Efficient Model Predictive Control Using a Sequential Semi-Definite Programming

Nadav Bar

Master of Science (Siv. Ing.) Thesis
Electrical Engineering Faculty

Department of Engineering Cybernetics

Norwegian University of Science and Technology

July 17, 2003

Preface

This report summarizes the main results of Master of Science (siv. ing.) thesis at the department of Engineering Cybernetics, during spring 2003.

I would very much like to thank Lars Struen Imsland, who motivated my interest in this topic, and provided many useful suggestions for improvements. The pleasant discussions with him guided and enlightened me during the last half an year. I thank Professor Bjarne A. Foss for his guidance in the last two years, who taught me how to be structural and organized. Thank to Manmeet, Who read parts of this text and gave me some useful comments.

I am very grateful to Eirin Marie, for her love and support, for her patience and dedication. She who waited many lonely nights, giving me the opportunity to finish my research successfully.

A very special Thanks to Einar S. Idsø, whom introduced me to the magically world of control.

Nadav Bar
Electrical Engineering faculty,
Norwegian University of Science and Technology

Contents

Summary

The contribution of this thesis is in the area of fast, efficient robust linear predictive control, especially for systems with high sampling-rates where other Model Predictive Control strategies are not usable.

Based upon previous work done by Kothare et al. (1996), Fares et al. (2000), Kouvaritakis et al. (2000) and others, a novel approach known as SSDMPC was first introduced by Drageset, Imsland and Foss (2003). Their paper describes the main subject of this approach, the SSDMPC algorithm, but only a simple SISO example is given. The algorithm experienced instabilities and was very sensitive to changes in parameters, especially for large systems. This thesis expands the work done, provides further analysis in this area, further stabilizes the algorithm, enables it to handle MIMO systems and provides applications and real-time experiments to verify the results.

Briefly, the approach taken in this thesis is divided in two parts. The first of these concentrates on solving an *Offline* problem, which maximizes the size of a region of attraction. This region guarantees the feasibility, invariance and stability of the system. Two offline types are presented. Efficient Robust Predictive Control (ERPC) is a convex optimization problem, while SSDMPC is more general, but a non-convex optimization problem due to a nonlinear equality constraint. This equation can be eliminated by including it in a Lagrangian. It is then possible to approximate the Lagrangian and transform the problem to a SDP problem, solved iteratively, in a SQP-manner. The SSDMPC problem remains, however, non-convex. Hence solving it is much more complex than solving ERPC, and while ERPC guarantees global solution, the SSDMPC finds only a local one. Nevertheless, it is shown that the region of attraction is larger for SSDMPC, and shown that the difference in size can be crucial for some cases.

The second part of this thesis involves solving an *Online* algorithm, which uses the solution found by the former offline section to ensure that the trajectories are maintained within the region and feasibility is preserved. The online algorithm solves an optimization problem fast and effectively.

It is shown that the combination of offline-online parts forms an alternative to a model predictive control based on quadratic programming. This approach is extremely fast online, hence can be used for systems with very high sampling rates, yet ensures feasibility and stability. QP-based model predictive control methods are expensive both in time and resources, and cannot usually be applied to high dynamic systems and/or resource-limited embedded systems.

Several applications are presented in this thesis, both for SISO and MIMO systems with high sampling-rates. The offline/online strategy is implemented successfully in real-time environment with very high sampling-rates of an unstable plant. Comparison between different approaches is given with extensive analysis. As this approach can be applied to a variety of systems to produce good results, it is hoped this study will encourage additional researches to continue work in this field.

Notation

Terminology

Throughout the thesis, a specific notation shall be adapted, to make the text more understandable. The expressions 'SSDMPC ellipsoid' and 'ERPC ellipsoid' shall refer to the ellipsoids generated by the SSDMPC and the ERPC respectively.

Another term to be adopted is the use of the algorithm name with the control horizon value in parentheses, e.g. ERPC(2). This is to avoid repetitive sentences such as '...solution of the ERPC algorithm using $N_c = 2$'.

The term *Systems with tight dynamic requirements* or *Fast (high) dynamic systems* will be referred to as systems which have high sampling-rates, due to rapid changes in the dynamics, such as missiles, aircrafts and so on.

General notation:

Symbol	**Description**	**Unit / dimensions**
ϵ	A threshold parameter	scalar
a_i	Ellipsoid's i axis length	scalar
k	Iteration number or index, both for offline algorithm and online	-
K	Optimal state feedback gain	
N_c	Control horizon length	scalar
T_x	The Perturbation matrix of a square matrix to the $\mathbf{x}$-space	$m_x \times (m_x + m_u N_c)$
$\mathcal{E}_z$	The ellipsoid $\mathcal{E}_z = \{\mathbf{z} \in \mathbb{R}^{m_x + N_c \cdot m_u} \vert \mathbf{z}^T P_z \mathbf{z} < 1\}$	-
$\mathcal{E}_{zx}$	The projection of $\mathcal{E}_z$ in the n dimensional $\mathbf{x}$-space, defined by $\mathcal{E}_{zx} = \{\mathbf{x} \in \mathbb{R}^n \vert\ \mathbf{x}^T P_{zx} \mathbf{x} < 1\}$	n
$\mathcal{E}_x$	The ellipsoid $\mathcal{E}_x = \{\mathbf{x} \in \mathbb{R}^n \vert\ \mathbf{x}^T P_x \mathbf{x} < 1\}$	areal/volume

Symbol	Description	Unit / dimensions
P_z	Defines the ellipsoid $\mathcal{E}_z$, used by online Newton Raphson problem	$(m_x+m_uN_c)\times(m_x+m_uN_c)$
Q_z	The inverse of P_z	$(m_x+m_uN_c)\times(m_x+m_uN_c)$
Q_{zx}	Projection of Q_z in $\mathbf{x}$-space, i.e. $T_xQ_zT_x^T$	$(m_x)\times(m_x)$
P_{zx}	The inverse of the projection of Q_z in the $\mathbf{x}$-space, i.e. Q_{zx}^{-1}, defines the ellipsoid $\mathcal{E}_{zx}$	$m_x\times m_x$
P_x	Defines the ellipsoid $\mathcal{E}_x$	$(m_x)\times(m_x)$
V_f	Volume factor, $(\det P_{zx})^{1/2}$	scalar
$\mathbf{x}_k$	State at iteration k	m_x
$\mathbf{z}_k$	Augmented state $(\mathbf{x}^T,\ \mathbf{f}^T)^T$ at iteration k	$m_x+N_cm_u$

Offline Variables and parameters:

Symbol	Description	Dimensions
α_k	Backtracking step size	$0<\alpha_k\leq 1$
β	$\mathcal{S}$-procedure variable	
η	Backtracking sufficient decrease parameter	$10^{-4}-10^{-1}$
ρ_l	Backtracking α_k reduction parameter	≈ 0.5
Λ	Lagrange multiplier	$(m_x+N_cm_u)\times(m_x+N_cm_u)$
μ	Lagrangian penalty parameter	Scalar
γ	Affects the increase rate of penalty parameter	
ρ	Increase factor of the penalty parameter	
H_k	Exact Hessian at iteration k	
H_m	Modified Hessian	
M	Variable, SSDMPC. Constant, ERPC	$N_c\times m_u$
p	Search Direction $[Q_z(k)-Q_z(k-1)\ \ P_z(k)-P_z(k-1)]^T$	
P_{proj}	Permutation matrix of the Hessian	
$\mathcal{L}_A$	Lagrange function	
R_A	An initial constant ellipsoid used by $\mathcal{S}$-procedure	
T	**Svec** operator matrix	
t	Slack variable	Scalar
v_k	Decision variable, a vector contains all the SSDMPC offline variables	
w	The vector $(Q_z,\ P_z)^T$	

Online Variables and parameters:

Symbol	Description	Dimensions
$\mathbf{c}_k$	Perturbation vector from the optimal control	m_u
$\mathbf{f}$	The vector future perturbations $\mathbf{f} = (\mathbf{c}_k^T,\ ..., c_{k+N_c-1}^T)^T$	$m_u N_c$
J_f	Performance index used by ERPC	
J_{LQ}	Traditional LQ performance index	
J_m	Performance index used by SSDMPC	
R	Input weighting matrix of the LQ	
Q	States weighting matrix of the LQ	
T_s	Time step	sec
W	Weighting matrix used by ERPC	
Γ	Weighting matrix used by SSDMPC	

Sets:

Symbol	Description
$\mathcal{P}$	Polytope set confined inside an ellipsoid
$Omega$	LDI polytope
Υ	Set of ellipsoids $\{\mathcal{E}_1,\ \mathcal{E}_2,\ ...\}$
$\mathcal{X}_{LMI}$	A set of LMIs
$\mathcal{O}_\infty$	Maximal admissible set

Abbreviation :

Notation	Description
BMI	Bi-Linear Matrix Inequality
Co	Convex Hull
ERPC	Efficient Model Predictive Control (offline algorithm)
LMI	Linear Matrix Inequality
LQ	Linear Quadratic
LDI	Linear Differential Inclusion
MPC	Model Predictive Control
NR	Newton Raphson (online algorithm)
SSDMPC	Sequential Semi-Definite Model Predictive Control
ROA	Region Of Attraction

Chapter 1

Introduction

The focus of this thesis is the control of constrained systems described by a linear model. Developing an accurate model of a physical system is a complex task. As physical systems are inherently nonlinear, any such model is also usually nonlinear. However, if the operating range of the control system is small, and if the involved nonlinearities are smooth, then the control system may be reasonably approximated by a linear model. As physical systems are inherently nonlinear.

Various control strategies have been suggested for linear models, but few can handle the problem of *constraints*. Any system possessing physical or operational limitations should be treated with care. These restrictions or constraints, if violated, can cause unexpected behavior. In some extreme cases they can even endanger lives. Consider, for example, a passenger plane, needing to approach a busy airport. Air-traffic restricts this plane from flying outside a certain 'corridor', or a collision may occur. We then say that the flight controller has placed a *constraint* on the altitude, which should not be violated. This thesis will offer a strategy to handle such constraints.

Another issue which will be discussed is robust control. A *robust* system can tolerate mismatch between a linearized internal model and a nonlinear plant, guaranteing good performance for large operation range. This thesis gives a strategy to assure the robustness of a system.

1.1 Model Class

This thesis will deal with discrete state-space model representation,

$$\begin{aligned} \mathbf{x}(k+1) &= A(k)\mathbf{x}(k) + B(k)\mathbf{u}(k), \qquad \mathbf{x}(0) = \mathbf{x}_0 \\ &\mathbf{x}(k) \in \mathbb{R}^{m_x} \\ &\mathbf{u}(k) \in \mathbb{R}^{m_u} \end{aligned} \tag{1.1}$$

subject to input and state constraints

$$\begin{aligned} \mathbf{u}_{min} \leq \mathbf{u}(k) \leq \mathbf{u}_{max}, \\ \mathbf{x}_{min} \leq \mathbf{x}(k) \leq \mathbf{x}_{max} \end{aligned} \tag{1.2}$$

Only linear constraints will be considered in this thesis. It will be assumed that perfect state knowledge is available and that the system is stabilisable. For uncertain systems a polytopic uncertainty model could be applied

$$[A_k \quad B_k] \in \Omega = \mathbf{Co}\{\left[A^1 \ B^1\right], ..., \ [A^p \ B^p]\} \tag{1.3}$$

where $\mathbf{Co}$ denotes the convex hull, and A^j and B^j are corners of the uncertainty set.

1.2 Model Predictive Control

Model Predictive Control (MPC) refers to a class of control algorithms that utilize an explicit process model in order to predict the future response of a plant. At each control interval an MPC algorithm attempts to optimize future plant behavior by computing a sequence of future control moves. The first input in the optimal sequence is then sent into the plant, and the entire calculation is repeated at subsequent control intervals. The ability to handle constraints is the most contributed to the success of MPC in industry (Qin and Badgwell (2002) and Maciejowski (2002)).

1.2.1 Control Objectives

The control objective will be to minimize a infinite horizon linear quadratic (LQ) cost function

$$J_{LQ} = \sum_{i=0}^{\infty} \mathbf{x}_{k+i+1}^T Q \mathbf{x}_{k+i+1} + \mathbf{u}_{k+i}^T R \mathbf{u}_{k+i} \tag{1.4}$$

where Q and R are positive semi-definite, symmetric matrices and $\mathbf{x}_{k+i+1} \in \mathcal{R}^{m_x}$ and $\mathbf{u}_{k+i} \in \mathcal{R}^{m_u}$ denote predicted values of states and control inputs.

1.2.2 MPC Problem Formulation

Consider the cost function

$$V(k) = \sum_{i=H_w}^{H_p} (\hat{x}_{k+i} - r_{k+i})^T Q (\hat{x}_{k+i} - r_{k+i}) + \sum_{i=0}^{N_c-1} \Delta\hat{u}_{k+i}^T R \Delta\hat{u}_{k+i}^T \tag{1.5}$$

A typical formulation, which will be called QP-MPC, of the optimization problem would then be

$$\begin{aligned}
\min \quad & V(k) && \text{(1.6a)}\\
s.t. \quad & && \\
\hat{x}_{k+i+1} &= f(\hat{x}_{k+i}, \hat{u}_{k+i}) \quad i \in 1, ..., H_p && \text{(1.6b)}\\
\Delta\hat{u}_{k+i} &= 0, \qquad i \geq N_c && \text{(1.6c)}\\
\underline{u} \leq \hat{u}_{k+i} &\leq \overline{u} && \text{(1.6d)}\\
Q, \quad R &> 0 && \text{(1.6e)}
\end{aligned}$$

where r_{k+i} is the reference trajectory, H_p is the prediction horizon length and N_c is the *control horizon*. We shall assume in this thesis that $N_c = H_p$ although the general case is $N_c \leq H_p$. The error vector $\hat{x}_{k+i} - r_{k+i}$ is penalized at every point in the prediction horizon, in the range $H_w \leq i \leq H_p$. We shall assume here that no tracking is required, i.e. $r_k = 0$, $\forall k$, and we start penalizing from the first iteration, i.e. $H_w = 0$. Then prediction horizon is given by

$$0 \leq i \leq N_c \tag{1.7}$$

The penalty matrices Q, R should be positive semi definite in order to assure that $V(k) \geq 0$ for stability issue.

Since the control strategy to be presented in this thesis assumes linear models, $f(x, u)$ must be linear approximated in case it is a nonlinear function. It can either be linearized around an equilibrium point, or by a family of linear time-invariant systems (LDI representation, see Boyd et al. (1994))

1.2.3 The Constraints

A significant problem that may can occur with the predictive control problem, as formulated above, is the optimizer may encounter an infeasible problem. This can happen when an unexpected large disturbance occurs, so the plant cannot be kept within the specified constraints. Alternatively, the real plant may behave differently from the model. There are numerous ways in which the predictive control problem can become infeasible, and most are difficult to anticipate.

One strategy for dealing with infeasibility is to 'soften' the constraints. That is, rather than regard the constraints as 'hard' boundaries which can never be violated, to allow them to be crossed occasionally, if absolutely necessary. There is an important distinction between input and output constraints. Output constraints can usually be 'softened' thus allowing them to cross the limitation for a short period. Input constraints, however, are usually 'hard' constraints, which can never be violated. Examples are valves, actuators and flaps, which have a limited range of action. The only way to exceed these ranges is to alter them physically, e.g. by installing more powerful actuators. Online controllers cannot just 'abort' in case of infeasibility, and various solutions are suggested, ranging from *ad hoc* measures such as outputting the same control signals in the previous step, to sophisticated strategies of *constraint management*, described by Rawlings and Muske (1993), in which one tries to relax the least-important constraints in an attempt to regain feasibility.

This thesis tries to deal with these hard constraints in such a way that the input, if satisfying certain properties, will never reach its bounds, hence preventing saturation and thus infeasibility of the problem.

