbance, and thereby filtered out.

For fault isolation, the UIO performs very well when trying to isolate parametric faults. Unfortunately the UIO cannot isolate sensor faults, as it is not possible to create a split sensor approach. If used in conjunction with another method that can isolate parametric faults, the UIO might be a good basis for fault detection and isolation.

4.5 Parametric Estimation Method

By using a parametric estimation method, it is possible to recursively verify that the variables in the linear model (ie. the A and B matrix) are correct. By using logged input to, and output from the plant, one of several parametric estimation methods can be used. Unfortunately many of the methods require a SISO (Single Input, Single Output) system, or at best a MISO (Multiple Input, Single Output) and are based on transfer functions. Based on this, it has been decided to use an offline parameter estimation method recursively, by which it can be perceived as an online method. The method is the ”PEM” function from MATLAB, which has also been used for estimation in the modeling chapter.

Another restriction in the current implementation is that all four states must be used

for the estimation to work. By developing the parameter estimation, and perhaps using another method, there is a chance this might not be necessary in a final version. In case the last states are still required, they can be estimated using a simple estimator or one of the methods developed in this report. With this said, the effect of using such an estimation, especially when a fault occurs, should be researched before any implementation.

4.5.1 PEM Theory

The PEM function needs at least two arguments when called; ”DATA” and ”Mi”. The ”DATA” argument must be the data for which the estimation is to be made - for example as an iddata object, and ”Mi” is the model structure. By adjusting the free parameters of the model, the PEM function tries to minimize a cost-function which is calculated recursively. The larger the difference between the measurements and the estimation using the model, the higher the cost will be.

An ”idss” object (state space model) is created in MATLAB, with the established model from Chapter 3 and then the PEM function is called the PEM function with a set of measured data. The model will need the matrices A, B, C, D and K and the vector X0 plus a predefined structure of the matrices. The initial A, B, C and D matrices are as mentioned given in the modeling chapter. Theoretically both the continous and discrete versions can be calculated, but the continous version has been chosen. The K-matrix is a disturbance matrix, but as $Q_{airload}$ is modeled as an input, the K-matrix will consist of only zeros. The vector X0 is the x-vector a single step before the first measurement, which can be estimated quite well by taking the first measurement used for estimation (as mentioned, all states are required for the current setup).

The structure of the model to be estimated will determine which parameters are free for the PEM function to tweak. The more free parameters, the higher are the chances that faults will ascribed to multiple parameters - some of which may have nothing to do with the fault. This is especially evident when some states are not sufficiently excited, and where model deviation exist (for example non-linearities described in a linear model). Based on this, it is wise to lock down as much of the model as possible. The D and K matrix should not change and can be locked to their initial values. The B and C-matrix could theoretically be allowed to change, but as the faults described in section 4.1 have no relation to the parameters in the B-matrix, and most sensor faults will not simply affect the C-matrix, both matrices will be locked as well.

The final element is the A-matrix, where it has been decided that all elements that are initially zero should be locked, as the structure of the matrix should not change. Regarding the remaining elements, there are two options; they can all be left to change during the parameter estimation, or all elements except the four central values can be locked. The motivation for leaving the four central elements free, and only them, is to freeze everything not related to the parametric fault which is saught for. The advantage is that when the parameters start to drift, the estimation process will not ascribe it to some other elements, but the disadvantage is that the PEM function will assume all deviations can be fixed by correcting the four central elements, which might make them very unstable. For the tests in this report, all elements (except zeros) of the A-matrix has been left free.

4.5.2 Implementation

Based on restrictions in time at the end of the project period, the parameter estimation has not been implemented in Simulink, but only as an M-file in MATLAB. The file can use "true" measurements where only the two measured states are available or it can use all four state measurements (ie. by using an estimator). The mfile, which is called "onlineparamest", takes the input arguments described in table 4.1, and is included in the CD [CD/Data/M-Files] and illustrated in Appendix D, with some used sub-functions.

Argument	Description
ydata	An array of measurements from the plant, either 2xN or 4xN (where N is the number of samples) depending on how many measurements are measured.
udata	An array of inputs to the plant, 3xN (where N is the number of samples).
ts	Sample-time for the above mentioned arrays of data.
RO	Stands for Reduced Order. This element determines if only T_{air} and T_{wall} are used as plant outputs (RO=1) or if all four states are available as measurements (RO=0).
Window	The amount of prior samples which should be used for each estimation.
Wait	The amount of samples to wait before the parametric estimation is run again.
Stop	As processing large arrays of data can take several hours, there is included a "Stop" parameter which makes the estimation stop when the process reaches that point.

Table 4.1: Parameter Estimation Implementation

When using the reduced order version (RO=1) the results become quite unpredictable. To solve this, it is (as mentioned) necessary to have data from all four states available. As this is obviously not possible for normal situations, the two remaining states (T_{goods} and M_{refrig}) could theoretically be found by an estimator. Unfortunately there has been no time to develop this, so the results obtained in this report are based on the "measurements" of T_{air} and T_{wall}, while the states T_{goods} and M_refrig are taken directly from Dymola. Based on this, the results here might very well be somewhat better than for a real scenario, and should therefore only be seen as a test of the concept.

4.5.3 Fault Detection

The parameter estimation technique using the PEM function has a high computational load, and is therefore not favourable to use as a detection method. Based on this, it has been decided not to develop a detection method using this technique. With this said, parameter estimation using a method with a lower computational load could yield good results.

4.5.4 Fault Isolation

As the dynamics and the magnitude of the sensor faults always will be unknown, it can be quite hard to isolate a sensor fault using the parameter estimation. Additive faults like offset and to some degree the drift fault should theoretically be possible to isolate,

but this would require that the elements of the C matrix were free. As mentioned earlier, many free parameters will make the estimated parameters more unstable, which some initial tests have verified. Thereby the estimation is not robust enough to actually trust. Based on this, is has been decided not to use the parameter estimation technique for detecting sensor faults.

Detection of parametric faults on the other hand, is where parameter estimation techniques are brilliant. Not only will the parametric estimation be able to isolate where the fault has occured, but it can also be used for establishing the magnitude of the problems, and thereby be used for controller reconfiguration. By using the established method, the estimated parameters from a simulation with a parametric fault can be seen in Figure 4.40. The function call used, was "onlineparamest(ydata,udata,1,0,1800,10)" which calculates the parameters each 10 seconds, using 1800 samples from ydata and udata.

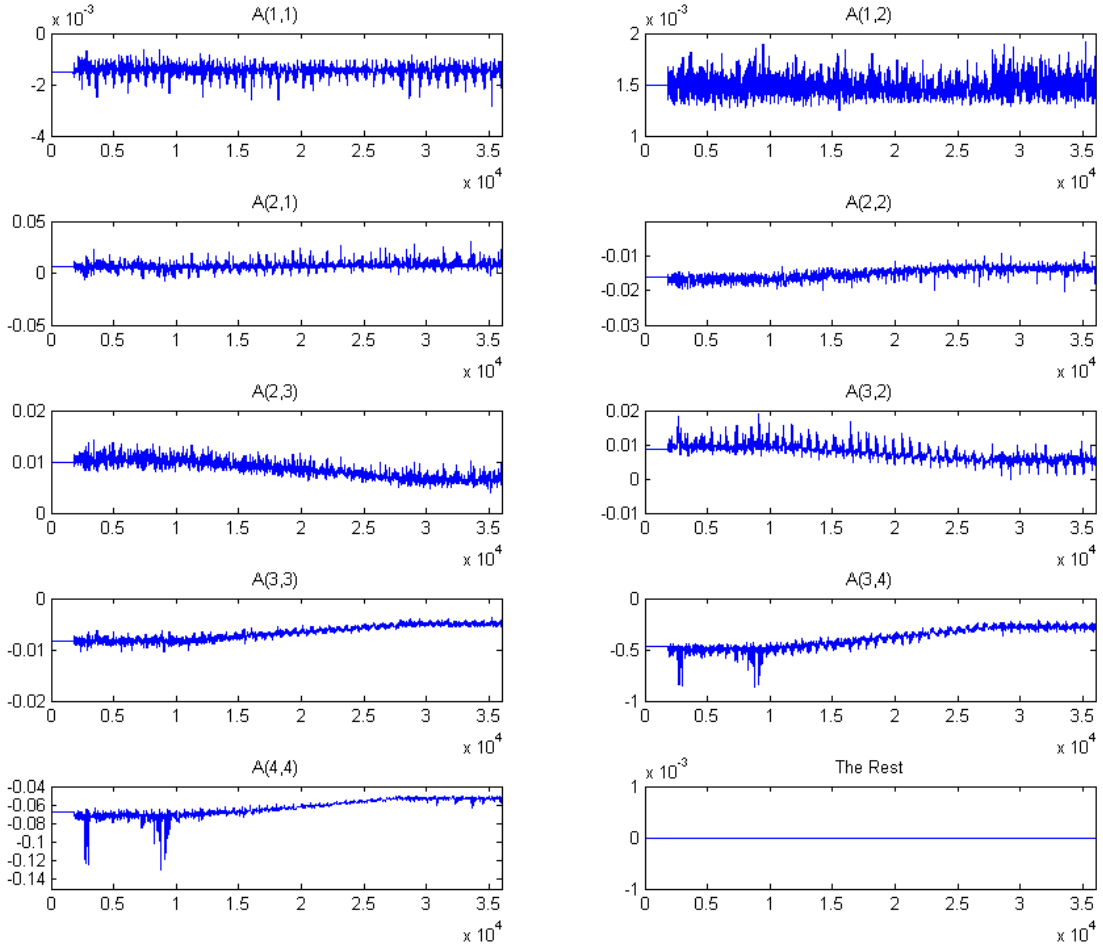


Figure 4.40: Parameters of A-matrix in parametric fault scenario

As can be seen, the changes in the parameters are clearly visible, even though some parameters not directly related to the changing parameter also change. In other words, if more than one parameter was allowed to change, it could easily be hard to determine which parameter was faulty.

Regarding sensor faults, there is no reason for testing the method. As mentioned, the two un-measurable states are taken directly from Dymola, so if a sensor fault is introduced, the remaining three states will still be correct (with or without noise) - which would not be the case if an estimator had been used. Furthermore, the development of the Parametric estimation method has been done in the very last of the project period, and there has not been enough time to evolve the method to one which is usefull. On these grounds, it has been decided not to go further with this approach.

4.5.5 Complete FDI System

As the parameter estimation technique used in this project is not developed enough, there is no reason for creating a Complete FDI System. Based on the results it would seem that a better estimation technique could give good results, especially for isolation purposes.

4.5.6 Conclusion

Parameter estimation used as fault detection is in general a very interesting appoach, not least because it not only allows detection and isolation, but also more detailed knowledge about the faults and their magnitude. This knowledge can be used to fit control efforts to the faulty system and perhaps to detect faults before they become a real problem.

The method used here is by no means developed enough to give any good results, and is thereby no success. On the other hand, the result was no complete failure, and has indicated that a good parameter estimation technique could be valuable for use with the refrigeration system.

4.6 Comparison of Above Methods

The different methods that have been developed each have their own advantages and disadvantages. To make it easier to compare the different methods, this section can give an idea of how good the methods are at detecting faults in comparison with each other and how much computational power they will require. Some of the methods are clearly inappropriate for some tasks while other properties can be harder to compare. The isolation times have not been compared, as they depend on many factors, and most of the methods have problems distinguishing between parametric and sensor faults.

4.6.1 Detection And Isolation

The main criteria for a good FDI method, is obviously the ability to detect and isolate faults. By comparing which method can be used for what, and how long it takes the different approaches to detect a fault, it is easier to choose the best method for further use.

Detection ability and time

For the Kalman Filter, Extended Kalman Filter and the Unknown Input Observer, it has been verified that detection is possible for all the faults mentioned in Section 4.1. The Parametric Estimation technique on the other hand, has not been testet for detection as it requires further development to be of any real use.

Table 4.2 illustrates the detection time for all methods and faults, presented in seconds. The times are based on the detection borders described in their respective sections, and all faults are introduced identically for each scenario, conforming to the following:

- Sensor Faults, all starting at time t=9000.
  - Drift: +0.001 Degree per sample.
  - Offset: +2.5 Degrees each sample.
  - Freeze: Ignore measurement, output same signal.
  - Hard-Over: Ignore measurement, output +25 Degrees.
- $UA_{air \rightarrow wall}$ Drift
  - Time t=0→6000: $UA_{air \rightarrow wall} = 500$
  - Time t=6000→16000: $UA_{air \rightarrow wall} = 500 - 0.025(t - 6000)$
  - Time t=16000→18000: $UA_{air \rightarrow wall} = 250$

As the parameter estimation function has not been tested for detection, no times will be given for PEM.

Fault\Method	KF	EKF	UIO	PEM
T_{air} Drift	2277	474	1408	N/A
T_{air} Offset	8	11	219	N/A
T_{air} Freeze	274	172	245	N/A
T_{air} Hard-Over	1	1	13	N/A
T_{wall} Drift	3050	1079	1414	N/A
T_{wall} Offset	101	10	2	N/A
T_{wall} Freeze	128	153	110	N/A
T_{wall} Hard-Over	1	1	1	N/A
$UA_{air \rightarrow wall}$ Drift	4474	569	567	N/A

Table 4.2: Detection Time (in seconds) Comparison

As can be seen, all tested methods can detect all tested faults. The detection time ranges from a single second up to more than an hour (4474 seconds). Based on the times, it can still be a little hard to select any specific method as being noticably better than the rest, but to give a better comparison base, the combined detection times can be calculated. The results can be seen in Table 4.3.

It is evident that the KF is no way near as effective as the EKF and UIO. The EKF has the fastest combined time but the UIO is also quite good. By including a CUSUM for

KF	EKF	UIO
10314	2470	3979

Table 4.3: Combined Detection Time (in seconds)

the UIO as well, there is a chance the detection times would be reduced somewhat, to make the UIO more competitive in comparison with the EKF.

Isolation ability

As mentioned, the time it takes to isolate an error cannot be compared reliably as the main part cannot distinguish between sensor- and parametric fault. Table 4.4 thereby only holds an indication of which algorithm can be used for which type of isolation. For a true comparison, the methods will need to be combined in some way to rule out other faults.

Fault\Method	KF	EKF	UIO	PEM
T_{air} Drift	-	$+^1$	-	-
T_{air} Offset	-	$+^1$	-	-
T_{air} Freeze	$+^1$	$+^1$	-	-
T_{air} Hard-Over	$+^1$	$+^1$	-	-
T_{wall} Drift	-	$+^1$	-	-
T_{wall} Offset	-	$+^1$	-	-
T_{wall} Freeze	$+^1$	$+^1$	-	-
T_{wall} Hard-Over	$+^1$	$+^1$	-	-
$UA_{air \rightarrow wall}$ Drift	$+^2$	$+^2$	+	+

Table 4.4: Isolation Ability Comparison

In Table 4.4,"+" indicates that the fault can be isolated using that method, while "-" indicates the opposite. The superscripts refer to:

1. A parametric fault might be identified as a sensor fault. Therefore, parametric faults must be ruled out before the method is used.
2. A sensor fault might be identified as a parametric fault. Therefore, sensor faults must be ruled out before the method is used.

Generally, all the methods can detect and isolate a parametric fault, but the only method which is able to isolate sensor faults reliably for all scenarios is the EKF. With this in mind, if a multiple-method approach is developed, the EKF is needed for isolation of sensor faults, while all methods can be used for parametric faults.

4.6.2 Night Time and Load

Two other important ways to compare the results are, the ability to adapt to night-operation, and the load of each process. These two are tested for in the following two

subsections.

Night Time Operation

In the introduction, it was specified that the FDI system could only be seen as successful if is could adapt to both night and day time operation, and any change between the two. As all tests to this point has been based on day-time operating conditions, a few different night-time scenarios have been tested for the FDI methods.

1. Simple night time test, 5 hours (18000 samples), $Q_{airload} = 1500$ and known.

   - A: Nominal run without faults.
   - B: T_{wall} Sensor Offset fault (+2.5 C) at time t=9000

2. Full 24 hour (86400 samples) test run with night/day change.

   - A: Store open 9.00 (t=32400) - 20.00 (t=72000), $Q_{airload}$ known.
   - B: Same as above, but faulty timer indicates store open 10.00 - 21.00.

Scenario 2B is tested by giving the models that need $Q_{airload}$ the signal delayed by an hour. In other words, when the store opens at 9.00, the display case still thinks the display case is covered, and at 10.00 the model is told it has been uncovered. Plots showing how the systems handle the scenarios are illustrated in Appendix A, in Figures A.2 to A.13. A success-chart of the results can be seen in Table 4.5.

	KF		EKF		UIO	
	A	B	A	B	A	B
Scenario 1	+	+	+	+	$+^{1}$	$+^{1}$
Scenario 2	+	-	+	-	$+^{1,2}$	$+^{1,2}$

Table 4.5: Ability to operate in night-time

The "+" indicates a success, while "-" indicates a failure. For scenario 1A a success is if no fault is detected, while 1B is succesfull if the Sensor Offset fault is detected. For scenario 2A and 2B the FDI must not detect a fault when the system changes from day to night time and vice versa. The UIO has an advantage here, as $Q_{airload}$ is a disturbance while the other models require a correct estimate of $Q_{airload}$.

