



**NTNU – Trondheim**  
Norwegian University of  
Science and Technology

# Fault-tolerant MPC for active mitigation of actuation faults in process systems

Jon Håman Brusevold

December 2014

PROJECT THESIS

Department of Engineering Cybernetics  
Norwegian University of Science and Technology

Supervisor 1: Professor Bjarne Anton Foss

Supervisor 2: Postdoc Researcher Brage Rugstad Knudsen



## **Project assignment**

Name of candidate: Jon Håman Brusevold

Subject: Engineering Cybernetics

Title: Fault-tolerant MPC for active mitigation of actuation faults in process systems

Title (in Norwegian): Feiltolerant MPC for aktive håndtering av feil i prossessystem.

### Task description:

1. Perform a literature study on fault-tolerant control (FTC). Classify and describe components in various FTC schemes. Look into differences in active and passive FTC, and physical vs. analytical redundancy in the FTC architectures.
2. Perform a brief literature study on different fault detection and isolation (FDI) schemes, and describe how this component may be integrated or kept as a separate component in the control scheme.
3. Describe briefly the components in model predictive control (MPC). Consider both conventional stabilizing MPC and economic MPC.
4. Consider different schemes for including fault tolerance in MPC. Focus on faults in actuators, but consider also faults in measurements and the process itself.
5. Implement fault-tolerant MPC on a suitable process-system test problem. Perform case studies on this test problem, implementing different techniques for handling faults in the system.

This project is designed as a basis for continuation on a master project on the same topic.

Starting date: 18.08.2014

End date: 20.12.2014

Co-supervisor: PhD student Brage Rugstad Knudsen, NTNU

Trondheim, 22.08.2014

Bjarne Foss  
Professor/supervisor



# Preface

This report is written in the fall of 2014 as a part of the final year project of the Master of Technology program in Engineering Cybernetics at the Norwegian University of Science and Technology.

The reader of of this report is assumed to have basic knowledge of control theory and linear systems, as well as a basic understanding of optimization and model predictive control.

I would like to thank my supervisor, Professor Bjarne Foss and my co-supervisor Postdoc Researcher Brage Rugstad Knudsen for providing me with guidance and valuable input during the course of this work.

This project leads to a master thesis on the same topic for the spring of 2015.

Trondheim, 2014-12-20

Jon Håman Brusevold



# Abstract

All modern technological systems are vulnerable to faults. It is of high importance that these systems avoid failure that can lead to great damage, or in the worst case, to the loss of lives. Many systems are expected to work under all potential faults and situations, and the theory of designing control strategies to handle these critical situations therefore demands a great amount of attention.

This report describes general concepts and methods in fault-tolerant control, and gives a brief introduction to fault detection and isolation. The commonly used model predictive controller is analyzed with respect to fault-tolerance, and the ease of which fault-handling can be included in this controller is illustrated. Additionally, the standard approach of ensuring feasibility in fault situations can in some cases be conservative. A new approach of ensuring feasible solutions when faults occur, using *soft constraints* and a *penalty function*, is introduced.

The proposed theory is verified by numerical simulations, and the approach to deal with infeasibility issues in the case of faults shows promising results. The report ends with a discussion where emphasis is placed on challenges and important assumptions that are made for the proposed method.





# Contents

Preface . . . . .	i
Abstract . . . . .	ii
Abbreviations . . . . .	vii
<b>1 Introduction</b>	<b>1</b>
1.1 Faults and failures . . . . .	1
1.2 Motivation and background . . . . .	2
1.3 Introduction to fault-tolerant Control . . . . .	2
1.4 Model Predictive Control for FTC . . . . .	3
1.5 Report outline . . . . .	3
<b>2 Fault-tolerant control</b>	<b>5</b>
2.1 Physical and analytical redundancy . . . . .	5
2.2 The fault-tolerant control problem . . . . .	6
2.3 Modeling of a faulty system . . . . .	6
2.4 Passive fault-tolerant control . . . . .	8
2.5 Active fault-tolerant control . . . . .	8
2.5.1 Reconfiguration . . . . .	9
2.5.2 Fault accommodation . . . . .	10
2.6 Fault-tolerant control methods . . . . .	10
2.6.1 Linear quadratic . . . . .	10
2.6.2 Pseudo-inverse . . . . .	12
2.6.3 Adaptive control . . . . .	13
2.6.4 Robust control ( $H_\infty$ control) . . . . .	13
<b>3 Fault detection and isolation</b>	<b>15</b>
3.1 Fault detection and isolation concept . . . . .	16
3.1.1 Residual generation . . . . .	16
3.1.2 Residual evaluation . . . . .	17
3.2 Fault detection and isolation methods . . . . .	17
3.2.1 Diagnostic observers . . . . .	18
3.2.2 Parity relations . . . . .	18
3.2.3 Kalman filters . . . . .	19

<b>4</b>	<b>Fault-tolerant Model Predictive Control</b>	<b>21</b>
4.1	Model Predictive Control . . . . .	22
4.1.1	Cost function . . . . .	22
4.1.2	Constraints . . . . .	23
4.1.3	Softening the constraints . . . . .	23
4.1.4	MPC principle . . . . .	24
4.1.5	Economic MPC . . . . .	25
4.2	Incorporating fault-tolerance in MPC . . . . .	27
4.2.1	Actuator faults . . . . .	27
4.2.2	System faults . . . . .	28
4.2.3	Sensor faults . . . . .	28
4.2.4	Reconfiguring the MPC . . . . .	29
4.3	Fault-tolerant MPC feasibility . . . . .	29
4.4	Proactive fault-tolerant model predictive control . . . . .	31
4.4.1	Economic MPC with fault-tolerance . . . . .	32
<b>5</b>	<b>Numerical results</b>	<b>35</b>
5.1	Active fault-tolerant MPC . . . . .	35
5.1.1	Actuator saturation fault . . . . .	36
5.1.2	Actuator freezing . . . . .	37
5.1.3	Actuator input loss and effectivity decrease . . . . .	37
5.2	Proactive fault-tolerant Economic MPC . . . . .	39
5.2.1	Without proactive control . . . . .	39
5.2.2	Proactive control with linear penalty . . . . .	41
<b>6</b>	<b>Discussion</b>	<b>47</b>
6.1	Active fault-tolerant MPC . . . . .	47
6.2	Proactive fault-tolerant Economic MPC . . . . .	48
<b>7</b>	<b>Conclusion</b>	<b>51</b>
<b>8</b>	<b>Future work</b>	<b>53</b>
	<b>Bibliography</b>	<b>54</b>

# Abbreviations

FTC	=	Fault tolerant control
AFTC	=	Active fault tolerant control
PFTC	=	Passive fault tolerant control
FDI/FDD	=	Fault detection and isolation/detection
MPC	=	Model predictive control
EMPC	=	Economic model predictive control
FTMPC	=	Fault tolerant model predictive control
FTEMPC	=	Fault tolerant economic model predictive control
RTO	=	Real time optimization



# Chapter 1

## Introduction

In safety-critical systems, it is crucial that some level of performance is maintained in the event of faults. Systems such as aircrafts and nuclear power plants need to have fault-handling as the top priority. Malfunctions in actuators, sensors or other components greatly reduce safety, and the economic losses can be severe. Methods for handling these type of critical situations are therefore needed to avoid potential catastrophic events. This chapter begins with a definition of faults and failures and gives an introduction to fault-tolerant control (FTC) and the motivation for developing effective algorithms for detecting and handling faults. Model predictive control is presented as a suitable framework for incorporating fault-tolerance, and the last section explains the outline of the report.

### 1.1 Faults and failures

Isermann and Ballé (1997) defines fault and failure, in compliance with the definitions given by the IFAC SAFEPROCESS technical committee, in the following way:

**Fault:**

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

**Failure:**

A permanent interruption of a system's ability to perform a required function under specified operating conditions.

It is clear from the definition that a failure is a condition that is much more severe than a fault. When a fault occurs in a component for example, the component may still be usable, but becomes less effective. For a failure however, a totally different component is needed to achieve the same objective. Failure might also lead to the need for system shut-down. Throughout this report, "fault" will be

used when something is wrong with a component, and "failure" will typically be described as the system becoming unstable due to a fault not being compensated for by the controller.

## 1.2 Motivation and background

A conventional feedback control design might result in poor performance, or even instability for a system that is affected by faults. This has motivated a considerable amount of research in the last decades on how to handle these potentially problematic events, and have the system retain an "acceptable" level of performance. Zhang and Jiang (2008) define a fault-tolerant control system as a closed-loop control system which can tolerate component malfunctions, while maintaining desirable performance and stability properties. The design of these controllers are critical for reliable operation of many systems.

The research can be divided into two branches: The part of fault detection and isolation/diagnosis (FDI/FDD), and the part of fault-tolerant control. While the research is somewhat separated, they are often used in a common scheme, and most books on FTC also include theory on FDI, e.g. Isermann (2006), Blanke et al. (2006).

Several different approaches to FTC have been discovered over the years, including the use adaptive control, linear quadratic,  $H_\infty$  control, methods based on model-matching, and model predictive control (Zhang and Jiang, 2008). The goal is to design a controller that is either robust to a certain set of faults, or is easily modified on-line when a fault occurs.

## 1.3 Introduction to fault-tolerant Control

Fault-tolerant control can be classified into two types: *active* and *passive*. Active fault-tolerant control (AFTC) relies on an FDI to detect and diagnose faults. The controller is then modified on-line in order to keep the system performing at a satisfactory level. Passive fault-tolerant control (PFTC) is designed, off-line, to be robust to a set of possible faults. This approach does not rely on FDI schemes, nor is any on-line modification required. The two approaches are described in more detail in Chapter 2.

In AFTC, information about the fault is used to modify the controller on-line. One of the most challenging issues with this approach is the integration of FDI and FTC. Effective fault-handling requires quick and exact information about a fault, as well as intelligent use of this information in the FTC. The main focus of this report is on FTC, and it is assumed that quick and reliable information from an FDI is available. Because of the rapid development in FDI schemes, this is a reasonable assumption (Venkatasubramanian et al., 2003).

Another critical assumption for fault-tolerant control is that the system is controllable after the fault has occurred. If this does not hold true, it is not possible to design a control law to control the system and shut-down is necessary.

## 1.4 Model Predictive Control for FTC

Model predictive control (MPC) has become one of the most commonly used control algorithms for multi-variable systems. This is due to the inherent property to easily handle constraints in an optimal manner (Mellichamp et al., 2010, Ch 20).

MPC also allows for effective ways of incorporating fault tolerance in the controller (Maciejowski, 1997). There are several examples in the literature on the use of MPC in this context, including Miksch et al. (2008). The advantage of using MPC for FTC is its property to handle faults like saturation and blocking of actuators, or faults that change the system itself. When information about a fault is received from the FDI, it is represented as a constraint or a change in the internal model in the optimization problem. The MPC then effectively calculates a new control law to accommodate the fault. Chapter 4 explains the use of fault-tolerant MPC (FTMPC) in more detail.

Infeasibility is a known issue with MPC, and this is especially true when faults instantly change the system's behaviour. An approach to handle infeasibility-issues after a fault, is to set tight operating regions for the system to make sure the problem is feasible when a fault occurs. However, this can be a rather conservative approach. Chapter 4 introduces an approach based on soft constraints that allows for more flexible operation.

## 1.5 Report outline

The contents of this report are organized as follows: In Chapter 2, the basic concepts and notations used in fault-tolerant control are presented, followed by examples of different schemes. Chapter 3 briefly introduces fault detection and isolation, and a few commonly used methods are described. In Chapter 4, the basic components of the MPC is introduced and fault-tolerant control using MPC is explained. Chapter 5 contains a numerical example with simulations based on the framework described in Chapter 4. The results and interpretation of the numerical example are discussed in Chapter 6. Finally, the conclusions are presented in Chapter 7, and a brief note about future work is included in Chapter 8.





## Chapter 2

# Fault-tolerant control

This section describes the basic concepts and notations used in fault-tolerant control, and presents some of the more commonly used methods for fault accommodation. Most of the notations used in this chapter are inspired by Blanke et al. (2006).

### 2.1 Physical and analytical redundancy

*Physical redundancy* is the duplication of physical components in the system in order to increase reliability. This is usually in the form of a backup component that is only used when the main component fails. Because of the very small chance of each component failing, the probability of all failing is extremely small. However, by using a high amount of extra components, the system quickly becomes more complex which is more prone to various issues. Another problem is that components usually have a high cost, and relying only on physical redundancy is therefore not cost effective.

*Analytical redundancy* is the design of fault-tolerant control to effectively make use of the redundancies in the system, without the use of duplicates for all components. This can save a lot of money, and avoid unnecessary complex system implementation. It is important to note that this type of redundancy of using less components to achieve reliability is only possible if more components are used than necessary to achieve an objective (Blanke et al., 2006, Chapter 7). The problem of designing a fault-tolerant controller in order to best utilize both types of redundancies can be a challenging issue (Zhang and Jiang, 2008). However, by intelligently using the remaining components after a fault has occurred, one can often achieve the same type of fault tolerance as with physical redundant components. This is the fault-tolerant control problem.

## 2.2 The fault-tolerant control problem

As described in Section 1.1, faults change the behaviour of the system. Actuators can lose effectivity, get stuck, or even lose their input. Pipes can get blocked and sensors can fall out. The standard control problem in the faultless case is to find a control law in order to make the system behave in a certain way while satisfying some constraints. Since faults change the system behaviour, this control problem therefore changes when a fault occurs. When this happens, one needs to make sure that the system is still behaving in a satisfactory way by solving the new control problem that arises. In some cases, the new objective might be different from in the nominal, fault-free case, if the old objective is not possible to be met. A new objective therefore needs to be defined. This can be to reach the old objective as closely as possible, or in the worst case just stabilizing the system.