1.2.4 Stability

Predictive control is a feedback control policy. There is therefore a risk that the resulting closed loop might be unstable. The performance of the plant is optimized over the predicted horizon, and is repeated each sampling-time. This does not guarantee stability since the strategy does not 'see' beyond the control horizon, and may put the plant in such a state that it eventually becomes impossible to stabilize. This is particulary likely when input constraints are presented.

Stability can, however, be guaranteed for a problem with an infinite prediction horizon (Maciejowski; 2002). The *principal of optimality* states that the tail (the optimal trajectory from the time no new information enters the optimization problem) of any optimal trajectory is itself the optimal trajectory from its starting point. If one defines the cost function as $V(k)$, then this function decreases as k decreases, under no-disturbance, perfect model assumption, enabling it to be used as an Lyapunov function, thus establishing stability.

1.3 Robust Predictive Control

Nearly all known formulations of MPC algorithms optimize, online, a *nominal* objective function to predict the future plant behavior. Feedback, in the form of plant measurement at the next sampling-time, is expected to account for plant model uncertainty. The extensive amount of literature on stability analysis of MPC algorithms is restricted to the *nominal case*, with no plant model mismatch (Kothare et al.; 1996).

1.3.1 Robust Feasibility

Softening the constraints is one strategy to deal with infeasibility of the MPC algorithms. However, the problem of avoiding constraints violation remains a very important one. If a system is frequently violating its constraints, then it is not doing what it is intended, even though the control algorithm is kept running due to soft constraints. Input constraints usually cannot be violated as they represent a physical constraint e.g. a valve or an actuator. Constraints that should bring an unstable plant to equilibrium cannot be violated if stability is to be guaranteed.

Most often infeasibility arises when there is uncertainty about the plant model or model mismatch. When a non-linear plant is linearized about its equilibrium point, a mismatch often occurs (e.g. when linearizing a model contains the *sin* function for angles larger than, for instance, 5^0). The user then faces a problem of *Robust Feasibility*: Designing the predictive controller so that it maintains feasibility despite uncertainty about the plant being controlled. This thesis will present a strategy for such design.

1.3.2 Why using Linear Matrix Inequalities?

Firstly, Linear Matrix Inequality (LMI)-based optimization problems are fast, and can be solved in practice very efficiently in polynomial time (Boyd et al.; 1994). This makes the LMI method a candidate for online algorithm.

The formulation of the ERPC and SSDMPC, which shall be studied here is different from the typical QP - MPC formulation (1.6a). It is formulated as an LMI optimization problem, which is a convex problem. It shall be shown that the LMI approach guarantees any plant in the uncertainty set will be stabilized. Moreover, satisfaction of constraints is also guaranteed by solving an LMI problem. If it is assumed that the current state satisfies $\mathbf{x}_k^T P \mathbf{x}_k < 1$ for some $P > 0$, the problem of finding $\mathbf{c}_k$ which ensures that $\mathbf{x}_{k+1}^T P \mathbf{x}_{k+1} < 1$ can be posed as an LMI problem.

LMI problems can be posed to handle input constraints and output constraints. LMI problems can also handle certain aspects of performances, with the right definitions (this shall be studied later). Additional LMIs require reformulation of the problem and there are added to the existed set in a simple manner.

1.4 Offline-Online Control strategy

To lower the computational burden from the online MPC, it is possible to 'split' the control tasks to two parts: An *Offline* part, where certain pre-computations are made, and an *Online* part - were the controller is updating the plant in real-time.

The offline portion can be a resource-rich and time-consuming process, but it is not subject to time limitations. It mathematically approximates a region which guarantees, among other things, feasibility and invariance.

The Online part, however, must be efficient, to achieve good system performance even for high dynamic requirements. The conceptual control strategy, to be used in this thesis is given by algorithm 1.1.

Algorithm 1.1 (Offline-Online control strategy, conceptual)

Step 1 (*Offline*): *For a given linear system or a LDI, maximize a region which satisfies certain properties (such as invariance, feasibility)*

Step 2 (*Online*): *Find an optimal control input which uses the solution found in step 1*

1.5 Contributions

The main contribution of this thesis is in the area of predictive control for systems with tight dynamic requirements. It explores a novel work, first presented by Imsland (2002) and Drageset (2002), based upon earlier reports by Kouvaritakis et al. (2000). Briefly, the strategy is to find a region of attraction where the states, once inside the region, will never leave it, and will be feasible. The region is described by mathematical means (ellipsoid). The novel features are as follows

1. The region is enlarged by adapting a nonlinear programming scheme. The algorithm is stabilized and conclusions regarding the Heuristic properties are given (chapter 4).

2. Analysis of a fast, efficient Online strategy, which can replace the slow QP-MPC approach in many cases.

3. Analysis of the Offline and Online by simulations for both SISO and MIMO systems, in several variables, with tight dynamic requirements.

4. A real-time experiment of a fast dynamic, (open-loop) unstable system is conducted, using both the offline and the online strategies, to demonstrate the applicability of the approach (chapter 6).

1.6 Thesis outline

Chapter 2 presents a theory with some important tools and concepts to be used throughout the thesis. The theory of LMI is introduced in section 2.1. Definitions and examples displays the mathematical instruments behind the offline algorithms which will be presented later. The mathematical description and some properties of the ellipsoid are introduced in section 2.2. A brief description of the Sequential Quadratic Programming theory is given by section 2.4.

Chapter 3 introduces an *offline* strategy known as ERPC, developed in the late 90's. The requirements of the offline problem are presented in sections 3.2 - 3.4. The solution to this offline problem is used as a 'hot start' to one in chapter 4. Sections 4.6 to 3.8 provide additional tools and details which can be useful for many systems.

An alternative offline approach, which enlarges the ERPC ellipsoid, is presented in chapter 4. It is a continuation of a novel approach, first introduced by Lars Imsland and Stian Drageset. By making a certain assumption regarding one of the variables, it is possible to introduce additional degrees of freedom, making it possible to enlarge the ellipsoid along certain axes. This assumption, however, also introduces non-linearity to the problem, converting it into a non-convex problem. Adapting some of the methods presented by Fares et al. (2000), a solution is found using an Augmented Lagrange method. Hessian modification, line search and termination criteria are some of the issues discussed in this chapter. Five different examples illustrate the SSDMPC procedure in section 4.8, together with an extensive analysis.

Chapter 5 presents the second part of the offline-online control strategy. The *online* control objectives are discussed first. Then the stability of the control scheme is discussed in section 5.4. Section 5.5 presents the complete offline-online control strategy, introduced by algorithm 1.1. The synthesis of an offline solution and online controller is then applied to some examples, and its behavior demonstrated and analyzed.

The online NR controller, using the offline SSDMPC solution, is tested in a real-time environment in chapter 6. The system tested has high sampling rates, and the open loop is highly unstable. The effectiveness of the online controller is compared to the unconstrained LQ controller, and to the NR controller using ERPC solution.

Chapter 7 discusses and evaluates the offline algorithm, and the NR online controller. It also presents extensions and potential-improvements of the offline-online strategy.

Chapter 2

Theory

A part of the theory behind the offline and online strategies is presented in this chapter. Section 2.1 introduces the Linear Matrix Inequalities theory, which will be used frequently in the next chapters. Section 2.2 explains briefly the mathematical framework of the ellipsoid. This will help to analyse the results and understand them better. The *maximal admissible set* theory is presented in section 2.3. Some concepts from SQP and nonlinear programming are given in section 2.4. Finally, the Lyapunov stability for discrete systems is presented in section 2.5.

2.1 Linear Matrix Inequalities

2.1.1 Definitions

Definition 2.1 (Affine functions)
A function $f : \mathbb{V} \to \mathbb{S}$ is affine if $f(x) = f_0 + T(x)$ where the mapping $T : \mathbb{V} \to \mathbb{S}$ is a linear map, i.e.

$$T(\alpha_1 x_1 + \cdots + \alpha_m x_m) = \alpha_1 T(x_1) + \cdots + \alpha_m T(x_m) \tag{2.1}$$

for all $\mathbf{x} \in \mathbb{V}$ and $\alpha_1 \cdots \alpha_m \in \mathbb{R}$.

Using the definition of the affine function above, we can define the most important concept of LMI in the next way:

Definition 2.2 (Linear Matrix Inequality (LMI))
A linear matrix inequality (LMI) has the form

$$F(x) > 0 \tag{2.2}$$

where F is an affine function mapping a finite dimensional vector space $\mathbb{V}$ to the set $\mathbb{S} := \{M \in \mathbb{R}^{n\times n} |\ M = M^T\}$

We can reformulate definition (2.2) to get a more useful form as done in the next proposition:

Proposition 2.1 (Linear matrix inequalities)
The affine mapping $F : \mathbb{V} \rightarrow \mathbb{S}$ necessarily takes the form $F(x) = F_0 + T(x)$ where $F_0 \in \mathbb{S}$ and $T : \mathbb{V} \rightarrow \mathbb{S}$ is linear transformation. Then we can write

$$T(x) = \sum_{i=1}^{m} x_i F_i$$

which leads to

$$\begin{aligned} F(x) &\triangleq F_0 + \sum_{i=1}^{m} x_i F_i > 0 \\ &= F_0 + x_1 F_1 + \cdots + x_m F_m > 0 \end{aligned} \tag{2.3}$$

where $\mathbf{x} \in \mathbb{R}^m$ is a vector called the decision variables and the symmetric matrices $F_i = F_i^T \in \mathbb{R}^{n \times n}$, $i = 0, \cdots, m$ are given. The inequality symbol in (2.2) means that $F(x)$ is positive definite, i.e. $u^T F(x) u > 0$ for all nonzero $u \in \mathbb{R}^n$.

A very important property of the LMI is that any system of linear matrix inequalities

$$F_1(x) > 0, \ \cdots, F_k(x) > 0$$

can be rewritten as one single LMI (2.2) in the next way

$$F(x) := \begin{bmatrix} F_1(x) & 0 & \cdots & 0 \\ 0 & F_2(x) & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & F_k(x) \end{bmatrix} > 0 \tag{2.4}$$

This has an important implication. If one defines a problem concerning a numerous LMI's, one does not need to reformulate the problem if additional LMI's need to be implemented. A new bigger LMI can be formulated, as a combination matrix of all the LMI's.

The linear matrix inequalities are important since many problems in control, and especially in the context of this thesis, can be formulated in (or reformulated to) this form. Then the problem can be solved in an efficient and reliable way. As will be shown later in the text, the LMI's defines a *convex constraint* and optimization problems involving the minimization or maximization of $f : \mathbb{V} \rightarrow \mathbb{S}$ with $\mathbb{V} := \{x | F(x) > 0\}$ belong to a class of convex optimization problems, and the full power of convex optimization theory can be employed.

The LMI in (2.2) is equivalent to a set of n polynomial inequalities in $\mathbf{x}$. An important property of the LMI (2.2) is that it is a *convex* constraint on $\mathbf{x}$.

Theorem 2.2 (convexity of LMI)
The set $\{x \in \mathbb{R}^{n \times n} | F(x) > 0\}$ is convex
Proof. *For the two dimensional case, if $x_1, \ x_2 \ \in \mathbb{S}$ then*

$$F(\alpha x_1 + (1 - \alpha) x_2) = \alpha F(x_1) + (1 - \alpha) F(x_2) > 0 \tag{2.5}$$

where the equality since F is affine and inequality since $\alpha \geq 0$, $(1 - \alpha) \geq 0$. ■

Theorem 2.2 is very important in the context of this thesis. As will be shown later, some of the optimization problems that we shall try to solve have convex cost functions, subjected to LMI sets. Since these sets are convex (as proven in theorem 2.2), the problems are then convex and a global solution is found. Even in cases where the problems are not fully convex, it can be transformed to another problems, with convex cost functions subjected to LMI constraints, and solved sequentially. This can be effective since each sequence solves a convex problem and finds a global solution.

Another important theorem is stated below.

Theorem 2.3 (Schur complements)
Let $F : \mathbb{V} \to \mathbb{S}$ be an affine function which is partitioned according to

$$F(x) = \begin{bmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{bmatrix} \tag{2.6}$$

where F_{11} is square. Then $F(x) > 0$ is equivalent to both

$$\begin{cases} F_{11} > 0 \\ F_{22} - F_{12}(F_{11})^{-1}F_{21} > 0 \end{cases}, \qquad \begin{cases} F_{22} > 0 \\ F_{11} - F_{12}(F_{22})^{-1}F_{21} > 0 \end{cases} \tag{2.7}$$

Theorem (2.3) is very useful since it allows us to convert some non-linear equations in the form of (2.7) to *linear* matrix inequalities (2.6). In particular, it follows that non-linear inequalities of the form (2.7) define a convex constraint on the variable $\mathbf{x}$. This can be illustrated in the next example:

Example 2.1 (Schur complement) *Consider the inequalities*

$$\begin{cases} Q_z > 0 \\ Q_z^{-1} - \Psi^T(Q_z)^{-1}\Psi > 0 \end{cases} \tag{2.8}$$

where $Q_z = Q_z^T$ is a positive definite matrix. The second inequality (2.8) is non-linear. However, it can be transformed to linear matrix inequality. Since Q_z is positive definite non-singular matrix, we can multiply the second inequality by Q_z from the left and right to get

$$\begin{cases} Q_z > 0 \\ Q_z - Q_z\Psi^T(Q_z)^{-1}\Psi Q_z > 0 \end{cases} \tag{2.9}$$

Then we can use theorem (2.3) to rewrite the inequalities above as

$$\begin{bmatrix} Q_z & Q_z\Psi^T \\ \Psi Q_z & Q_z \end{bmatrix} > 0 \tag{2.10}$$

and the last inequality is linear, and can be a part of a convex set $\mathbf{x} \in \mathcal{X}_{LMI}$.

2.1.2 Solving the LMI's

Only general ideas of solving the LMI systems are given in this section. For more details about LMI solvers and the theory behind them the reader is referred to Scherer and Weiland (1999) and Boyd et al. (1994).

The two most used algorithm today are the Ellipsoid method and the Interior Point method. The former is very simple and not always as effective as the interior point, but can detect infeasible point easily. The Ellipsoid algorithm starts with an ellipsoid $\mathcal{E}^{(0)}$ that is guaranteed to contain an optimal point. If such point does not exist, the problem is infeasible and the algorithm stops. If the point exists, then a *cutting plane* is computed, which slices the ellipsoid to two parts. The optimal point belongs to one of the halves. In the next iteration of the algorithm another ellipsoid $\mathcal{E}^{(1)}$ that contains this half is computed. $\mathcal{E}^{(1)}$ is then guaranteed to contain an optimal point. The process is then repeated.

The Interior Point method has been developed recently and is much more effective than the ellipsoid method. The main idea is to replaced the constrained optimization problem

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & F(x) > 0 \end{aligned} \tag{2.11}$$

with the unconstrained optimization problem

$$F_t(x) := tf(x) + \phi(x) \tag{2.12}$$

where $t > 0$ is the penalty parameter and ϕ is a barrier function. The main idea is to determine the minimizer $x(t)$ of f_t and to consider the behavior of $x(t)$ as a function of the penalty parameter $t > 0$

The Matlab *LMI toolbox* has ready packages which solve the feasibility problem (*feasb.m*) and solving a linear objective optimization problem subjected to a set of LMI's i.e.

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & F(x) > 0 \end{aligned} \tag{2.13}$$

The package is called *mincx* and will be used in the LOP algorithm. For more details about the Matlab toolbox the reader is referred to Gahinet et al. (1995).