The superscripts in Table 4.5 represents:

1. New detection threshold required.
   The UIO model does not fit as well for night time operation as daytime. With this said, the magnitude of the signal is noticably larger for nominal operation during night-time, but the ratio between a faulty signal and a non-faulty signal is still good. The solution could be either fixing the model to fit better, or to use a higher detection threshold.

2. No difference in results.
   As the UIO does not have any elements describing $Q_{airload}$ (it is an unknown input), there is no difference between these two scenarios.

Based on this, it is evident that the UIO has a clear advantage if the system has no clear way of knowing if the display case is covered up. The UIO is thereby also more robust in scenarios where the airload can change more dynamically thoughout the day. Having said that, if the system has access to precise knowledge of whether it is covered or not, the results for both the KF and EKF are good as well.

Computational Load

Another very interesting element, is the computational load. As this can be quite hard to measure, all three fault detection algorithms (PEM not included as it is not used for detection) have been tested in the same scenario, namely a 10 hour simulation with all variables at their nominal values ($Q_{airload}$ fixed at 3000). The Simulink models are stripped down to contain only the necessary elements (Dymola and data conversion blocks) and a single detection algorithm for each method. The execution time is measured with the 'tic-toc' function in MATLAB and rounded, and the fastest time is set as index 1, as the specific time is not of interest (as it is based on the test-computers specific hard- and software setup). Based on the execution time, the load cannot be determined, but a feeling of the relation between the different methods can be established. As the results are not completely reliable (not real time and based on a Microsoft Windows XP PC) three tests are made for each scenario and an average is calculated. The results are given in Table 4.6.

	KF	EKF	UIO
Test 1	1.13	1.47	1
Test 2	1.2	1.47	1
Test 3	1.2	1.47	1
Average	1.18	1.47	1

Table 4.6: Execution Time Comparison With Dymola

These tests were made with the Dymola link active. To eliminate any effect by Dymola, a similar test has been run with a much larger amount of data and both detection and isolation systems active. It is a total of 10 consecutive simulations with a combined simulation time of 128 hours (2x24 hours, 8x10 hours) where all scenarios are simulated independantly. As before, the numbers are converted to an index and given in Table 4.7. The Dymola block is replaced with a block which reads logged data from the workspace. All used files can be found on the CD in [CD/Data/From Dymola] and [CD/Models/Simulink] including sub-folders.

KF	EKF	UIO
3.22	19.34	1

Table 4.7: Execution Time Comparison Without Dymola

As can be seen from the tables, there is a large difference in whether the Dymola block is included or not. The EKF now takes far longer than the two other methods, which is due to the fact that the model is non-linear and based on a large amount of additions and multiplications in order to give a prediction. Meanwhile, the KF and UIO are liniear, and are based on a state space equation where the most of the calculations have been made pre-simulation.

As a comparison it can be said that the 128 hour UIO non-Dymola test took approximately two times the time as the 10 hour UIO simulation with Dymola. As the Dymola-link is not going to be used in any implementation, there is no need to examine the effect of this delay. It should be noted, that the current implementation of the parameter estimation would take several hours to do the same simulation. Knowing how the methods compare, it is possible to create a complete FDI system combining the different methods.

4.7 Multiple Method FDI

As all the described methods have different problems with isolating one or more faults, a multiple-method approach can be developed instead of using a single method for detection and isolation. As was seen in Section 4.6, the Kalman Filter generally has too poor results in comparison with the remaining methods to be of any use. The Extended Kalman Filter and Unknown Input Observer on the other hand perform quite well and have their own distinct advantages. The parametric estimation has not been developed to a level where it is thought usefull for implementation, but is considered as it could theoretically be used as well.

The detection system should at all times be active, and should be quite robust against disturbances while sensitive to faults. To save computational power, there is no reason for having the isolation process active while no fault has been detected. The isolation system should obviously be able to locate where the fault has occured, but this can be handled in several ways. One approach is to have a single method (generally multiple model) that based on the measured signals can determine which fault is the reason for any given faulty behaviour. Another method is to have several parallel isolators that will only trigger a fault in case the fault fits to the specific fault which each isolator is built to detect. A third option, is a serial isolation routine, where faults are ruled out and then passed to the next isolator which then evaluates if the fault fits its own parameters.

The only developed method which is theoretically able to isolate both sensor and parametric faults is the MMAE (Multiple Model Adaptive Estimator) used for both the Kalman filter and the Extended Kalman Filter. Unfortunately this approach will require a substancial amount of models to describe faulty sensors as they can fail in several ways with different dynamics to follow. This would require a large amount of development, not to mention fitting to new systems and computational load, which all together makes the MMAE a bad choice. Using parallel isolation methods is not possible either, as all the different faults are visible in the output of the main part of the developed isolation approaches. This leaves the serial method, which is used, and described in Section 4.7.2.

4.7.1 Detecting Faults

The Kalman Filter, as mentioned, has too poor results to be of any use. Even though it can be used for detection, the combined detection time calculated in Section 4.6 indicate that it is far to slow. The parametric estimation could theoretically detect faults, but the computational load using the developed method is extremely high, and therefore a poor choice as a detection method.

Both the Extended Kalman Filter and the Unknown Input Observer generally have good results for fault detection, but the UIO has the upper hand for this task. Even though the EKF is generally faster at detecting faults, in the UIO both goods and airload are decoupled as disturbances. Changes in these parameters can thereby not trigger a fault detection, which makes the fault detection more robust to disturbances and other faults. Furthermore, the large difference between faulty and non-faulty signals (with only a normalization) makes it ideal for detection.

4.7.2 Isolating Faults

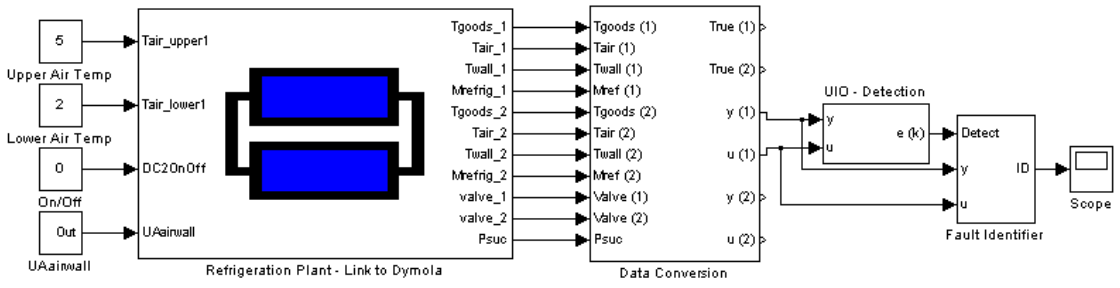
As mentioned, the isolation procedure is based on a serial approach where a fault, when it has been detected, is ruled out as being of each known kind iteratively. As there is two types of fault (parametric and sensor), one of them must initially be ruled out. The parametric fault can be ruled out by using some of the developed methods. The parameter estimation technique can be used to determine if any of the parameters have drifted away from the expected values. Relatively small changes could indicate that is a parametric fault, for example a change in $UA_{air \rightarrow wall}$ which could be brought on by ice or dirt on the evaporator. Large changes on the other hand, will most likely indicate some other fault, which can then be assumed to be a sensor fault.

Theoretically the split signal EKF could be used with the assumption that both fault indicators, and thereby their CUSUM, will indicate a fault if the parameter drifts. Then if both signals indicate a fault, it can be assumed that the fault is parametric. Unfortunately, this method is not feasible, as the assumption does not hold.

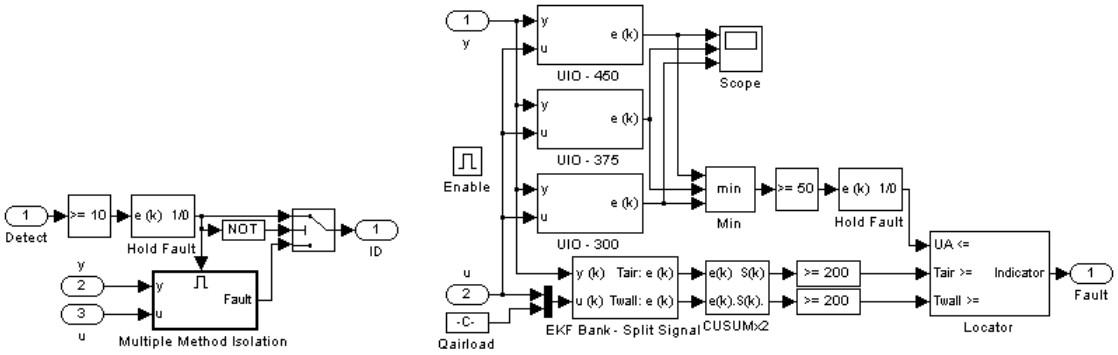
Some faults will be large enough to make one CUSUM grow, while the other does not change. This was also visible in Figure 4.32a, where $UA_{air \rightarrow wall}$ drops to 250 over some time and only one of the two signals is claiming a fault at the beginning.

What seems to be the best approach is to use the UIO again, but with multiple models based on different values of $UA_{air \rightarrow wall}$. By verifying that at least one of the models fits reasonably, it can be assumed that the fault is due to a parametric change. By having models for $UA_{air \rightarrow wall} = 450$, 375 and 300, the ice/dirt build-up fault should be possible to isolate, as the value should not drop below 300 in normal operation. This method was developed and tested in Section 4.4.4 and can be re-used here.

In case all the isolation-models for the UIO claim they do not fit, it can be assumed that the fault is not due to a drop in $UA_{air \rightarrow wall}$. As the remaining faults considdered for this project are sensor faults, the next step is to isolate which sensor is faulty. The only good method developed for this, is the EKF split signal, which can then be inserted after the multiple models.



(a) Kalman Method (Main Model)



(b) Fault Identifier (Submodel) (c) Fault Isolator (Sub-Submodel)

Figure 4.41: Multiple Method FDI Implementation

In case both CUSUM's from the EKF split signal are below or above the border at the same time, the data does not fit with the expected faults, and an "unknown" fault must be claimed.

4.7.3 Implementation

The detailed implementation of the multiple method approach can be found in Appendix C, while this section gives a short description. As the UIO setup used for detection is identical to the one described in Section 4.4.3, it will be described no further. The output signal from the UIO is sent to a subsystem called "Fault Identifier" which generates the ID signal. As the UIO has not changed, the detection border of 10 can also be re-used. The top level of the setup can be seen in Figure 4.41a.

The insides of the "Fault Identifier" block is illustrated in Figure 4.41b. The purpose of the "Hold Fault" subsystem is to keep the isolation system active in case of a non-stable fault indicator signal. For some faults, the fault indicator from the UIO can be very unstable, so by "holding" the isolation subsystem active for 7.5 minutes after the last fault detection, it is not continuously turned on and off for small or intermittent faults. The used time was selected by introducing the different faults, and verifying that the fault was detection was continuously enabled.

The output of the "Hold Fault" subsystem is connected to the "enable" input of the fault isolation subsystem which is illustrated in Figure 4.41c, which also takes the input and output from the plant ($y = [T_{air} \;\; T_{wall}]^T$ and $u = [V_p \;\; P_{suc}]^T$). Inside the subsystem is the fault isolation routine from the UIO chapter, and two EKF's in a single subsystem

using the split input method. As the EKF also uses $Q_{airload}$ as input, this is added to the input signal as a fixed value. $Q_{airload}$ can be estimated for night- and day-time operation and the change between the two values can be made automatically by knowing when the store closes.

The remaining blocks in the isolation subsystem create a logic which can be described by

Case:

1. Claim parametric fault if Multiple UIO has a fit for at least "wait" seconds.
   In other words, a fault is only claimed to be parametric if it has fitted underneath the border for "wait" consecutive seconds. The reason for including a wait-timer is that some faults will for short periods of time generate signals that are close to the expected. If the wait-period was excluded the isolation procedure would be undecisive for this type of fault. The "wait" integer has as mentioned earlier been defined to 450 seconds (7,5 minutes) for testing purposes, which has yielded good results.

2. If **one** EKF claims to be faulty, claim a fault for that sensor.
   If the fault is not claimed as parametric and one (and only one) EKF model claims to be faulty, it can be assumed the specific sensor is faulty. In case of a faulty sensor, only one EKF will generate a fault while the other will fit.

3. If nothing else fits, claim unknown fault.
   If both sensors claim to be faulty, or none of them claim to be faulty while the multiple UIO does not fit, none of the known scenarios fit to the fault. Therefore an "unknown" fault should be claimed.

4.7.4 Testing the Method

By testing all faulty scenarios described in Section 4.1, it can be checked if the system can detect and isolate all the faults correctly. Furthermore, a few other faults have been tested on the system, to check how the system will respond. The signal illustrated in the figures below is the fault descision which is an integer corresponding to:

0. No fault found.
1. Dirt or Ice buildup on evaporator.
2. T_{air} Sensor is faulty.
3. T_{wall} Sensor is faulty.
4. An unknown fault has been discovered.

The first faults which are shown in Figure 4.43 are all four sensor faults for T_{air}. The faults are all introduced at time t=9000, and (at the latest) detected approximately 1000 seconds later. Isolation takes a little longer, and (for the slowest) takes around 3000 seconds.

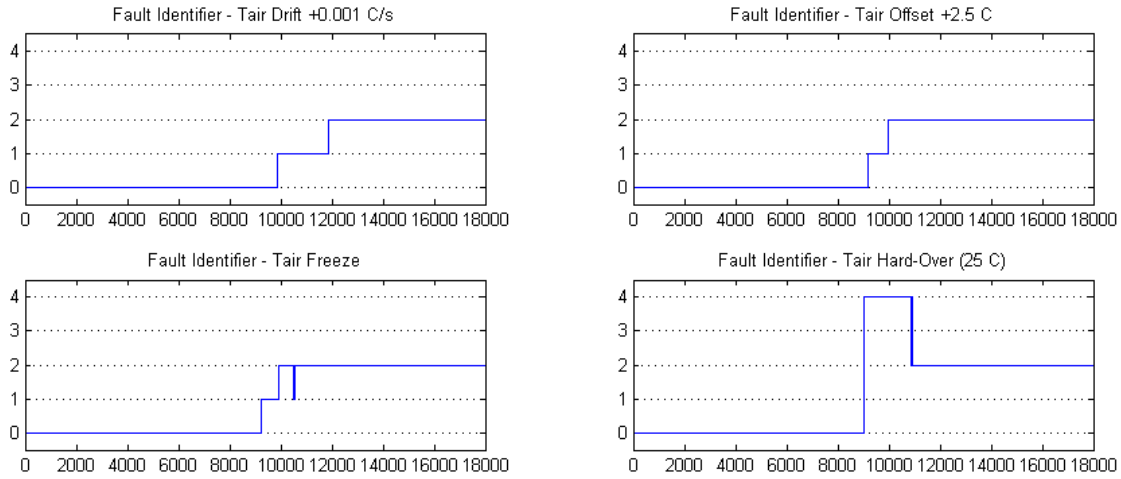


Figure 4.42: Multiple Methods - Fault identifier - T_{air} sensor faults

The next figure, Figure 4.43 is the sensor faults for T_{wall}. Again the faults are introduced at time t=9000 and all are detected within approximately 900 seconds, and the last is isolated after around 1800 seconds.

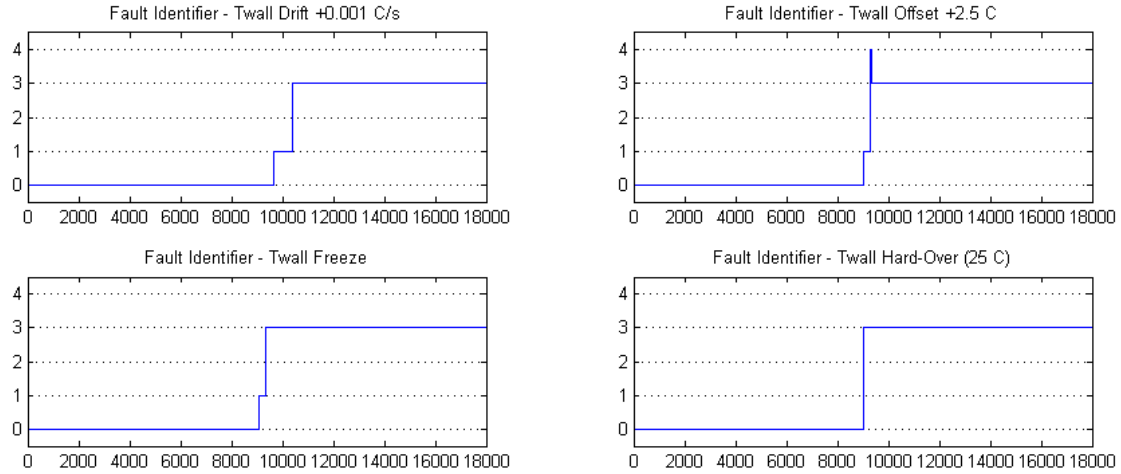


Figure 4.43: Multiple Methods - Fault identifier - T_{wall} sensor faults

Figure 4.44 shows four other scenarios.