As described in Blanke et al. (2006), the nominal control problem can be defined as

$$\langle O, C, U \rangle,$$

where the set  $O$  defines the control objectives that the system needs to meet. This might be to stabilize the system, or to reach a reference point.  $C$  is the set of constraints that should not be violated, examples of these are physical constraints on actuators, constraints on the system states and, safety boundaries.  $U$  is the set of admissible control laws. Thus, the solution to the control problem will be to find a control law,  $u \in U$ .

Faults can change the constraints of the system, as well as the feasible control laws. When a fault occurs, the control problem thus becomes

$$\langle O, C_f, U_f \rangle,$$

where the subscript  $f$  denotes the new constraints and the sets of feasible control laws when a fault has occurred. The fault-tolerant control problem will therefore be to solve  $\langle O, C_f, U_f \rangle$ .

## 2.3 Modeling of a faulty system

In order to solve the fault-tolerant control problem, models defining faults and their impact need to be defined. It is common to classify a fault in one of the three main categories (Blanke et al., 2000):

- Plant faults
- Sensor faults
- Actuator faults

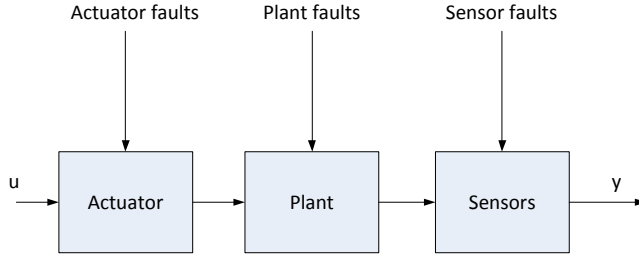


Figure 2.1: Illustration of the different fault types. From Blanke et al. (2006)

A graphical illustration of the fault types are shown in figure 2.1. Plant faults change the dynamics of the system itself. Sensor faults give rise to faulty sensor readings, and actuator faults change how the actuator acts, or the influence of the actuator on the system. It is critical for a controller to be able to handle these faults, as they can otherwise make the system not behave properly, or in the worst case make the system unstable (Mahmoud and Xia, 2014, Chapter 9).

Consider the continuous linear system

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \quad (2.1)$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) \quad (2.2)$$

where  $\mathbf{x} \in \mathbb{R}^n$  is the state vector,  $\mathbf{u} \in \mathbb{R}^m$  is the input vector, and  $\mathbf{A}$ ,  $\mathbf{B}$  and  $\mathbf{C}$  are matrices of appropriate dimensions. The set of possible faults is denoted  $\mathbb{F}$ . A typical way to represent a fault in the system is by changing the system parameters. This means that faults in the process, actuators, and sensors are modeled with new system matrices. When a fault  $f \in \mathbb{F}$  occurs, the new model becomes:

$$\dot{\mathbf{x}}(t) = \mathbf{A}_f\mathbf{x}(t) + \mathbf{B}_f\mathbf{u}(t) \quad (2.3)$$

$$\mathbf{y}(t) = \mathbf{C}_f\mathbf{x}(t) \quad (2.4)$$

where  $\mathbf{A}_f$ ,  $\mathbf{B}_f$ ,  $\mathbf{C}_f$  are the new system matrices after the fault has occurred. Thus we get:

$$\dot{\mathbf{x}}(t) = \begin{cases} \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) & t < t_f \\ \mathbf{A}_f\mathbf{x}(t) + \mathbf{B}_f\mathbf{u}(t) & t \geq t_f \end{cases} \quad (2.5)$$

$$\mathbf{y}(t) = \begin{cases} \mathbf{C}\mathbf{x}(t) & t < t_f \\ \mathbf{C}_f\mathbf{x}(t) & t \geq t_f, \end{cases} \quad (2.6)$$

where  $t_f$  is the time the fault occurs. This representation links directly back to the types of faults that can occur.  $\mathbf{A}_f$  describes the system faults. When a fault

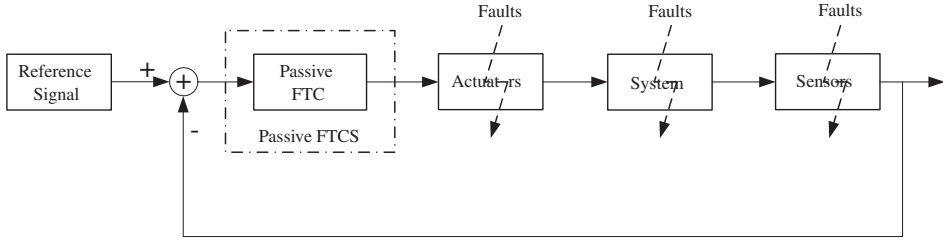


Figure 2.2: Passive fault-tolerant controller. From Jiang and Yu (2012)

happens in the system itself, the dynamics of the system changes, and are represented by the new system matrix.  $\mathbf{B}_f$  is the input matrix, and describes the effect of faults in actuators and how the system responds to inputs when a fault has occurred.  $\mathbf{C}_f$  describes how the states are measured when sensor faults are present. It is important that this system representation is as precise as possible in order for the controller to handle the fault. This relies on an effective fault diagnosis and identification tool, which is described in Chapter 3.

## 2.4 Passive fault-tolerant control

In passive fault-tolerant control, the faults that can occur are considered before the controller is implemented. This means that no fault detection and isolation or on-line modification scheme is needed, and the same controller is always used, even when a fault occurs. Thus the PFTC is passive in the sense that it deals with faults without making any adjustments to the controller. Since the PFTC needs to be able to work for a range of different faults, it is more difficult to make it optimal and its main goal will therefore be stabilization of the system. Figure 2.2 shows an overview of the PFTC. Using a PFTC requires that there is a common solution to the problem for the nominal system and for the system with a fault,  $f \in \mathbb{F}$ . In Figure 2.3, this corresponds to the shadowed area where the admissible solution sets intersect, and the controller will always operate in this region.

## 2.5 Active fault-tolerant control

Active fault-tolerant control uses an FDI scheme to detect and give information about the fault that has occurred. The controller is "active" in the sense that it is modified or changed in real-time based on the information received from the FDI. Thus AFTC consists of three steps to handle faults: (1) Effectively detect and provide information about the fault; (2) modify or change the controller so that it keeps the system stable and keeps it performing on an acceptable level; and (3) implement the new or modified controller. Figure 2.4 shows an overview of the AFTC. A critical part is that the detection, modification and implementation need to happen fast enough to avoid the system entering a state it cannot recover from.

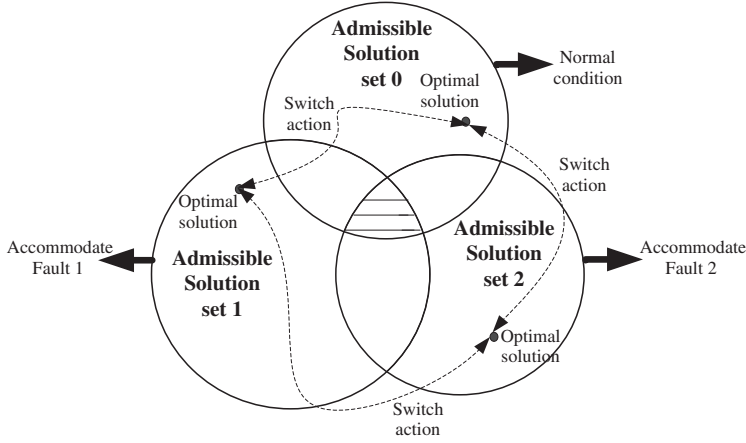


Figure 2.3: Admissible solution sets for active and passive FTC. From Jiang and Yu (2012)

The key issues of AFTC therefore become to use a controller that can easily be modified and having an FDI scheme with high sensitivity to faults, and robustness with regard to model uncertainties and changes in operating conditions. In order for effective fault handling, the modification mechanism also needs to modify the control based on the constraints of the system, and the new constraints introduced by a fault. In some cases, the reference input to the system might also need to be adjusted if the nominal reference is not possible to reach for the faulty system. The dotted line in Figure 2.3 illustrates the control switching to accommodate faults.

There are two different strategies for active fault-tolerant control, *fault accommodation* and *reconfiguration*, which will be investigated in the following sections. Which one is used depends on the accuracy of the data from the FDI.

### 2.5.1 Reconfiguration

If the FDI is not able to give an estimate of the fault magnitude, the new control problem is solved by completely switching off the faulty parts. The controller then tries to achieve the same control objectives as before, but from only using the healthy components. In some cases, the faulty components are replaced by healthy ones. An example of this is when an actuator fault is detected. The new input matrix,  $\mathbf{B}_f$ , is in this case either changed by zeroing the column corresponding to the faulty actuator, or the column is replaced by the dynamics of the new healthy replacement. It is clear that the new input matrix does not need to be estimated by an FDI, as the faulty components are either switched off, or replaced by new ones with known dynamics.

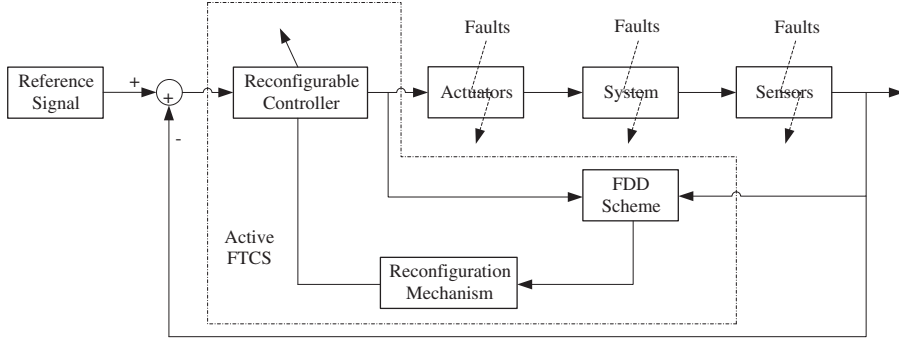


Figure 2.4: Active fault-tolerant controller. From Jiang and Yu (2012)

### 2.5.2 Fault accommodation

When the FDI is able to give an estimate of the fault impact, the fault can be accommodated by changing the controller parameters and constraints such that it achieves the same objectives as from before the fault occurred. An example of this is when an actuator is known to have lost 50% effectivity. The column of the input matrix corresponding to the faulty actuator is multiplied by 0.5, and denoted  $\mathbf{B}_f$ . The control problem with the modified input matrix is then solved. The rest of this text will focus on fault accommodation, where precise information from an FDI is available.

The next section will describe a few of the more commonly used fault-tolerant control methods, with a focus on active schemes.

## 2.6 Fault-tolerant control methods

Many different schemes have been developed for satisfactory control of systems with faults. Zhang and Jiang (2008) gives a review of existing methods, and an overview is shown in Figure 2.5. The goal of this section is not to go into details for all methods, but rather give an overview of the most used fault-handling schemes.

### 2.6.1 Linear quadratic

Linear quadratic control has become a very popular method for control of linear systems. It is fairly well known that this controller employs a feedback control law  $\mathbf{u}(t) = -\mathbf{K}\mathbf{x}(t)$ , where  $\mathbf{K}$  is calculated to minimize the cost function

$$J = \int_0^\infty (\mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{u}^T \mathbf{R} \mathbf{u}) dt. \quad (2.7)$$

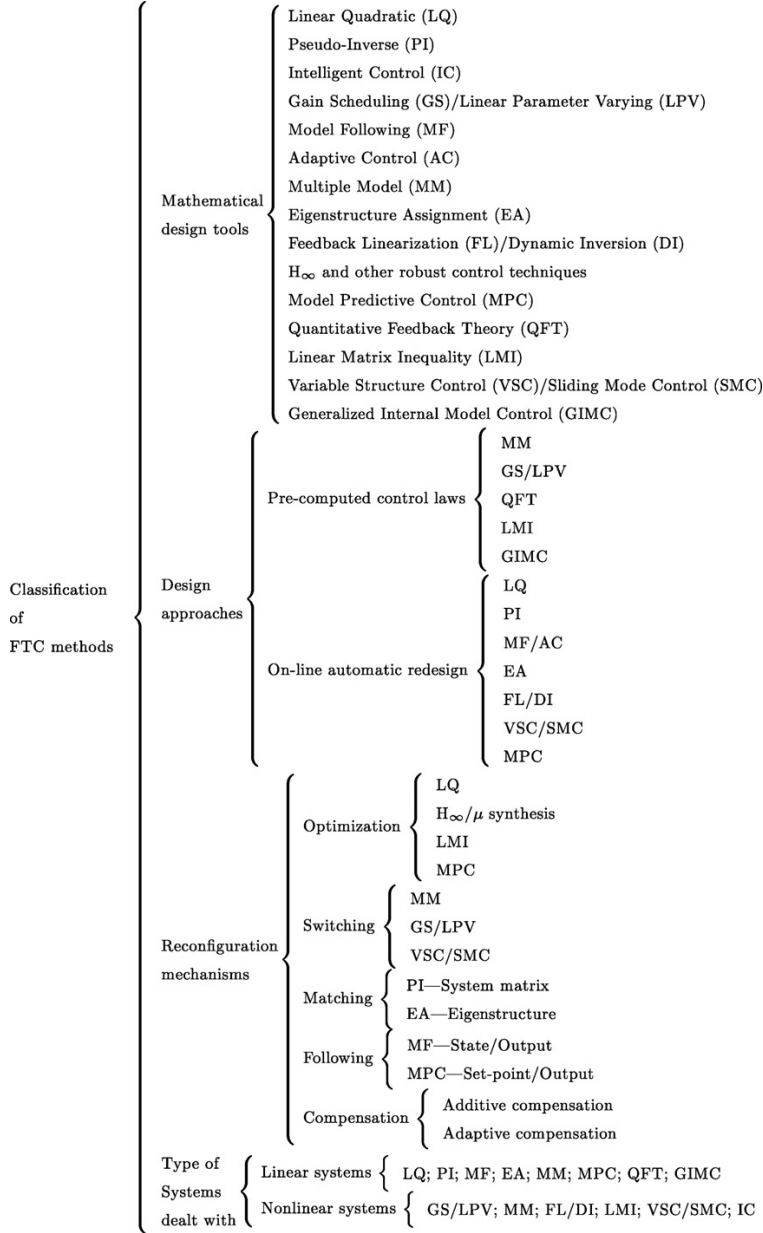


Figure 2.5: Classification of FTC. From Zhang and Jiang (2008)

The solution is given by

$$\mathbf{K} = \mathbf{R}^{-1}\mathbf{B}\mathbf{P}, \quad (2.8)$$

where  $\mathbf{P}$  is found by solving the Riccati equation

$$\mathbf{A}^T\mathbf{P} + \mathbf{P}\mathbf{A} - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{P} + \mathbf{Q} = 0. \quad (2.9)$$

$\mathbf{Q}$  and  $\mathbf{R}$  are diagonal, positive definite weighting matrices.  $\mathbf{A}$  and  $\mathbf{B}$  are the system matrices for the linear system. When a fault,  $f \in \mathbb{F}$  occurs, the new feedback gain  $\mathbf{K}_f$  is calculated with 2.8 and 2.9 by replacing  $\mathbf{A}$  and  $\mathbf{B}$  with  $\mathbf{A}_f$  and  $\mathbf{B}_f$ . The new feedback control law  $\mathbf{u}(t) = -\mathbf{K}_f\mathbf{x}(t)$  will then stabilize the new system given that the pair  $(\mathbf{A}_f, \mathbf{B}_f)$  is controllable. Because of the on-line modification of the controller to accommodate the fault, this is classified as an active fault-tolerant control method. Note that the LQ controller does not take system constraints into account. Constraints can be handled by using model predictive control, which is described in Chapter 4.