2.2 The Ellipse

The parametric description of a two dimensional ellipse is given by

$$\begin{cases} x_1 = a\cos(t) \\ x_2 = b\sin(t) \end{cases} \tag{2.14}$$

where $t \in \mathbb{R}$ is the parameter, a and b are constants. Then the ellipse curve is the intersection of the two lines in (2.14). An ellipsoid can be described as the image of the unit ball under an affine mapping with symmetric positive definite matrix:

$$\mathcal{E} := \{\mathbf{x} \mid (\mathbf{x}-\mathbf{x}_c)^T P(\mathbf{x}-\mathbf{x}_c) \leq 1 \ \} \tag{2.15}$$

where $P = P^T > 0$ and $\mathbf{x},\ \mathbf{x}_c \in \mathbb{R}^n$. Define $P = Q^{-1}$, the volume of $\mathcal{E}$ is then proportional to $\det Q^{1/2}$ and its diameter is $2\lambda_{max}(Q^{1/2})$ (Boyd et al. (1994)). For the two dimensional case, it is possible to calculate the parameters a and b in (2.14) from the Q^{-1} matrix by the following way

$$a = \frac{1}{\sqrt{\lambda_1}}\ , \quad b = \frac{1}{\sqrt{\lambda_2}} \tag{2.16}$$

and the curve itself is found by

$$\mathbf{x} = \mathbf{V} \cdot [\ a\cos(t) \ \ b\sin(t)\ ]^T \tag{2.17}$$

where $\mathbf{V} \in \mathbb{R}^{2\times 2}$ is the eigenvectors matrix of Q^{-1}. Notice that (2.15) describes a n dimensional ellipsoid whilst (2.17) is the projectile of the ellipsoid's curve in the $x_1,\ x_2$ plane.

2.2.1 The Volume

As has been mentioned above, the volume of the ellipsoid $E_x = \{x | x^T P_x x \leq 1\}$ is proportional to the term $(\det P_x)^{\frac{1}{2}}$ (proof is given in Boyd et al. (1994)). Using this fact, it is possible to compare between sizes of different ellipsoid (or area in 2D case and volume in three dimensional). We shall denote the term $(\det\ P_x)^{\frac{1}{2}}$ as the *Volume Factor*, and use it later to evaluate whether a size of ellipsoid is satisfactory.

Since the volume of the ellipsoid is proportional to the volume factor, it is straight forward to define an optimization problem for maximizing the volume of the ellipsoid. Define the objective as

$$\begin{aligned} \max\ \text{Volume}\ \mathcal{E} &\sim \max\ (\det P_x)^{-\frac{1}{2}} && (2.18)\\ &= \max (\det\ P_x)^{-\frac{1}{2}} && (2.19)\\ &\sim \max \log\det (P_x)^{-1} && (2.20)\\ &= \min\ -\log\det (P_x)^{-1} && (2.21)\\ &= \min\ \log\det (P_x) && (2.22) \end{aligned}$$

Thus, the maximum volume of ellipsoid can be found by solving the next optimization problem

$$\begin{array}{ll} \min & \log\det(P_x) \\ s.t. & P_x > 0, \quad P_x = P_x^T \ \in \mathbb{R}^{n\times n} \\ & x^T P_x x \leq 1, \quad x \in \mathbb{R}^n \end{array} \tag{2.23}$$

This will be the basic problem of finding maximum volume of an ellipsoid subjected to certain constraints.

2.2.2 The P Condition Number

Recall that the diameter of $\mathcal{E}$ is defined as twice the maximum semi-axis length, or

$$2\lambda_{max}(P) \tag{2.24}$$

and each of the ellipsoid axis is found by

$$a_i = \frac{1}{\sqrt{\lambda_i}}, \qquad i = 1, ..., n \tag{2.25}$$

If we consider i in (2.25) to be the index of the largest (or smallest) eigenvalue of P, we can write

$$a_{max} = \frac{1}{\sqrt{\lambda_{min}}}, \tag{2.26}$$

$$a_{min} = \frac{1}{\sqrt{\lambda_{max}}} \tag{2.27}$$

The division of the largest ellipsoid axis by the smallest axis can be a measure for the shape of the ellipsoid. If this ratio is near the value 1, then the shape is rounded, since the largest axis and the smallest one are nearly equal. This ratio is written as follow

$$\frac{a_{max}}{a_{min}} = \sqrt{\frac{\lambda_{max}(P)}{\lambda_{min}(P)}} \tag{2.28}$$

but since $P = P^T > 0$, it can be written as

$$\begin{aligned} \left(\frac{a_{max}}{a_{min}}\right)^2 &= \left(\frac{\lambda_{max}(P)}{\lambda_{min}(P)}\right) \\ &= \left(\frac{\lambda_{max}(P^TP)}{\lambda_{min}(P^TP)}\right)^{1/2} \\ &\triangleq \kappa(P) \end{aligned} \tag{2.29}$$

where $\kappa(P)$ is the condition number of the matrix P. Hence, it is most desirable that the condition number of the ellipsoid matrix P will be close to one, or at least of a low value. It can indicate which ellipsoid will be thin and long, and which one will be rounded. The higher the condition number is, the thinner the ellipsoid becomes. This can be demonstrated by the next example:

Example 2.2 (Condition number and volume factor) *Consider the two ellipsoids $\mathcal{E}_1$ and $\mathcal{E}_2$ created using the matrices P_1 and P_2 accordingly,*

$$P_1 = \begin{bmatrix} 0.912 & -0.227 & -0.059 \\ -0.227 & 0.364 & 0.115 \\ -0.059 & 0.114 & 0.04 \end{bmatrix} \qquad P_2 = \begin{bmatrix} 0.13 & 0 & 0 \\ 0 & 0.011 & 0.012 \\ 0 & 0.012 & 0.033 \end{bmatrix} \tag{2.30}$$

The condition number of P_1 *is* $\kappa(P_1) = 309.8$ *while the condition number* $\kappa(P_2) = 22.2$. *Thus we can expect that* $\mathcal{E}_2$ *will be rounded ellipsoid, i.e the radius which creates a ball around the origin is large, while* $\mathcal{E}_1$ *will be long and thin ellipsoid, as it has large condition number. It will therefore have relatively small radius around the origin. This can be verified by figure 2.1. The blue ellipsoid has larger condition number than the grey, it is thus thinner and longer. Notice that the volume of* $\mathcal{E}_2$ *is larger than the volume of* $\mathcal{E}_1$. *This can be verified by computing the volume factor* $V_f(P_2) = 183.83$ *while* $V_f(P_2) = 31.48$. *These values are not the actual volume of the ellipsoids, but are a tool for comparing the sizes.*

Example 2.2 illustrates the significant of the volume factor and the condition number of P, and these quantities will be used as measurement tools later in the analysis of the examples and the case study. Notice that the projections of $\mathcal{E}_1$ in example 2.2 can be deceptive: The $x - y$ projection is rounded and can (along with the $x - z$ projection) lead to the wrong conclusion that $\mathcal{E}_1$ is a rounded ellipsoid. It is easy to see from the $y - z$ projection that this is not the case. However, it will not be as easy (or even impossible) to draw any conclusion on the shape of a higher dimension ellipsoids (e.g. five or six dimensions) just by examine the different 3D projections. The condition number is therefore an excellent measurement tool for that.

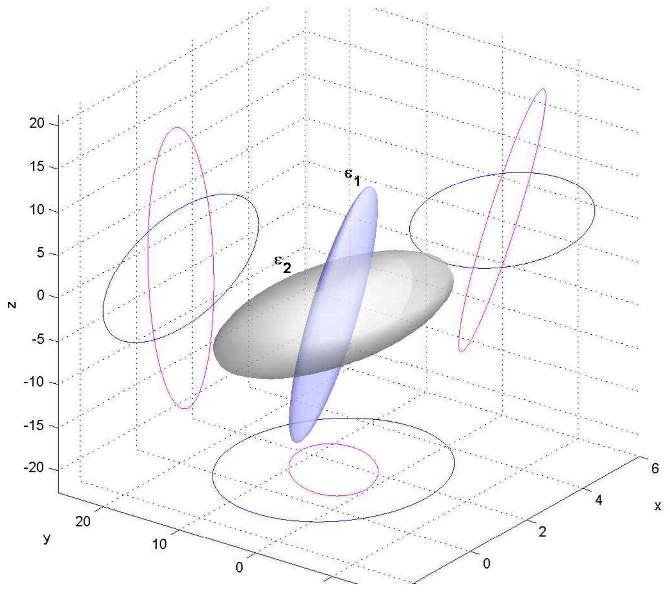


Fig. 2.1: Ellipsoids $\mathcal{E}_1$ and $\mathcal{E}_2$, example 2.2

2.3 Maximal Output Admissible Sets

The methods that will be presented in chapters 3 and 4 finds sets, regions (invariant and feasible) whose nature is chosen to fit with the mathematical framework being used, rather than the *actual* regions. This fact makes the sets conservative, sometimes too conservative to be used in practice. That can lead to the wrong conclusion that no such practical set exists. Consider for instance the ellipsoid region $\Omega = \{x \in \mathbb{R}^{m_x} | x^T P x < 1\}$ discussed in section 2.2. If only a local solution P is used to describe the ellipsoid set, then there might be many other larger ellipsoids that can be found. Even if the largest ellipsoid is found, it is still an ellipsoid, not necessary the shape of the *real* region.

The key for reduction of such conservation is to estimate the real set in which the state must be confined, as accurate as possible. Gilbert and Tan (1991) presented in their paper a strategy for such approximation. If for a time-invariant linear system

$$\begin{aligned}\mathbf{x}(k+1) &= \Phi\mathbf{x}(k) = (A - BK)\,\mathbf{x}(k)\\ \mathbf{y}(k) &= H\mathbf{x}(k)\end{aligned}$$

has to satisfy an output constraint $\mathbf{y}(k) \in \mathbf{Y}$ for each k, then a set $\mathcal{O}$ is output admissible if

$$\mathbf{x}(0) \in \mathcal{O} \Rightarrow \mathbf{y}(k) \in \mathbf{Y} \quad \forall k > 0$$
$$\Leftrightarrow$$
$$\mathbf{x}(0) \in \mathcal{O} \Rightarrow H\Phi^k\mathbf{x}(0) \in \mathbf{Y} \quad \forall k > 0$$

Furthermore, the *maximal* output admissible set $\mathcal{O}_\infty$ is the largest of such sets. If $\mathbf{Y}$ is a convex polytope then the problem of finding $\mathcal{O}_\infty$ is a maximization linear programming problem, so it can be solve efficiently.

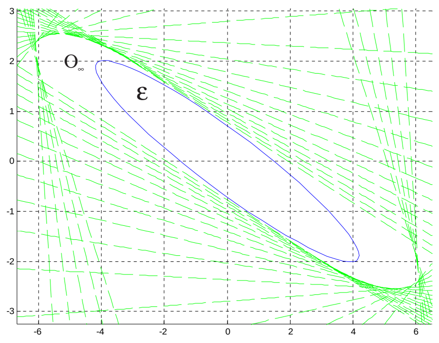


Fig. 2.2: Maximal admissible set (green) and another (conservative) set (blue).

Example 2.3 (Maximal Admissible set) *This concepts can be illustrated by the next simple linear closed-loop system*

$$\begin{aligned}\mathbf{x}(k+1) &= \Phi\mathbf{x}(k) = \begin{pmatrix} 1 & 0.1 \\ -0.073 & 0.85 \end{pmatrix}\mathbf{x}(k) \\ \mathbf{y}(k) &= H\mathbf{x}(k) = (1 \quad 0)\,\mathbf{x}(k)\end{aligned}$$

then the maximal admissible set $\mathcal{O}_\infty$ is shown in figure 2.2 bounded by the green dashed lines. Another set $\mathcal{E}$, which is more conservative, is the blue ellipsoid.

Gilbert and Tan (1991) shows and proves that this maximal set $\mathcal{O}_\infty$ is convex, bounded is Φ, H is observable pair, and containing the origin in it interior (see figure 2.2). Moreover, $\mathcal{O}_\infty$ scales with $\mathbf{Y}$: if $\mathbf{Y}$ is scaled by a constant γ, then $\mathcal{O}_infty$ is scaled by the same ratio, without changing shape.

The set $\mathcal{O}_t$ is defined as the set where $H\Phi^k\mathbf{x}(0) \in \mathbf{Y}, \quad \forall k = 1, 2, ...t$. It is obvious that is $t_2 > t_1$ then $\mathcal{O}_{t_2} \subset \mathcal{O}_{t_1}$, that is, $\mathcal{O}_{t_2}$ is at least smaller than the former. Now if we increase t at each iteration, $t_i > t_{i-1}$, then the sets should get smaller and smaller. If they stop getting smaller, i.e. $\mathcal{O}_t = \mathcal{O}_{t-1}$ then the maximal admissible set is found $\mathcal{O}_\infty = \mathcal{O}_t$, assuming it can be determined. The complete algorithm to find $\mathcal{O}_\infty$ is given by Gilbert and Tan (1991) and will not be repeated here.

2.4 Concepts from Optimization Theory

This section introduces some of the concepts behind the Non-Linear Programming (NLP), the penalty methods. It is important to understand this concept as it is a essential components of the algorithm presented in chapter four.

2.4.1 Penalty functions

Consider the next optimization problem:

$$\begin{aligned}&\min f(\mathbf{x}) \\ &\text{subject to } h_i(\mathbf{x}) = 0 \qquad i \in \mathcal{E}\end{aligned} \tag{2.31}$$

We consider here only equality constraints, since the problem for inequality constraints can be solved by introducing a slack variable s_i and replacing the inequalities $h_i \geq 0, \quad i \in \mathcal{I}$ as is done by Nocedal and Wright (1999) (chapter 17). The main idea in the penalty methods is to approximate a constrained minimization problem (2.31) by a problem which is considerably easier to solve. Naturally by solving an approximation problem, we can only expect to obtain an approximate solution of the original problem. However, it is possible to construct a sequence of approximate problems which converges to the original problem in the limit. In practice, a solution can be obtained after a *finite* number of approximate problems. It is possible to utilize information obtained from each approximate problem in the solution of the next approximate problem, to converge to the solution faster.

The penalty methods eliminate some or all of the constraints and add to the objective function a penalty term which prescribes a high cost to infeasible points. The parameter μ, associated with these methods, determines the severity of the penalty and as a consequence the extent to which the resulting unconstrained problem approximates the original constrained problem.

The choice of the penalty function is not obvious. A penalty (or 'merit') function that is found suitable to a certain system can be unsuitable for another. The ℓ_1 merit function given by

$$\phi_1(\mathbf{x};u) = f(\mathbf{x}) + \mu \left\| h(\mathbf{x} \right\|_1 \tag{2.32}$$

where $\mu > 0$ is a penalty parameter. We shall exclude this merit function as it is not differential everywhere, those has problem with global convergence.

Augmented Lagrange method

The second penalty function is Fletcher's augmented Lagrangian function, given by

$$\mathcal{L}_A = f(\mathbf{x}) - \lambda h(\mathbf{x}) + \frac{1}{2}\mu h^2(\mathbf{x}) \tag{2.33}$$

where $\mu > 0$ denotes a *penalty parameter* and λ is the Lagrange multiplier. The augmented Lagrangian (2.33) differs from the standard Lagrangian by the presence of the quadratic penalty function $\frac{1}{2}\mu h^2(\mathbf{x})$.

An important disadvantage of the Lagrangian methods is that they require a good starting point in order to converge to an optimal solution, i.e. they converge only locally. Furthermore, if the number of approximate problems is large, then the problem can become ill-conditioned, making it difficult to obtain a solution due to numerical inaccuracies (Bertsekas; 1996).

2.5 Lyapunov Stability of Discrete-Time systems

In the section follows, the Lyapunov stability theorem for discrete-time systems is given. For discrete-time systems we use the forward difference

$$\Delta V(\mathbf{x}(k)) = V(\mathbf{x}(k+1)) - V(\mathbf{x}(k)) \tag{2.34}$$

The next theory gives the conditions necessary for a discrete-time system to be stable.

Theorem 2.4 (Stability of discrete-time systems)
Consider the discrete time system

$$\mathbf{x}(k+1) = \mathbf{f}(\mathbf{x}(k)) \tag{2.35}$$

where

$$\mathbf{x} \in \mathbb{R}^n$$
$$\mathbf{f}(\mathbf{x}) \in \mathbb{R}^n \text{ with property that } \mathbf{f}(\mathbf{0}) = \mathbf{0}$$

suppose there exists a scalar function V(x) continuous in x such that

- $V(\mathbf{x}) > 0 \quad \forall \quad \mathbf{x} \neq 0$
- $\Delta V(\mathbf{x}) < 0 \quad \forall \quad \mathbf{x} \neq 0$
- $V(\mathbf{0}) = 0$

locally in region Ω, then the equilibrium state $\mathbf{x} = \mathbf{0}$ is asymptotically stable and $V(\mathbf{x})$ is a Lyapunov function.