- Top Left Graph.
  This graph shows nominal operation, which in other words is no fault. As can be seen no fault is detected during nominal operation.
- Top Right Graph.
  A drift in $UA_{air \rightarrow wall}$ from 500 to 250 starting at time t=6000 and ending at time t=16000. As can be seen, when the parameter goes below approximately 275, the fault is no longer claimed to be parametric, but instead unknown. This is not a bad decision, as a value of below 250-300 for $UA_{air \rightarrow wall}$ is most likely not due to ice or dirt buildup.

- Bottom Left Graph.
  Similar to the one mentioned above, but the value goes from 500 to 450 instead.
- Bottom Right Graph.
  A drift identical to the one in the top right graph, with a sensor offset fault (+2.5 C) added at time t=9000.

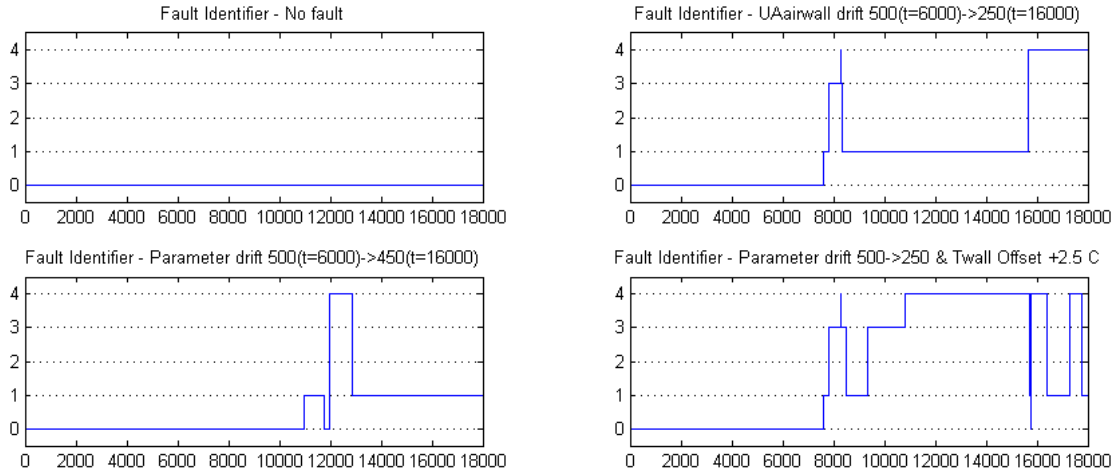


Figure 4.44: Multiple Methods - Fault identifier - parametric faults

As can be seen, the fault is detected for the three faulty scenarios but no fault is detected for the nominal situation. The fourth sub-graph with a combination of sensor and parametric fault makes the isolation unreliable, but the detection is still very useful.

4.7.5 Conclusion

The multiple method approach has very good results. By using multiple methods in different setups, it is possible to detect and isolate faults within few minutes of the faults occurance. The results seem to be quite robust as all faults are isolated after a few minutes, and continue to be so. Another thing noticable from the tests, are that the speed in the lack of a fault are substancially higher that after a fault has been detected. This indicates that the computational load is low for detection, and somewhat higher for isolation, which is very acceptable.

On the other hand, the approach requires a lot of setup, and assumes that faulty scenarios can be tested on the display case to give the faulty models. Based on this, the multiple method approach would probably be most useful as a part of a complete system, and not as an ”install and setup” solution for existing refrigeration systems.

5 Conclusion

The objective of the project was to create a fault detection and isolation system that could be implemented into new display cases, and perhaps added to existing systems. With the multiple method approach given in Section 4.7, the objective was fulfilled, but the system can become better. The main way to improve the performance would be to replace the multiple method UIO with a approach that was easier to implement. But there are also other ways to improve the final solution.

The basis of any good work within model-based fault detection schemes, is obviously the model of the plant itself. The model used as a basis for this project is generally very good, but has some issues specifically related to FDI methods. Because the model was originally created for controller optimization/creation in a plant which is assumed to be ideal, the potential for introducing faults to the plant was not considdered. The lack of detail makes it hard to impose realistic faults on the system, and thereby test how the plant and any developed FDI method will respond.

The three main methods developed in the project are very different in their results and requirements. Generally, the EKF and UIO outperform the KF with a large margin. The Kalman Filter and Extended Kalman Filter both need the $Q_{airload}$ value as an input and have the goods as part of their dynamics. Thereby they are not robust to unknown changes in the dynamics of the goods (for example a change in type or amount of goods in the display case), or to changes in the effect of the ambient air. On the other hand, by knowing the air temperature in the store (a simple temperature sensor outside the display case) and knowing when the display case is covered up (a simple contact where the cover is fitted) a good estimate of the $Q_{airload}$ value should be available at all times. Based on this, and the fact that the EKF generally performs quite well and is faster at detecting the faults than any of the other methods, the Extended Kalman Filter cannot be ruled out.

The Unknown Input Observer on the other hand, does not need any knowledge of $Q_{airload}$

to estimate the states - and does not get affected by changes in the goods. Even with this lack of information, the detection is generally still very fast, so if one method was to be selected as the best for FDI in a supermarket display case, based on this project, it would have to be the Unknown Input Observer. Having said that, the initial tests with the parameter estimation also provided some good results, and aided by its ability to draw out the parameters of the system, it could prove an important element in a FDI-implementation.

The developed multiple method FDI approach generally performs well for the faults that have been tested on it. Using the UIO for detection, it should be able to detect the vast majority of faults without generating false alerts. The main disturbances are disconnected while the important dynamics are kept intact, which should make the UIO detect all faults that not only affect T_{air} directly (as these will be ignored as a disturbance).

For implementation, the isolation system should be adjusted based on what kind of faults it must be able to isolate. By developing a true online parameter estimation, it should be able to take the place of the multiple UIO. The reason for preferring the parameter estimation over the UIO, is that the Unknown Input Observer method will require several models that are only obtainable if the faults can be testet on the plant. The parameter estimation on the other hand does not require any prior knowledge other than the structure of the model, and an initial guess. Furthermore, the parameter estimation can be used when the FDI system must be fitted to a new plant. By enabling the parameter estimation with an initial guess, the true parameters can be extracted and used for the EKF and UIO, which would make the fitting process to any given plant an easy task.

Appendix A
Additional Plots

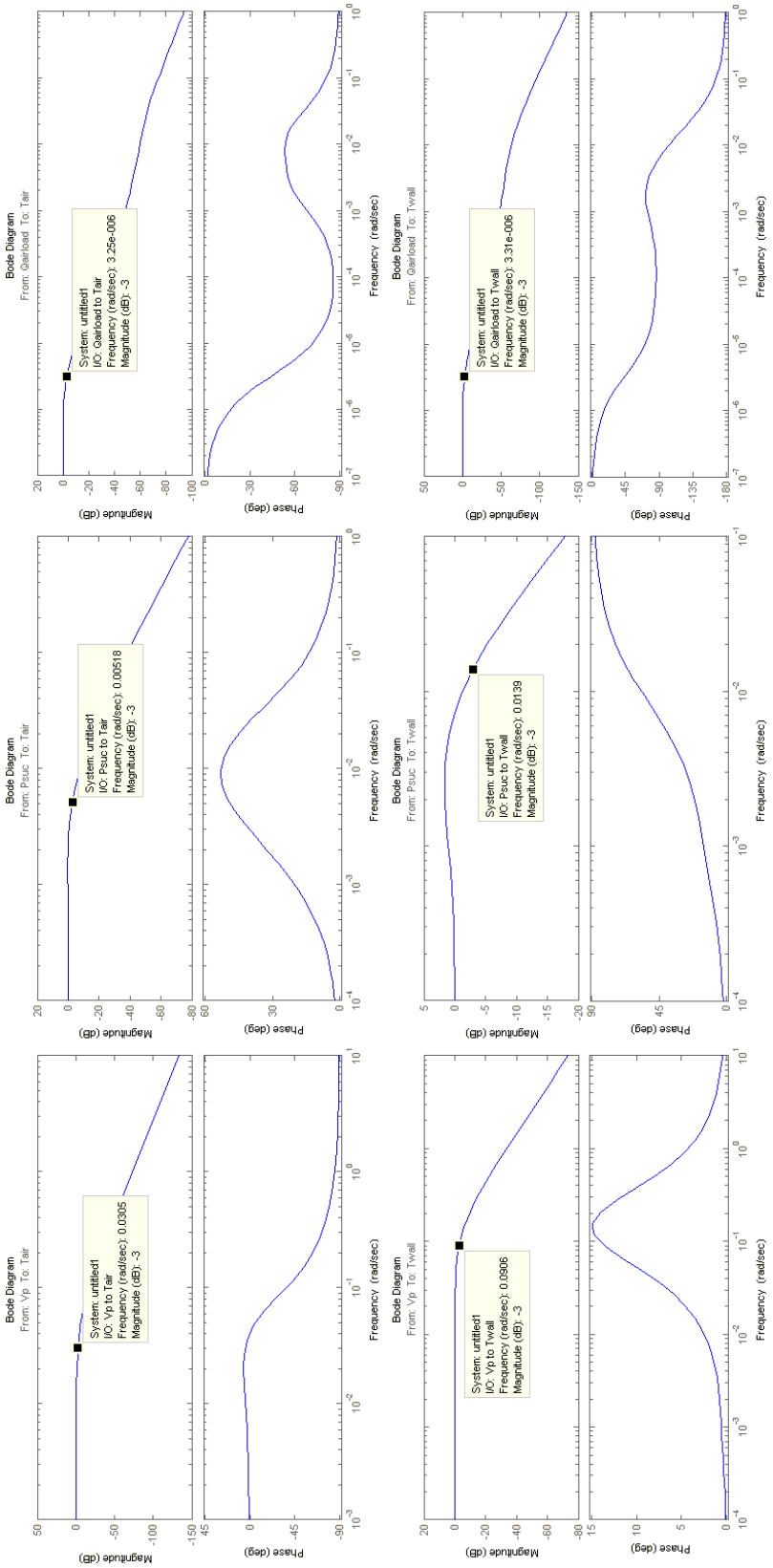


Figure A.1: Bode plot of the linear continous model

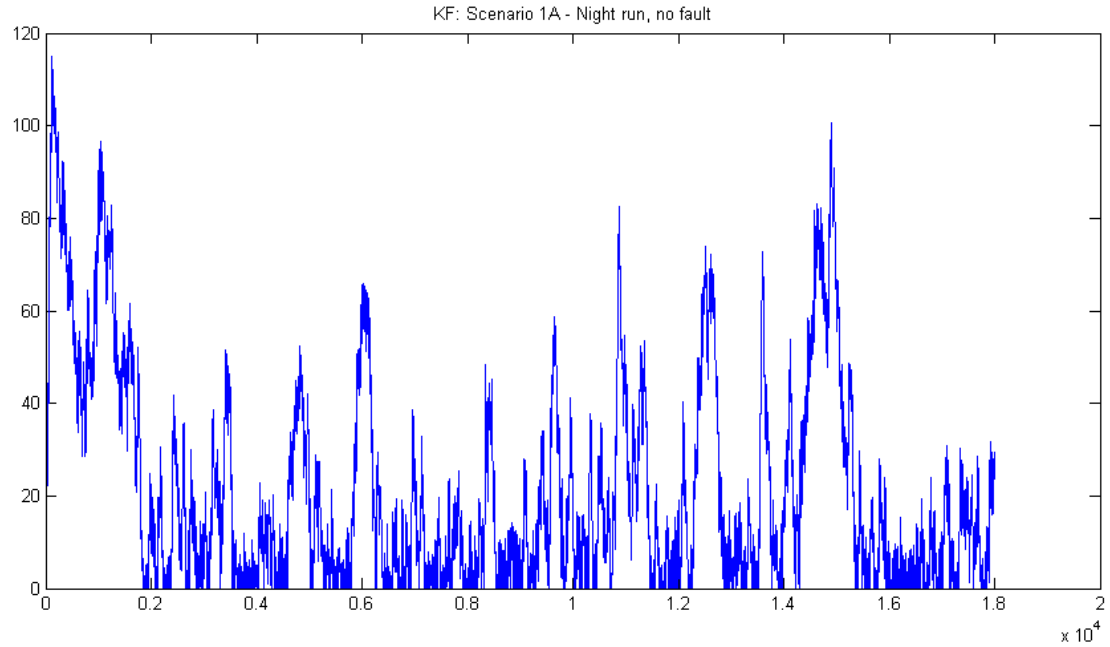


Figure A.2: Kalman Filter - Scenario 1A - Night operation, No fault

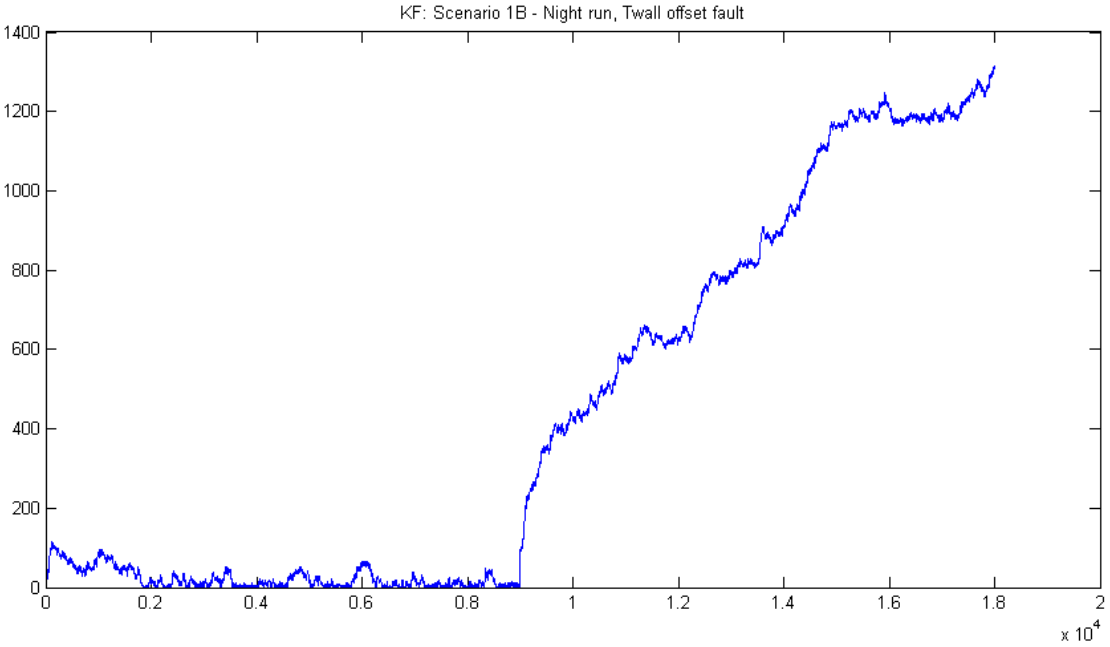


Figure A.3: Kalman Filter - Scenario 1B - Night operation, T_{wall} sensor offset

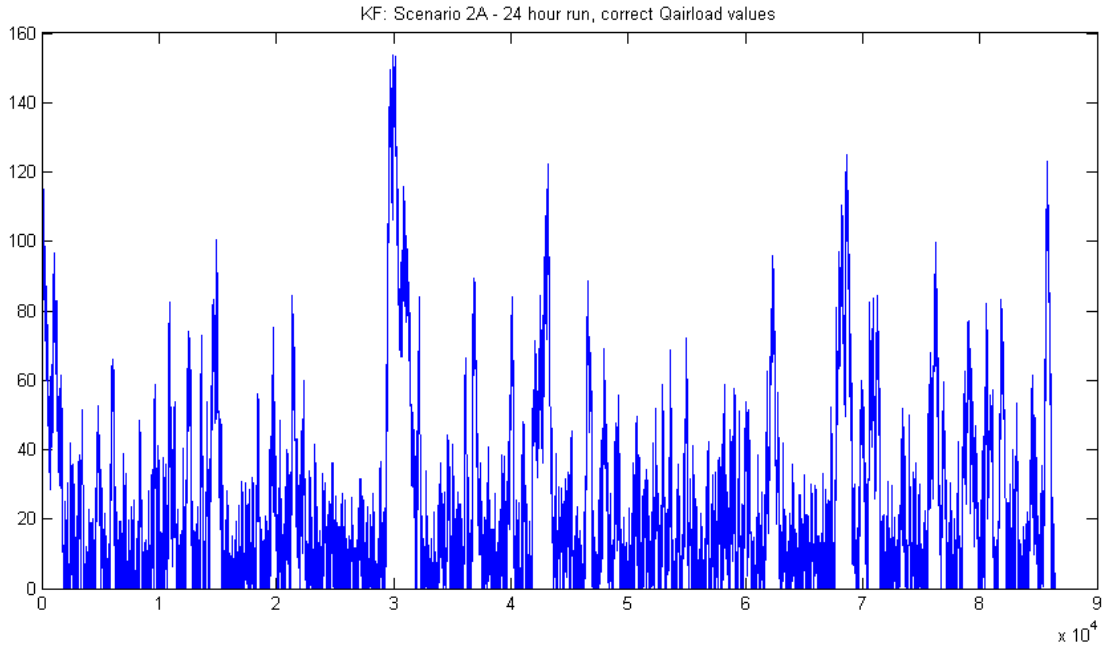


Figure A.4: Kalman Filter - Scenario 2A - 24 hour run, Correct $Q_{airload}$ values