## 2.6.2 Pseudo-inverse

An early method for fault-tolerant control, pseudo-inverse, is based on the principle of model-matching design. This is a method that uses the fact that the nominal closed-loop system model is known, and designs a new control law to handle the faulty system. Consider again the continuous linear system 2.1, and a state feedback controller  $\mathbf{u}(t) = -\mathbf{K}\mathbf{x}(t)$  which results in the closed-loop system

$$\dot{\mathbf{x}}(t) = (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x}(t). \quad (2.10)$$

When a fault occurs, the system is modeled as in 2.3. The first goal is to design a control law for the faulty system, so that it behaves in a similar manner as the nominal system when an actuator fault occurs. The new state feedback controller is given as  $\mathbf{u}(t) = -\mathbf{K}_f\mathbf{x}(t)$ . This yields the closed-loop system model:

$$\dot{\mathbf{x}}(t) = (\mathbf{A}_f - \mathbf{B}_f\mathbf{K}_f)\mathbf{x}(t) \quad (2.11)$$

To make the faulty system behave like the nominal one, the new feedback gain,  $\mathbf{K}_f$ , is designed to minimize the difference

$$\|(\mathbf{A} - \mathbf{B}\mathbf{K}) - (\mathbf{A}_f - \mathbf{B}_f\mathbf{K}_f)\|. \quad (2.12)$$

This solves for

$$\mathbf{K}_f = \mathbf{B}_f^+(\mathbf{A}_f - \mathbf{A} + \mathbf{B}\mathbf{K}) = (\mathbf{B}_f^T\mathbf{B}_f)^{-1}\mathbf{B}_f^T(\mathbf{A}_f - \mathbf{A} + \mathbf{B}\mathbf{K}) \quad (2.13)$$

where  $\mathbf{B}_f^+$  is the pseudo-inverse of  $\mathbf{B}_f$ . This new control law is activated when a fault is detected and the FDI has provided information and magnitude of the fault. It minimizes the difference between the faulty and the nominal system's behaviour.



### 2.6.3 Adaptive control

Adaptive controllers automatically adjust the control parameters in order to achieve the desired performance. There are two approaches in adaptive control, direct and indirect. In the indirect approach, model parameters are first estimated and used in the controller in order to effectively track a reference. For a linear system, the pair  $(\mathbf{A}, \mathbf{B})$  is estimated. In the case of changes in operating conditions, e.g. faults, the new system parameters are automatically estimated and used in the controller. In the direct approach, the system parameters are not explicitly estimated, but control parameters are chosen on-line based on the response of the system to a given input. Model-reference adaptive control (MRAC) is a popular methodology (both direct and indirect), where the controller is adapted on-line in order for the system to track a reference trajectory by making the tracking error converge to zero. Therefore, in the case of faults that change the system parameters, the controller is automatically adapted and able to track the reference. However, it is important to note that adaptive control often requires certain assumptions about the system, which may no longer hold if a fault occurs. Another problem with adaptive control for fault-tolerance is the fact that these types of controllers require the system parameters to be slowly varying. For abrupt, severe faults, this will not be the case and the adaptive controllers are not effective. For a more detailed explanation of MRAC, the reader is referred to Ioannou and Sun (1996).

### 2.6.4 Robust control ( $H_\infty$ control)

The controllers listed so far have been using active fault-tolerant approaches, where the fault is estimated on-line, and the controller is modified to accommodate the fault. Another approach to incorporate fault-tolerance is to use the passive  $H_\infty$  controller.  $H_\infty$  is one of the most popular robust control methods, and has proved to inherit strong robustness properties. The basic concept of this controller is to minimize the infinity norm of the transfer function for the closed-loop system, while including model uncertainties and unknown disturbances. Because of the strong robustness properties, it proves a good candidate for passive fault-tolerant control, where it can be designed to guarantee stability for a range of different potential faults.

The main advantage of this type of controller is that it does not require information about faults on-line, and can therefore deal with faults without the need of an accurate FDI. However, because it needs to be able to deal with all potential faults, it is fairly conservative for nominal operating conditions. In Skogestad and Postlethwaite (2005),  $H_\infty$  design is covered in great detail.



## Chapter 3

# Fault detection and isolation

An important step in fault-tolerant control is to quickly identify where a fault has occurred, as well as its severity. To do this, a fault detection and isolation scheme is used. The different FDI methods can be divided in two different parts: Model-based and data-based (model-free) methods (Zhang and Jiang, 2008). Figure 3.1 gives an overview of the methods that can be implemented for the different categories.

The majority of research in the FDI area are for monitoring or diagnostics purposes (Zhang and Jiang, 2008), and not all FDI schemes might be suitable in the context of fault-tolerant control. Since most controllers are model-based, the focus of this text will be on quantitative model-based approaches, which utilizes a mathematical model to carry out FDI on-line.

Blanke et al. (2006) classifies the diagnostic steps as follows:

**Fault detection:**

Decide whether or not a fault has occurred. This step determines the time at which the system is subject to some fault.

**Fault isolation:**

Find in which component the fault has occurred. This step determines the location of the fault.

**Fault identification and fault estimation:**

Identify the fault and estimate its magnitude. This step determines the kind of fault and its severity.

The following sections give a brief introduction to the concept of FDI, and present some of the more commonly used methods.

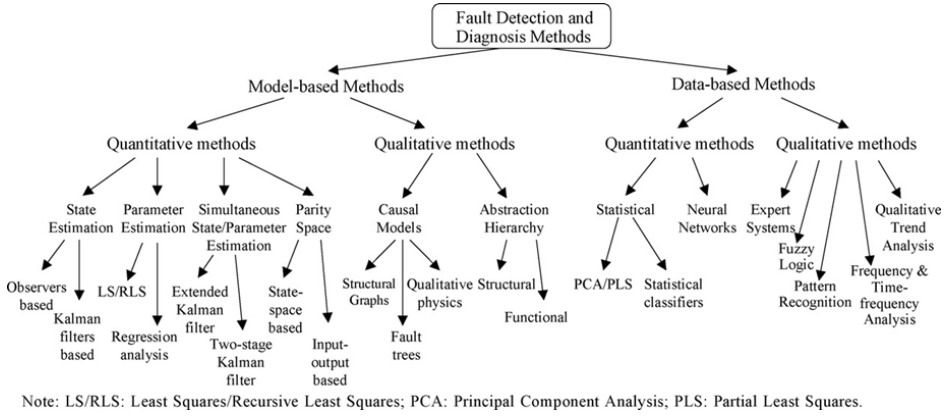


Figure 3.1: Classification of the FDI methods. From Zhang and Jiang (2008)

## 3.1 Fault detection and isolation concept

### 3.1.1 Residual generation

The first step of handling a fault is to recognize that it is there. In order to detect a fault, it is common to check the consistency of the relationship between the input and output signals compared to a reference model. Consider a dynamical system with input  $u$  and output  $y$  subject to some fault  $f$ . The behaviour of the system will depend on the fault that has occurred. The diagnostic system uses the pair

$$U = (u(0), u(1), u(2), \dots, u(N)) \quad (3.1)$$

$$Y = (y(0), y(1), y(2), \dots, y(N)), \quad (3.2)$$

where  $N$  is a given time horizon, to determine on-line if a fault has occurred. If the response of the system does not match the reference model, the fault detection scheme will conclude that a fault has occurred.

The main concept of model-based fault detection is residual generation, which is defined as a fault indicator based on a deviation between measurements and model-equation based computations (Isermann, 2006). The residual is calculated at every time step, by

$$r(t) = y(t) - \hat{y}(t) \quad (3.3)$$

In a faultless case, the residual vanishes or is close to zero. A non-vanishing residual indicates the existence of a fault. Figure 3.2 illustrates the residual generation concept.

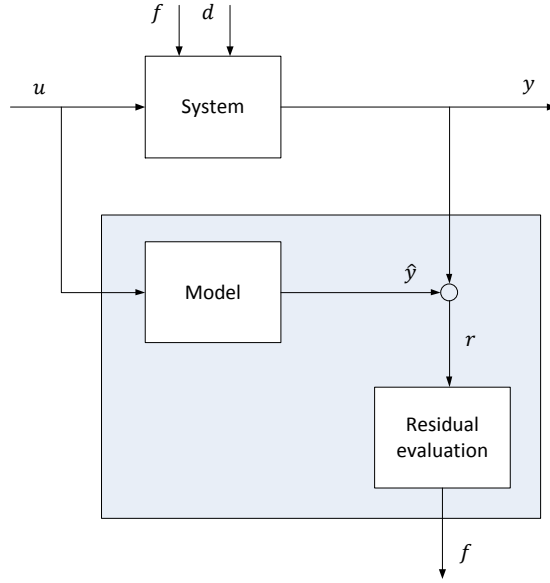


Figure 3.2: Residual generation concept. From Blanke et al. (2006)

### 3.1.2 Residual evaluation

When the residual is generated, the next step is to evaluate if its value indicates a fault. For a perfectly modeled system without disturbances, this will simply be to check if it is zero. However, potential disturbances and model uncertainties will affect the residual value, and therefore need to be taken into account when evaluating whether a fault has occurred. It is also important to note that in order to be able to isolate a fault, a high residual value is not enough. Modeling of each potential fault before the system is set in action is needed to pinpoint exactly where the fault has occurred. The goal is to identify deviations from the nominal plant, which is not possible without a list of faults (Blanke et al., 2006, Ch 1). The basic concept of residual evaluation is illustrated in Figure 3.3.

## 3.2 Fault detection and isolation methods

Methods commonly used in FDI to generate residuals are diagnostic observers, parity relations and Kalman filters (Venkatasubramanian et al. (2003)). The following sections give a brief explanation of these methods. As in Section 2.6, the goal is not to go into detail about any of the methods, but rather give an overview of different approaches that can be used. References to more detailed articles are included.

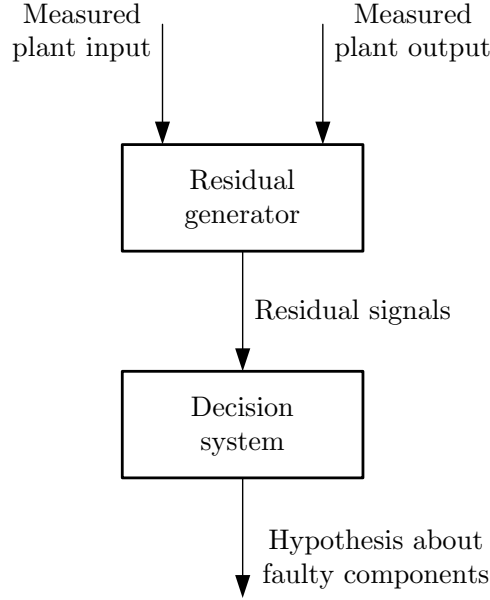


Figure 3.3: Residual evaluation concept. From Blanke et al. (2006)

### 3.2.1 Diagnostic observers

The basic idea of diagnostic observers is to develop a set of observers where each one is sensitive to a subset of potential faults and insensitive to the remaining faults. In the fault-free case, all observers track the process with a low residual value which is only affected by unknown inputs such as noise. When a fault occurs, the observers that are made insensitive to the fault will continue to track the process with a low residual. The other observers, however, will have large errors with residuals of significant values. The set of observers are designed such that a given fault results in a distinct residual pattern, which makes fault isolation possible. In Frank (1994), the method is explained in detail, where the authors give a review of the diagnostic observer for non-linear systems.

### 3.2.2 Parity relations

Parity equations are found by rearranging the state-space models of the plant, and fault isolation using this method requires more sensors than states. The model structure is rearranged to get the best possible fault isolation, and by having more sensors than states, there is more freedom to the design of residual generating equations. In order for fault isolation, this redundancy is exploited and residual vectors that are orthogonal to each other for different faults are generated (Venkata-subramanian et al., 2003). Thus by having redundant sensors, fault detection and isolation is implemented by algebraic relations. The derivation of the residual using

parity relations is shown in Patton and Chen (1994).