If the system is described in terms of state-space equations, the asymptotic stability of a discrete-time system obtained by discretizing a continuous-time system is equivalent to that of the original continuous-time system (Ogata; 1995).

Consider a continuous-time system

$$\dot{\mathbf{x}} = A\mathbf{x}$$

and the corresponding discrete time system

$$\mathbf{x}((k+1)T) = \Phi\mathbf{x}(kT), \qquad \Phi = e^{AT}$$

If the continuous-time system is asymptotically stable then

$$\|\Phi^n\| \to 0 \qquad \text{as } n \to \infty$$

and the discretized system is also asymptotically stable. This is because if the λ_iś are the eigenvalues of A then the $e^{\lambda_i T}$ś are the eigenvalues of Φ and $|e^{\lambda_i T}| < 1$ if $\lambda_i T$ is negative.

Chapter 3

ERPC

The introduction chapter indicated that online MPC based on quadratic programming can be demanding, both resources and computational time. Short control horizons can reduce the computational time, but risk of infeasibility increases. Long control horizon yield good performance and guarantee stability, but on the cost of time and resources. An alternative strategies should be considered for systems which have limited resources and/or high update frequency.

The Efficient Robust Predictive Control (ERPC) is an algorithm which computes *offline* an ellipsoid, centered about the origin. This ellipsoid will later be used by a fast online controller which is one alternative strategy to the QP-MPC. The ERPC algorithm maximizes such an ellipsoid, in order to expand the feasible operating range of system. ERPC is discussed with details in Kouvaritakis et al. (2000). It will be used later in the thesis to initialize another offline strategy, and will be used as a comparison tool.

3.1 Problem formulation

Consider the next discrete-time state space system:

$$\mathbf{x}(k+1) = A\mathbf{x}(k) + B\mathbf{u}(k), \qquad \mathbf{u} \in \mathbb{R}^{m_u}, \quad \mathbf{x} \in \mathbb{R}^{m_x} \tag{3.1}$$

$$-\bar{u}_i \leq u_i(k) \leq \bar{u}_i, \qquad i = 1, ..., m_u \tag{3.2}$$

$$-\bar{x}_i \leq x_i(k) \leq \bar{x}_i \qquad i = 1, ..., m_x \tag{3.3}$$

where $\mathbf{x}(k)$ is the state vector at iteration k, $\mathbf{u}(k)$ is the input vector, both are bounded within symmetric constraints. The constraints on the inputs and the states introduce non-linearities to the system. If the eigenvalues of matrix A lies outside the unit circle, then the system is unstable. We can calculate feedback controller gain K which will move the eigenvalues to a desired place, inside the unit circle, assuming the system is controllable. It can be done by pole-placement method or Linear Quadratic controller which gives optimal K matrix ignoring the constraints of course. Introducing the feedback control:

$$\mathbf{u}_k = -K\mathbf{x}_k + \mathbf{c}_k \tag{3.4}$$

with $\mathbf{c}_k$ being the vector of perturbations away from the optimal control. We can then rewrite the system (3.1) to

$$\mathbf{x}_{k+1} = \Phi \mathbf{x}_k + B\mathbf{c}_k \tag{3.5}$$

where $\Phi = A - BK$. Letting $\mathbf{f}$ denote the vector of future perturbations away from the optimal control, $\mathbf{c}_k$, over the control horizon N_c, the dynamics of (3.5) can be described as

$$\mathbf{z}_{k+1} = \Psi_k \mathbf{z}_k \tag{3.6a}$$

where $\mathbf{z} \in \mathbb{R}^{m_x + m_u N_c}$ and

$$\mathbf{z}_k = \begin{bmatrix} \mathbf{x}_k \\ \mathbf{f}_k \end{bmatrix}, \quad \mathbf{f}_k^T = \begin{bmatrix} \mathbf{c}_k^T & \mathbf{c}_{k+1}^T & \cdots & \mathbf{c}_{k+N_c-1}^T \end{bmatrix} \tag{3.6b}$$

$$\Psi_k = \begin{bmatrix} \mathbf{\Phi} & [\mathbf{B} \ \ 0 \cdots 0] \\ \mathbf{0} & M \end{bmatrix}, \quad M = \begin{bmatrix} 0_{m_u} & I_{m_u} & 0_{m_u} & 0_{m_u} & \cdots & 0_{m_u} \\ 0_{m_u} & 0_{m_u} & I_{m_u} & 0_{m_u} & \cdots & 0_{m_u} \\ \vdots & \ddots & & \ddots & & 0_{m_u} \\ 0_{m_u} & \ddots & 0_{m_u} & 0_{m_u} & 0_{m_u} & I_{m_u} \\ 0_{m_u} & \cdots & 0_{m_u} & 0_{m_u} & 0_{m_u} & 0_{m_u} \end{bmatrix} \tag{3.6c}$$

here $\mathbf{z}_k$ is the augmented state vector, Ψ is the augmented transition vector, and both I_{m_u} and 0_{m_u} have dimensions $m_u \times m_u$. The matrix M accommodates the time recession of $\mathbf{f}_k$. The equivalent gain vector for the augmented system, $\tilde{K}$ is

$$\tilde{K} = [K \ \ I_{m_u} \ \ 0_{m_u} \cdots \ 0_{m_u}] \tag{3.7}$$

3.2 Invariant set

The concept of invariant set is important in this context, since it allows us to show stability of the system (3.1).

The region $\mathcal{E}_{zx}$ is an invariant set for $\mathbf{x}_{k+1} = \Phi_k \mathbf{x}_k$ if

$$\mathbf{x}_k \in \mathcal{E}_{zx} \quad \Rightarrow \quad \mathbf{x}_{k+1} \in \mathcal{E}_{zx} \tag{3.8}$$

where the ellipsoid is defined as $\mathcal{E}_{zx} := \{\mathbf{x} | \mathbf{x}^T (T_x Q_z T_x^T)^{-1} \mathbf{x} \leq 1\}$, and T_x is defined by $\mathbf{x} = T_x \mathbf{z}$.

This can be interpreted in the next way: If at some iteration k, $\mathbf{x}_k$ is in the set $\mathcal{E}_{zx}$, then all the future values of $\mathbf{x}_{k+i} \quad i \in \{0, 1, 2...\}$ remain in this set.

Recall that the objective is to find a region, ellipsoid for (3.6a) where certain conditions apply within it. One of the conditions which the ellipsoid must fulfill is the invariance property. The ellipsoid will be invariant if after some time-step $k \geq k_0, \quad z_k \in \mathcal{E}_z$, the next inequality is feasible:

$$\mathbf{z}_k^T Q_z^{-1} \mathbf{z}_k \geq \mathbf{z}_{k+1}^T Q_z^{-1} \mathbf{z}_{k+1} \tag{3.9}$$

using (3.6a), the equation above can be rewritten as

$$\mathbf{z}_k^T Q_z^{-1}\mathbf{z}_k - \mathbf{z}_{k+1}^T Q_z^{-1}\mathbf{z}_{k+1} \geq \mathbf{0} \tag{3.10}$$
$$\mathbf{z}_k^T Q_z^{-1}\mathbf{z}_k - (\Psi\mathbf{z}_k)^T Q_z^{-1}(\Psi\mathbf{z}_k) \geq \mathbf{0} \tag{3.11}$$
$$\mathbf{z}_k^T Q_z^{-1}\mathbf{z}_k - \mathbf{z}_k^T \Psi^T Q_z^{-1}\Psi\mathbf{z}_k \geq \mathbf{0} \tag{3.12}$$
$$\mathbf{z}_k^T \left(Q_z^{-1} - \Psi^T Q_z^{-1}\Psi\right)\mathbf{z}_k \geq \mathbf{0} \tag{3.13}$$

thus the only possible way for (3.13) to be positive semi-definite is when $\mathbf{z}_k \neq 0$ and the term

$$Q_z^{-1} - \Psi^T Q_z^{-1}\Psi \geq \mathbf{0} \tag{3.14}$$

is positive semi-definite. The Schur complement theorem (2.3) can now be applied to (3.14) in order to get the next LMI in a matrix form:

$$\begin{pmatrix} Q_z & \Psi^T \\ \Psi & Q_z^{-1} \end{pmatrix} \geq 0 \tag{3.15}$$
$$Q_z > 0$$
$$\Leftrightarrow \begin{pmatrix} Q_z & Q_z^T\Psi^T \\ \Psi Q_z & Q_z \end{pmatrix} \geq 0 \tag{3.16}$$
$$Q_z > 0$$

The last term was proven in 2.1. This LMI equation will be used to find the largest set $\mathcal{E}_x$ in order to assure that the set will be invariant.

3.3 Feasibility

Feasibility of the solution obtained by setting an upper and lower bounds on $\mathbf{u}_k$:

$$|u_i| \leq d_i \tag{3.17}$$

where $\mathbf{d}$ is the bound vector and i represents the rows of $\mathbf{u}_k$ and $\mathbf{d}$. Equation above can be rewritten as

$$|u_i|^2 = |\tilde{K}_i z|^2 = |\tilde{K}_i Q_z^{1/2} Q_z^{-1/2}\mathbf{z}|^2 \leq \|\tilde{K}_i Q_z^{1/2}\|^2 \cdot \|Q_z^{-1/2}\mathbf{z}\|^2 \leq d_i^2 \tag{3.18}$$

The first norm can be written as

$$\|\tilde{K}_i Q_z^{1/2}\|^2 = (\tilde{K}_i Q_z^{1/2}) \cdot (\tilde{K}_i Q_z^{1/2})^T = \tilde{K}_i Q_z \tilde{K}_i^T \tag{3.19}$$

while the second norm has an upper bound

$$\|Q_z^{-1/2}\mathbf{z}\|^2 = (Q_z^{-1/2}\mathbf{z})^T \cdot (Q_z^{-1/2}\mathbf{z}) = (\mathbf{z}^T (Q_z^{-1/2})^T Q_z^{-1/2}\mathbf{z}) = \mathbf{z}^T Q_z^{-1}\mathbf{z} \leq 1 \tag{3.20}$$

this holds since Q_z is symmetric and the last inequality is the ellipsoid definition (2.15). Then (3.18) can be rewritten as

$$\tilde{K}_i Q_z \tilde{K}_i^T \cdot \|Q_z^{-1/2}\mathbf{z}\|^2 \leq \tilde{K}_i Q_z \tilde{K}_i^T \leq d_i^2 \tag{3.21}$$

We can thus define the bound of u_i by

$$
\begin{aligned}
& d_i^2 - \tilde{K}_i \; Q_z \; \tilde{K}_i^T \geq 0 \\
\Leftrightarrow \;\; & d_i^2 - [K_i \;\; e_i \;] \; Q_z \; [K_i \;\; e_i]^T \geq 0
\end{aligned} \tag{3.22}
$$

where e_i denotes the ith column of the identity matrix. This bound is somewhat conservative, since the term $\|Q_z^{-1/2}\mathbf{z}\|^2$ can be low.

3.4 Maximizing the ellipsoid

The objective is to find a maximum ellipsoid $\mathcal{E}_{zx}$, $\mathbf{x} \in \mathbb{R}^{mx}$, centered at the origin of the $\mathbf{x}$-space and defined as

$$
\mathcal{E}_{zx} = \{\mathbf{x} | \mathbf{x}^T Q_{zx}^{-1} \mathbf{x} < 1\} \tag{3.23}
$$

It is equivalent to finding the maximum of the projection of $\mathcal{E}_z$ in the $\mathbf{x}$-space. Recalling that $\mathbf{z} = T\mathbf{x}$, we can rewrite the ellipsoid to

$$
\mathcal{E}_{xz} = \{\mathbf{x} | \mathbf{x}^T (TQ_zT)^{-1} \mathbf{x} < 1\} \tag{3.24}
$$

where $Q_{zx}^{-1} = TQ_zT^T$ (Kouvaritakis et al. (2000)). Since the volume of $\mathcal{E}_{zx}$ is proportional to $(det \; Q_{zx})^{-1/2}$, maximizing $log \; det \; Q_{zx}^{-1}$ is the same as maximizing the volume of $\mathcal{E}_{zx}$. We can then define the maximum problem as

$$
\max Vol(\mathcal{E}_{zx}) \tag{3.25}
$$

$$
\Leftrightarrow \;\; \min \; - log \; det \; (TQ_zT^T) \tag{3.26}
$$

$$
\Leftrightarrow \;\; \min \; log \; det \; (TQ_zT^T)^{-1} \tag{3.27}
$$

Equation 3.27 is the objective function as given in Kouvaritakis et al. (2000). Notice that log det is a *convex* problem and have a global solution (Boyd et al.; 1994).

3.5 Offline problem

We can now summarize all the results as the offline problem in the following way:

$$
\begin{aligned}
\min \quad & log \det \; (TQ_zT^T)^{-1} && \text{Max. volume} && (3.28a) \\
s.t. \quad & && && \\
& \begin{bmatrix} Q_z & Q_z\Psi^T \\ \Psi Q_z & Q_z \end{bmatrix} \geq 0 && \text{Invariance} && (3.28b) \\
& d_i^2 - [K_i \;\; e_i \;] \; Q_z \; [K_i \;\; e_i]^T \geq 0 && \text{Feasibility} && (3.28c) \\
& F(Q_z) \geq 0 && && (3.28d)
\end{aligned}
$$

where Ψ is given by (3.6a), $\mathbf{x} = T\mathbf{z}$. The inequality (3.28d) should be interpreted as an affine function representing polytopic state constraints, if presented (see section

3.6). This problem is convex as well as *tractable* and can be solved very efficiently in polynomial-time (Boyd et al. (1994)). Since (3.28) does not depend on the current state, the algorithm can be implemented offline.

We shall denote the set of LMIs (3.28b-3.28d) as $\mathcal{X}_{LMI}$. Any additional LMI which contains Q_z as a variable can be added to the set and be solved by the ERPC.