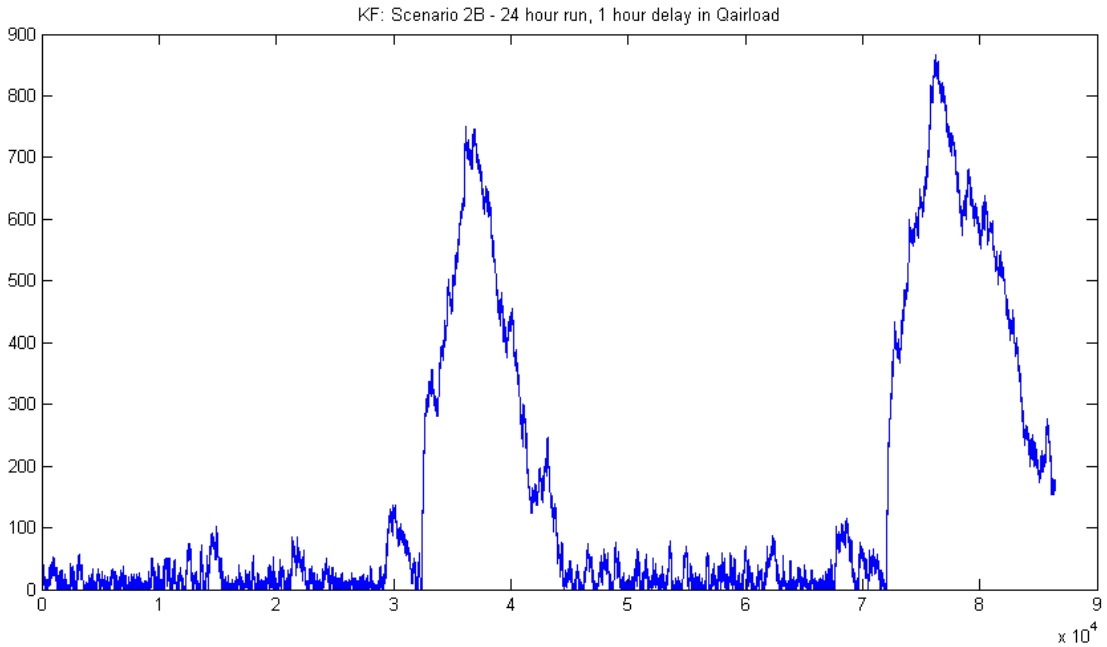


Figure A.5: Kalman Filter - Scenario 2B - 24 hour run, 1 hour delay in $Q_{airload}$

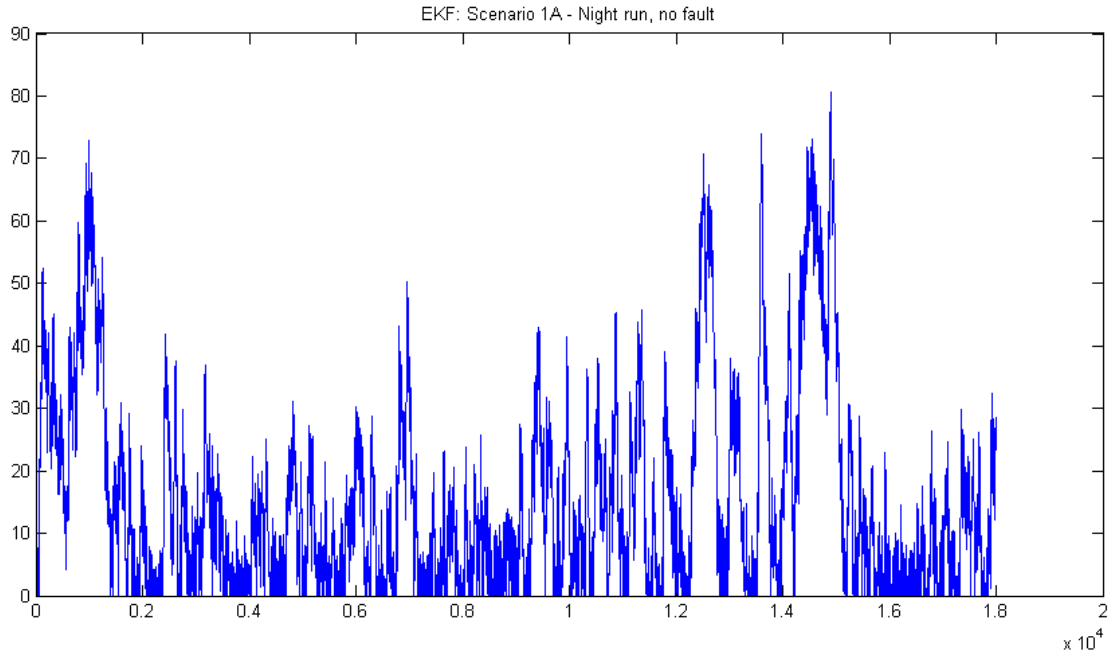


Figure A.6: Extended Kalman Filter - Scenario 1A - Night operation, No fault

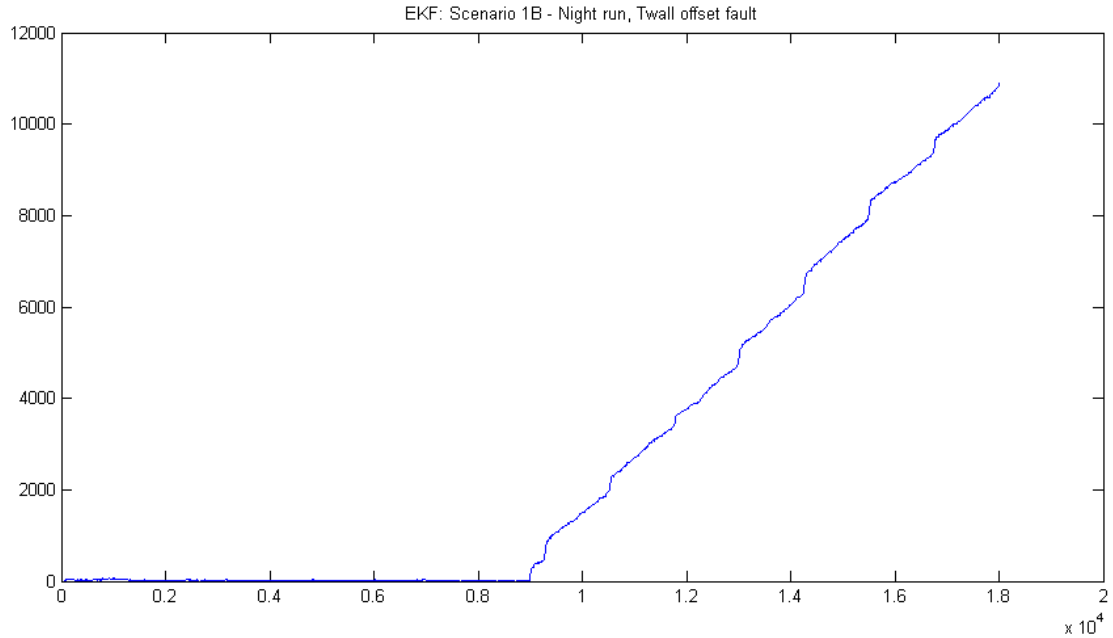


Figure A.7: Extended Kalman Filter - Scenario 1B - Night operation, T_{wall} sensor offset

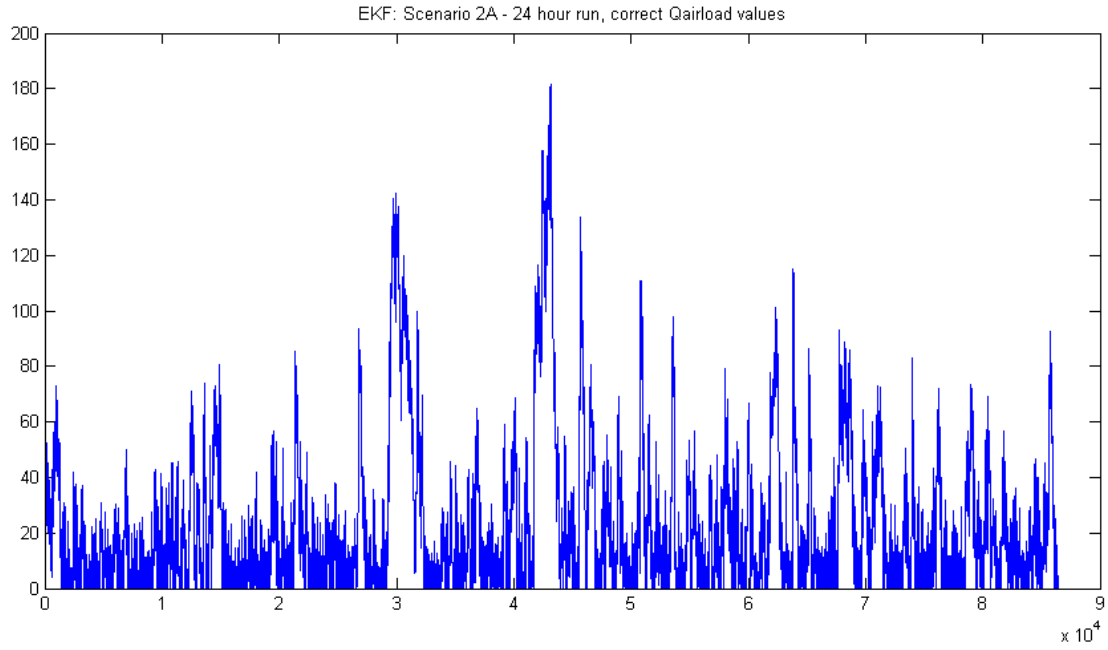


Figure A.8: Extended Kalman Filter - Scenario 2A - 24 hour run, Correct $Q_{airload}$ values

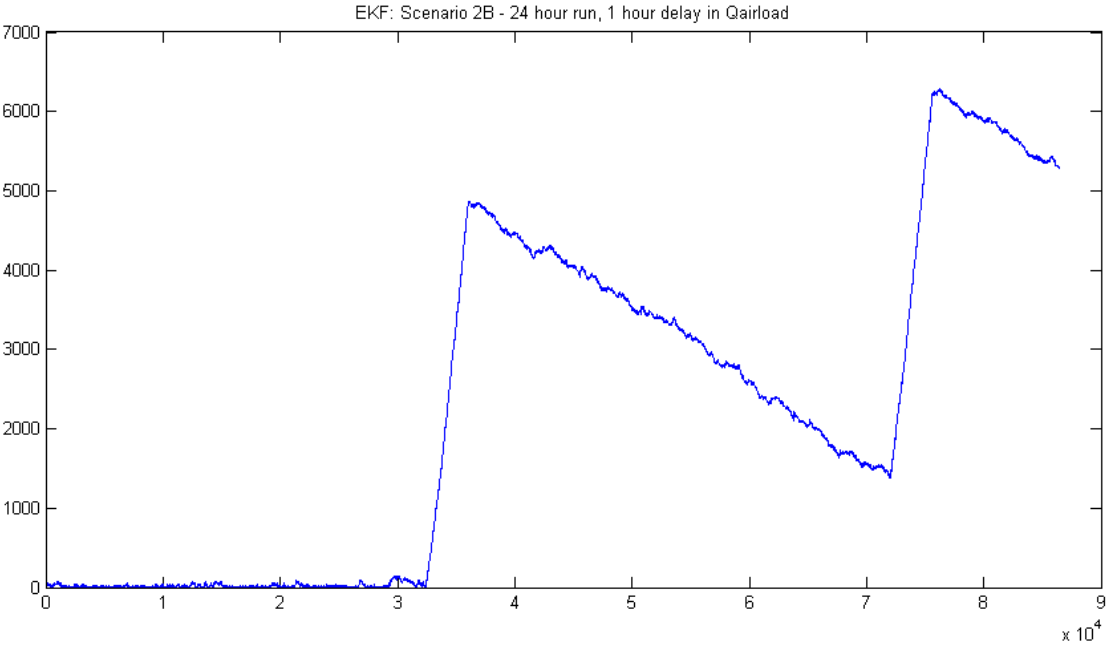


Figure A.9: Extended Kalman Filter - Scenario 2B - 24 hour run, 1 hour delay in $Q_{airload}$

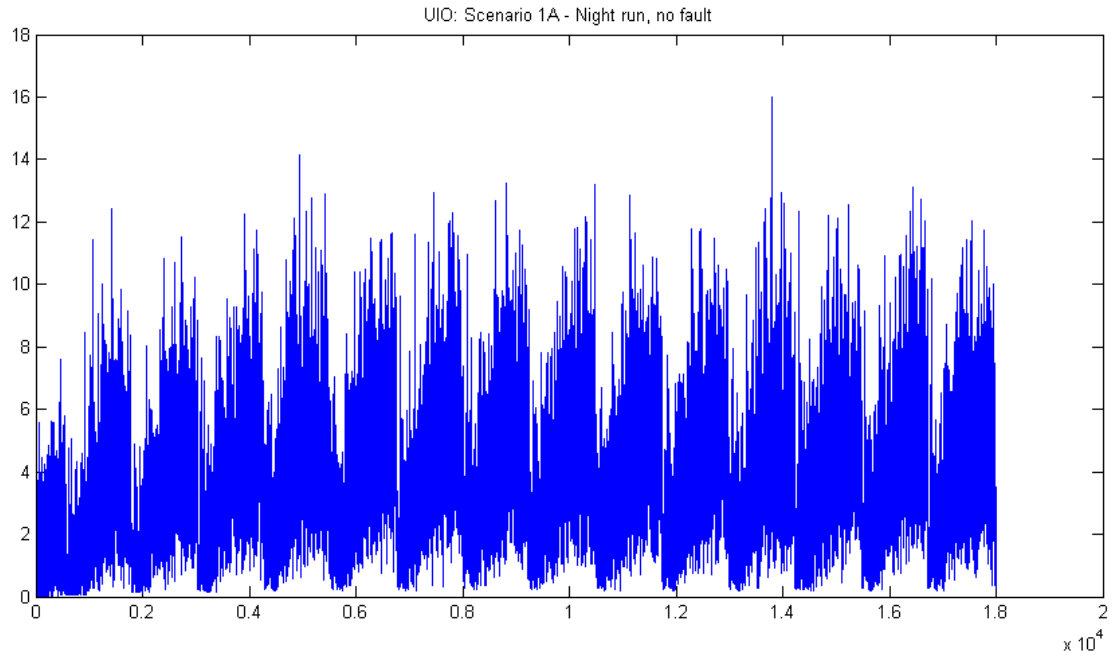


Figure A.10: Unknown Input Observer - Scenario 1A - Night operation, No fault

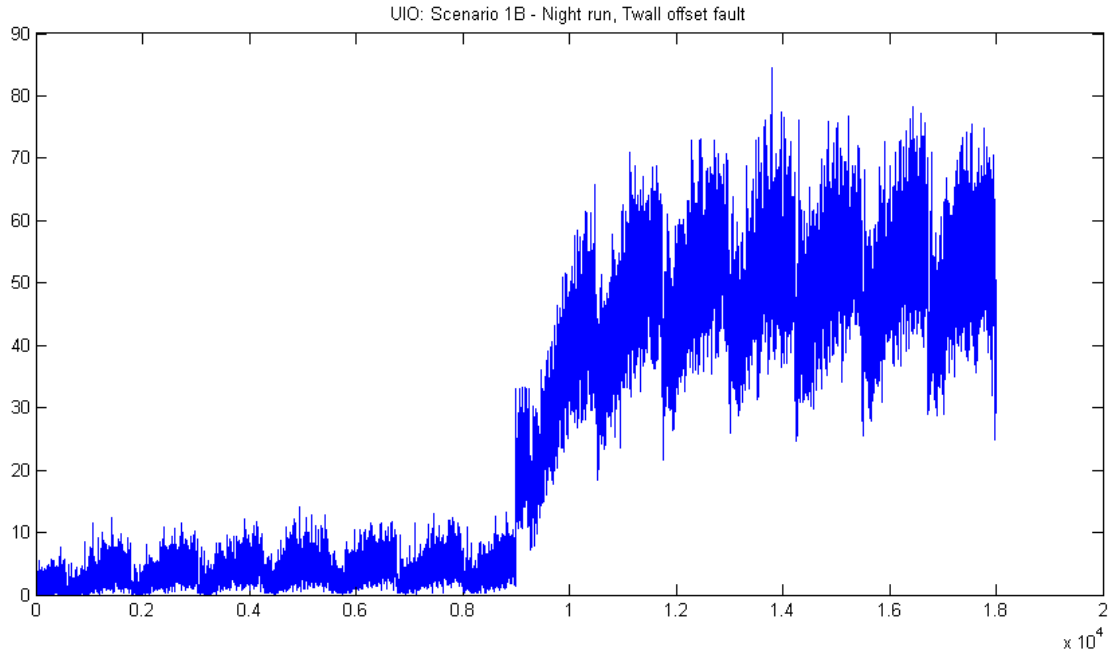


Figure A.11: Unknown Input Observer - Scenario 1B - Night operation, T_{wall} sensor offset