### **3.2.3 Kalman filters**

The Kalman filter is a recursive algorithm for state estimation, and is one of the most commonly applied methods in process systems. If the statistical parameters for the disturbances to the system are known, the Kalman filter will provide an estimate with minimal estimation error (Venkatasubramanian et al., 2003). For fault diagnosis, the Kalman filter is designed on the basis of nominal operating conditions for the plant. If the differences between the Kalman estimates and the measured values are large, a fault is present. Fault isolation can be implemented by using a bank of Kalman filters, where each filter is designed on the basis of a certain set of faults. When a fault occurs, only the Kalman filter corresponding to this fault will have a large residual. Xue et al. (2007) gives an example of the use of a bank of Kalman filters for fault detection and isolation for an aircraft with potential sensor and actuator faults.





## Chapter 4

# Fault-tolerant Model Predictive Control

Model predictive control, is one of the most commonly used control algorithms for multi-variable control problems. Mellichamp et al. (2010) summarizes the MPC concept as follows: A multiple input, multiple output system is to be controlled while satisfying constraints on the input and output variables. The algorithm uses a model of the system to predict the future states. At each time step, an optimization problem is solved that takes future events into account and calculates a sequence of control moves to take the system from the current state to the reference state while minimizing some cost function. The first control move is then applied to the system, and the whole problem is recomputed in the next time step. The basic concept is illustrated in figure 4.1. The main reason for the popularity of the MPC controller is its effectiveness in handling large multi-variable systems with constraints on both inputs and states.

As mentioned Section 1.4, model predictive control allows for easy ways to incorporate fault-tolerance in the controller. Because the optimization problem incorporates the system model as well as the constraints when calculating a control input, the controller can easily be modified on-line if a fault in the system is known. This can be done by either changing the system model to match the new faulty system, or the input constraints in the case of actuator faults. When a fault and its magnitude is known, updating the MPC formulation to accommodate the fault is relatively straight forward (Maciejowski, 1997). It has also been shown that MPC inherits some implicit fault-tolerance for over-actuated systems (Maciejowski, 1998).

This chapter begins by describing the basic components of the MPC that are necessary to describe for the implemented fault-tolerant MPC.

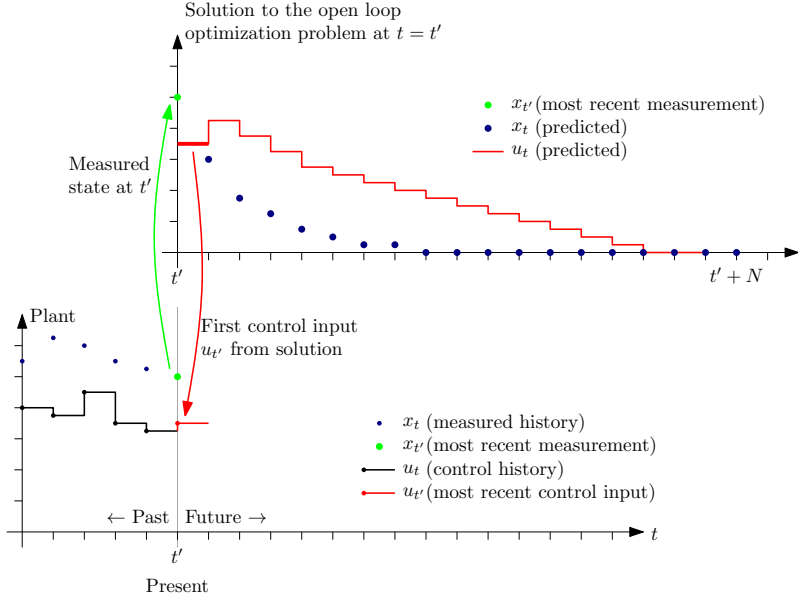


Figure 4.1: The principle of MPC. From Foss and Heirung (2013)

## 4.1 Model Predictive Control

### 4.1.1 Cost function

The cost function is a major component in MPC theory. This is a function that is minimized, subject to some constraints. In the most general case where the controller is designed to track a certain reference, the cost function is chosen to penalize the difference between the current and the desired state. The most commonly used cost function is the quadratic:

$$J = \sum_{t=0}^{N-1} \ell(\mathbf{x}_{t+1}, \mathbf{u}_t) = \sum_{t=0}^{N-1} (\mathbf{x}_{t+1}^T \mathbf{Q} \mathbf{x}_{t+1} + \mathbf{u}_t^T \mathbf{R} \mathbf{u}_t), \quad (4.1)$$

where  $N$  is the prediction horizon, and  $\mathbf{Q}$ ,  $\mathbf{R}$  are positive real, symmetric, weighting matrices. These matrices describe the relative importance of their respective signals. Note that in this definition of the cost function, it is assumed that all states are correctly measured. This is often not the case, and the system states need to be estimated, these estimates are then used in this optimization problem. *The rest of this text will assume that there are exact measurements of all states available.*

Additionally, a penalty of the control input change can be added to 4.1:

$$\Delta \mathbf{u}_t^T \mathbf{P} \Delta \mathbf{u}_t, \quad (4.2)$$

where  $\mathbf{P}$  is a positive real, symmetric, weighting matrix. The rest of this text will focus on the cost function defined in 4.1, without the control input change penalty. An economic cost function approach is described in Section 4.1.5.

### 4.1.2 Constraints

As described in the previous sections, the controller will drive the system to the states that minimize the cost function, while satisfying some constraints. These are typically constraints on the inputs to the system, as well as constraints on the states. Additionally, in order for the controller to design state-trajectories that are possible for a given system, the system model needs to be included as a constraint to the optimization problem. This gives rise to the following set of constraints

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) \quad (4.3a)$$

$$\mathbf{x}_{min} \leq \mathbf{x}_t \leq \mathbf{x}_{max} \quad (4.3b)$$

$$\mathbf{u}_{min} \leq \mathbf{u}_t \leq \mathbf{u}_{max}, \quad (4.3c)$$

$$-\Delta \mathbf{u}_{min} \leq \Delta \mathbf{u}_t \leq \Delta \mathbf{u}_{max}, \quad (4.3d)$$

These can be hard constraints, such as physical limitations on the system and actuators. Examples of these are the maximum volume of a tank, or the capacity of a pump. It is clear that it is impossible for these to be violated. Constraints can also be soft, this means that they can be violated for short periods of time, but these violations are punished in the cost function. Obviously, constraints that are linked to physical limitations of a system can not be softened. However, many constraints represent *desired* operating regions for a system, which are physically possible to violate. The softening of such constraints is described in the next section. The constraints on states and inputs define an operating window for the process (Mellichamp et al., 2010, Ch 20).

For stability analysis, it is also common to include a terminal constraint of the form

$$\mathbf{x}_N \in \mathcal{X}^N, \quad (4.4)$$

where  $\mathcal{X}^N$  is a set in which the states should lie at the end of the horizon, which is elaborated on in Section 4.1.4.

### 4.1.3 Softening the constraints

A problem with hard constraints in the optimization problem is that problems can become infeasible if the constraints are not satisfied in the beginning of the optimization problem (Maciejowski, 2002, Ch 3). In a physical system, this might be due to noise, or a change in operating conditions. If the system states are pushed outside of the feasible region, or the feasible region suddenly changes, the optimization problem will become infeasible and the optimization algorithm will give up with an "infeasible problem" error message. This is clearly not acceptable

for an on-line controller, as it will lose control of the system. To handle this infeasibility, the constraints can be softened for a period of time, and the violation of the constraints is penalized in the cost function. This is done by adding "slack variables" to the constraints that need to be softened, and penalize their values in the cost function. The controller will then push the system into a region where the constraints are not violated. As noted earlier, this kind of constraint softening is not possible if the constraints are related to hard, physical limitations. Given the constraint

$$\mathbf{x}_{min} \leq \mathbf{x}_t \leq \mathbf{x}_{max}$$

If the system finds itself outside the feasible region of  $\mathbf{x}$ , the optimization problem will be infeasible. A slack variable  $\epsilon$  is then added to the constraint, and the violation is penalized in the cost function. The new, softened constraint thus becomes:

$$\mathbf{x}_{min} - \epsilon_t \leq \mathbf{x}_t \leq \mathbf{x}_{max} + \epsilon_t \quad (4.5)$$

and the cost function

$$J = \sum_{t=0}^{N-1} \ell(\mathbf{x}_{t+1}, \mathbf{u}_t) + \epsilon_t^T \rho_2 \epsilon_t + \rho_1 \|\epsilon_t\|_1, \quad (4.6)$$

where  $\ell(\mathbf{x}_{t+1}, \mathbf{u}_t)$  is the original cost function,  $\epsilon$  is a nonnegative vector with the same dimension as  $\mathbf{x}$ , and  $\rho_1 \geq 0$ ,  $\rho_2 \geq 0$  are weighting constants for the slack variable.

Equation 4.6 includes both a linear ( $l_1$  norm) and a quadratic penalty term. This is not necessary in all cases. However, only using the quadratic term can lead to the cost function finding an optimal point outside the original "hard constrained" region. When using the linear term, it can be shown that with large enough  $\rho_1$ , the constraint violations will not occur unless there is no feasible solution to the original hard problem (Maciejowski, 2002, Ch 3). Thus, if the hard problem is infeasible, the controller will push the system into the feasible region so that  $\epsilon$  becomes zero. This is known as exact penalty functions.

A well known result of the exact value of  $\rho_1$ , is that it should be larger than the corresponding Lagrange multiplier (Kerrigan and Maciejowski, 2000b).

#### 4.1.4 MPC principle

Say the goal is to drive the system

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) \quad (4.7)$$

from the state  $\mathbf{x}_0$  to the state  $\mathbf{x}_r$ , while satisfying the constraints 4.3.

This can be done by replacing  $\tilde{\mathbf{x}}_{t+1} = \mathbf{x}_{t+1} - \mathbf{x}_r$  in the cost function 4.1. The MPC will then drive the system as close to the reference state as possible. Based on this, the optimization problem that is solved at every time step becomes

$$\min_{\mathbf{z}} \quad J = \sum_{t=0}^{N-1} \left( \tilde{\mathbf{x}}_{t+1}^T \mathbf{Q} \tilde{\mathbf{x}}_{t+1} + \mathbf{u}_t^T \mathbf{R} \mathbf{u}_t \right) \quad (4.8a)$$

$$\text{s.t.} \quad \mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t), \quad \forall t \in \{0, \dots, N-1\} \quad (4.8b)$$

$$\mathbf{x}_0 = \mathbf{x}_{init}, \quad (4.8c)$$

$$\mathbf{x}_{min} \leq \mathbf{x}_t \leq \mathbf{x}_{max}, \quad \forall t \in \{0, \dots, N\} \quad (4.8d)$$

$$\mathbf{u}_{min} \leq \mathbf{u}_t \leq \mathbf{u}_{max}, \quad \forall t \in \{0, \dots, N-1\} \quad (4.8e)$$

$$-\Delta \mathbf{u}_{min} \leq \Delta \mathbf{u}_t \leq \Delta \mathbf{u}_{max}, \quad \forall t \in \{0, \dots, N-1\} \quad (4.8f)$$

$$\mathbf{x}_N \in \mathcal{X}^N \quad (4.8g)$$

Where  $\mathbf{z} = (\mathbf{x}_1^T, \dots, \mathbf{x}_N^T, \mathbf{u}_0^T, \dots, \mathbf{u}_{N-1}^T)^T$ . The current state is used as the initial state, and the first control move is implemented. The optimization problem is then recalculated at the next time step. If the system is linear and the cost function quadratic, the optimization problem is convex and solution methods are known to solve the problem quickly and reliably (Maciejowski, 2002).

**Definition 1** (Positively invariant set (Blanchini, 1999)). *The non-empty set  $\Omega \subset \mathbb{R}^n$  is positively invariant for the autonomous system  $\mathbf{x}_{t+1} = f(\mathbf{x}_t)$  if and only if  $\forall \mathbf{x}_0 \in \Omega$ , the system evolution satisfies  $\mathbf{x}_t \in \Omega$ ,  $\forall t \in \mathbb{N}_+$ .*

A closed-loop positively invariant set is therefore a set which satisfies Definition 1 for a closed-loop system. When  $\mathcal{X}^N$  is chosen to be a closed-loop positively invariant set for the controller, and  $(\mathbf{A}, \mathbf{Q}^{\frac{1}{2}})$  is detectable, feasibility of the optimization problem with a quadratic cost will guarantee stability (Mayne et al. (2000)). *This is assumed to hold true for the rest of this report.*

For the economic MPC described in the next section, similar criteria exist. These are also assumed to hold true. In Rawlings et al. (2012), criteria for stability of economic MPC is elaborated on.

### 4.1.5 Economic MPC

The standard way of maximizing profit in process systems is to have an external management system often referred to as real-time optimization (RTO), to calculate the desired set-points. These set-points are sent to a reference tracking MPC, which drives the system to the desired set-points. However, for some applications this hierarchical separation might not be optimal nor desirable. Recent studies show that much can be gained by formulating the economic cost function directly into the MPC (Rawlings et al., 2012). In economic model predictive control (EMPC), the quadratic cost function is replaced by an economic cost function, so that the set-points that maximize profit are automatically calculated by the MPC. Figure

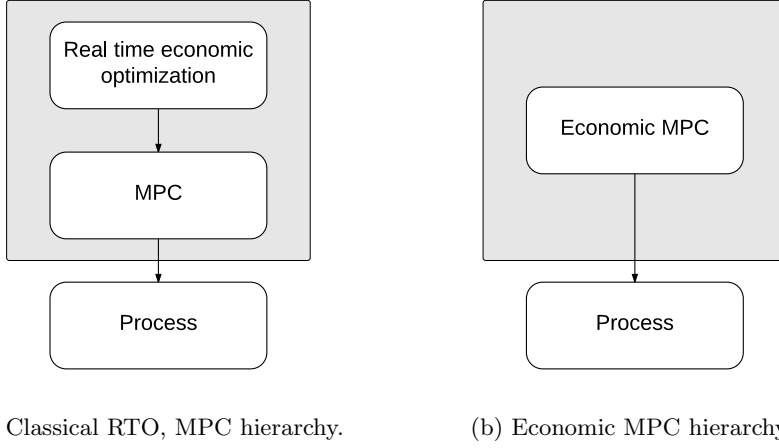


Figure 4.2: Figure showing the difference between the EMPC, and RTO-MPC hierarchies.