3.6 Largest invariant ellipsoid containing a polytope

If the initial position $\mathbf{x}_0$ is not on the set i.e. $\mathbf{x}_0 \not\subseteq \mathcal{E}_{zx}$, then it is possible to increase the control horizon N_c such that the ellipsoid $\mathcal{E}_{zx}$ becomes larger. But as have been shown by Drageset (2002), the ERPC method tends to produce long narrow ellipsoids with increasing N_c. This sets tend to be unuseful in most cases as they cover narrow areas, usually which contain values that cannot be used due to physical limitations in the states x_i (angles, deflections, current etc.). This can be demonstrated by the next example:

Example 3.1 *Consider the next single-variable system used in Kouvaritakis et al. (2000) and Kothare et al. (1996)*

$$\begin{aligned}
&\begin{bmatrix}\theta_{k+1}\\ \omega_{k+1}\end{bmatrix} = A\begin{bmatrix}\theta_k\\ \omega_k\end{bmatrix} + Bu_k\\
&A = \begin{pmatrix}1 & .1\\ 0 & 1\end{pmatrix}, \qquad \mathbf{B} = \begin{pmatrix}0\\ 0.0787\end{pmatrix}\\
&Q = diag(10\ ,10) \quad R = 0.1
\end{aligned} \tag{3.29}$$

with $-1 < u < 1$. The results for $N_c = 1, 2, 4$ are given in figure 3.1. It is easily seen

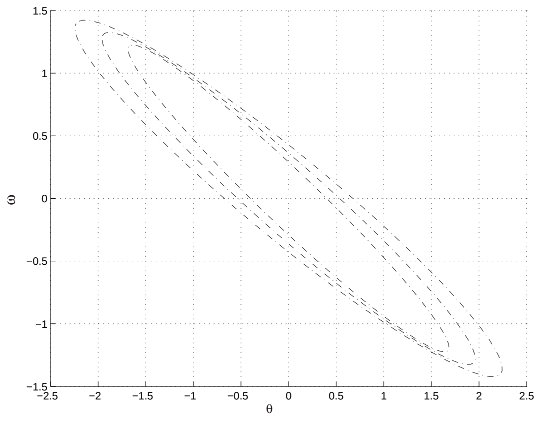


Fig. 3.1: The ellipsoids are getting bigger, with increasing $N_c = 1, 2, 4$ values, but are still narrow.

that although the initial value $\mathbf{x}_0 = [-1\ 0]^T$ lies not far from the origin, it is outside the *feasible region (the set $\mathcal{E}_{zx}$) and thus it cannot be solved by the efficient online problem. Moreover, points that do not lie in the vicinity of the origin (example $\mathbf{x} = (-2, 1.3)^T$) might not be usable since the system might not operate at such high angular velocities ($x_2 = \omega = 1.3\frac{rad}{sec} = 74.4\frac{deg}{sec}$) because of physical operating range of the system.*

Thus it is often required to find a feasible solution in the vicinity of the origin, e.g in a certain radius from the origin. This can be done in several ways. One strategy is to enforce certain states to be bounded inside certain values. In Example 3.1 it is possible to confine the state x_2 to be $|x_2| < 1$, and hope that the algorithm will find a solution $\mathcal{E}_{zx}$ which is more round than the narrow ellipsoid in figure 3.1. This approach however cannot guarantee good results and will usually reduce the ellipsoid size. Another strategy is to add certain LMI's which will enforce the ROA to contain certain points inside. This new set of LMI's may be infeasible if the points are too far from the origin, thus problem (3.28) will not have any feasible solution at all.

It is possible to contain a polytope inside the region of attraction. By adjusting the vertexes of the polytope in a proper way, it is possible to achieve a certain control on the size and shape of the region.

Consider a polytope described by its vertices,

$$\mathcal{P} = \mathbf{Co}\{v_i, ..., v_p\} \tag{3.30}$$

The ellipsoid E_{zx} contains the polytope $\mathcal{P}$ if and only if

$$v_i^T Q^{-1} v_i \leq 1, \quad i = 1, ..., p \tag{3.31}$$

This result is based on (Boyd et al. (1994) page 69) but has been adapted for *largest* invariant ellipsoid. Inequality (3.31) may be expressed as an LMI using Schur complement

$$\begin{pmatrix} 1 & v_i^T \\ v_i & Q \end{pmatrix} \geq 0, \quad i = 1, ..., p \tag{3.32}$$

Confining a polytope inside $\mathcal{E}_{zx}$ is done by adding the LMI (3.32) to the ERPC problem definition (3.28) or any other similar offline algorithm.

3.7 Scaling the system

In many cases, the system has to be scaled in order to use the ERPC algorithm effectively. We shall consider here the state space representation of a linear system 3.1. If the states are not properly scaled, then any resulting invariant set will have large condition number, and can yield problems in online computation and feedback control. Scaling affects the performances of the system, and should be performed with care. A model of a DC motor is a good example for the importance of scaling. In this model, one can define the states as the induced current and the angular velocity. The current is usually given in units of Amperes, which are very small (magnitudes of 10^{-3}) while the angular velocity is given in units of rounds per seconds, usually few thousands per seconds. Furthermore, the small dimensions of the motor results in a very small inertia moment, which causes the system matrix A to be ill-conditioned. This might result in failure of the *maxdet* algorithm, due to linear dependencies of the LMI's. Hence proper scaling can be a good remedy for such cases.

If the unscaled states are given as $\mathbf{x}$, then scaling can be performed in the following way:

$$\tilde{\mathbf{x}} = L\mathbf{x} \tag{3.33}$$

where L is the scaling matrix. Then the invariant set can be reformulate as

$$\mathcal{E}_{zx} = \{\tilde{\mathbf{x}} | \tilde{\mathbf{x}}^T \tilde{Q}_{zx} \tilde{\mathbf{x}} \leq 1\} \tag{3.34}$$

thus the unscaled solution is given by

$$Q_z = L^T \tilde{Q}_{zx} L \tag{3.35}$$

3.8 Robust MPC formulation

Uncertain systems can be represented in a few ways. One possible presentation is a 'multi model' paradigm, in which the model is a part of a polytope. Consider the next system representation

$$\begin{aligned} \mathbf{x}(k+1) &= A\mathbf{x}(k) + B\mathbf{u}(k) \\ \mathbf{y}(k) &= C\mathbf{x}(k) \\ [A \quad B] &\in \Omega \end{aligned} \tag{3.36}$$

where the polytope Ω is given as

$$\Omega = Co\{[A_1 \ B_1], \ [A_2 \ B_2], \ ..., \ [A_p \ B_p]\} \tag{3.37}$$

where Co denotes a convex hull. If the Jacobian $\left[\frac{\partial f}{\partial x}, \ \frac{\partial u}{\partial x}\right]$ of a nonlinear discrete system $\mathbf{x}(k+1) = f(\mathbf{x}(k), \mathbf{u}(k))$ is known to lie on the polytope Ω, then every trajectory $(\mathbf{x}, \mathbf{u})$ of the original system is also trajectory of (3.36) (Liu; 1968). This means that it is possible to approximate the non-linear system by polytopic uncertain of LTI systems.

The robust model predictive control strategy is then to find an invariant set, which will also guarantee feasibility, for the polytope Ω. This can be done by replacing (3.28b) of the ERPC offline problem by the next LMI

$$\begin{bmatrix} Q_z & Q_z\Psi_i^T \\ \Psi_i Q_z & Q_z \end{bmatrix} \geq 0, \qquad i = 1, 2, ..., p \tag{3.38}$$

where

$$\Psi_i = \begin{bmatrix} (A_i - B_i K) & [B_i \ 0 \ ... \ 0] \\ 0 & M \end{bmatrix} \tag{3.39}$$

and p is the number of vertices in the polytope Ω. Since each vertex of the polytope describes a worst case situation, we can expect a conservative solution, where the ellipsoid (and the maximal admissible set) will be small.

3.9 Summary

This chapter presented the first strategy of solving a offline part. The convex ERPC offline algorithm finds a global solution Q_z which is used to create an invariant ellipsoid $\mathcal{E}_z = \{z| \; z^T Q_z^{-1} z\}$. The ellipsoid guaranties feasibility of the system (3.1) or alternatively, the LDI (3.36) in case of uncertainties. The solution Q_z^{-1} is then used as an alternative to the QP problem in standard model predictive control. This is done online, using a very effective Newton Raphson algorithm. Further details about the online part is given in chapter 5. Drageset (2002) gives a good discussion about the ERPC algorithm. The performance of the ERPC offline is illustrated by a few examples in chapter 4. Some very good discussions are given by Drageset (2002), and Kouvaritakis et al. (2000) so they will not be repeated here, since it is not central in this thesis.

As will be shown next chapter, the ERPC plays important roll in the Sequential SDP algorithm performance, as its solution is used as an initial condition.

Chapter 4

SSDMPC

4.1 Introduction

This chapter investigates a new method of MPC, enlarging the ERPC ellipse discussed in the previous chapter. The offline problem is altered and a series of subproblems solved to find an optimal solution that produces a larger ellipsoid, thus enlarging the invariance feasible area.

Section 4.2 presents the guidelines of the SSDMPC offline algorithm. The sequential SDP is developed step by step in section 4.3. This section explores each component of the SSDP, derives the exact gradient and Hessian, and discusses techniques to modify the Hessian. The convergence properties of the SSDMPC algorithm are analyzed in section 4.4, where line-search backtracking and updating rules for the main parameters are given. Section 4.5 discusses the termination criteria, followed by section 4.7, which gives the full offline problem formulation and states the SSDMPC offline algorithm. Numerical examples and applications are presented in section 4.8. An analysis and discussion concludes this chapter.

4.2 The method

4.2.1 Extra degrees of freedom

Consider the offline problem formulation given by (3.28). The M matrix for the SISO case was chosen to be

$$M = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & & \ddots & & 0 \\ 0 & \ddots & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.1}$$

and this gives the next iteration of the future perturbations vector

$$f_{k+1}^T = \begin{bmatrix} \mathbf{c}_{k+1}^T & \mathbf{c}_{k+2}^T & \cdots & \mathbf{c}_{k+N_c}^T \end{bmatrix}^T \tag{4.2}$$

The main idea of the new method is that M matrix is set as a general matrix, so more degrees of freedom can be achieved. The problem 3.28 is then changed and optimized in such a way that the ellipsoid $\mathcal{E}_{zx} = \{\mathbf{x}|\ \mathbf{x}^T Q_z^{-1} \mathbf{x} \leq 1\}$ can be larger than the ERPC case.

The M matrix shall be denoted as

$$M = \begin{bmatrix} m_{11} & m_{12} & \cdots & m_{N_c \cdot m_u} \\ m_{21} & & & \vdots \\ \vdots & & \ddots & \vdots \\ m_{N_c \cdot m_u, 1} & \cdots & & m_{N_c \cdot m_u, N_c \cdot m_u} \end{bmatrix} \tag{4.3}$$

4.2.2 The Bilinear Matrix Inequality

Since the matrix Ψ is now a variable containing the unknown $m_{11}, m_{12}, ..., m_{N_c \cdot m_u, N_c \cdot m_u}$, (3.28b) can no longer be defined as an LMI i.e. the next matrix

$$\begin{pmatrix} Q_z & Q_z^T \Psi^T \\ \Psi Q_z & Q_z \end{pmatrix} > 0 \tag{4.4}$$

is no longer an LMI but a Bilinear Matrix Inequality, BMI, since the two variables are multiplied together, ΨQ_z. Recall from (3.15) that using the Schur complement the BMI above is equivalent to

$$\begin{pmatrix} Q_z & \Psi^T \\ \Psi & P_z \end{pmatrix} > 0 \tag{4.5}$$

where $P_z = Q_z^{-1}$. We shall therefore use the LMI (4.5) to ensure that the ellipsoid $\mathcal{E}_{zx}$ will be an invariant one and the system will be asymptotically stable. This, however, introduces a new difficulty, namely to require that $P_z = Q_z^{-1}$, which is a nonlinear equation.

4.2.3 Requirements

Chapter 3 presented the absolute requirements for the ellipsoid $\mathcal{E}_{zx}$, mainely to be invariant and feasible, in addition to optional requirements such as a polytope being confined inside the ellipsoid. The LMI set $\mathcal{X}_{LMI}$ which defines these requirements is

$$\begin{array}{lll} \begin{pmatrix} Q_z & \Psi_i \\ \Psi_i^T & P_z \end{pmatrix} > 0 \qquad i = 1, ..., p & & \text{Invariance} \\ d_i^2 - [K_i \ \ e_i\ ]\ Q_z\ [K_i \ \ e_i]^T \geq 0 & & \text{Feasibility} \\ F(Q_z, P_z) \geq 0 & & \end{array} \tag{4.6}$$

We can then define $\{Q_z,\ \ P_z\} \in \mathcal{X}_{LMI}$ which means that $\exists M$ such that Q_z and P_z satisfies (4.6). $F(Q_z,\ P_z)$ is an affine function which can be any relevant constraint which depends on Q_z, P_z.

4.3 SSDP

As with the ERPC problem, the objective is to find a maximum invariant ellipsoid $\mathcal{E}_{zx}$. Here, however, the Ψ matrix is no longer a constant matrix as in ERPC problem, but a variable, as given by (4.3). Hence, we are no longer dealing with LMIs alone, but with a Bilinear Matrix Inequality as well. As it is not the only variable in the invariance inequality (4.5), the standard LMI solvers can no longer be used to find the Q_z that maximize the ellipsoid $\mathcal{E}_{zx}$. We can use and alternative strategy, presented in the following sections, to find a *local* solution Q_z^*. Consider the next optimization problem

$$\begin{aligned} \min \quad & f := \log\det(T_x^T Q_z T_x)^{-1} \\ \text{s.t.} \quad & Q_z = P_z^{-1} \\ & \{Q_z,\ P_z\} \in \mathcal{X}_{LMI} \end{aligned} \tag{4.7}$$

where the set $\mathcal{X}_{LMI}$ is a convex set given by (3.28b-3.28d). This problem is non-convex due to the nonlinear term in the equality constraint. This section will investigate strategies for solving problem (4.7) in an efficient manner. Note that the equality constraint can be written as $Q_zP_z - I$ assuming that the P_z matrix is invertible (which it is, since it is a symmetric positive definite matrix). For convenience, the vector of variables shall be denoted as

$$w = \begin{pmatrix} Q_z \\ P_z \end{pmatrix} \tag{4.8}$$

the w notation will be used in the following sections.

4.3.1 Augmented Lagrange Method

The key idea in solving problem (4.7) is to eliminate the non-linear equality constraint $Q_z = P_z^{-1}$. This is done by defining a new objective function which includes the constraint and the old objective function from (4.7). Consider the next lagrange function

$$\mathcal{L}_A = f(Q_z) + \mathrm{Tr}(\mathbf{\Lambda}(Q_zP_z - I)) + \frac{\mu}{2}\mathrm{Tr}\left[(Q_zP_z - I)^T(Q_zP_z - I)\right] \tag{4.9}$$

where $f(Q_z) = \log\det(T_x^T Q_z T_x)^{-1})$ and $\mathbf{\Lambda} \in \mathbb{R}^{(m_x+m_u\cdot N)\times(m_x+m_u\cdot N)}$ is the Lagrange multiplier matrix and μ is a positive constant. Then we can formulate a (non-convex) subproblem as follows

$$\begin{aligned} \min \quad & \mathcal{L}_A \\ \text{s.t.} \quad & w \in \mathcal{X}_{LMI} \end{aligned} \tag{4.10}$$

where $\mathcal{L}_A$ is the same Lagrangian (4.9) and w is a vector of variables $w =\in \mathcal{X}_{LMI}$ means that the vector w should satisfy some LMI constraints involving $Q_z,\ P_z$ (that is $\mathcal{X}_{LMI}$ is convex). This subproblem will be identical to (4.7) when $\mu \to \infty$ and $\mathbf{\Lambda}_k = \mathbf{\Lambda}^*$, where $\mathbf{\Lambda}^*$ is the optimal Lagrangian. The last statement shall be proven in section 4.4.