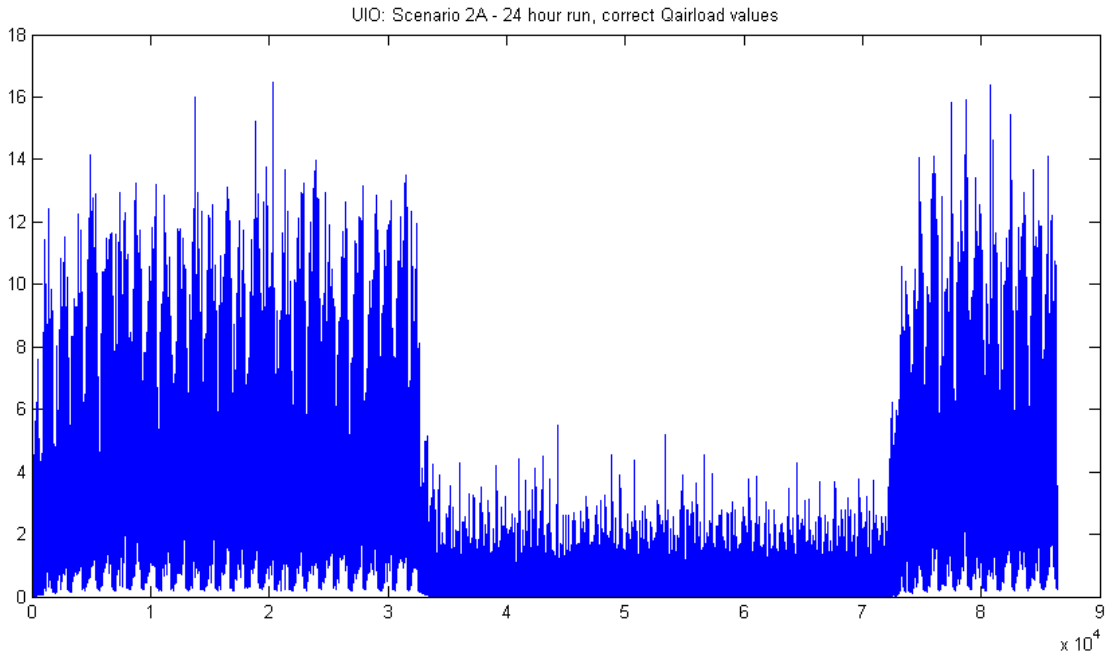


Figure A.12: Unknown Input Observer - Scenario 2A - 24 hour run, Correct $Q_{airload}$ values

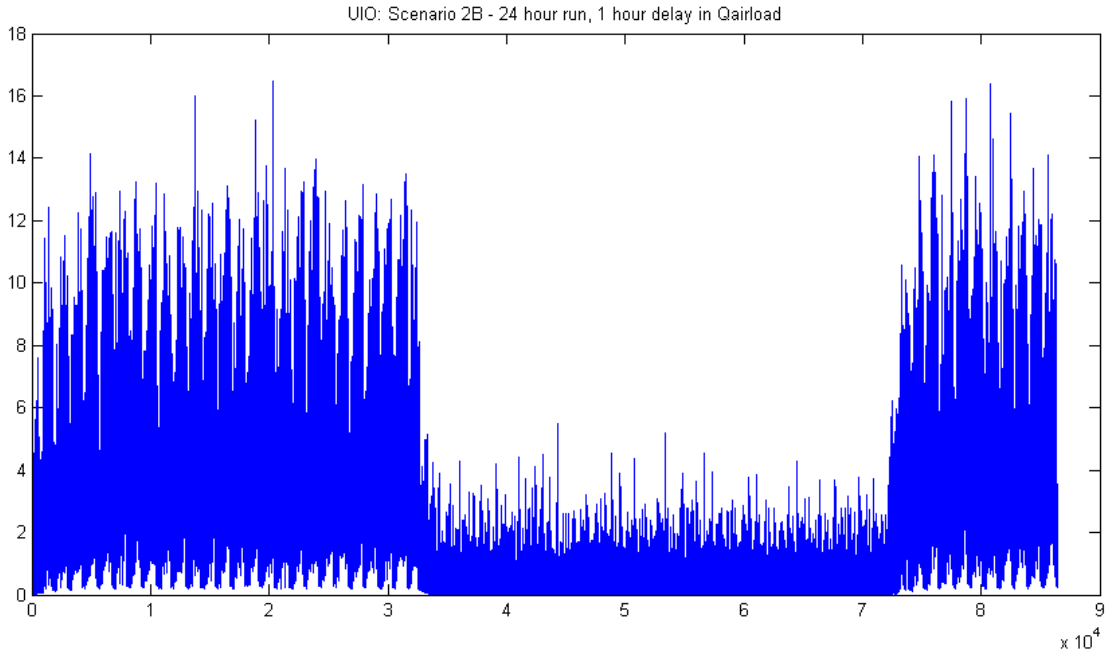


Figure A.13: Unknown Input Observer - Scenario 2B - 24 hour run, 1 hour delay in $Q_{airload}$

Appendix B
Dymola Documentation

In the original Dymola model, there has been made some small changes to adapt it for use in this project. The model "Supermarket_system_2d2c_matlab" from the file "Refrigeration_HYCON.mo" has been used as a basis. The changes are:

- General small changes - See section B.1.
- $UA_{air \rightarrow wall}$ changes to be an input - See section B.2.
- Compressor Controller - See section B.3.

B.1 General small changes

There are three small changes made to the Dymola model given from Danfoss

1. As all four states (T_{goods}, T_{air}, T_{wall}, M_{refrig}) are needed for some simulations in Simulink, all four states have been set as outputs from each display case.
2. It is assumed that all display cases will run with identical upper and lower limits, so the inputs to the two display cases are combined.
3. There is included an option to force the expansion valve of display case two open. This was included as the second display case changes the dynamics of the first display case before for compressor controller was implemented.

B.2 $UA_{air \rightarrow wall}$ as an input

To be able to impose the parametric fault on the Dymola model, it was necessary to be able to control the $UA_{air \rightarrow wall}$ parameter from Simulink. This was acheived by creating an extra input to the display cases, and then using the input in the place of the value

$UA_{air \rightarrow wall}$. It only required a small change in the source code, where the input 'fault' is used.

```
Qair_wall = UAair_wall*(T_air-T_wall); <- Old code
Qair_wall = fault*(T_air-T_wall); <- New code
```

B.3 Compressor Controller

In the Dymola implementation from Danfoss, there is a hysteresis controller built in for the Expansion Valve which is based on the air temperature of the display case. As it is important that the suction pressure lies within the range for which it is linearized, a simple compressor controller has been created. The controller has an upper pressure limit of 1.45 and a lower limit of 1.25. If the pressure goes over the top limit, the compressor is set to 100% (corresponding to two compressors running at full speed). If the pressure goes below the lower limit, the compressor is set to 0%. Both extremes (0 and 100%) are kept until the suction pressure is again right between the top and bottom limit (1.35 bar). The funtionality is in Dymola described by:

```
block CompHysteresis

  input Modelica.Blocks.Interfaces.RealInput Psuc;
  output Modelica.Blocks.Interfaces.RealOutput CompSpeed;

  Boolean high(start=false);
  Boolean low(start=false);

equation
  low = Psuc<1.25 or pre(low) and Psuc <= 1.35;
  high = Psuc>1.45 or pre(high) and Psuc >= 1.35;

  if high then
    CompSpeed = 100;
  elseif low then
    CompSpeed = 0;
  else
    CompSpeed = 50;
  end if;
```

Appendix C
Simulink Documentation

As there are a lot of blocks that are re-used in the implementation, the description of the blocks might seem somewhat unorganised. The used approach is to start describing from one end. When a block is used that has already been described earlier, the previous version is just referred to, and thereby no block is described more than once.

C.1 The Kalman Filter

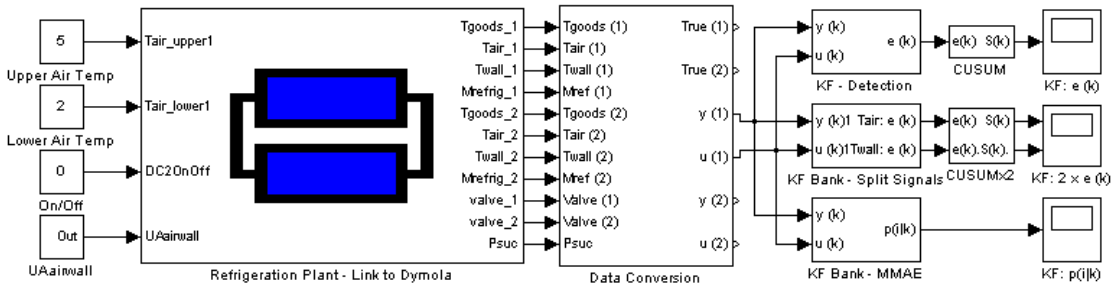


Figure C.1: KF: Main Window

Figure C.1 illustrates the top-model view of the Kalman implementation. Starting from top left to bottom right

- Three constants; 'Upper Air Temp', 'Lower Air Temp' and 'On/Off'
  These three constants determine the upper and lower bound of the display case hysteresis controller, and determine if the second display case should be enabled respectively.

- 'UAairwall' block
  This block generates the $UA_{air \rightarrow wall}$ value, and is described in Section C.1.1.

- Dymola Interface
  The link to the Dymola software is obtained using this block. The used model has been selected within, and compiled. No values have been changed, and therefore there is no further description of this block.

- Data Conversion Block
  As the data from the Dymola model needs to be structurized, this block is added, which generates the $y(k)$ and $u(k)$ signals, and gives the true states out for comparison if needed. The block is described in Section C.1.2.

- KF - Detection
  This block represents the detection system for the Kalman filter, and is followed by a scope where the result can be seen. A description of the block can be found in Section C.1.6.

- KF Bank - Split Signal
  Two parallel Kalman filters are inside this block, each getting only one of the two measured signals from the plant. A description of the block can be found in Section C.1.11.

- KF Bank - MMAE
  This block is the Multiple Model Adaptive Estimator bank. A description of this block can be found in Section C.1.12.

- CUSUM and CUSUMx2
  The final elements to be described are the CUSUM blocks. As they are identical (CUSUMx2 just has 2 CUSUM inside), so only one description is given, which can be found in Section C.1.18.

C.1.1 $UA_{air \rightarrow wall}$ Builder

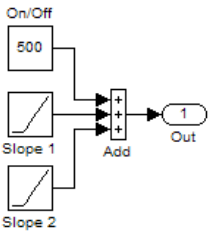


Figure C.2: $UA_{air \rightarrow wall}$ Builder

The purpose of the $UA_{air \rightarrow wall}$ Builder, is obviously to create the $UA_{air \rightarrow wall}$ signal for the Dymola model. The block consists of three elements. A fixed value which can be set to any desired value (is set to 500 in the figure, as this is the nominal value), and two slopes. The intention of having two, and not just one, is the option to 'counter' the addition by the first slope, which will give the option of stopping the change in the value.

C.1.2 Data Conversion Block

The Data conversion block is used to arrange the simulated data from the Dymola model, including the introduction of noise and faults. The four states related to each display case

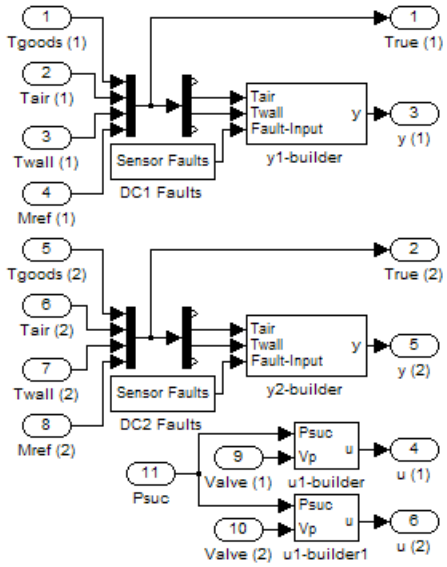


Figure C.3: Data Conversion Block

are combined into a mux. The resulting signal is an output from the block, holding the true states in case they are needed. T_{air} and T_{wall} are isolated and sent with the signal from the 'Sensor Faults'-block (described in Section C.1.3) into the 'yz-builder'. The 'yz-builder' (where z is a integer representing the display case number) block is described in Section C.1.4, while the 'uz-builder' block, which takes the two plant inputs, is described in Section C.1.5.

C.1.3 Sensor Faults

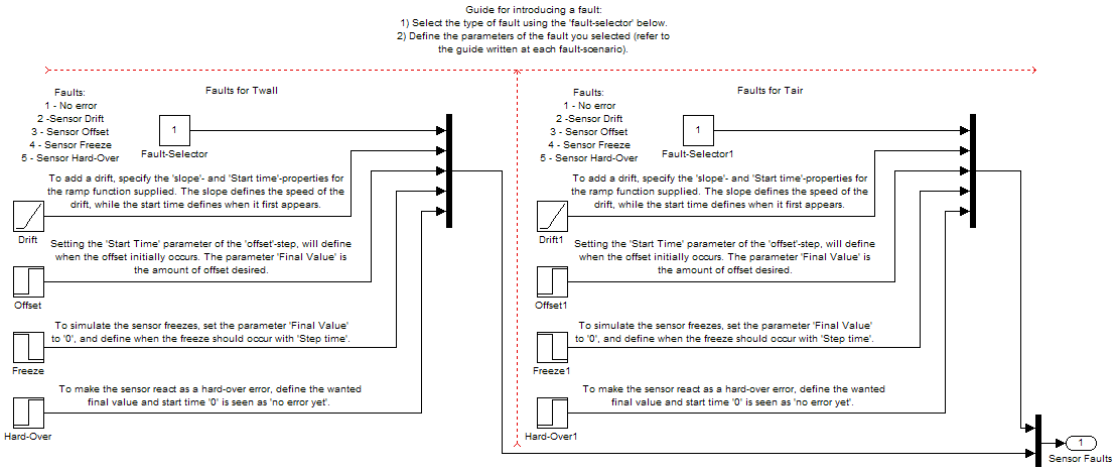


Figure C.4: Sensor Faults

The sensor faults block is used to select sensor faults, their starting time, and their dynamics if relevant. The desired fault is selected by defining the constant at the top and the dynamics in the block related to the fault. In this way, the next simulation will have a sensor fault introduced. The constants represent the following faults:

1. No fault is introduced.
2. A drift fault is introduced to the sensor. By altering the 'start time' and 'slope' parameters of the ramp-block, the behaviour of the fault can be defined.
3. Offset fault. Defining 'Step Time' and 'Final Value', the dynamics of the offset fault can be specified.
4. The 'Step Time' parameter can be adjusted to the time where a sensor freeze is desired.
5. The Hard-Over fault will make the sensor jump to the parameter 'Final Value' at time 'Step Time'.

The selected sensor fault is then sent on to the 'yz-builder' described in Section C.1.4.

C.1.4 yz-builder

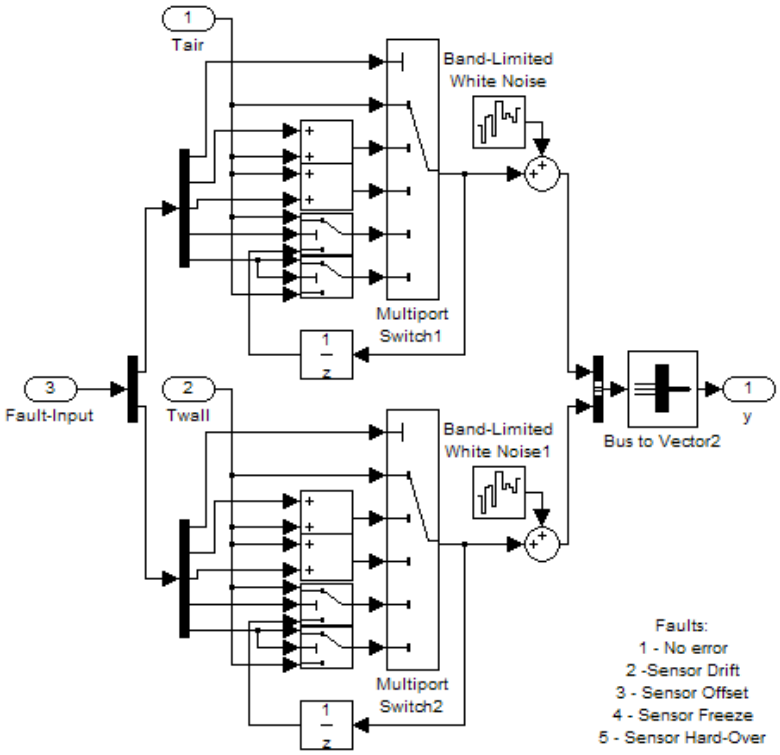


Figure C.5: yz-builder

The 'z' in the name 'yz-builder' defines what display case it is for. For instance, 'y1-builder' will generate the y-signal, that is the simulated measurements, from display case 1, including faults and noise. The sensor input is taken from the Dymola block and is introduced to noise, and based on the scenario selected in the sensor faults block, faults as well. The white noise (noise power=0.25) is added just before the signals are combined into the y-signal, and will therefore be present no matter what fault is introduced. The faults are introduced using the case-block, which for the first scenario is just the simulated

signal. For the two next scenarios (drift and offset), the fault is added to the simulated signal, while the freeze fault uses an unit delay block, so the signal will be the same at all times. The final fault is created by checking the value of the signal from the sensor faults block. As soon as it is no longer zero, the simulated value is ignored, and the fixed value is used as a measurement instead.