4.2 shows the difference between the standard scheme, and the EMPC. Say the economic profit over a time horizon with length  $N$  is given as,

$$- \sum_{t=0}^{N-1} \ell_e(\mathbf{x}_{t+1}, \mathbf{u}_t) \quad (4.9)$$

and the goal is to maximize this profit, which will be the same as minimizing the negative. The system is given as in Equation 4.7, and the constraints are the same as in Equation 4.3. Instead of formulating the optimization problem as in Problem 4.8, the economic function is used as the objective function. The economic MPC problem can therefore be formulated as follows:

$$\min_{\mathbf{z}} \quad J = \sum_{t=0}^{N-1} \ell_e(\mathbf{x}_{t+1}, \mathbf{u}_t) \quad (4.10a)$$

$$\text{s.t.} \quad \mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t), \quad \forall t \in \{0, \dots, N-1\} \quad (4.10b)$$

$$\mathbf{x}_0 = \mathbf{x}_{init}, \quad (4.10c)$$

$$\mathbf{x}_{min} \leq \mathbf{x}_t \leq \mathbf{x}_{max}, \quad \forall t \in \{0, \dots, N\} \quad (4.10d)$$

$$\mathbf{u}_{min} \leq \mathbf{u}_t \leq \mathbf{u}_{max}, \quad \forall t \in \{0, \dots, N-1\} \quad (4.10e)$$

$$-\Delta \mathbf{u}_{min} \leq \Delta \mathbf{u}_t \leq \Delta \mathbf{u}_{max}, \quad \forall t \in \{0, \dots, N-1\} \quad (4.10f)$$

$$\mathbf{x}_N \in \mathcal{X}^N \quad (4.10g)$$

The controller will then drive the system to the states that optimize the economic profit. Several papers discuss the different uses of EMPC, including stability issues and feasibility, including Rawlings et al. (2012), Bø and Johansen (2014), Amrit et al. (2013), among others.

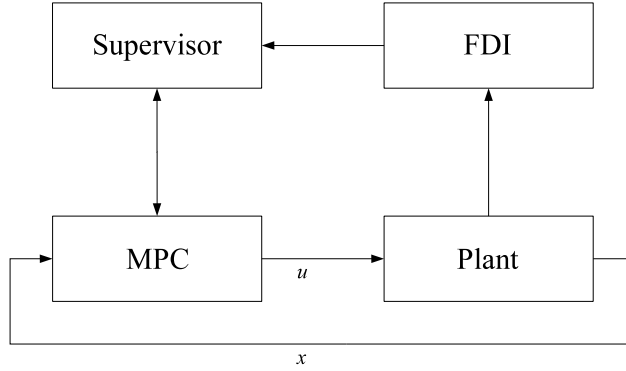


Figure 4.3: Fault-tolerant MPC. Edited from Ocampo-Martínez et al. (2005)

## 4.2 Incorporating fault-tolerance in MPC

In the following sections, methods to accommodate the different fault types listed in Section 2.3 using MPC, is described. The representations have strong correlations with actual physical faults that can occur in systems, e.g. stuck valves, broken sensors and blocked pipes. Figure 4.3 shows the principle of fault-tolerant model predictive control (FTMPC). The FDI detects and estimates the fault magnitude, the information is sent to a supervisor which then modifies the MPC to accommodate the fault.

### 4.2.1 Actuator faults

A way to handle actuator faults in the MPC scheme is to modify the constraints in the optimization problem based on the information about the fault. Consider the input constraints from Equation 4.3, repeated here for convenience:

$$\begin{aligned} \mathbf{u}_{min} &\leq \mathbf{u}_t \leq \mathbf{u}_{max} \\ -\Delta \mathbf{u}_{min} &\leq \Delta \mathbf{u}_t \leq \Delta \mathbf{u}_{max}. \end{aligned}$$

Inspired by Miksch et al. (2008), the different actuator faults can be classified and modeled by change of constraints in the following way:

#### Actuator region decrease

$$\mathbf{u}_{min}^f \leq \mathbf{u}_t \leq \mathbf{u}_{max}^f$$

#### Actuator freezing

$u_j = \text{constant}$ , where the subscript denotes actuator  $j$ .

#### Control-rate decrease

$$-\Delta \mathbf{u}_{min}^f \leq \Delta \mathbf{u}_t \leq \Delta \mathbf{u}_{max}^f.$$

**Loss of actuator**

$u_j = 0$ . This fault could also be reflected by zeroing the corresponding column in the  $\mathbf{B}$  matrix.

As described in Chapter 2.3, a reduction in actuator effectivity can be modeled by a change in the input matrix of the system. For a linear system, this will be a change in  $\mathbf{B}$ . By updating the constraints and system model based on the fault that occurs, the MPC will distribute the control in a way that handles the fault Maciejowski (1997).

**4.2.2 System faults**

System faults are faults that change the dynamics of the system. As described in Chapter 2.3, this is modeled by changing the system matrix. For a linear system, this will mean changing  $\mathbf{A}$  to  $\mathbf{A}_f$ . When a fault occurs, it is important that  $(\mathbf{A}_f, \mathbf{B}_f)$  is controllable, or at least stabilizable, in order for a new control law to be found. The control problem then becomes to change the tuning parameters, constraints, and internal model of the MPC in order for the faulty system to behave like the nominal system. In Huzmezan and Maciejowski (1999), a solution to choosing new tuning parameters was proposed by matching the eigenvalues of the closed-loop faulty system to the nominal one through a nonlinear minimization with the predicted weighting matrix  $\mathbf{Q}$  as decision variables. However, this requires a high amount of computational resources, and so this problem is still to be solved effectively. For the rest of this text the tuning parameters will not be changed after a fault occurs, and the focus will be on modifying constraints and the internal model to accommodate faults.

**4.2.3 Sensor faults**

Sensor faults are potentially the most difficult to deal with from the view point of correcting them within the MPC framework (Maciejowski, 1999). There is not much information available in the literature about handling sensor faults in the MPC. Since this controller uses the current states (or state estimates) as the initial condition for the system in each iteration, the whole prediction sequence will be wrong if the state values/measurements are incorrect. If the measurements that are subject to sensor faults are available elsewhere, e.g. from an observer, Kalman filter, or redundant sensors, the original system performance could be recovered. This, however, would be a matter of changing the estimator algorithms to accommodate the fault instead of using the MPC directly. Maciejowski (1999) notes that there might be ways of changing the cost function, or substituting the control of another variable for the unavailable one in order to achieve satisfactory performance. These methods have not been well studied in the literature, and the focus of this text will therefore be on system and actuator faults.



#### 4.2.4 Reconfiguring the MPC

Reformulating the standard MPC model 4.8 with the fault information the MPC has been provided, changes the optimization problem to

$$\min_{\mathbf{z}} \quad J = \sum_{t=0}^{N-1} \ell(\mathbf{x}_{t+1}, \mathbf{u}_t) \quad (4.11a)$$

$$\text{s.t.} \quad \mathbf{x}_{t+1} = f_f(\mathbf{x}_t, \mathbf{u}_t), \quad \forall t \in \{0, \dots, N-1\} \quad (4.11b)$$

$$x_0 = x_{init}, \quad (4.11c)$$

$$\mathbf{x}_{min}^f \leq \mathbf{x}_t \leq \mathbf{x}_{max}^f, \quad \forall t \in \{0, \dots, N\} \quad (4.11d)$$

$$\mathbf{u}_{min}^f \leq \mathbf{u}_t \leq \mathbf{u}_{max}^f, \quad \forall t \in \{0, \dots, N-1\} \quad (4.11e)$$

$$-\Delta \mathbf{u}_{min}^f \leq \Delta \mathbf{u}_t \leq \Delta \mathbf{u}_{max}^f, \quad \forall t \in \{0, \dots, N-1\} \quad (4.11f)$$

$$\mathbf{x}_N \in \mathcal{X}_f^N, \quad (4.11g)$$

where the sub- and superscript  $f$  denotes the values when a fault has occurred. Note that for the parts of the system that are not influenced by the fault, these values will be the same as for nominal operation. When the MPC receives information about a fault and is reconfigured on-line, it will automatically change its inputs to the system to reach the objective. Numerical examples of this type of on-line reconfiguration is shown in Huzmezan and Maciejowski (1999), Maciejowski and Jones (2003), Ocampo-Martínez et al. (2005) among others. Chapter 5.1 of this text also considers a numerical example of the fault-tolerant MPC.

It is clear that the MPC is suitable for fault-tolerant control. The fault-tolerant control problem of chapter 2.2 can elegantly be included in the MPC framework by modifying the constraints and objective function.

### 4.3 Fault-tolerant MPC feasibility

As described in Section 4.1.3, optimization problems can become infeasible when the operating conditions are changed. System and actuator faults can often contribute to large and instant changes in the operating conditions, and may therefore cause infeasibility problems. It is critical that the controller can handle these types of events, as they can render the system unstable if no action is taken. A common way to handle infeasibility issues is by the use of soft constraints, as described in Section 4.1.3. There are several approaches in the literature to effectively recover the system from these complicating states, including Vada et al. (2001), Kerrigan and Maciejowski (2000b), Hovd (2011), among others. However, a problem arises when some of the constraints that need to be softened are actually hard constraints. This means that these constraints either are physically defined constraints on the system or defines *hard* safety constraints that would require a shut-down of the plant if violated.

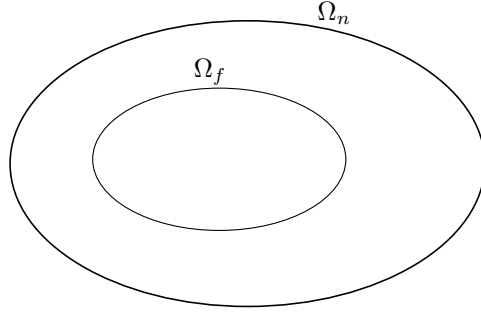


Figure 4.4: Feasible regions  $\Omega_n$  and  $\Omega_f$ , for nominal, and fault operation respectively.

An example of the aforementioned feasibility issue is a sudden dropout of one of the actuators in the system. Due to the (nominal) bounds on the remaining actuators, the controller may no longer be able to control the system from the state the system is in when the fault occurs. Hence, the feasible region for the optimization problem will often cases be reduced. This situation is illustrated in Figure 4.4. *Note that stability and feasibility is used interchangeably because of the inclusion of the terminal constraint 4.4 in the MPC formulation 4.8, and the assumptions made in Section 4.1.4. The feasible set is therefore the set in which the MPC can control the system, and this will change for different input constraints.*

To guarantee that the system is still controllable when a fault occurs, a requirement is that the system's nominal operating states lie in the feasible set for a potential fault. Let

$$\Omega_n = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{G}\mathbf{x} \leq \mathbf{h}\}, \quad (4.12)$$

and

$$\Omega_f = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{G}_f\mathbf{x} \leq \mathbf{h}_f\} \quad (4.13)$$

denote the feasible region for the MPC with nominal operating conditions and when a fault,  $f$ , has occurred, respectively. In order to guarantee feasibility and stability for the fault, the system needs to operate in  $\Omega_n \cap \Omega_f$ . When the fault then hits the system, the controller can accommodate the fault. This is, however, a rather conservative approach. It sets a tight operating window for the plant that might be unnecessary in nominal operation. If information of an incipient fault can be retrieved, an alternative approach is to let the plant operate outside of this region, but drive the system back in to  $\Omega_f$  to ensure stability in the event of an actuator fault in the system. This approach is called proactive fault-tolerant control, and is elaborated in the next section.

## 4.4 Proactive fault-tolerant model predictive control

State and control constraints can be satisfied if and only if the initial state belongs to a positively invariant set for the closed-loop system (Kerrigan and Maciejowski, 2000a). Methods to calculate these types of positively invariant sets can therefore be used to define operating regions for the MPC. In order to guarantee feasibility for a fault, a closed-loop positively invariant set can be calculated for the different faults, and then ensure that the system operates within this set. Approaches for computing these positively invariant sets can be found in Kerrigan and Maciejowski (2000a). Bø and Johansen (2014) describe an approach to quickly drive the system into a "safe" set where the MPC problem is feasible for a set of potential faults. However, always operating in this "safe" set is a rather conservative approach. If knowledge about when a fault will occur is available from an FDI, or statistical/historical data, the system can operate outside this set and the controller can drive the system back in before the fault occurs. Lao et al. (2013) suggests a proactive approach to deal with known incipient faults by using a Lyapunov-based MPC to drive the system into a region where a certain feedback controller can guarantee stability for the faulty system.