4.3.2 Lagrangian approximation

Since solving subproblem (4.10) is very complicated, a quadratic approximation to the Lagrangian will be used, as suggested by Fares et al. (2000) and Imsland (2002). This is done by finding the second order Taylor expansion to (4.9), given by

$$\mathcal{L}_A(p)|_{p_0} \approx \mathcal{L}_A(p_0) + \nabla\mathcal{L}_A(p_0)^T p + \frac{1}{2}p^T\nabla^2\mathcal{L}_A(p_0)p \tag{4.11}$$

where p is a vector of variables to be defined later, and $\nabla\mathcal{L}_A(p_0)$ and $\nabla^2\mathcal{L}_A(p_0)$ denote the gradient and the Hessian of the Lagrangian in a given vector p_0. It is possible then to solve subproblem (4.10) by replacing the exact Lagrangian with the approximation (4.11). Notice that the first term in the left hand side of (4.11) is constant and does not affect the solution. The subproblem can then be rewritten as

$$\begin{aligned} \min \quad & \nabla\mathcal{L}_A^T p + \frac{1}{2}p^T\nabla^2\mathcal{L}_A p \\ \text{s.t.} \quad & \mathcal{X}_{LMI} \end{aligned} \tag{4.12}$$

where $\mathcal{X}_{LMI}$ is the basic LMI set 4.6. Assuming that the Hessian $\nabla^2\mathcal{L}_A$ is positive definite, subproblem (4.12) is convex and can be solved very effectively since the objective function is quadratic and the inequality set $\mathcal{X}_{LMI}$ is a convex set. Even if the Hessian is not positive definite, it will be modified in such a way that the subproblem will still converge in at least linear rate. One way of solving (4.12) is to introduce a slack variable t and to transform the objectives to an SDP problem. Using the Schur complement (2.7):

$$t - \left(\nabla\mathcal{L}_A^T p + \frac{1}{2}p^T\nabla^2\mathcal{L}_A p\right) \geq 0$$

$$\Updownarrow$$

$$\begin{cases} \begin{bmatrix} t - \nabla\mathcal{L}_A p & p^T \\ p & (\frac{1}{2}\nabla^2\mathcal{L}_A\ )^{-1} \end{bmatrix} \geq 0 \\ \nabla^2\mathcal{L}_A \geq 0 \end{cases}$$

The last term (i.e a positive definite Hessian) is assured from the modification of the Hessian. The subproblem which finds a *local* solution v_k^* at each iteration can now be defined. Denote a line search vector

$$p(k) = \begin{bmatrix} Q_z(k) \\ P_z(k) \end{bmatrix} - \begin{bmatrix} Q_z(k-1) \\ P_z(k-1) \end{bmatrix} \tag{4.13}$$

Then transform subproblem (4.12) to

$$\min \quad t \tag{4.14a}$$

$$\text{s.t.} \quad \begin{bmatrix} t - \nabla\mathcal{L}_A \cdot p(k) & p(k)^T \\ p(k) & (\frac{1}{2}\nabla^2\mathcal{L}_A\ )^{-1} \end{bmatrix} \geq 0 \tag{4.14b}$$

$$\{Q_z(k) \quad P_z(k)\}^T \in \mathcal{X}_{LMI} \tag{4.14c}$$

where t is the slack variable (which is to be found), $\nabla\mathcal{L}_A$ and $\nabla^2\mathcal{L}_A$ are the gradient and the Hessian respectively, and $\mathcal{X}_{LMI}$ is the set of LMI's which ensure the feasibility and invariance of the solution. This problem is a Semi Definite Programming (SDP) problem, and can be solved efficiently using LMI solvers, for example *mincx* of Mathworks. A very good explanation about the SDP theory, solvers and applications is given by Wolkowicz et al. (2000). For the Matlab *mincx* solver, see Gahinet et al. (1995).

At each iteration, the SDP problem (4.14) is solved using the current data to find a local solution $v(k)^* = (Q_z(k)^*, P_z(k)^*, M(k)^*, t(k)^*)^T$. v_k^* is then used to compute a new search direction $p(k)$ (using (4.13)) and then to reduce v_k^* in the direction p_k by step length α_k, found using backtracking. If not satisfying certain termination criteria, v_k is used as an initial value for the next iteration. We shall refer to this method as a Sequential Semi Definite Programming (SSDP). The SSDP can be conceptually described by the next algorithm:

Algorithm 4.1 (SSDP conceptual)

1. *$k = 1$: Find the initial values Q_z, P_z and M using, for example the ERPC offline algorithm. Define the decision variable $v_0 = [Q_z,\ P_z,\ ,M,\ \Psi,\ t]^T$ and the initial Lagrange multiplier matrix $\mathbf{\Lambda}_1$, and the penalty parameter μ using the procedure section 4.4.4.*

2. *Calculate the gradient and the Hessian, and modify the Hessian to ensure it is a positive definite matrix, using one of the methods presented in section 4.3.4.*

3. *Solve the SDP problem (4.14) using a LMI solver (e.g. mincx) to find a local solution $v_k = (Q_z(k),\ P_z(k),\ M(k),\ \Psi(k),\ t(k))$, and search direction p_k.*

4. *Use line-search backtracking to compute the step length α_k that reduces v_k in the direction p_k as described in section 4.4.3.*

5. *Update the decision variable vector $v_{k+1} = v_{k-1} + \alpha_k(v_k - v_{k-1})$, the penalty parameter μ_{k+1} and the Lagrange multiplier $\mathbf{\Lambda}_{k+1}$ using rules in section 4.4.4.*

6. *If the termination criteria are not satisfied, go to step 2, otherwise STOP with the solution Q_z^* and P_z^*.*

The gradient and the Hessian will be computed explicitly, as explained in the subsequent sections.

4.3.3 The Gradient

The gradient of the Lagrangian is computed using the **svec** operator. This operator is a linear operator which maps the set of symmetric matrices $\mathcal{S}^n$ into $\mathbb{R}^{n(n+1)/2}$ and is defined by

$$\mathbf{svec}\ X = [X_{11}, X_{12}..., X_{1n}, X_{22}, X_{23}, ..., X_{nn}]^T \tag{4.15}$$

svec operator essentially maps the upper right half of the symmetric matrix into a vector. The diagonal matrix is defined as

$$T = \text{diag}\ \left[1, \sqrt{2}, ..., \sqrt{2}, 1, \sqrt{2}, ..., 1\right]^T \tag{4.16}$$

where the 1's corresponds to the elements on the diagonal of a symmetric matrix mapped with **svec**, and the elements $\sqrt{2}$ refer to the upper diagonal places.

Example 4.1 *consider the* 3×3 *matrix,*

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{bmatrix}$$

Then the **svec** *operator will map the matrix* A *to the vector:*

$$\mathbf{svec}(A) = [\ a_{11}\ a_{12}\ a_{13}\ a_{22}\ a_{23}\ a_{33}\ ]^T$$

and the corresponding T *matrix can be written as*

$$T = diag\left[1,\ \sqrt{2},\ \sqrt{2},\ 1,\ \sqrt{2},\ 1\right]$$

For convenience, define a new vector $w = (Q_z,\ P_z)^T$. Using these definitions the partial derivatives of $\mathcal{L}_A$ with respect to w can be found. The Lagrangian (4.9) has two terms: the object function $f(w)$ and the constraints part of the Lagrangian. The gradient is

$$\nabla_w \mathcal{L}_A = \nabla_w f(w) + \begin{bmatrix} \nabla_{Q_z} \mathcal{L}_{const.} \\ \nabla_{P_z} \mathcal{L}_{const.} \end{bmatrix} \tag{4.17}$$

The set $\mathcal{X}_{LMI}$ is in terms of Q_z and P_z in the **z**-space, with dimensions $(m_x + m_u N_c) \times (m_x + m_u N_c)$, while the objective function $f(w) = \log\det(T_x Q_z T_x^T)^{-1}$ has dimensions $m_x \times m_x$. Therefore, a permutation matrix P_{proj} is needed to adapt the gradient and Hessian to **z**-space. The gradient of the objective function is given by

$$\nabla_w f(w) = P_{proj}\mathbf{svec}\left[(-T_x Q_z T_x^T)^{-1}\right] \tag{4.18}$$

This result is discussed in detail in Drageset (2002) and will not be repeated here. Notice that the projection matrix P_{proj} is given in Drageset (2002) only for *SISO* systems. The dimensions of P_{proj} in the general case is $P_{proj} \in \mathbb{R}^{m \times n}$ where $m = (m_x + N_c m_u) \times (m_x + N_c m_u - 1)$ and $n = m_x(m_x + 1)/2$. The matrix is given as

$$P_{proj} = \begin{bmatrix} G_{n_x} & 0 & & \cdots & 0 \\ 0 & G_{n_x - 1} & 0 & \cdots & 0 \\ 0 & 0 & \ddots & & 0 \\ 0 & \cdots & & 0 & G_1 \end{bmatrix} \tag{4.19}$$

$$G_{n_x - i} = \begin{bmatrix} I_{(n_x - i)\times(n_x - i)} \\ \mathbf{0}_{(n_x - i)\times(N_c m_u)} \end{bmatrix}, \quad i = 0, 1..., (n_x - 1) \tag{4.20}$$

Note that P_{proj} given in Drageset (2002) has an extra zero line. This is not correct as the elements G contain already the zero matrices, see equation (4.20). The dimensions of the zero matrices, given in Drageset (2002) are also not true, since they decrease as i increases. The correct dimensions are given by (4.20). The constraints part of the Lagrangian is computed (in a similar manner to Fares et al. (2000)) as

$$\nabla_{Q_z}\mathcal{L}_{const.} = T^2\left(\mu \cdot \mathbf{svec}\left[Q_z P_z^2 + P_z^2 Q_z - 2P_z\right] + \mathbf{svec}\left[P_z \mathbf{\Lambda} + \mathbf{\Lambda}^T Q_z\right]\right) \tag{4.21}$$

$$\nabla_{P_z}\mathcal{L}_{const.} = T^2\left(\mu \cdot \mathbf{svec}\left[P_z Q_z^2 + Q_z^2 P_z - 2Q_z\right] + \mathbf{svec}\left[Q_z \mathbf{\Lambda}^T + \mathbf{\Lambda} P_z\right]\right) \tag{4.22}$$

4.3.4 The Hessian

The Hessian is computed in an analytic fashion, using the same tools used for the gradient computation. The Hessian is the partial derivation of the gradient (4.17) and can be written as

$$\nabla^2\mathcal{L}_A(w) = \nabla_w^2 f(w) + \begin{bmatrix} \nabla^2_{Q_z}\mathcal{L}_{cnstr.} & \nabla^2_{Q_z P_z}\mathcal{L}_{cnstr.} \\ \nabla^2_{P_z Q_z}\mathcal{L}_{cnstr.} & \nabla^2_{P_z}\mathcal{L}_{cnstr.} \end{bmatrix} \tag{4.23}$$

Notice that the right hand term of the Hessian defines a symmetric matrix, i.e. $\nabla^2_{Q_z P_z}\mathcal{L}_{cnstr.} = \nabla^2_{P_z Q_z}\mathcal{L}_{cnstr.}$.

Exact Hessian

The analysis shall begin with the computation of the second derivatives of the $f(w)$ term, i.e. $\log\det(T_x Q_z T_x^T)^{-1}$. Denote $Q_{zx} = T_x Q_z T_x^T$ the projection of Q_z in the $\mathbf{x}$-space and define the symmetric Kronecker product operator $\circledast$ as (Alizadeh et al.; 1998)

$$(U \circledast V)T\mathbf{svec}X = T\mathbf{svec}\frac{1}{2}(UXV^T + VXU^T) \tag{4.24}$$

Then the second derivative of $f(w)$ is computed as

$$\begin{aligned} \frac{\partial^2 \log\det Q_{zx}^{-1}}{\partial \mathbf{svec} Q_{zx}^2} &= -T^2 \frac{\partial \mathbf{svec} Q_{zx}^{-1}}{\partial \mathbf{svec} Q_{zx}} \\ &= T(Q_{zx}^{-1} \circledast Q_{zx}^{-1})T \\ &= H_f \end{aligned} \tag{4.25}$$

The same permutation matrix given by (4.19) shall be used to adapt the term (4.25) of the Hessian to $\mathbf{z}$-space as follows:

$$\nabla_w^2 f(w) = P_{proj} H_f P_{proj}^T \tag{4.26}$$

The other term in the Hessian (4.23) contains the constraint section and is computed in a similar way to Fares et al. (2000). The elements in the matrix term of (4.23) are computed as

$$\begin{aligned}
\nabla^2_{Q_z}\mathcal{L}_{cnstr.} &= \mu T(P_z \circledast I)T && (4.27)\\
\nabla^2_{Q_zP_z}\mathcal{L}_{cnstr.} &= T(\mathbf{\Lambda} \circledast I + \mu((P_zQ_z - I) \circledast I + P_z \circledast Q_z)T && (4.28)\\
\nabla^2_{P_z}\mathcal{L}_{cnstr.} &= \mu T(Q_z \circledast I)T && (4.29)\\
& && (4.30)
\end{aligned}$$

Indefinite Hessian

In cases where the Hessian is a positive definite matrix in each iteration, then a convex problem is solved and a global solution $v = [Q_z,\ P_z,\ M,\ t]$ is found. Unfortunately, the majority of cases produce an indefinite Hessian matrix and not positive definite as was mentioned by Drageset (2002). This fact is highly significant as we are required to solve a sequence of non-convex problems. However, it is possible to modify the Hessian in such a way so as it becomes positive definite, making the subproblem convex. In each subproblem (iteration), a global solution $v_k = [Q_z(k),\ P_z(k),\ M(k),\ t(k)]$ is acquired. Note that the main problem (4.7), however, remains non-convex and can only find a local solution. The modified Hessian shall be denoted as $H_m(k)$. It is important to modify the Hessian in such a manner that it will be well conditioned (at least not too ill conditioned) while the modification remains as small as possible, so that the second order information in the Hessian is preserved as far as is possible.

Some modification strategies shall be explored: (1) Eigenvalues modification (2) Modified Cholesky algorithm (3) Modified symmetric indefinite factorization. Fares et al. (2000) and Bertsekas (1996) suggest other modification strategies which make the Hessian positive definite while attempting to ensure convergence.

Eigenvalue modification

As the name suggests, this strategy decomposes the Hessian in the following way

$$\nabla^2\mathcal{L}_A = Q\Lambda Q^T \tag{4.31}$$

where Q is the eigenvectors matrix and Λ is a diagonal matrix of the eigenvalues. The negative eigenvalues are then replaced by a small positive number δ that is somewhat larger than the machine accuracy, say $\delta = 10^{-6}$. Using a very small value can result in a problem, presented in the next example (adapted from Nocedal and Wright (1999))

Example 4.2 *Consider a problem where the gradient at iterate k is $\nabla\mathcal{L}_A(k) = (1, -3, 2)$ and the Hessian is $\nabla^2\mathcal{L}_A(k) = diag(10, 3, -1)^T$. This is an indefinite matrix since the last eigenvalue is negative. Using equation (4.31), we find that $Q = I$ and $\Lambda = diag\ (\lambda_1, \lambda_2, ..., \lambda_6)$. The solution of the pure Newton step, i.e. $\nabla^2\mathcal{L}_A(k)p_k = -\nabla\mathcal{L}_A(k)$ gives the vector $p_k^N = [-0.1,\ 1,\ 2]^T$, which is not a decent direction since $\nabla\mathcal{L}_A(k)^T p_k^N =$*

$11.41 > 0$. *We can modify the last eigenvalue to be* $\delta = 10^{-16}$ *and the modified Hessian becomes*

$$H_m(k) = \sum_{i=1}^{2} \lambda_i q_i q_i^T + \delta q_3 q_3^T = diag(10,\ 3,\ 10^{-8}) \ > \ 0 \tag{4.32}$$

which is positive definite. The search direction becomes $p_k \approx [0,\ 0,\ -(2\times 10^8)q_6\ ]^T$ *which is nearly parallel to* q_3*, with very small contributions from the other directions. The extreme length of the direction is questionable since the Newton method relies on a quadratic approximation in a neighborhood of the current iterate.*

The above example illustrates a problem that can be encountered when choosing small values of δ. Introducing very small value(s) to the Hessian usually produces an ill-conditioned modified Hessian $H_m(k)$ due to differences in the size of the elements. Therefore care should be used when modifying the eigenvalues.

Modified Cholesky Factorization

The main idea of the Cholesky factorization is every symmetric positive definite matrix A can be written as

$$A = LDL^T \tag{4.33}$$

where L is the lower triangular matrix with unit diagonal elements and D is a diagonal matrix with positive definite elements on the diagonal. If A is indefinite, the factorization (4.33) may not exist or it may be numerically unstable. To avoid this, the matrix A is modified in such a way that all elements in D are sufficiently positive and the elements in D and L are not too large. The modified Cholesky algorithm guarantees that the modified Hessian is positive definite and bounded. The algorithm used in the simulations is given by appendix A. It produces a non-negative diagonal matrix E with the elements $e_j = d_j - c_{jj}$, where

$$A + E = LDL^T \tag{4.34}$$

The diagonal E matrix is zero if A is sufficiently positive definite. To control the quality of the modification, all the elements of D should be sufficiently positive, and elements in L and D are not too large. This is done by choosing two positive constants δ and β which define bounds on the elements of L and D. Nocedal and Wright (1999) give suggestions for choosing the constants δ and β, which should generally be small. It was observed from the simulations that reducing the condition number (by choosing low β values) causes the modified Hessian H_m to be significantly different from the Hessian $\nabla^2\mathcal{L}_A(k)$ thus losing the second order information of the exact Hessian $\nabla^2\mathcal{L}_A(k)$. This reduces the chance of finding a good decent direction (the term *good* direction means one that is not excessively long, or not an ascent direction) to the problem (4.10). However, if the modification is applied sporadically (i.e. only at some iterations), the problem will eventually converge to a (local) solution (see section 4.4)

Modified Symmetric Indefinite factorization

A third strategy, discussed in detail by Cheng and Higham (1998) uses the fact that the Hessian is symmetric. Any symmetric matrix, whether positive definite or not, can be written as

$$PAP^T = LDL^T \tag{4.35}$$

where P is a permutation matrix, L is unit lower triangular and D is block diagonal with diagonal blocks of dimensions 1 or 2. Then the factorization

$$P(A+E)P^T = L(D+F)L^T \tag{4.36}$$

is found where F is chosen so that $D+F$ (and hence also $A+E$) is positive definite.