C.1.5 uz-builder

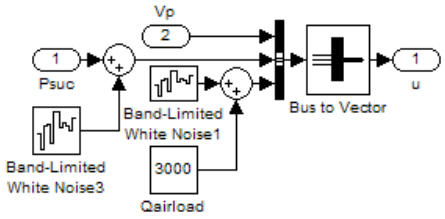


Figure C.6: uz-builder

The 'uz-builder' ('z' is again the display case number) is similar to the 'yz-builder', but instead of creating the measured outputs, it creates the inputs. There are three inputs to each display case, the suction pressure, the valve position and the airload. The valve position is a digital on/off (1/0) value, and is assumed noise-free (as a filter should be reasonably good at filtering such values) while the P_{suc} has noise introduced as well, with a power of 0.05. $Q_{airload}$ is actually not a real input, but a disturbance, which for the case of the Kalman Filter is introduced as an input with white noise added (noise power=250).

C.1.6 KF - Detection

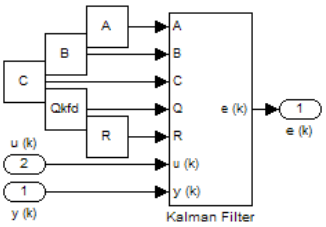


Figure C.7: KF - Detection

The Kalman filter is implemented both as detection and isolation. The detection part consists of several submodels, which are also described shortly in the report itself. The detection block is actually just a Kalman filter block with defined inputs. The A, B, C, R and Qkfd (kfd=kalman filter detection) matrices are defined in the MATLAB workspace, and therefore only needs to be referred to from Simulink. $y(k)$ and $u(k)$ are taken as input, while the fault indicator $e(k)$ is the only output. The Kalman filter block, is described in section C.1.7.

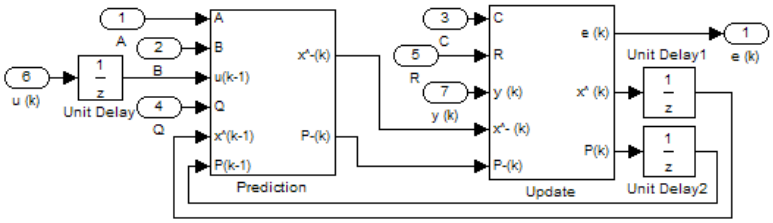


Figure C.8: Kalman Filter for Detection

C.1.7 Kalman Filter for Detection

In agreement with the theory for the Kalman Filter, the implementation has two sub-models containing the prediction (described in Section C.1.8) and update (described in Section C.1.9) procedures respectively. As the plant input used for calculation is $u(k-1)$, the signal is sent through a unit delay. The remaining elements should be self-explanatory.

C.1.8 Kalman Filter for Detection - Prediction

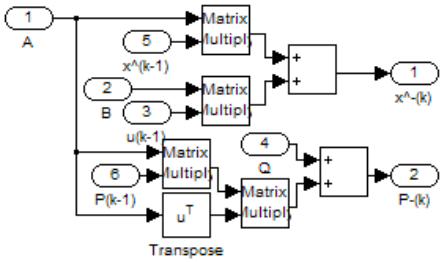


Figure C.9: Kalman Filter for Detection - Prediction

In agreement with the theory relevant to the Kalman filter, an *a priori* estimate is made of the states based on the previous *a posteriori* estimate. Afterwards, the predicted estimation covariance is calculated, which can also be verified against the Kalman theory.

C.1.9 Kalman Filter for Detection - Update

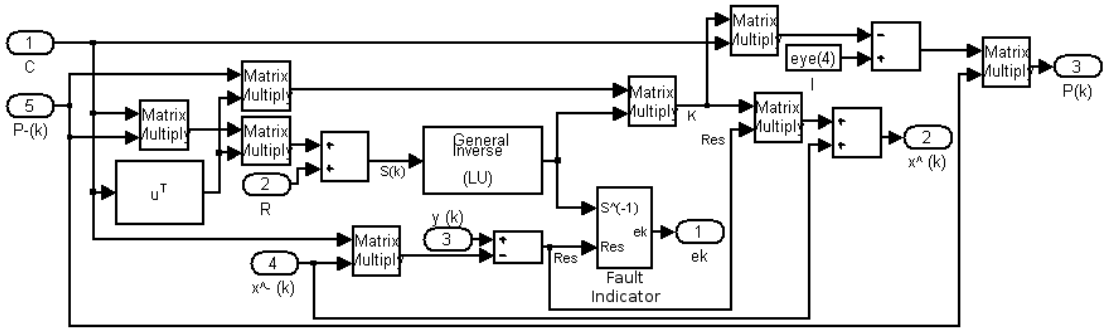


Figure C.10: Kalman Filter for Detection - Update

The calculations here are, again, based on the equations relevant for the Kalman filter. As this is the case, they will not be described, as this has already been done in the FDI chapter. A simple visual inspection is sufficient to verify that the calculations are correct. The fault indicator submodel is described in Section C.1.10.

C.1.10 Kalman Filter for Detection - Fault Indicator

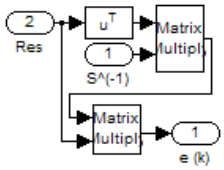


Figure C.11: Kalman Filter for Detection - Fault Indicator

The fault indicator function is based on the Kalman theory, and is not furhter described here.

C.1.11 KF Bank - Split Signal

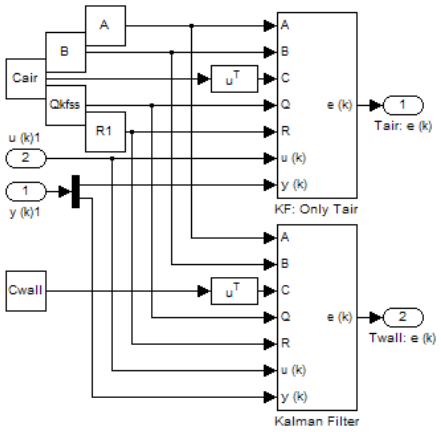


Figure C.12: KF Bank - Split Signal

The split signal bank of kalman filters, uses two of the kalman filters described in Section C.1.7, including all submodels. The only difference is the inputs to the filters. For the split signal setup, each filter has its own C-matrix, while R1 and Qkfss (kfss=kalman filter split signal) is used instead of R and Qkfd. The reason for including the transpose blocks for Cair and Cwall, is that Simulink assumed the values to be a column-vector and not a row-vector.

C.1.12 KF Bank - MMAE

The MMAE block contains four Kalman filters that are almost identical to the ones used for detection and split signal, with the differences described in section C.1.13 and a probability block described in Section C.1.17. The C, Q and R matrices are identical

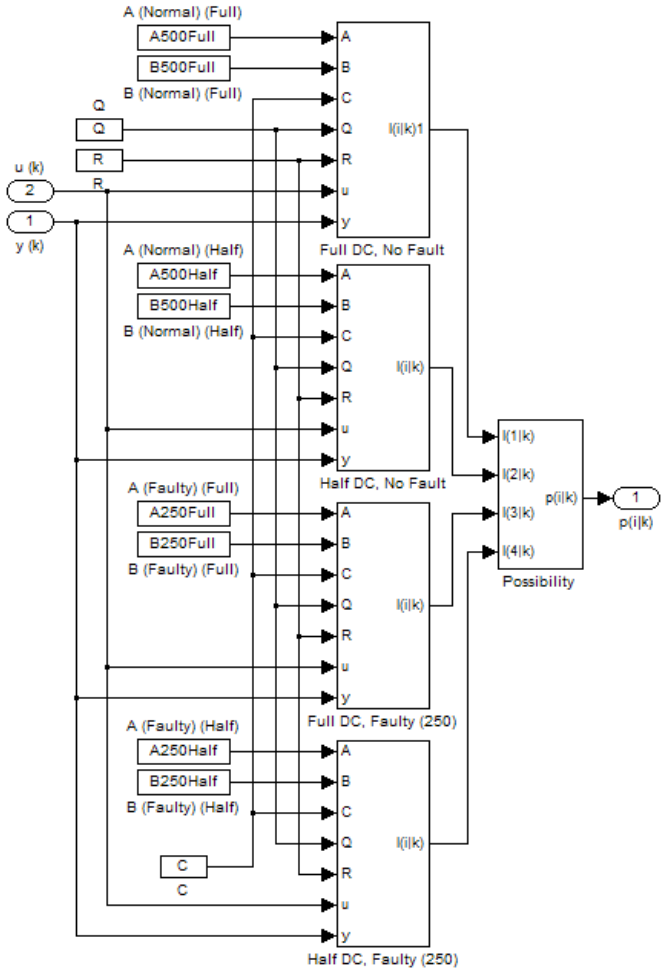


Figure C.13: KF Bank - MMAE

for all models, but the A and B matrix are different, depending on the scenario for the specific model.

C.1.13 Kalman Filter for MMAE

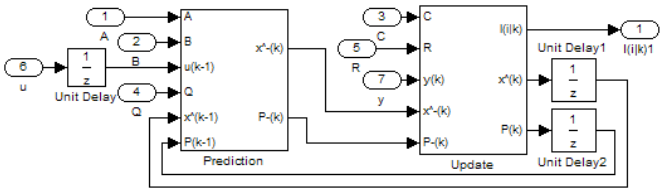


Figure C.14: Kalman Filter for MMAE

The Kalman filter used in the MMAE is almost identical to the one used for detection and split signal. The only real difference is the update procedure now generates a likelihood function, and not a fault indicator. Therefore, the prediction process is the same, and the description of it can be found in Section C.1.8. The update process is described in Section C.1.14.

C.1.14 Kalman Filter for MMAE - Update

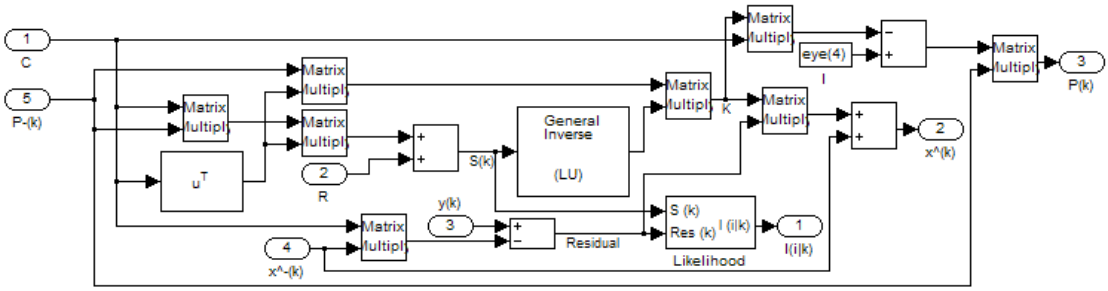


Figure C.15: Kalman Filter for MMAE - Update

The update process is identical to the one described in Section C.1.9 with the exception that the fault indicator calculation has been exchanged with the likelihood calculation, and obviously, so has the output. The likelihood calculation can be seen in Section C.1.15.

C.1.15 Kalman Filter for MMAE - Likelihood

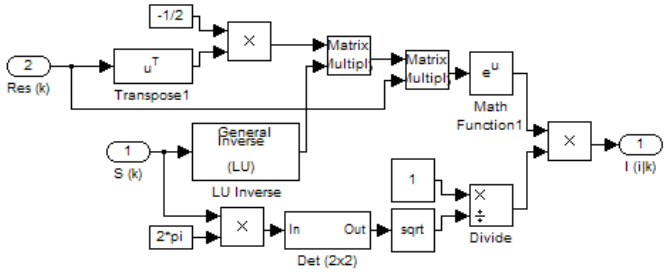


Figure C.16: Kalman Filter for MMAE - Likelihood

The likelihood is calculated based on the equations described in the FDI chapter, and needs to calculate a determinant of a matrix. As Simulink has no built in method of

calculating this (there only is a 3x3 determinant) a simple block has been created, which is described in Section C.1.16. The remaining elements can be visually verified to be in agreement with the used equations.

C.1.16 Kalman Filter for MMAE - Determinant

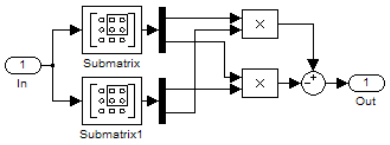


Figure C.17: Kalman Filter for MMAE - Determinant

The determinant is calculated by taking the first element of the first row multiplied with the second element of the second row, and then subtracting the second element of the first row multiplied with the first element of the second row. This is agreement with what can be seen in Figure C.17.

C.1.17 Probability for MMAE

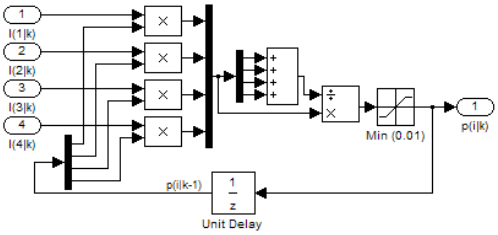


Figure C.18: Probability for MMAE

For the probability calculation the implementation is in complete agreement with the equations used, with one addition, namely the saturation block. As the MMAE can lock itself down when using digital numbers (with a limit to the amount of decimals), it is necessary to limit how low any single probability can go. By adding a saturation with a lower border of 0.01 and a maximum of 1, the system cannot lock itself.

C.1.18 CUSUM (Cumulative Sum)

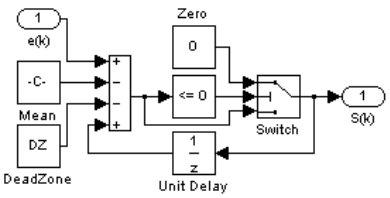


Figure C.19: CUSUM (Cumulative Sum)

The final block to be described for the Kalman filter, is the CUSUM. This block is used to increase the robustness of the fault indicator, by cumulating the sum. The block

is just a direct implementation of the equation for CUSUM, which can be verified by comparing the two.

C.2 The Extended Kalman Filter

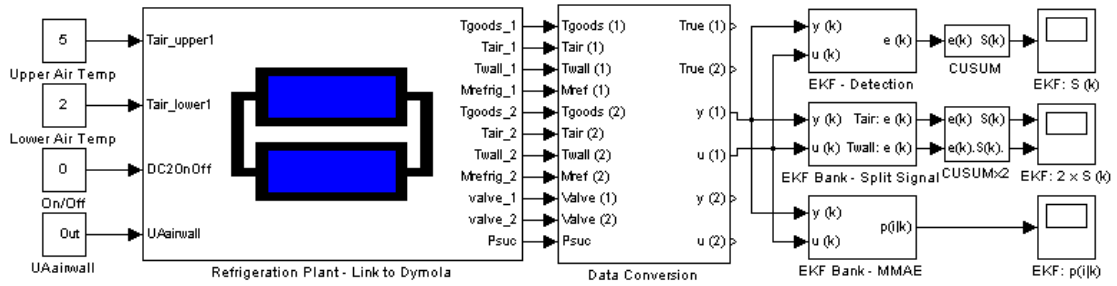


Figure C.20: EKF: Main Window

Figure C.20 illustrates the top-model view of the Kalman implementation. Starting from top left to bottom right

- Three constants; 'Upper Air Temp', 'Lower Air Temp' and 'On/Off'
  These three constants are identical to the ones used for the KF. They determine the upper and lower bound of the display case hysteresis controller, and determine if the second display case should be enabled respectively.

- 'UAairwall' block
  This block generates the $UA_{air \rightarrow wall}$ value, and is described in relation to the Kalman filter, in Section C.1.1.

- Dymola Interface
  The link to the Dymola software is obtained using this block. The used model has been selected within, and compiled. No values have been changed, and therefore there is no further description of this block.

- Data Conversion Block
  The data conversion block, which is identical to the one used for the KF, generates the $y(k)$ and $u(k)$ signals, and gives the true states out for comparison if needed. The block is described in Section C.1.2.