This section focuses on using *soft constraints* and a *penalty function* to drive the system into a positively closed-loop invariant set for the incipient fault. Consider a system that under nominal conditions operate in the feasible set defined by 4.12, and a known incipient fault will change the feasible set to 4.13. The goal is then, before the fault occurs, to drive the system into the set defined by 4.13. Note that it is critical for the system to enter this set before the fault occurs, as it will otherwise render the system unstable. When the MPC receives information about an incipient fault, the following constraint

$$\mathbf{G}_f \mathbf{x}_t \leq \mathbf{h}_f + \epsilon_t, \quad (4.14)$$

and the cost

$$\rho \epsilon_t, \quad (4.15)$$

$\rho > 0, \epsilon_t \geq 0$ , are added to the optimization problem, which then becomes

$$\min_{\mathbf{z}} \quad J = \sum_{t=0}^{N-1} \ell(\mathbf{x}_{t+1}, \mathbf{u}_t) + \rho \|\epsilon_t\|_1 \quad (4.16a)$$

$$\text{s.t.} \quad \mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t), \quad \forall t \in \{0, \dots, N-1\} \quad (4.16b)$$

$$\mathbf{x}_0 = \mathbf{x}_{init}, \quad (4.16c)$$

$$\mathbf{x}_{min} \leq \mathbf{x}_t \leq \mathbf{x}_{max}, \quad \forall t \in \{0, \dots, N\} \quad (4.16d)$$

$$\mathbf{u}_{min} \leq \mathbf{u}_t \leq \mathbf{u}_{max}, \quad \forall t \in \{0, \dots, N-1\} \quad (4.16e)$$

$$-\Delta \mathbf{u}_{min} \leq \Delta \mathbf{u}_t \leq \Delta \mathbf{u}_{max}, \quad \forall t \in \{0, \dots, N-1\} \quad (4.16f)$$

$$\mathbf{G}_f \mathbf{x}_t \leq \mathbf{h}_f + \epsilon_t, \quad \forall t \in \{0, \dots, N\} \quad (4.16g)$$

$$\epsilon_t \geq 0, \quad \forall t \in \{0, \dots, N\} \quad (4.16h)$$

Due to the use of a linear penalty function, the system will be driven into the set defined by 4.13 if  $\rho$  is chosen large enough, as explained in Section 4.1.3. The use of the linear penalty is an important part of this approach, as a quadratic penalty would not guarantee the system entering the set defined by 4.13. Note that the terminal constraint 4.4 has been replaced by the soft constraint. The impact of this approach is illustrated in a numerical example in Section 5.2. This can be implemented by setting  $\rho = 0$  before information about the incipient fault is received, and then choosing a value that will guarantee exactness of the penalty function when the FDI as provided information about the upcoming fault.

By assuming the FDI or historical data can provide information about an incipient fault, one can therefore design the controller such that the system is allowed to operate outside the feasible region defined by 4.13, and thereby allow for improved economical operations, while driving the system into the preassigned safety region when necessary to guarantee stability. The main difference between this type of approach and the standard fault-tolerant approach, where faults are only dealt with after they occur, is that the system might lose stabilizability if no proactive action is taken.

How to optimally choose  $\rho$ , both in terms of guaranteeing that the system ends up inside  $\Omega_f$  before the fault occurs, and preserving economic operation of the system as long as possible, requires more work, and is mentioned in chapter 8.

#### 4.4.1 Economic MPC with fault-tolerance

Section 4.1.5 discusses model predictive control with an economic cost function. The goal for this type of controller is to drive the system to a state that maximizes economic profit, it is therefore desirable to operate at this point for as long as possible. By using the soft-constraint approach described in Section 4.4 for proactive fault-tolerant control, the plant can operate at this point until an incipient fault is detected. This is advantageous to the conservative approach of always operating in a "safe set" where the MPC problem is feasible for the potential fault, as that set

might not include the optimal economic point. However, it is important to note that entering the safe set before the fault occurs is critical, as a plant shut-down will be extremely costly.

When the soft constraint 4.14 is added to the optimization problem, the MPC will drive the system to a new point that will optimize profit inside the safe set 4.13, where the MPC will be feasible for the upcoming fault. Then when the fault is fixed, the constraint 4.14 will be removed, and the system will be driven to the original economic optimum. A numerical example can be found in Section 5.2.



# Chapter 5

## Numerical results

This chapter considers a numerical example based on the theory from Chapter 4. Simulations have been done for different actuator-fault scenarios in order to see how the MPC responds. The example is separated into two different parts, where the first considers standard on-line modification of the MPC, based on instant information from the FDI. The second example deals with the proactive approach, where incipient faults are dealt with using penalty functions. The first example uses the traditional quadratic cost function, while the second example uses the economic MPC where the goal is to maximize profit while still handling an upcoming fault. The reason for this is that it is helpful to see how the standard MPC can respond to faults before looking more into fault-handling from an economically optimal viewpoint.

All simulations are done in MATLAB, using YALMIP (Löfberg, 2004) and Multi-Parametric Toolbox 3.

### 5.1 Active fault-tolerant MPC

Consider the linear, time invariant system

$$\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t \quad (5.1)$$

where

$$\mathbf{A} = \begin{bmatrix} 1.2045 & 0.2647 \\ 1.0588 & 1.6015 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 0.2346 & 0.3369 \\ 0.8301 & 0.2706 \end{bmatrix} \quad (5.2)$$

This is a discretized system with sampling time  $T_s = 0.1$ . The eigenvalues of  $\mathbf{A}$  are  $\lambda_1 = 0.8376$ ,  $\lambda_2 = 1.9684 > 1$  and so the system is open-loop unstable. The goal is to design a model predictive controller that drives the system to a reference state, while also being able to compensate for a fault happening at some time instant.

Two critical assumptions are made: it is assumed that there exists an external FDI algorithm that immediately detects the fault and sends the data to the MPC. It is also assumed that the new control problem is feasible when the fault happens, which is not always the case, as described in Chapter 4.

The constraints on the system are

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} \leq \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 5 \\ 5 \end{bmatrix}, \quad (5.3)$$

and

$$\begin{bmatrix} -5 \\ -5 \end{bmatrix} \leq \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \leq \begin{bmatrix} 5 \\ 5 \end{bmatrix}. \quad (5.4)$$

The initial condition is set to be

$$\mathbf{x}_0 = \begin{bmatrix} 0 \\ 2 \end{bmatrix}, \quad (5.5)$$

and the reference state is

$$\mathbf{x}_r = \begin{bmatrix} 1 \\ 1 \end{bmatrix}. \quad (5.6)$$

The following tuning parameters are used:

$$\mathbf{Q} = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}, \quad (5.7)$$

$$\mathbf{R} = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}, \quad (5.8)$$

and the prediction horizon is set to  $N = 10$ , with the quadratic cost function 4.1.

The following fault cases are simulated: Saturation fault on  $u_1$  (input region decrease), input  $u_1$  freezing, and loss of  $u_2$  with  $u_1$  effectivity loss.

### 5.1.1 Actuator saturation fault

This simulation represents a saturation fault on  $u_1$ , which occurs at  $t_f = 1.5$ . The new constraint on  $u_1$  becomes

$$-1 \leq u_1^f \leq 1. \quad (5.9)$$

Figure 5.1a shows how the system responds when the MPC actively changes its control-formulation to accommodate the fault, which in this case will be updating the constraints used in the optimization problem. Figure 5.1b shows the system response when no active changes are made. With fault accommodation, the system is kept stable, but with some degraded performance on the reference tracking. However, without fault accommodation the system quickly becomes unstable. The MPC then has no information about the fault on  $u_1$ , and will not compensate for the fault by using  $u_2$  more aggressively.



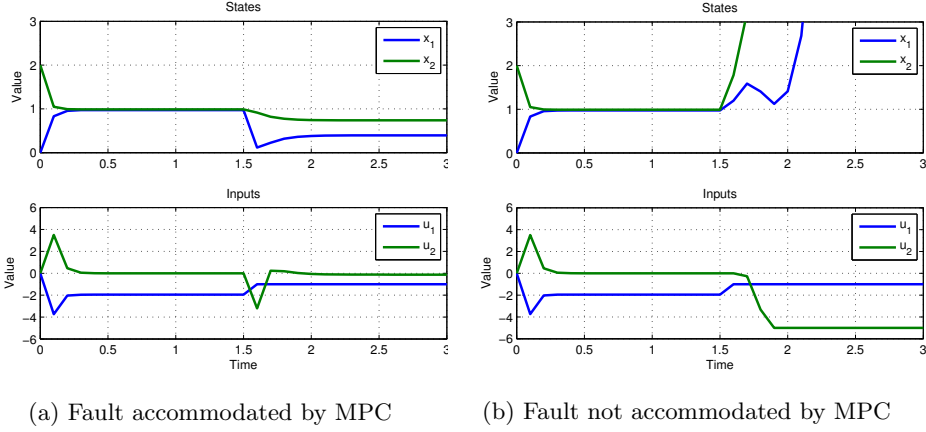


Figure 5.1: Figure showing the response of the system when the operating region of  $u_1$  decreases to  $-1 \leq u_1 \leq 1$  at  $t_f = 1.5$ . The top figures show the states, and the lower show the input values.

### 5.1.2 Actuator freezing

This simulation represents actuator-freezing for  $u_1$ , which occurs at  $t_f = 1.5$ . The new constraint on  $u_1$  becomes

$$u_1^f = -3. \quad (5.10)$$

Figure 5.2a shows how the system responds when the MPC actively changes its control-formulation to accommodate the fault, and 5.2b shows the system response when no active changes are made. As in the previous example, the MPC is able to stabilize the system when knowledge about the fault is available.

### 5.1.3 Actuator input loss and effectivity decrease

This simulation represents actuator-loss of  $u_2$ , and a 30% effectivity loss of  $u_1$ . The fault occurs at  $t_f = 1.5$ . The new constraint on  $u_2$  becomes

$$u_2^f = 0, \quad (5.11)$$

and the input matrix:

$$\mathbf{B}_f = \begin{bmatrix} 0.2346 \cdot 0.7 & 0.3369 \\ 0.8301 \cdot 0.7 & 0.2706 \end{bmatrix} = \begin{bmatrix} 0.1642 & 0.3369 \\ 0.5810 & 0.2706 \end{bmatrix} \quad (5.12)$$

Figure 5.3a shows how the system responds when the MPC actively changes its control-formulation to accommodate the fault, which in addition to updating the constraint in the optimization problem, is to update the prediction model with  $\mathbf{B}_f$ . 5.3b shows the system response when no active changes are made. Again, the MPC stabilizes the system when fault information is available.

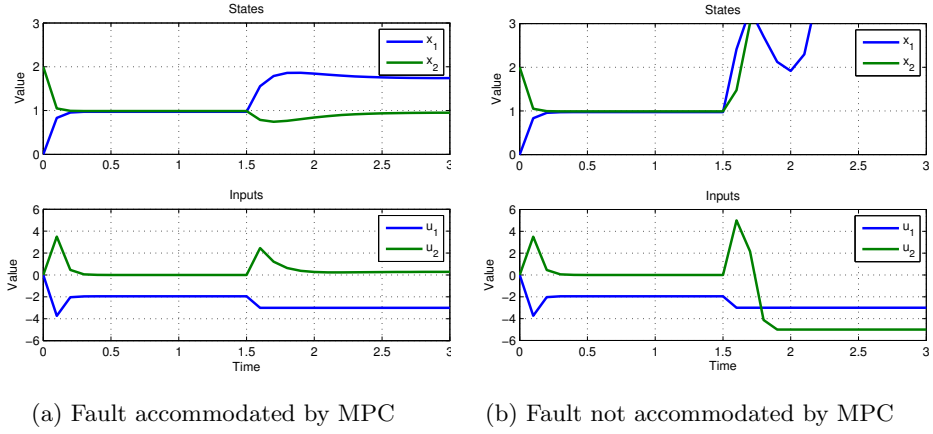


Figure 5.2: Figure showing the response of the system when  $u_1$  freezes at input  $u_1 = -3$  at  $t_f = 1.5$ . The top figures show the states, and the lower show the input values.

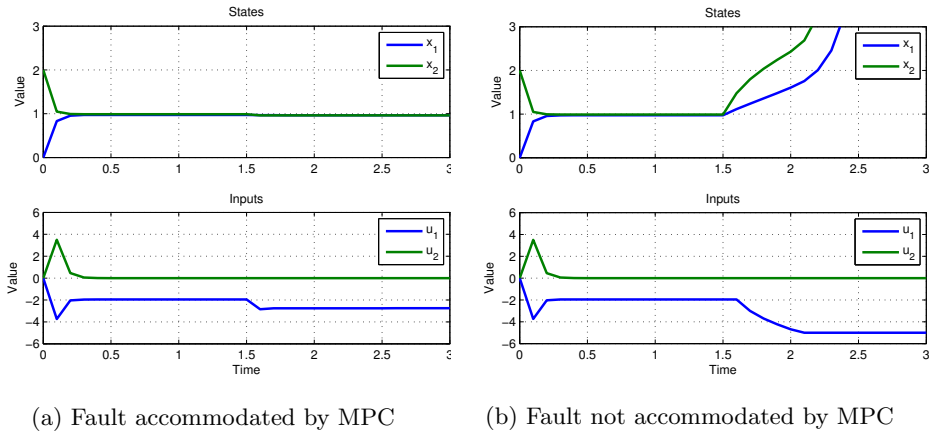


Figure 5.3: Figure showing the response of the system when  $u_1$  loses 30% effectiveness, and  $u_2$  loses input at  $t_f = 1.5$ . The top figures show the states, and the lower show the input values.

## 5.2 Proactive fault-tolerant Economic MPC

Multi-parametric toolbox 3 is used to calculate the positively closed-loop invariant sets. The toolbox takes the system as well as the state- and input constraints as inputs, and calculates the maximal region where a control sequence to control the system exists.

Consider again the system 5.1-5.2, with the constraints 5.3 and 5.4, and the initial state 5.5. In this example, the following economic cost function is used:

$$\begin{aligned}
 J &= \sum_{t=0}^{N-1} \ell(\mathbf{x}_{t+1}, \mathbf{u}_t) = \sum_{t=0}^{N-1} (-\mathbf{C} \|\mathbf{x}_{t+1}\|_1 + \mathbf{R} \|\mathbf{u}_t\|_1) \\
 &= \sum_{t=0}^{N-1} (-10|x_{t+1}^1| - 10|x_{t+1}^2| + |u_t^1| + |u_t^2|)
 \end{aligned} \tag{5.13}$$

which is the negative of a profit function that should be maximized. *Note that for ease of notation,  $x^i$  and  $u^i$  are used instead of  $x_i$  and  $u_i$ ,  $i = 1, 2$ .*

This example will include the same type of on-line MPC modification as in the previous section. However, in addition, feasibility after an incipient fault is dealt with using penalty functions to drive the system into a positively closed-loop invariant set, as described in Section 4.4.

In all cases, when a fault occurs, it is assumed that the MPC receives instant and accurate information about the fault. It will then change its optimization formulation to match the faulty system, as described in Chapter 4.2, and illustrated in Section 5.1.