4.4 Convergence properties

The Lagrangian (4.9) penalizes the constraint violations by squaring the infeasibilities and scaling them by the factor $\frac{\mu}{2}$, while also including an explicit estimate of the lagrange multipliers Λ. This inclusion of Λ helps avoid numerical problems associated with large values of μ. This can be demonstrated by the following analysis. Consider the gradient of the Lagrangian

$$\begin{aligned}\nabla_w \mathcal{L}_a(w,\ \Lambda;\mu) &= \nabla f(w) + \nabla \text{Tr}(\Lambda(Q_zP_z - I)) + \mu \text{Tr}(Q_zP_z - I)^T \nabla \text{Tr}(Q_zP_z - I) && (4.37)\\ &= \nabla f(w) + \text{Tr}\left[\Lambda + \mu(Q_zP_z - I)^T\right] \nabla \text{Tr}(Q_zP_z - I) && (4.38)\end{aligned}$$

where w is defined by (4.8). We can use the proof of Nocedal and Wright (1999) Theorem 17.2 to deduce that

$$\text{Tr}\Lambda^* \approx \text{Tr}\Lambda_k + \mu \text{Tr}(Q_zP_z - I) \tag{4.39}$$

where Λ_k is the Lagrange multipliers at the current iteration and Λ^* is the optimal Lagrange multiplier matrix. A similar proof can be found at Berteskas (1982, p. 101). The expression (4.39) can be rearranged to achieve

$$\text{Tr}(Q_zP_z - I) \approx \frac{1}{\mu}\text{Tr}(\Lambda^* - \text{Tr}\Lambda_k) \tag{4.40}$$

so it can be concluded that if $\text{Tr}\Lambda_k$ is close to the optimal $\text{Tr}\Lambda^*$, the infeasibility in w will be very small since $\lim\limits_{\mu\to\infty} \frac{1}{\mu} \to 0$ and the term $(\text{Tr}\Lambda^* - \text{Tr}\Lambda_k)$ is also very small. This means infeasibility in w will be much smaller than $\frac{1}{\mu}$, rather than being proportional to $\frac{1}{\mu}$, and the numerical problems arising from very large values of μ are avoided. Thus is not strictly correct in practice, since $\text{Tr}\Lambda$ usually differs from $\text{Tr}\Lambda^*$. Fares et al. (2000) proves global convergence for a similar SSDP algorithm. These results are not true in practice since the modification of the Hessian, and the approximation of the lagrange result in local convergence, which depends on the penalty parameter and the efficiency of the line search. In practice, it may prove difficult to find Lagrange multiplier estimate $\text{Tr}\Lambda$ close to the optimal $\text{Tr}\Lambda^*$, especially for initial conditions far from the solution.

4.4.1 The problem of ill-conditioning

There is a degree of difficulty in solving the problem (4.10). It depends on the eigenvalue structure of the Hessian matrix $\nabla^2\mathcal{L}(w, \Lambda)$. For simplicity, we shall define $h(w_k) = \text{Tr}(Q_z P_z - I)$ the equality constraint at iteration k. The Hessian of the Lagrangian (4.9) is then

$$\nabla^2\mathcal{L}(w, \Lambda) = \nabla^2 f(w) + (\text{Tr}\Lambda + \mu h(w))\,\nabla^2 h(w) + \mu \nabla h(w) \nabla h(w)^T$$

By using the notation $\text{Tr}\tilde{\Lambda} = \text{Tr}\Lambda + ch(w)$, we can write

$$\nabla^2\mathcal{L}(w_k, \Lambda_k) = \nabla^2\mathcal{L}_0(x_k,\ \tilde{\Lambda_k}) + \mu_k \nabla h(w_k) \nabla h(w_k)^T$$

The minimum eigenvalue $\underline{\gamma}$ of $\nabla^2\mathcal{L}(w_k,\ \Lambda_k$ satisfies

$$\underline{\gamma}_k = \min \frac{z^T \nabla^2 \mathcal{L}_k(w_k,\ \Lambda_k) z}{z^T z} \leq \min \frac{z^T \nabla^2 \mathcal{L}_0(w_k,\ \tilde{\Lambda}_k) z}{z^T z} \tag{4.41}$$

where a vector $z \neq 0$ with $\nabla h(w_k)^T z = 0$ always exists (proof in Bertsekas (1996)). The maximum eigenvalue $\overline{\gamma}$ of the Hessian $\nabla^2\mathcal{L}(w_k,\ \Lambda_k)$ satisfies

$$\begin{aligned} \overline{\gamma}_k &= \max \frac{z^T \nabla^2 \mathcal{L}_k(w_k,\ \Lambda_k) z}{z^T z} \\ &\geq \min \frac{z^T \nabla^2 \mathcal{L}_0(w_k,\ \tilde{\Lambda}_k) z}{z^T z} + \mu_k \max \frac{z^T \nabla h(w_k) \nabla h(w_k)^T z}{z^T z} \end{aligned} \tag{4.42}$$

Since the gradient $\nabla h(w^*) \neq 0$ (see section 4.3.3), it follows that

$$\lim_{k \to \infty} \frac{\overline{\gamma}_k}{\underline{\gamma}_k} = \infty \tag{4.43}$$

In other words, the condition number of the subproblem (4.10) becomes progressively worse and tends to infinity as $k \to \infty$ and μ_k becomes large. Ill conditioning can be avoided only by using an initial condition which is sufficiently close to the solution. To avoid this problem at the early stages of iteration, one should use a low increase rate of the penalty parameter. ρ can be controlled dynamically, high in the first iterations to assure fast convergence, then reducing the rate as the condition number increases (denote it ρ_k then). Thus, ρ_k should be kept as small as possible when μ_k is large (equation 4.45). Typical values are $\rho_k = 1.01$ when $\mu_k > 10^7$. The low update rate of μ may slow down the rate of convergence, but when is used together with intelligent Hessian updating, the convergence rate is satisfactory.

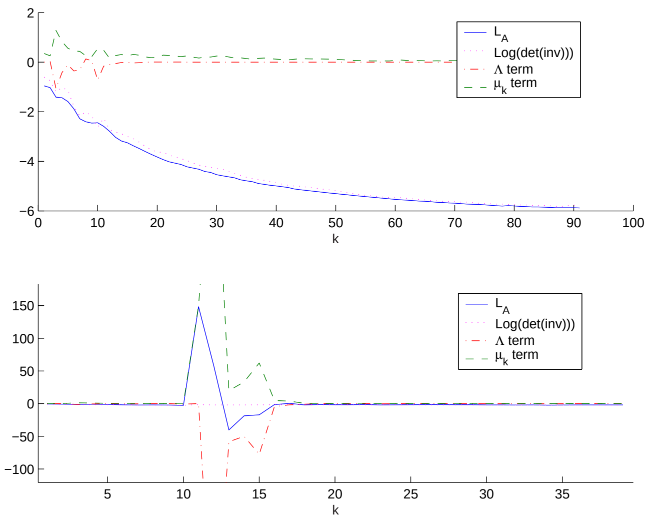
4.4.2 Penalty parameter

The choice of μ_k is very important, since it affects the convergence and the size of the ROA. If the increase rate of μ_k is monotonic, then large problems are likely to have a poor convergence rate. If many iterations are made, ill-conditioning problems may occur, resulting in an unbounded SDP solution, and failure of the algorithm. It is suggested then to augment the penalty parameter after a few iterations, say $k > 11$, to a large number, for instance $\mu_k = 1000$. This enforces faster convergence, as the residual terms in the Lagrangian (4.9) are heavily penalized. Figure 4.1 presents the results of two cases: μ_k was increased at monotonic rate (left top), resulting in 91 iterations, while in the second case it was augmented at $k > 11$ to $\mu_k = 1000$, and was increased at a monotonic rate there on. In this case it took only 39 iterations, but the ROA is smaller (green solid, right). Hence 'trade-off' must be achieved between the size of the ROA and the convergence for large systems.

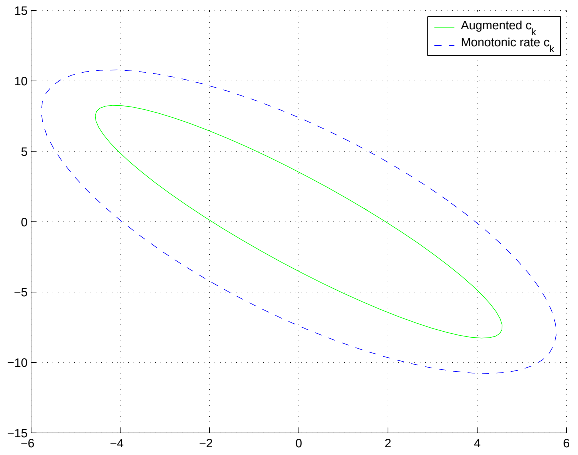
4.4.3 Line Search Backtracking

SSDMPC uses a basic *Backtracking* which satisfies both Wolfe conditions, i.e. the *sufficient decrease* and the *curvature*. This is done in order to find a suitable step size α_k which causes further reduction of v_k found by the SSD problem, along direction p_k (4.13). Nocedal and Wright (1999) suggest a two-stage algorithm which finds a trial estimate of α_k and then uses a so called *zoom* algorithm which increase this α_k until an acceptable step length is found. This algorithm is expensive to solve, and was not tested. The basic linesearch algorithm is given below, where η is sufficient decrease of $\mathcal{L}_A$ and ρ_l reduces the step length:

Algorithm 4.2 (SSDMPC Backtracking)
Initializing: *Set $\rho_l = 0.5$. If $k < k_0$ then set $\eta = 10^{-4}$, for some k_0, for instance $k_0 = 15$. Otherwise increase $\eta = 10^{-1}$.*



(a) Convergence



(b) ROA

Fig. 4.1: Convergence properties. Top left figure: μ_k increase at monotonic rate. Bottom left figure: μ_k augmented k=11 to $\mu_k = 10^3$. Right figure: different in ROA sizes

```
While $\mathcal{L}_A(v_k + \alpha_k \cdot p_k;\ \Lambda_k, \mu_k) \leq \mathcal{L}_A(v_k;\ \Lambda_k, \mu_k) + \eta \cdot \alpha_k \nabla \mathcal{L}_A(v_k; \Lambda_k, \mu_k) \cdot p_k$
        $\alpha_{k+1} = \rho_l \cdot \alpha_k$
        $k = k + 1;$
Repeat
```

where the value of ρ_l is kept around $0.9 - 0.5$. Lower values cause the backtracking to converge faster, but they may over-reduce the step length, requiring more SDP iterations. Simulations reveal that the step length is usually around 1 and are rarely blocked by constraints (see section 4.8 for examples). It is worth using more backtracking iterations than using more iterations in SSDMPC, since solving the SDP (4.55) is expensive. η should be chosen so that α_k is not too much smaller than its predecessor α_{k-1}. This varies with the iterations; it is very small, 10^{-4}, at the beginning to allow sufficient decrease of $\mathcal{L}_A$, but later increases to $0.1 - 0.01$.

4.4.4 Update parameters

Initializing

The choice of initial parameter is critical to the behavior of the algorithm. The μ_0 value should be kept small only in the few first iterations, to allow the algorithm to iterate away from the initial condition. Typical values are between $0.5 - 1$, but if the system is large, higher μ_0 values are suggested, usually between $1 - 10$. This will allow fewer iterations, and numerical problems are avoided.

Updating rules

The convergence properties of the SSDMPC are affected by the rate μ_k increases and the values. Fares et al. (2000) suggest an intelligent updating rule of the Lagrange multiplier, based on Fletcher (1987). A better method for this algorithm is to control the increase rate of the penalty parameter, as discussed in section 4.4.2. To ensure Λ tends to Λ^* during the iterations, the next update rule is suggested:

$$\Lambda_{k+1} = \Lambda_k + \mu_k(Q_{k+1}P_{k+1} - I) \tag{4.44}$$

where the penalty parameter μ_k is updated by the following rule:

$$\mu_{k+1} = \begin{cases} \rho\mu_k & \text{If } \|P_{k+1} - Q_{k+1}^{-1}\|_\infty > \gamma \|P_k - Q_k^{-1}\|_\infty \\ \mu_k & \text{If } \|P_{k+1} - Q_{k+1}^{-1}\|_\infty \leq \gamma \|P_k - Q_k^{-1}\|_\infty \\ 1000 & \text{If } k = k_0 \end{cases} \tag{4.45}$$

for given $\rho > 1$ and $0 < \gamma < 1$, k_0. The update rule for μ_k is different from that presented by Fares et al. (2000). The norm $\|P_z(k) - Q_z(k)^{-1}\|_\infty$ is usually very different from $\|Q_z(k)^{-1}P_z(k) - I\|_{fro}$, particulary for iterations far from the solutions. The difference can clearly be seen in figure 4.2, when $k < 12$. γ is a *sufficient decrease* parameter, which affects the increase rate of μ. In other words, if the norm at the current iteration k is not lower than norm at last iteration, by factor of γ, then the penalty μ should be increased.

- Values in the range $\rho = 1.3 - 1.6$ have shown good results for most systems. If many iterations are expected, for instance if the initial condition lies far from the solution, then lower ρ values are preferred to prevent numerical problems in the early stages. It is important to keep ρ as low as possible when μ_k is getting too high (e.g. $\mu_k = 10^8$).
- If μ_k is increased too quickly, numerical problems may occur at an early stage, so ρ should not be too high.
- If μ_k is increased too slowly in the early stages, the update rate of the Lagrange multipliers is very slow, resulting in a poor convergence rate.
- μ_k is kept low in the first few iterations, e.g. $k < 15$, to allow the the decision variable to iterate away from the initial conditions. Too high μ_k values at the start of the algorithm can result in ellipsoids which are almost identical to the initial one (ERPC).
- μ_k is set to high value, say $\mu_k = 1000$ after k_0 steps, e.g. 15 iterations. This heavily penalizes the residual term of the Lagrangian (4.9) at early steps, forcing the subproblem (4.10) to come closer to the optimal solution. It prevents a large number of steps from taking place, and is particulary effective for large problems.
- γ affects the increase rate of μ_k and is chosen to be close to one, for instance $\gamma = 0.95$. Lower values of γ will increase the update rate of the penalty parameter μ_k, and ill-conditioning can occur at early stages.

4.5 Termination criteria

This section discusses the termination criteria required such that the variable vector $v_k = (Q_z, P_z, M, t)^T$ will be as close to the solution v^* as possible. Consider the equality constraint from the offline formulation (4.55)

$$Q_z P_z - I = 0 \tag{4.46}$$

This equation must be satisfied in order for the solution to be feasible (in addition to feasibility of the LMI set $\mathcal{X}_{LMI}$). However, due to the non-linearity of the term $Q_z P_z$, it will not be possible to achieve an exact solution there $Q_z = P_z^{-1}$. Moreover, if the variable v_k is not in the vicinity of the solution v_k^* at some iteration, then there may be a large difference between the size of (4.46) described by the norm, and the size of $\|Q_z - P_z^{-1}\|$. By the matrix norm Submultiplicative axiom,

$$\|Q_z P_z - I\| \leq \|Q_z - P_z^{-1}\| \cdot \|P_z\| \tag{4.47}$$

and the difference between the two norms depends on the size of norm P_z. Ergo, the norm $\|Q_zP_z - I\|$ which Fares et al. (2000) suggest as a termination criteria, is suitable only if $\|P_z\|$ is low, which is usually *not* the case for the offline SSDMPC.