- EKF - Detection
  This block represents the detection system for the Extended Kalman filter, and is followed by a scope where the result can be seen. A description of the block can be found in Section C.2.1.

- EKF Bank - Split Signal
  Identical to the KF, two parallel Extended Kalman filters are inside this block, each getting only one of the two measured signals from the plant. A description of the block can be found in Section C.2.9.

- EKF Bank - MMAE
  A description of the Multiple Model Adaptive Estimator bank block can be found in Section C.2.10.

- CUSUM and CUSUMx2
  These are identical to the ones used for the KF and is thereby described in Section C.1.18.

C.2.1 EKF - Detection

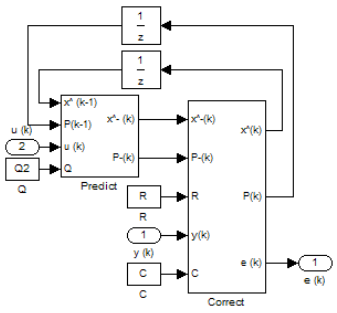


Figure C.21: EKF - Detection

The Extended Kalman filter is implemented both as detection and isolation. The detection part consists of several submodels, more than is the case for the KF. The A and B matrices are not used for the EKF, but C, R and Q2 matrices are defined in the MATLAB workspace, and therefore only needs to be referred to from Simulink. $y(k)$ and $u(k)$ are taken as input, while the fault indicator $e(k)$ is the only output, just like the KF. The Prediction block is described in Section C.2.2 and the Correction (another word for Update) block is described in Section C.2.8.

C.2.2 EKF for detection - Prediction

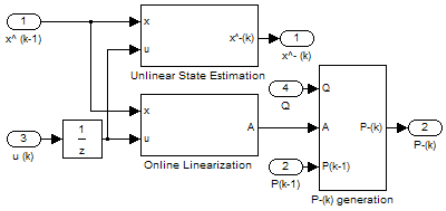


Figure C.22: EKF for detection - Prediction

In contrast to the Kalman filter, the calculation of the *a priori* estimate of the states in the EKF is not that simple. There are several steps, as the calculation is based on the non-linear model. The Unlinear State Estimation is described in Section C.2.3 and the Online Linearization is described in Section C.2.5. Finally, the generation of the predicted estimate covariance is illustrated in Section C.2.7.

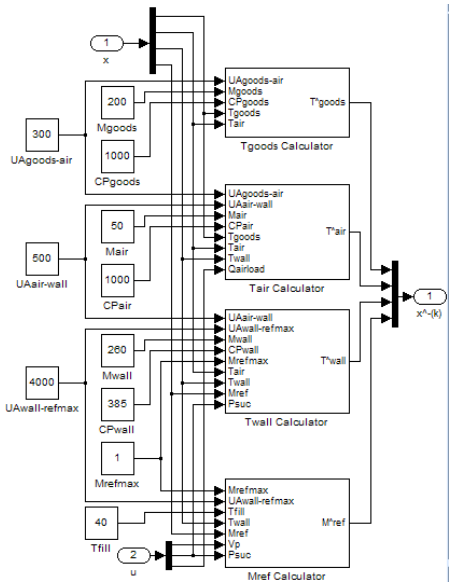


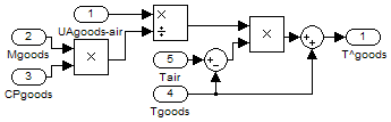
Figure C.23: EKF - Unlinear State Estimation

C.2.3 EKF - Unlinear State Estimation

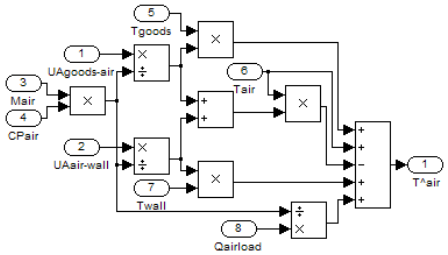
The Unlinear State Estimation consists of four seperate 'calculators' that determine what their specific state will be at the next sample, (k), whereafter they are combined to the *a priori* state estimate, written as $\hat{x}^-(k)$. The 'calculators' share several variables and use the previous estimates and input. The four blocks are illustrated in Section C.2.4, from Figure C.24a to C.24d on the next page.

C.2.4 EKF - State Estimation Calculators

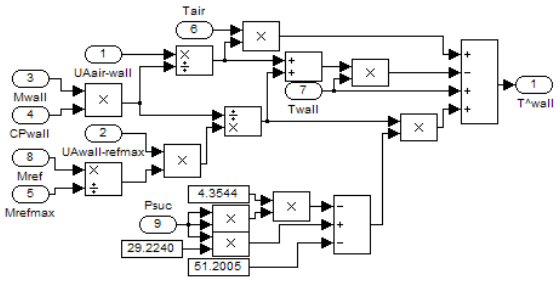
The calculators are not described as this would be cumbersome. If one wants to verify that they are true, they can be compared to the equations for the model, defined in the Modeling Chapter.



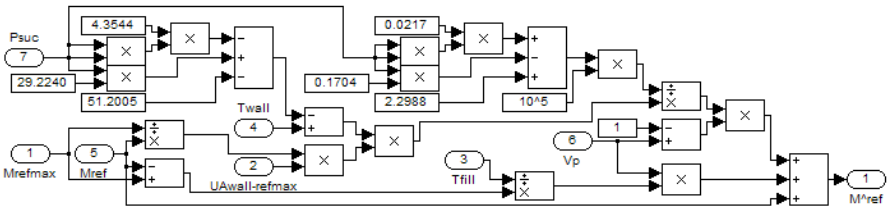
(a) T_{goods} calculator



(b) T_{air} calculator



(c) T_{wall} calculator



(d) M_{refrig} calculator

Figure C.24: EKF - State Estimation Calculators

C.2.5 EKF - Online Linearization

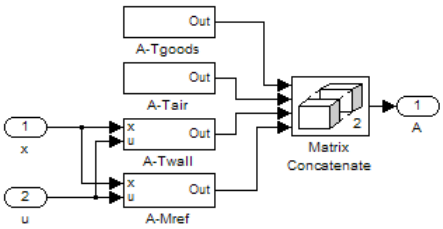
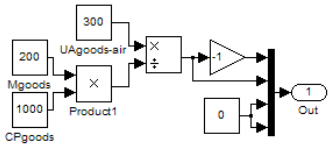


Figure C.25: EKF - Unlinear State Estimation

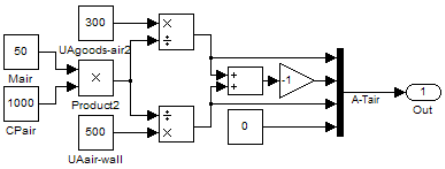
In the same way as the State Estimation Calculators, the Online Linearization has been divided into four submodels. The submodels can be seen in Figures C.26a to C.26d on the facing page as a part of Section C.2.6.

C.2.6 EKF - Online Linearization of States

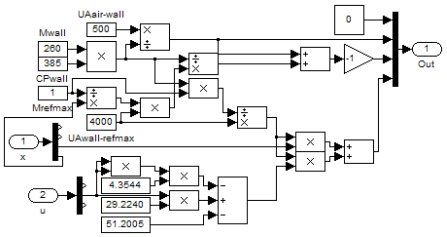
Again, it would seem cumbersome to describe how each submodel works, as they are built up of simple blocks, and it can easily be verified to be correct.



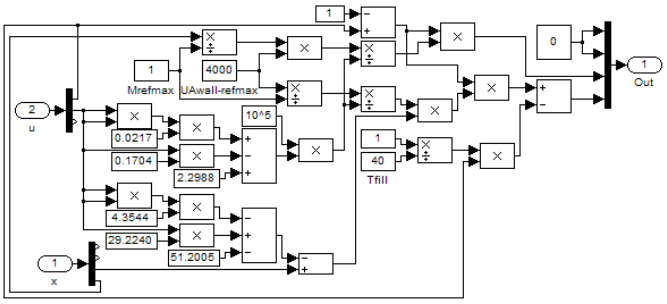
(a) T_{goods} Linearization



(b) T_{air} Linearization



(c) T_{wall} Linearization



(d) M_{refrig} Linearization

Figure C.26: EKF - Online Linearization of States

C.2.7 EKF - Prediction Estimate Covariance

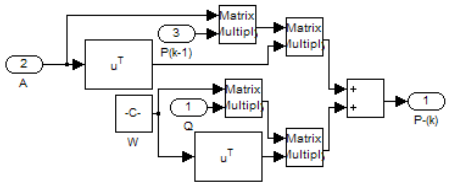


Figure C.27: EKF - Prediction Estimate Covariance

The Prediction Esimate Covariance can be calculated as shown in Figure C.27. W is a identity matrix of 2x2 as described in the report. The remaining elements are self-explanatory.

C.2.8 EKF - Correction/Update procedure

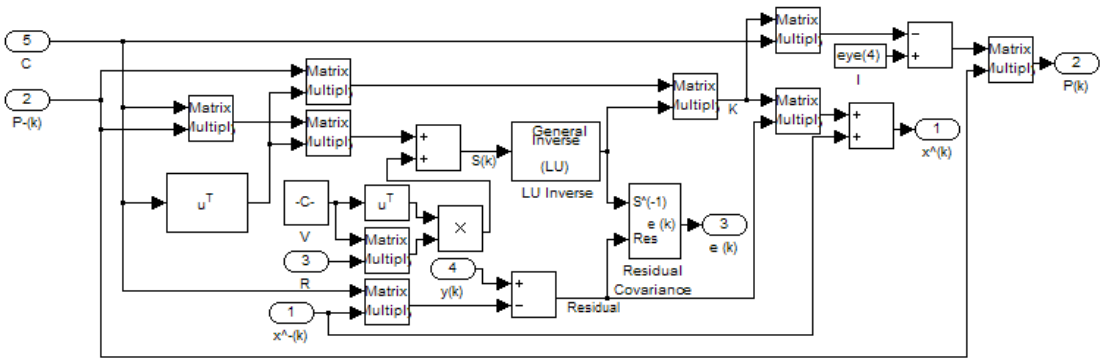


Figure C.28: EKF - Correction/Update procedure

The correction procedure of the Extended Kalman filter is almost identical to the update procedure of the Kalman filter described in Section C.1.9. The only difference is that for the EKF, V and V^T is multiplied with R, which can be seen from the figure. Even the fault indicator calculation is identical, and can therefore be seen in Section C.1.10.

C.2.9 EKF Bank - Split Signal

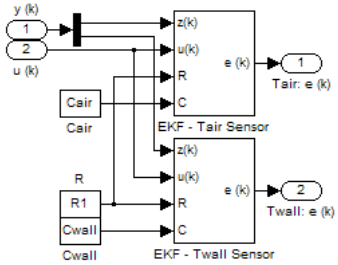


Figure C.29: EKF Bank - Split Signal

The split signal bank of Extended Kalman Filters uses two of the Extended Kalman Filters described in Section C.2.1, including all submodels. The only difference is the inputs to the filters. For the split signal setup, each filter has its own C-matrix, while R1 and Q2 are re-used.

C.2.10 EKF Bank - MMAE

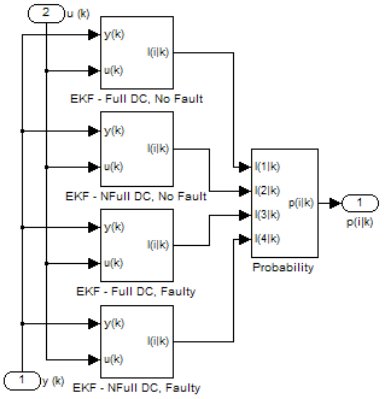


Figure C.30: EKF Bank - MMAE

The MMAE bank has four seperate Extended Kalman Filters that are almost identical to the one described in Section C.2.1. An example of one of these is shown in Section C.2.11. The probability calculation is identical to the one for the Kalman filter, which was described in Section C.1.17.

C.2.11 Extended Kalman Filter for MMAE

As mentioned, the Extended Kalman Filters used here are close to identical to the ones described in Section C.2.1. The only difference is that now Rekf and Qekf is used

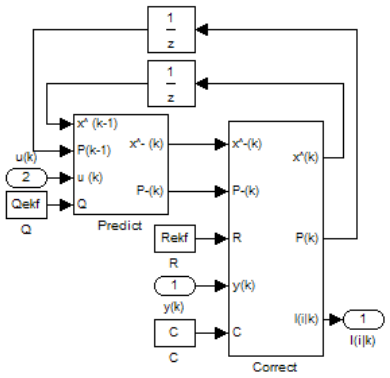


Figure C.31: Extended Kalman Filter for MMAE

instead of R and Q2, and the likelihood is calculated instead of the fault indicator. The calculation of the likelihood is identical to the one for the Kalman Filter, and the description can therefore be found in Section C.1.15. Obviously the four models need to be different to simulate different scenarios. This is achieved by changing the values Mgoods and UAairwall shown in Figures C.26a to C.26d on page 115 and C.23. The values used are

- Model 1: Mgoods=200, UAairwall=500
- Model 2: Mgoods=25, UAairwall=500
- Model 3: Mgoods=200, UAairwall=250
- Model 4: Mgoods=25, UAairwall=250

which are implemented in the same order as shown in Figure C.31.

C.3 The Unknown Input Observer

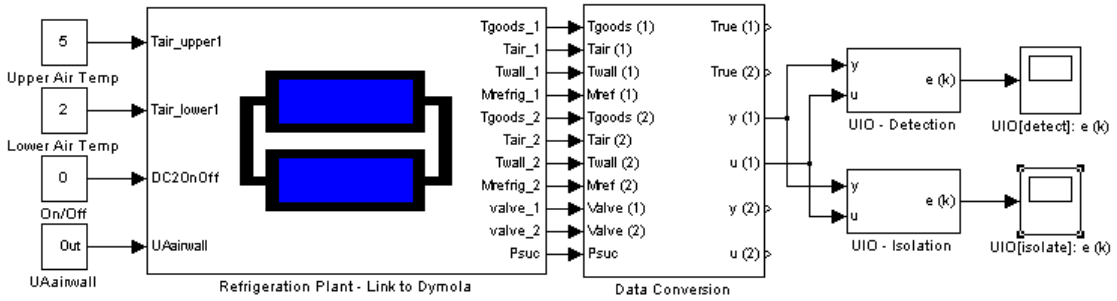


Figure C.32: UIO: Main Window

Figure C.32 illustrates the top-model view of the Unknown Input Observer implementation, which as almost identical to the two previous. Starting from top left to bottom right

- Three constants; 'Upper Air Temp', 'Lower Air Temp' and 'On/Off'
  These three constants are identical to the ones used for the KF and EKF. They determine the upper and lower bound of the display case hysteresis controller, and determine if the second display case should be enabled respectively.

- 'UAairwall' block
  This block generates the $UA_{air \rightarrow wall}$ value, and is described in relation to the Kalman filter, in Section C.1.1.

- Dymola Interface
  The link to the Dymola software is obtained using this block. The used model has been selected within, and compiled. No values have been changed, and therefore there is no further description of this block.

- Data Conversion Block
  The data conversion block generates the $y(k)$ and $u(k)$ signals, and gives the true states out for comparison if needed. The block is a little different from the one used for the KF and EKF, and is described in Section C.3.1.

- UIO - Detection
  This block represents the detection system for the Unknown Input Observer, and is followed by a scope where the result can be seen. A description of the block can be found in Section C.3.3.

- UIO - Isolation
  The isolation procedure for the UIO is somewhat different from the two other approaches, as is described in the FDI chapter. A description of the isolation block can be found in Section C.3.5.

C.3.1 Data Conversion Block

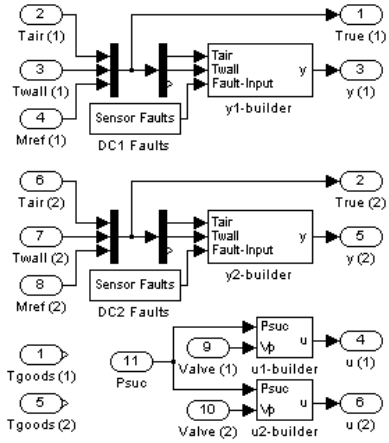


Figure C.33: Data Conversion Block

The Data conversion block is used to arrange the simulated data from the Dymola model, including the introduction of noise and faults. The three states T_{air}, T_{wall} and M_{refrig} are used for the UIO and are combined into a mux. The resulting signal is split into two, one which holds the true states in case they are needed and one where T_{air} and T_{wall}

are isolated and sent with the signal from the 'Sensor Faults'-block (identical to the one described for the Kalman filter in Section C.1.3) into the 'yz-builder'. The 'yz-builder' (where z is a integer representing the display case number) block is described in Section C.1.4 (as it has not changed), while the 'uz-builder' block, which has been changed a little, is described in Section C.3.2.

C.3.2 uz-builder

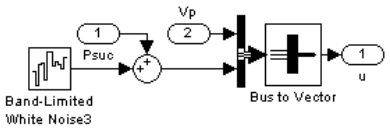


Figure C.34: uz-builder

The only difference in the uz-builder (compared to the one used for KF and EKF described in Section C.1.5), is that $Q_{airload}$ has been removed. This is because it is now a disturbance, or unknown input.