First a simulation done without proactive control is illustrated, in order to show how the system responds when it is not proactively driven into a positively closed-loop invariant set where the MPC is feasible when the fault occurs. The next subsections show how different penalty weightings affect the response.

*All simulations show the system response when  $u^2$  loses input ( $u_f^2 = 0$ ) at  $t_f = 3$ , and regains its input at  $t = 5$ .*

### 5.2.1 Without proactive control

First consider the case where no action is taken to drive the system into a positively closed-loop invariant region, where the MPC is feasible after the known fault  $u_f^2 = 0$  at  $t_f = 3$ . When the fault happens, the system lies outside the feasible region, and so the MPC problem becomes infeasible. The system quickly becomes unstable. The state trajectory is shown in Figures 5.4 and 5.5, and Figure 5.6 clearly shows that the system becomes unstable.

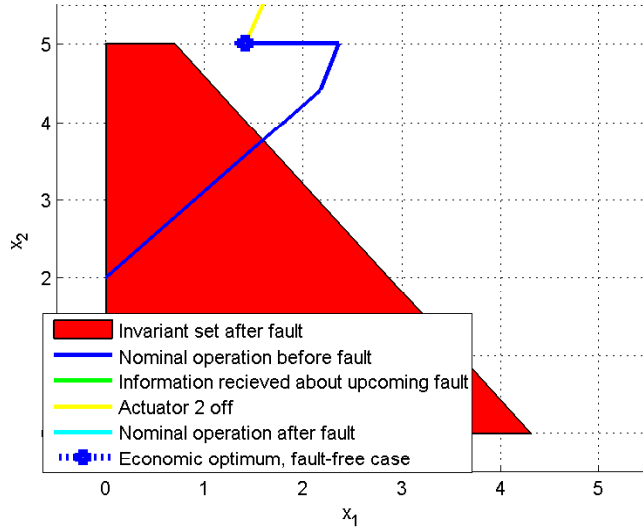


Figure 5.4: State trajectory without proactive control. The colored field shows the positively closed-loop invariant set for  $u_2^f = 0$ .

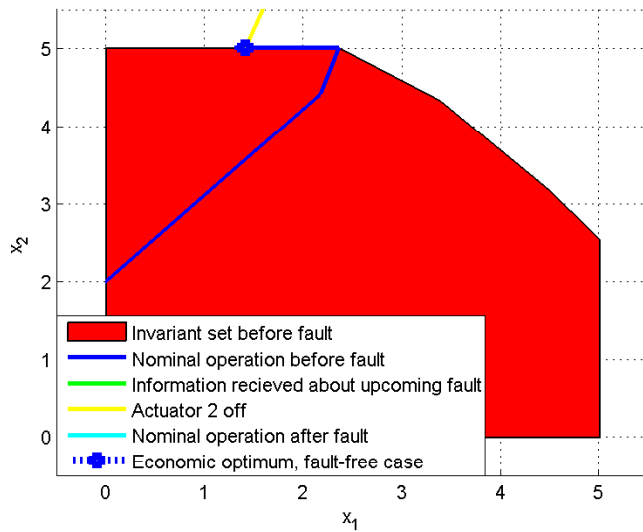


Figure 5.5: State trajectory without proactive control. The colored field shows the positively closed-loop invariant set for the nominal fault-free case.

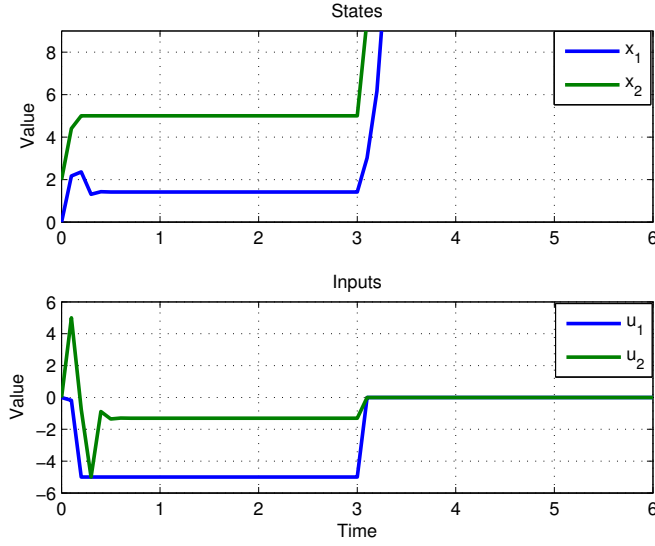


Figure 5.6: The system response without proactive control. Fault  $u_f^2 = 0$  occurs at  $t_f = 3$ .

### 5.2.2 Proactive control with linear penalty

In this section, a situation where the MPC receives information about an incipient fault is considered. The goal is then for the controller to drive the system into a region where the MPC problem will be feasible when the fault occurs. The positively closed-loop invariant set is calculated, and set as a soft region constraint for the states, as described in Section 4.4. Thus, when the information about the upcoming fault is retrieved, the constraints

$$\begin{aligned} \mathbf{G}_f \mathbf{x}_t &\leq \mathbf{h}_f + \epsilon_t \\ \epsilon_t &\geq 0, \end{aligned}$$

where  $\mathbf{G}_f \mathbf{x} \leq \mathbf{h}_f$  denotes the feasible set when the fault occurs, are added to the problem. The cost function then becomes

$$J = \sum_{t=0}^{N-1} (-10|x_{t+1}^1| - 10|x_{t+1}^2| + |u_t^1| + |u_t^2| + \rho \|\epsilon_t\|_1) \quad (5.14)$$

It will be clear that with an exact penalty function, the system will be driven into this region, and the MPC problem will be feasible when the fault occurs. The two different penalty weightings  $\rho = 12$  and  $\rho = 11$  were found by trial and error.

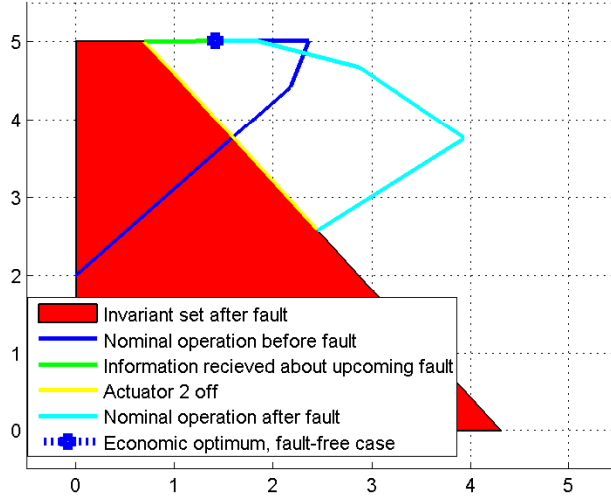


Figure 5.7: State trajectory with proactive control and exact penalty function. The colored field shows the positively closed-loop invariant set for  $u_f^2 = 0$ .

### Simulation with $\rho=12$

Figures 5.7 and 5.8 show the system trajectory  $\rho$  is chosen large enough so that the penalty function is exact. The system operates in the economic optimal point until the controller receives information about the upcoming fault. The system is then driven into the new feasible set. When the fault occurs, the MPC is modified and able to control the system, it then drives the system to a new optimal point inside the new feasible region. At  $t = 5$ , when  $u_2$  goes back to normal operation, the system is driven back to its original optimal point. Figure 5.9 shows the system response versus time.

### Simulation with $\rho = 11$

With a smaller  $\rho$ , the penalty weighting is not large enough, and the penalty function is inexact. The system then stays in the original optimal point, and the MPC problem becomes infeasible when the fault occurs. Figures 5.10, 5.11 and 5.12 illustrate the response and system trajectory.

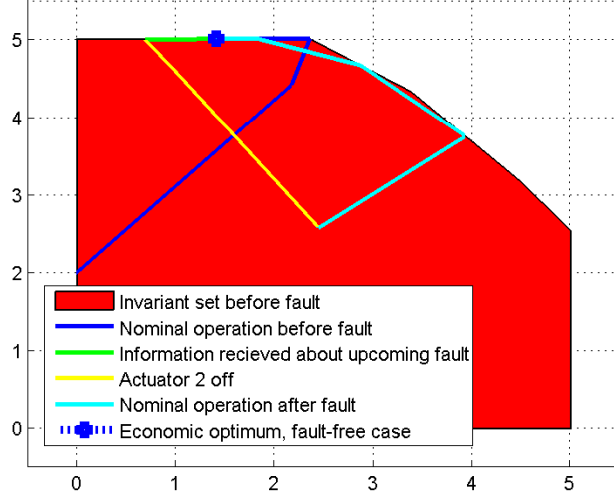


Figure 5.8: State trajectory with proactive control and exact penalty function. The colored field shows the positively closed-loop invariant set for the nominal fault-free case.

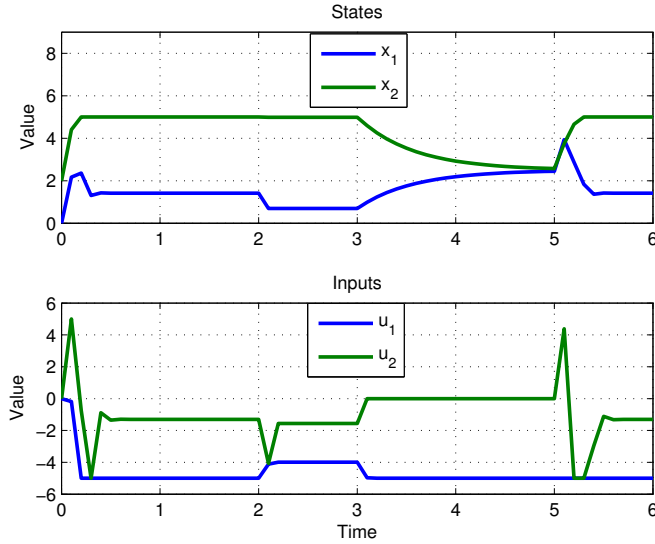


Figure 5.9: System response with exact penalty function. The information about the incipient fault  $u_f^2 = 0$  is received at  $t = 2$ . The fault occurs at  $t_f = 3$ , and  $u^2$  is fixed at  $t = 5$ . Penalty weighting  $\rho = 12$ .

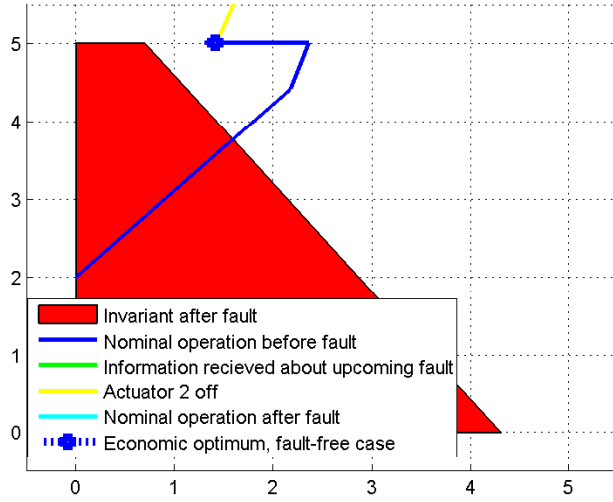


Figure 5.10: State trajectory with proactive control and inexact penalty function. The colored field shows the positively closed-loop invariant set for  $u_f^2 = 0$ .

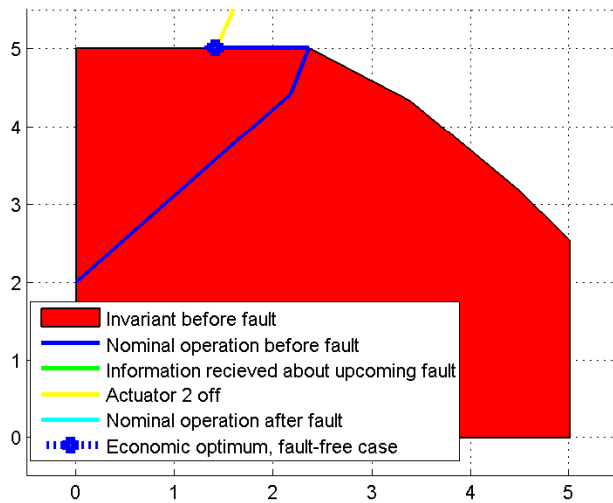


Figure 5.11: State trajectory with proactive control and inexact penalty function. The colored field shows the positively closed-loop invariant set for the nominal fault-free case.



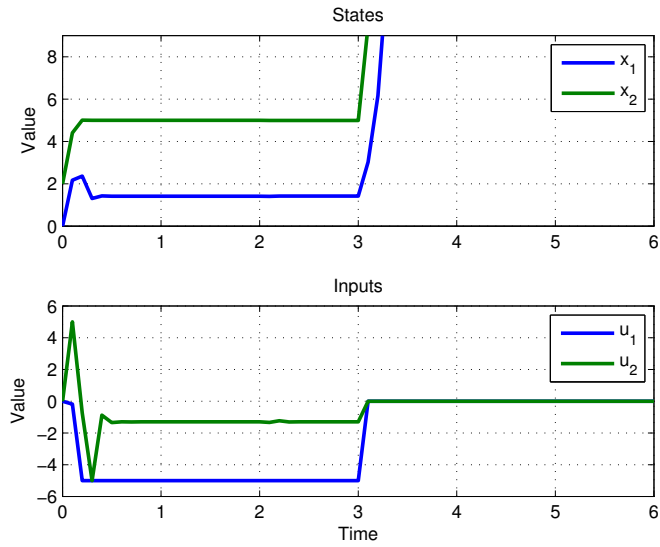


Figure 5.12: The system response when information about the incipient fault of actuator 2 is received at  $t = 2$ , but the penalty function is inexact. The fault  $u_2^f = 0$  occurs at  $t = 3$ . Penalty weighting  $\rho = 11$ .



# Chapter 6

## Discussion

The previous section illustrates that MPC can be used to obtain fault-tolerant control under basic assumptions. This chapter discusses the obtained results and the assumptions that were made. The main contribution of this report, using soft constraints to ensure feasibility after an incipient fault, is discussed in the last section.