C.3.3 UIO - Detection

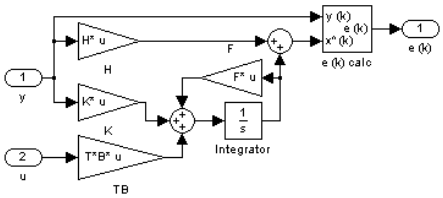


Figure C.35: UIO - Detection

The detection block of the Unknown Input Observer is quite simple. The calculated matrices K, H, T and F are introduced in gain blocks, and the complete setup can be verified visually to comply with the relevant equations. The only sub-model is the 'e (k) calc' block, which generates a fault indicator. The block is described in Section C.3.4.

C.3.4 UIO - Fault Indicator

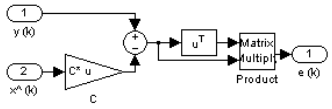


Figure C.36: UIO - Fault Indicator

The fault indicator block takes two inputs. The first is the measurements from the system, while the other is the estimated states. By multiplying the C matrix with the estimated states, the estimated output can be found. Subtracting the estimated states from the measured states, will give the residual. By multiplying the residual vector transposed with itself (not transposed), the result is a scalar which can be used for fault detection. The scalar is then sent on as an output.

C.3.5 UIO - Isolation

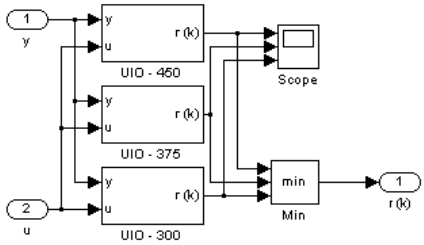


Figure C.37: UIO - Isolation

The isolation block consists of three UIO blocks identical with the detection block shown in Figure C.35 with only a few small exceptions. Instead of referring to K and F in the gain blocks, the values KXXX and FXXX are used (where XXX is either 300, 375 or 450 for each of the models) which represent the scenarios where $UA_{air \rightarrow wall}$ is faulty.

C.4 The Multiple Method Approach

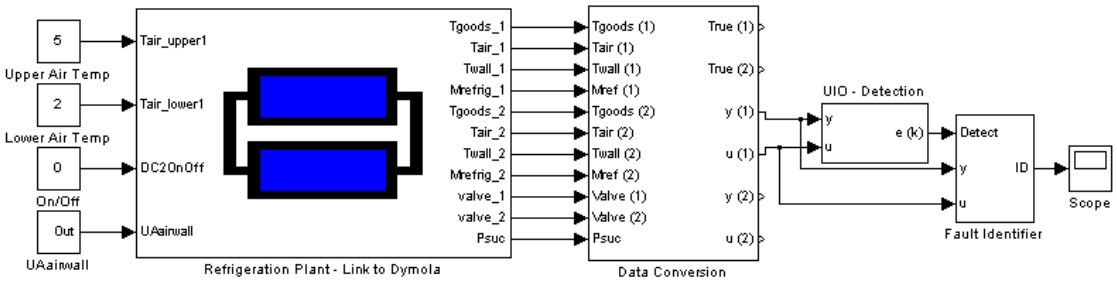


Figure C.38: Multiple Method: Main Window

Figure C.32 illustrates the top-model view of the Multiple Method implementation, which as almost identical to the other setups, and actually re-uses several of the previously described blocks. Starting from top left to bottom right

- Three constants; 'Upper Air Temp', 'Lower Air Temp' and 'On/Off'
  These three constants are identical to the ones used for the the other methods. They determine the upper and lower bound of the display case hysteresis controller, and determine if the second display case should be enabled respectively.

- 'UAairwall' block
  This block generates the $UA_{air \rightarrow wall}$ value, and is described in relation to the Kalman filter, in Section C.1.1.

- Dymola Interface
  The link to the Dymola software is obtained using this block. The used model has been selected within, and compiled. No values have been changed, and therefore there is no further description of this block.

- Data Conversion Block
  The data conversion block generates the $y\,(k)$ and $u\,(k)$ signals, and gives the true

states out for comparison if needed. The block is identical to the one used for the UIO, and is thereby described in Section C.3.1.

- UIO - Detection
  This block represents the detection system for the Multiple Method approach. As the block is identical to the one used in the UIO section, a description of the block can be found in Section C.3.3.

- Fault Identifier
  The Fault Identifier is a block which based on the 'UIO - Detection' block and some internal blocks gives an output representing the current fault-scenario. Section C.4.1 gives a description of the blocks contents.

C.4.1 Multiple Method - Fault Identifier

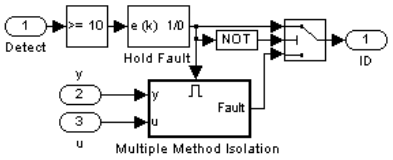


Figure C.39: Multiple Method - Fault Identifier

As mentioned, the Fault Identifier block gives an output which can be used to determine if a fault has occurred, and in that case, which fault it is. To do this, it uses the 'Hold Fault' block described in Section C.4.2 and the 'Multiple Method Isolation' block described in Section C.4.3. The functionality of the block is first to check if the detection signal goes above 10, as a fault will then be claimed. Hereafter, the signal is put into the 'Hold Fault' block which keeps the detection signal live for 5 minutes since last detection. At the left side is a selector, which gives a zero as output when no fault is detected. If a fault is detected, the output signal from the isolation block is given as output instead.

C.4.2 Multiple Method - Hold Fault

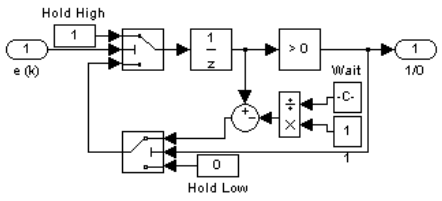


Figure C.40: Multiple Method - Hold Fault

The block actually just acts as a timer giving out a output of 1 that resets each time a high (1) signal is received on the e(k) input. When a low (0) signal is input, it starts counting down for 'Wait' samples, which for this setup is seconds. After the time has passed by, the output is set to 0.

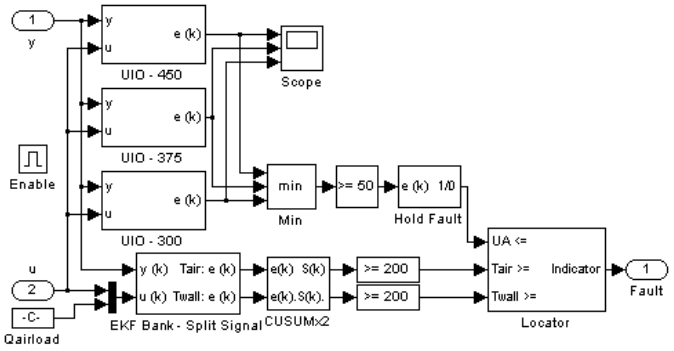


Figure C.41: Multiple Method - Isolation

C.4.3 Multiple Method - Isolation

Starting from the top left, the UIO isolation system can be seen, using multiple models, described in Section C.3.5. Underneath is an Extended Kalman Filter block using the split sensor setup described in Section C.2.9. As the EKF model needs $Q_{airload}$ as input, it is added just before the block. After the EKF block, the CUSUM is used, which is described in Section C.1.18. The signals from the two isolation methods are compared the their respective borders and the UIO isolator is sent though a 'Hold Fault' block, which was described in Section C.4.2. Finally the signals are sent on to the 'Locator' block, which is described in Section C.4.4.

C.4.4 Multiple Method - Locator

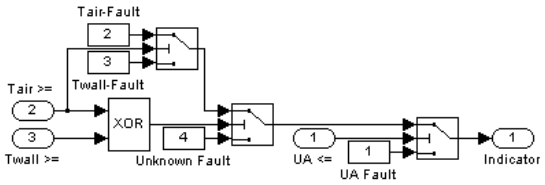


Figure C.42: Multiple Method - Locator

The 'Locator' is actually just some logic which determines which fault to claim based on the input signals. This time starting from right to left, the first block is a selector which, if a parameter drift is determined, gives a 1 as output. If the parameter drift is not determined, the output signal instead comes from another selection block which, if both or none of the EKF's claim a fault at the same time, gives out a 4, an indication of an unknown fault. In case one (and only one) of the EKF filters are claiming a fault, depending on which sensor is faulty, either a 2 or a 3 is given as output.

Appendix D
MATLAB Documentation

D.1 File: bodeplot.m

```
A=[-0.00150059361782129,0.00149834614470013,0,0;
    0.00587807061053345,-0.0160509598641797,0.00997931089551525,0;
    0,0.00854458281472063,-0.00837466657679920,-0.463434255739900;
    0,0,0,-0.0677419514798366;];
B=[0,0,0;0,0,2.00744285953520e-05;0,-0.0129282632891503,0;0.0454110728662820,0,0;];
C=[0 1 0 0; 0 0 1 0;];

for m=1:3
    Bnew=B(:,m);
    for n=1:2
        Cnew=C(n,:);
        BaseModel=idss(A,Bnew,Cnew,0);
        set(BaseModel,'ts',0);
        FBModel=feedback(BaseModel,unity);
        switch n
            case 1
                set(FBModel,'OutputName','Tair');
            case 2
                set(FBModel,'OutputName','Twall');
        end;
        switch m
            case 1
                set(FBModel,'InputName','Vp');
            case 2
                set(FBModel,'InputName','Psuc');
            case 3
                set(FBModel,'InputName','Qairload');
        end;
        plot(FBModel);
    end;
end;
```

D.2 File: onlineparamest.m

```
function[A B mA]=onlineparamest(ydata,udata,ts,RO,Window,Wait,Stop)
  %Offline version of online parameter estimation. By giving a large array
  %of input and output data as ydata and udata respectively (including the
  %sampling time ts), a recursive parameter estimation is run. By setting
  %RO (Reduced Order) to zero, a 4x4 c-matrix is used. If RO is one (1), it
  %will be assumed that only Tair and Twall are measurable. The variable
  %'Window' tells how many samples should be used for the estimation, while
  %'Wait' defines how often the parameter estimation should be run. As an
```

```
%example, if Wait=1, the parameter estimation will be run for each sample
%and if Wait=10, it is run once for every 10 samples. The final parameter
%called 'Stop' is introduced to be able to stop the process after a given
%number of samples. If 'Stop' is not defined, the calculations are run
%til the end of the data.

[N b]=size(ydata);

if nargin==7 && Stop<=N
    N = Stop;
end;

%Initialization
k=1;
A=[-0.0015 0.0015 0 0; 0.0059 -0.0161 0.0100 0;
    0 0.0085 -0.0084 -0.4634; 0 0 0 -0.0677;];
B=[0 0 0; 0 0 2.074*10^(-5); 0 -0.0129 0; 0.0454 0 0;];
BaseA=[NaN NaN 0 0; NaN NaN NaN 0; 0 NaN NaN NaN; 0 0 0 NaN;];
BaseA(4,4)=A(4,4); %Fix Mref value
%BaseB=[0 0 0; 0 0 NaN; 0 NaN 0; NaN 0 0;];
BaseB=B; %This makes the B-matrix fixed.

if RO
    C=[0 1 0 0; 0 0 1 0;];
    D=[0 0 0; 0 0 0;];
    K=[0 0; 0 0; 0 0; 0 0;];
    X0=[3.5 0 0 0]';
else
    C=[1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1;];
    D=[0 0 0; 0 0 0; 0 0 0; 0 0 0;];
    K=[0 0 0 0; 0 0 0 0; 0 0 0 0; 0 0 0 0]';
end;

%Actual Process
while k<=N
  k=k+1;
  if k>(Window+1) && mod(k,Wait)==0
    DataModel=iddata(ydata((k-Window):k-1,:),udata((k-Window):k-1,:),ts);
    if RO
        X0(2:3,1)=ydata(k-(Window+1),:);
    else
        X0=ydata(k-(Window+1),:)';
    end;
    PrePem=idss(A(:,:,k-1),B(:,:,k-1),C,D);
    set(PrePem,'ts',0); %Using the continous model!
    setstruc(PrePem,BaseA,BaseB,C,D,K,X0);
    PostPem=pem(DataModel,PrePem);
    A(:,:,k)=PostPem.A;
    B(:,:,k)=PostPem.B;
    display(k);
  else
    A(:,:,k)=A(:,:,k-1);
    B(:,:,k)=B(:,:,k-1);
  end;
end;
plotfuncs(A);
mA=meanify(A,(1-1/600));
plotfuncs(mA,4);
end
```

D.3 File: meanify.m

```
function [mA]=meanify(data,factor)
%The meanify function (ignore the name) uses a method similar to an IIR
%filter to filter out high frequenzy changes, which are caused by noise.
%The value 'factor' determines how high a weight the 'old' data will have
%and the new measurement will have an effect of (1-factor).

[a b N]=size(data); %Find the size of the array

mA(:,:,1:N)=data(:,:,1:N); %Pre-allocate memory

    for n=2:N %Start from 2, so the first value is real data.
        mA(1,1,n)=factor*mA(1,1,n-1)+(1-factor)*data(1,1,n);
        mA(1,2,n)=factor*mA(1,2,n-1)+(1-factor)*data(1,2,n);
        mA(2,1,n)=factor*mA(2,1,n-1)+(1-factor)*data(2,1,n);
        mA(2,2,n)=factor*mA(2,2,n-1)+(1-factor)*data(2,2,n);
        mA(2,3,n)=factor*mA(2,3,n-1)+(1-factor)*data(2,3,n);
        mA(3,2,n)=factor*mA(3,2,n-1)+(1-factor)*data(3,2,n);
        mA(3,3,n)=factor*mA(3,3,n-1)+(1-factor)*data(3,3,n);
        mA(3,4,n)=factor*mA(3,4,n-1)+(1-factor)*data(3,4,n);
```

```
    end; %Obviously, only the un-locked values are calculated.
end
```

D.4 File: plotfuncs.m

```
function[]=plotfuncs(A,offset)
%This function is made for plotting the parameters for the four states
%when using the recursive parameter estimation. The matrix A must hold
%the A-matrix (4x4) of parameters from the first iteration (:,:,1) to
%the final iteration (:,:,I).
%
%The offset parameter is used to make sure the function does not use
%the same figures more than once in case several calls are made to this
%function, or other figures are to be protected. If offset is set to 4,
%the first used figure-numer will be 5.

StateNames={'Tgoods' 'Tair' 'Twall' 'Mrefrig'};

if nargin==1
    offset=0;
end;

[a b I]=size(A);

for n=1:4
    figure(offset+n);
    for m=1:4
        subplot(4,1,m); %Plot 4 rows, 1 column
        plot(reshape(A(n,m,:),I,1));
        V = AXIS;
        axis([0 I V(3) V(4)]);
        if m==1
            title(StateNames(n))
        end;
        ylabel(StateNames(m));
    end;
end;
end
```

Bibliography

[CPZ96] J. Chan, R. J. Patton, and H. Y. Zhang. Design of unknown input observers and robust fault-detection filters. *International Journal of Control*, 63:pp. 85–105, 1996.

[DH01] Roy J. Dossat and Thomas J. Horan. *Principles of Refrigeration.* Prentice Hall, 5 edition, 2001.

[Din03] Ibrahim Dinçer. *Refrigeration Systems and Applications.* John Wiley & Sons, Ltd., 2003.

[FPW98] Gene F. Franklin, J. David Powell, and Michael L. Workman. *Digital Control of Dynamic Systems.* Addison Wesley Longman, 3 edition, 1998.

[IB97] R Isermann and P Balle. Trends in the application of model-based fault detection and diagnosis of technical processes. *Control Engineering Practice*, Vol. 5:pp. 707–719, 1997.

[Lar07] Lars F. S. Larsen. *Model Based Control of Refrigeration Systems.* PhD thesis, Department of Control Engineering - Aalborg University, 2007.

[LIZW] Lars F. S. Larsen, Roozbeh Izadi-Zamanabadi, and Rafael Wisniewski. Supermarket refrigeration system - benchmark for hybrid system control. Paper made by Danfoss and AAU which is the basis for the plant model.

[REQ] `http://www.et.web.mek.dtu.dk/WinDali/Files/RefEqns_3.10.ZIP`. Software for calculating refrigerant properties by Skovrup, M. J., Version 3.10.

[TRM] `http://www.sztaki.hu/conferences/safeprocess2000/termfr.html`. Applied Terminology of Fault Detection, Supervision and Safety for Technical Processes.

[WB06] Greg Welch and Gary Bishop. An introduction to the kalman filter. Good description of the Kalman Filter and Extended Kalman Filter, July 2006.

[WCU] `http://en.wikipedia.org/wiki/Cusum`. Wikipedia: Cusum (2008/2009).

[WKF] `http://en.wikipedia.org/wiki/Kalman_filter`. Wikipedia: The Kalman Filter (2008/2009).

[WRE] `http://en.wikipedia.org/wiki/Refrigeration`. Wikipedia: Refrigeration - includes the Vapour-Compression cycle.