### 6.1 Active fault-tolerant MPC

The numerical results from Section 5.1 illustrate the effectiveness of using MPC as a fault-tolerant control scheme when coupled with a reliable fault detection and isolation algorithm. The simulations show that the MPC is able to stabilize the system after a fault if the assumptions about feasibility and correct fault estimates hold true. When a fault in an actuator occurs, the controller effectively redistributes input to the other actuator.

For the actuator 1 saturation fault in Section 5.1.1, the MPC is modified to accommodate the fault. Actuator 2 quickly compensates for the fault before the system becomes unstable. From Figure 5.1a it is clear that the system is stable when the MPC is modified, and Figure 5.1b shows the system becoming unstable when no modifications to the MPC are made. Similarly for the other actuator faults in Figures 5.2 and 5.3 the control is distributed effectively to stabilize the faulty system, and the system loses stability if no modification of the MPC is made. This has an intuitive explanation: the MPC then calculates the control law based on a wrong model, and the inputs will therefore not be correct for the actual faulty system. The controller has no information about the fault, and will continue to try and apply input on the faulty actuator. This illustrates the importance of an accurate FDI.

As can be seen in the numerous simulations, the reference tracking suffers after the fault. This is to be expected, since the faults reduce the degrees of freedom

of the controller. As noted in Section 4.2.2, re-tuning the MPC parameters (horizon and weighting matrices) in an effective manner could potentially increase the reference tracking after the fault. Effective and reliable methods for doing this remains to be studied in the literature, but can potentially improve performance for a faulty system. However, the main goal of Section 5.1 was to show how the MPC can maintain stability for a faulty system, which was possible without changing its tuning-parameters in the provided examples.

It is important to note that these examples only show how the MPC handles faults as an isolated part of a larger scheme. Many new challenges arise when this is put into use in real-time with other components, such as an FDI. Firstly, there will always be some delay from the FDI as well as in the modification of the MPC. A challenge in the design phase will therefore be to make sure that the system does not reach an unrecoverable state and become unstable before the fault is detected. Secondly, the information from the FDI needs to be as accurate as possible. Since the proposed scheme of fault-handling in MPC is the same as modifying the MPC to work on a "new" system with new constraints, accurate models for the faulty system is important. However, as mentioned in Section 1.3, the assumption of quick and reliable information from the FDI is not unreasonable, because of the rapid development and improvement of FDI schemes.

In all the examples, perfect state feedback was assumed. In most real systems, this assumption does not hold true, and an estimator needs to be used to give the MPC information about the current states of the system, as mentioned in Section 4.1.1. This is, however, a problem for many other fault-tolerant control schemes, and does not reduce the effectiveness of using the MPC compared to other methods.

Fault-tolerant control using MPC is achieved by effectively modifying the optimization problem so it matches the new, faulty, system. It is clear that the MPC is a great choice for the controller as part of a larger fault-tolerant scheme, that also includes FDI and state estimation. This supports the theory from earlier research, described in Section 1.4.

## 6.2 Proactive fault-tolerant Economic MPC

The numerical examples in Section 5.2 illustrate that using soft constraints to drive the system into a positively closed-loop invariant region (*from here on denoted the feasible region, from the reasons mentioned in Section 4.4*) for the faulty system is an effective way to handle potential infeasibility if knowledge about the incipient fault is available.

In the case of a sudden actuator dropout, it can clearly be seen from Figures 5.4 and 5.5 that the feasible region is reduced. As illustrated in these figures, if no proactive action is taken, the system operates outside the new feasible region when the fault occurs. The system then becomes unstable when the fault occurs, as illus-

trated in Figure 5.6. In this region there does not exist a control sequence to control the faulty system. An intuitive interpretation of this is that the remaining actuator does not have enough capacity to still control the system from the current state.

By using soft constraints with an exact penalty function to drive the system into the feasible region for the fault before it occurs, as illustrated in Figure 5.7, the remaining actuator is able to control the system after the other renders unusable. Figure 5.9 shows the stability of the system. When the faulty actuator becomes available, e.g. is repaired, the system is driven back to its original optimum. This illustrates the importance of the state the system is in when a fault occurs, and the flexibility of incorporating proactive fault-tolerant control in the MPC to compensate for potential infeasibility.

Figure 5.10 illustrates that by using a lower value on the penalty, the objective function finds an optimum outside the feasible region for the fault. This yields the same response as if no proactive action is taken, and therefore clearly shows the importance of using an exact penalty function. The penalty weightings used in this numerical example were found by trial and error when doing the simulations. However, there exist effective methods to calculate these penalties, as noted in Section 4.3, which can potentially be used in the future to guarantee exactness.

Furthermore, in Figure 5.9, the system reaches the feasible region for the fault long before the fault occurs. Optimizing the process of driving the system into the feasible region with respect to preserving operation in the nominal optimal state for as long as possible, while still entering the feasible region before the fault occurs, is something that needs more work. This can potentially maximize profits while still ensuring fault-tolerance.

Another important note is that this approach relies heavily on the calculation of positively closed-loop invariant sets. As mentioned in Section 4.4, methods exist to calculate these effectively. These calculations can be heavy for systems of many variables, and can potentially introduce a problem with this approach. However, as computation power becomes greater and greater and more effective algorithms for invariant set estimation are designed, this approach can effectively be implemented. Some methods that calculate exact penalty functions also implicitly calculate the feasible region, e.g. Hovd (2011). This can potentially be implemented for the proactive FTMPC.

All the numerical examples in Section 5.2 ignored noise in the system. Because of the importance of the system operating inside the feasible region for the fault when the fault occurs, noise can cause a potential issue. A conservative feasible region should then be used, so that the effect of noise can not drive the system out of the feasible region for the fault.

In contrast to other methods in the literature, where the conservative approach

of always operating in the feasible region for a potential fault is implemented, this approach lets the system operate at a state that yields higher profit while still being able to handle an upcoming fault. Of course, this assumes an extremely effective FDI that can detect the incipient fault, or the need of making an estimate based on historical data of when a fault will occur. The conservative approach is therefore necessary if such knowledge is not available.

## Chapter 7

# Conclusion

In this report, general concepts and methods in fault-tolerant control, and fault detection and isolation were described, with a focus on using model predictive control for fault-tolerance. Actuator- and system faults showed to be easily incorporated in the MPC framework, and the MPC proved to effectively redistribute control actions to healthy actuators once one became faulty. There are many challenges for a complete fault-tolerant control scheme, but the MPC showed good results if information about faults were available. The effectiveness of the MPC to handle faulty scenarios was illustrated with a numerical example.

Furthermore, a way to handle potential infeasibility issues for incipient faults using soft constraints was introduced. In contrast to the normal approach, which is rather conservative, this approach showed great potential for more flexible operation. From an economic standpoint, this approach could be used for potential higher profits. Several challenges including computation time were discussed, and future work is therefore needed to make the method effective. A numerical example illustrated the concept, which showed promising results.





## Chapter 8

# Future work

The numerical example for the proactive fault-tolerant approach using soft constraints showed good results. However, several improvements need to be made to improve its effectiveness. First of all, fast and reliable computations of optimal and exact penalty functions with regards to guaranteeing feasibility when a fault occurs, and preserving optimal economic operation for as long as possible should be studied. Several methods for designing optimal penalty functions are available in the literature, and should be tested in cooperation with the proactive method. Secondly, as mentioned in the discussion, the invariant set calculations can be computationally heavy, and intelligent ways of incorporating these calculations need to be worked on.

In this report, it was assumed that the systems were not disturbed by noise. This is clearly not the case for real systems, and using the proactive approach for systems that include noise should be looked in to. Another important assumption that was made was that the FDI was able to provide instant and accurate information about the fault. Methods to handle delays and uncertainties from the FDI should be studied. Clearly, the proactive approach combines methods from many different theories, and developing an overall scheme where all these methods work together will require significant amounts of work.

Future work also includes the study of automatically tuning the MPC parameters after a fault occurs, in order to improve reference tracking in the case of a standard reference tracking MPC.



# Bibliography

- Amrit, R., Rawlings, J. B., and Biegler, L. T. (2013). Optimizing process economics online using model predictive control. *Computers and Chemical Engineering*, 58:334–343.
- Blanchini, E. (1999). Set invariance in control. *Automatica*, 35:1747-1767.
- Blanke, M., Frei, C., Kraus, F., Paton, R., and Staroswiecki, M. (2000). What is fault tolerant control? In *IFAC Symposium on Fault Detection Supervision and Safety for Technical Processes*, pages 40–51, Budapest, Hungary.
- Blanke, M., Kinnaert, M., Lunze, J., and Staroswiecki, M. (2006). *Diagnosis and fault-tolerant control*. Springer, 2 edition.
- Bø, T. I. and Johansen, T. A. (2014). Dynamic safety constraints by scenario based model predictive control. *Proceedings of the 19th IFAC World Congress*.
- Foss, B. and Heirung, T. (2013). *Merging Optimization and Control*. Technical report, Norwegian University of Science and Technology.
- Frank, P. M. (1994). On-line fault detection in uncertain nonlinear systems using diagnostic observers: a survey. *International Journal of Systems Science*, 25(12):2129–2154.
- Hovd, M. (2011). Multi-level programming for designing penalty functions for mpc controllers. *IFAC Proceedings Volumes (IFAC-PapersOnline)*, 18:6098–6103.
- Huzmezan, M. and Maciejowski, J. M. (1999). Automatic Tuning for Model Based Predictive Control During Reconfiguration. In *Automatic Control in Aerospace 1998*, page 237.
- Ioannou, P. and Sun, J. (1996). *Robust Adaptive Control*. Number v. 1 in Control theory. PTR Prentice-Hall.
- Isermann, R. (2006). *Fault-Diagnosis Systems*. Springer Berlin Heidelberg.
- Isermann, R. and Ballé, P. (1997). Trends in the application of model-based fault detection and diagnosis of technical processes. *Control Engineering Practice*, 5:709–719.

- Jiang, J. and Yu, X. (2012). Fault-tolerant control systems: A comparative study between active and passive approaches. *Annual Reviews in Control*, 36(1):60–72.
- Kerrigan, E. and Maciejowski, J. (2000a). Invariant sets for constrained nonlinear discrete-time systems with application to feasibility in model predictive control. *Proceedings of the 39th IEEE Conference on Decision and Control*, 5:4951–4956.
- Kerrigan, E. C. and Maciejowski, J. M. (2000b). Soft constraints and exact penalty functions in model predictive control. In *Proc. UKACC International Conference (Control)*.
- Lao, L., Ellis, M., and Christofides, P. (2013). Proactive fault-tolerant model predictive control. *AIChE J.*, 59(8), 2810–2820.
- Löfberg, J. (2004). Yalmip : A toolbox for modeling and optimization in MATLAB. In *Proceedings of the CACSD Conference*, Taipei, Taiwan.
- Maciejowski, J. (1997). Modelling and predictive control: Enabling technologies for reconfiguration. In *Proceedings of 3rd IFAC Conference on System Structure and Control*, Bucharest. Also in *Annual Reviews in Control*, vol.23, 1999, pp.13–23.
- Maciejowski, J. (1998). The implicit daisy-chaining property of constrained predictive control. *Applied Mathematics and Computer Science*, 8(4):101–117.
- Maciejowski, J. (1999). Fault-tolerant aspects of mpc. *IEE Seminar on Practical Experiences with Predictive Control*, (1):1–4.
- Maciejowski, J. (2002). *Predictive Control with Constraints*. Prentice Hall, England.
- Maciejowski, J. and Jones, C. (2003). Mpc fault-tolerant flight control case study: Flight 1862. *International Federation of Automatic Control on Safeprocess Symposium*, pages 119–124.
- Mahmoud, M. S. and Xia, Y. (2014). *Analysis and Synthesis of Fault-Tolerant Control Systems*. Wiley Publishing, 1st edition.
- Mayne, D., Rawlings, J., Rao, C., and Scokaert, P. (2000). Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814.
- Mellichamp, D. A., Edgar, T. F., Doyle, F. J., and Seborg, D. E. (2010). *Process Dynamics and Control*. John Wiley & Sons.
- Miksch, T., Gambier, A., and Badreddin, E. (2008). Real-time implementation of fault-tolerant control using model predictive control. *World Congress*, pages 11136–11141.
- Ocampo-Martínez, C., Puig, V., Quevedo, J., and Ingimundarson, A. (2005). Fault tolerant model predictive control applied on the Barcelona sewer network. In *Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference, CDC-ECC '05*, volume 2005, pages 1349–1354.

- Patton, R. and Chen, J. (1994). Review of parity space approaches to fault-diagnosis for aerospace systems. *Journal of guidance control and dynamics*, 17(2):278–285.
- Rawlings, J. B., Angeli, D., and Bates, C. N. (2012). Fundamentals of economic model predictive control. *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 3851–3861.
- Skogestad, S. and Postlethwaite, I. (2005). *Multivariable Feedback Control: Analysis and Design*. John Wiley & Sons.
- Vada, J., Slupphaug, O., Johansen, T. A., and Foss, B. A. (2001). Linear mpc with optimal prioritized infeasibility handling: Application, computational issues and stability. *Automatica*, 37(11):1835–1843.
- Venkatasubramanian, V., Rengaswamy, R., Yin, K., and Kavuri, S. N. (2003). A review of process fault detection and diagnosis. *Computers and Chemical Engineering*, 27(3):293–311.
- Xue, W., Guo, Y.-q., and Zhang, X.-d. (2007). A Bank of Kalman Filters and a Robust Kalman Filter Applied in Fault Diagnosis of Aircraft Engine Sensor/Actuator. *Second International Conference on Innovative Computing, Information and Control (ICICIC 2007)*.
- Zhang, Y. and Jiang, J. (2008). Bibliographical review on reconfigurable fault-tolerant control systems. *Annual Reviews in Control*, 32(2):229–252.