This is demonstrated in the next example:

Example 4.3 *Consider a MIMO system, given by (4.57). This angular positioning system will be discussed in details in section 4.8.1. Figure 4.2 shows the differences between the Frobenius norm $Q_zP_z - I$ and the norm $Q_z - P_z^{-1}$. Although the two norms are almost equal near the solution, there are iterations where the difference is more than 10^3 (at $k = 10$ for instance). For some larger systems a difference of more than 10^4 was observed outside the vicinity of the solution.*

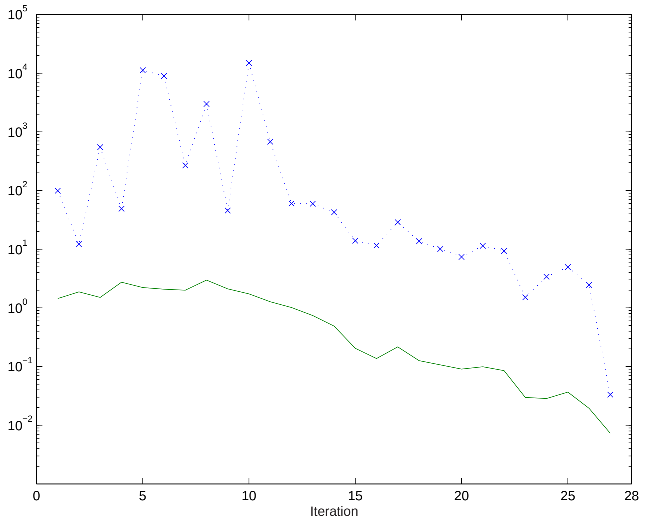


Fig. 4.2: The difference between the norm $\|Q_z - P_z^{-1}\|$ $(\cdot\cdot x \cdot\cdot)$ and the norm $\|Q_zP_z - I\|$ (solid)

It is thus recommended to consider the equation $Q_z - P_z^{-1} = 0$ as one of the termination criteria. However, due to the non linearity of Q_zP_z, it is not possible to achieve the exact solution $Q_z - P_z^{-1} = 0$. It is enough to check whether the inequality

$$\|Q_z - P_z^{-1}\|_\infty < \epsilon \tag{4.48}$$

holds for some threshold $\epsilon > 0$. Another question that arises is where to put the termination criteria. If the criteria is placed immediately after solving the SDP problem (4.14) then all that needs to be done is to check the norm (4.48), as the rest of the LMI set is satisfied anyway as long as the SDP is feasible. If the termination criteria is placed after the line search algorithm, then the LMI set does not necessarily remain feasible. This is because the line search alters $Q_z(k)$, $P_z(k)$ and $M(k)$. Then the fact that these variables have to satisfy the basic LMI constraints (to be given in eqn. set 4.55) must be considered, so that the solution will be invariant and will guarantee stability.

Considering all of these factors, a termination criteria can be formulated.

Algorithm 4.3 (Termination criteria)
Given ϵ. Then at each iteration $k > k_0$:

1. *Check whether $\|Q_z - P_z^{-1}\| < \epsilon$*
2. *Check feasibility of the solution v_k with regard to the LMI set $\mathcal{X}_{LMI}$ given by (4.6).*

k_0 is chosen so that the algorithm iterates away from the initial condition (usually $k_0 = 10$ is enough). This is important because the initial condition is the ERPC solution, which is a local solution. Experiments show that $\epsilon = 0.05$ will usually satisfy the feasibility check, but obviously lower values can also be used.

4.6 The $\mathcal{S}$-Procedure

Recall from section 3.6, that it is desired in certain cases to force the ellipsoid to be larger than a certain polytope $\mathcal{P}$, in order to avoid long narrow ellipsoids. This section presents a different strategy, which confines an ellipsoid, here denoted as $\mathcal{E}_A := \{x | \, x^T R_A x \leq 1\}$, inside the region of attraction $\mathcal{E}_{zx}$, that is, $\mathcal{E}_A \subseteq \mathcal{E}_{zx}$. This is equivalent using the $\mathcal{S}$-procedure, to $\exists \beta$ such that $\forall x$

$$\mathbf{x}^T P_{zx} \mathbf{x} - 1 - \beta(\mathbf{x}^T R_A \mathbf{x} - 1) \leq 0 \tag{4.49}$$

$$\mathbf{x}^T (T_x Q_z T_x^T)^{-1} \mathbf{x} - 1 - \beta(\mathbf{x}^T R_A \mathbf{x} - 1) \leq 0 \tag{4.50}$$

$$\Leftrightarrow \quad \mathbf{x}^T (T_x Q_z T_x^T)^{-1} \mathbf{x} - 1 - \beta(\mathbf{x}^T R_A \mathbf{x} - 1) \leq 0 \tag{4.51}$$

Since $Q_{zx}^{-1} = (T_x Q_z T_x^T)^{-1}$, one can use the matrix inversion lemma (Horn and Johnson (1985) p. 472) to conclude that

$$Q_{zx}^{-1} = P_{xx} - P_{xf} P_{ff}^{-1} P_{fx} \tag{4.52}$$

where the partition of P_z is

$$P_z = \begin{pmatrix} P_{xx} & P_{xf} \\ P_{fx} & P_{ff} \end{pmatrix}, \quad P_{xx} \in \mathbb{R}^{m_x \times m_x}, \quad P_{xf} = P_{fx}^T \in \mathbb{R}^{m_x \times N_c m_u}, \quad P_{ff} \in \mathbb{R}^{N_c m_u \times N_c m_u} \tag{4.53}$$

then inequality (4.52) is equivalent via Schur complement to $\exists \beta > 0$ such that

$$\begin{bmatrix} P_{xx} - \beta R_A & P_{xf} & 0 \\ P_{fx} & P_{ff} & 0 \\ 0 & 0 & \beta - 1 \end{bmatrix} \leq 0 \tag{4.54}$$

The proof is given by Slupphaug (1998). However, inequality (4.54) cannot be used in ERPC in its present form. This is since (4.54) involves the parts of the term P_z, and cannot be applied to ERPC, which does not contain P_z as a variable.

Consider the SSDMPC offline algorithm which *does* uses P_z as a variable (in contrast to ERPC). By adding the LMI (4.54) to the set $\mathcal{X}_{LMI}$ defined by (4.6), and adding the additional variable $\beta > 0$ to the decision variable vector $v(k)$, it is possible to confine an ellipsoid $\mathcal{E}_A$ inside $\mathcal{E}_{zx}$, as is demonstrated by the next example:

Example 4.4 *Consider the system given by (3.29), Chapter 3. Choose the initial ellipsoid $\mathcal{E}_A$ to be a circle, with radius $r = 1.5$, centered in the origin given by:*

$$R_A = \begin{bmatrix} \frac{1}{r^2} & 0 \\ 0 & \frac{1}{r^2} \end{bmatrix}$$

This enforced the ellipsoid to get more round shape, and as can be seeing in figure 4.3, the area around the origin is feasible.

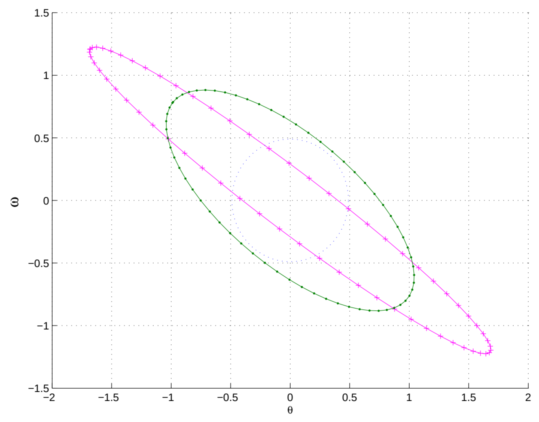


Fig. 4.3: The ellipsoid found by SSDMPC using $\mathcal{S}$-procedure (-*-) with $N_c = 2$, the $\mathcal{E}_A$ is in dotted line $(\cdots)$ and the same ERPC ellipse from Example 3.1 using $N_c = 2$ (-+-)

4.7 The Offline problem

Now the offline problem shall be summarized, and the offline algorithm given. For the system $\mathbf{x}_{k+1} = \Phi_i \mathbf{x}_k + B\mathbf{c}_k$, where $i = 1, .., p$, the problem of finding the invariant ellipsoid $\mathcal{E}_z$ at the iteration k is formulated as an SDP, given by the next set of equations:

$$\begin{aligned}
\min \quad & t && && (4.55a)\\
\text{s.t.} \quad & \begin{bmatrix} t - \nabla \mathcal{L}_A p & p^T \\ p & (\frac{1}{2}\nabla^2 \mathcal{L}_A\ )^{-1} \end{bmatrix} \geq 0 && && (4.55b)\\
& \begin{pmatrix} Q_z & \Psi_i \\ \Psi_i^T & P_z \end{pmatrix} \geq 0 \qquad i = 1, ..., p && \text{Invariance} && (4.55c)\\
& d_i^2 - [K_i \ \ e_i\ ]\ Q_z\ [K_i \ \ e_i]^T \geq 0 && \text{Feasibility} && (4.55d)\\
& F(Q_z, P_z) \geq 0 && && (4.55e)
\end{aligned}$$

where LMI (4.55e) should be interpreted as an affine function representing polytopic state constraints, if presented. Again, the SDP problem (4.55) is solved at each iteration of the Sequential SDP algorithm. The solution is then reduced even further by means of line search methods. When satisfying termination criteria, the algorithm finishes with the local solution v^*.

The affine function $F(Q_z, P_z)$ can, for instance, be a polytope confined inside the solution ellipsoid, or an approximation of an initial ellipsoid confined inside the solution ellipsoid using $\mathcal{S}$-procedure, or constraints on the states. It can then be written as the whole set, a combination of it or any other affine function which depends on Q_z or P_z:

$$\begin{aligned}
& \begin{pmatrix} 1 & v_i^T \\ v_i & Q \end{pmatrix} \geq 0 \qquad i = 1, .., p && \text{Polytope } \mathcal{P} \text{ inside } \mathcal{E}_{zx} && (4.56a)\\
& \begin{bmatrix} P_{xx} - \beta R_A & P_{xf} & 0 \\ P_{fx} & P_{ff} & 0 \\ 0 & 0 & \beta - 1 \end{bmatrix} \leq 0 && \mathcal{S}\text{-procedure} && (4.56b)\\
& \bar{x}_i - [e_i \ \ \mathbf{0}] Q_z [e_i \ \ \mathbf{0}]^T \geq 0, \quad i = 1, .., p && \text{Constr. on states} && (4.56c)
\end{aligned}$$

To conclude this chapter, the final SSDMPC offline algorithm shall be formulated. Given the discrete system $\mathbf{x}_{k+1} = \Phi \mathbf{x}_k + B\mathbf{u}_k$, with constraints on the input and/or states, the algorithm for calculation of the solution Q_z is given as:

Algorithm 4.4 (SSDMPC Offline algorithm)
Step 1- initialization:

Find initial Q_z, P_z, M for example from the ERPC algorithm discussed in chapter 3.

Initialize the Lagrange multiplier matrix Λ, the penalty μ_k and the backtracking parameters ρ_l, η.

Choose a threshold $\epsilon < 1$.

Step 2: Gradient, Hessian

Compute the Gradient using (4.21) and the Hessian, using (4.23).

Check whether the Hessian is Positive Definite. If not, use one of the Hessian modifying methods discussed in section 4.3.4 (e.g. Modified Cholesky).

Step 3: SDP solution

Define a search direction p_k, equation 4.13.

Solve the SDP problem (4.55) to obtain a local solution v_k, using an LMI solver (for instance $mincx$ if using Matlab).

Perform line search backtracking to find the step length α_k (see section 4.4.3).

Step 4: Updating

Update the decision variable $v_{k+1} = v_{k-1} + \alpha_k(v_k - v_{k-1})$.

Update the penalty parameter μ_{k+1} according to the updating rules given in section 4.4.4.

Update the Lagrange multiplier matrix $\Lambda_{k+1} = \Lambda_k + \mu_{k+1}(Q_z P_z - I)$.

Step 5: Termination

If the termination criteria discussed in section 4.5 are not satisfied for ϵ, update $k = k + 1$ and go to step 2. Otherwise, stop iterating with $M_k = M^*$ and execute the ERPC algorithm, now using the full matrix M^* found, to obtain the global solution Q_z^*.

4.8 Numerical Examples

The results from the previous sections shall be demonstrated using a few examples. The volume factor given in the following results is not the *actual* volume of the ellipsoid, but a factor which is proportional to the n dimensional volume of a given ellipsoid. Recall from sections 2.2.1 and 2.2.2 that the volume of ellipsoid $\mathcal{E}_{zx} = \{x | \; x^T P_{zx} x \leq 1\}$ is proportional to the term $(\det P_{zx})^{\frac{1}{2}}$ while the condition number of P_{zx} gives an indication of the shape of the ellipsoids. These tools will be used to compare solutions.

4.8.1 Example 1: Angular positioning system

The next MIMO system is a discrete angular positioning system adapted from Kouvaritakis et al. (2000) and modified to include a second input:

$$\mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k \tag{4.57}$$

$$A = \begin{bmatrix} 1 & 0.1 \\ 0 & 1 \end{bmatrix}, \qquad B = \begin{bmatrix} 0.2 & 0.1 \\ 1.1 & 0.1 \end{bmatrix} \tag{4.58}$$

where $\mathbf{x} = [\theta \quad \dot{\theta}]^T$ is the vector of the angular position and the angular rate respectively and $u_k = [u_1 \quad u_2]^T$ is the input vector, in Voltage, to the two motors. The control constraints $|u_1| < 1V$ and $|u_2| < 1V$ are imposed. Further description about the system is given by Kothare et al. (1996). The optimal feedback gain matrix (the LQ controller) is calculated to be

$$K = \begin{pmatrix} -0.079 & -0.831 \\ -2.508 & 0.123 \end{pmatrix}$$

using $Q = diag(1, 1)$ and $R = diag(0.1, 0.1)$. An invariant ellipsoid $\mathcal{E}_{zx}$ for this system was calculated using the SSDMPC algorithm in section 4.7 with control horizon $N_c = 2$. The Ψ_k matrix (see eq. 3.6a) gets the form

$$\Psi_k = \begin{bmatrix} \mathbf{\Phi}_{2x2} & \begin{matrix} 0.1 & 0.1 & 0 & 0 \\ 1.1 & 0.1 & 0 & 0 \end{matrix} \\ \mathbf{0}_{4x2} & M_{4x4} \end{bmatrix} \qquad \mathbf{\Phi} = \begin{pmatrix} 0.73 & -0.05 \\ -0.33 & 0.09 \end{pmatrix}$$

Figure 4.4 presents three different ROA $\mathcal{E}_{zx}$: (1) ERPC using $N_c = 1$, (2) the ROA found at termination phase of SSDMPC *before* executing the ERPC using the SSDMPC solution M^* and (3) SSDMPC $N_c = 1$ using augmented $\mathbf{c}_k$ as described in section 4.4.2. Both SSDMPC ROA are larger than the ERPC, which is not surprising since extra degrees of freedom allow increase of the ellipsoid in both axes direction. An interesting observation from the figure is that the final SSDMPC ellipsoid $\mathcal{E}_{zx}$ is much larger than the SSDMPC ellipsoid at termination phase $\mathcal{E}_{termination}$. The reason for that is simple: While SSDMPC algorithm finds local solution P_z, M, the convex ERPC algorithm finds a *global* P_z which maximizes the ROA, for the given M from SSDMPC. The motivation of presenting the $E_{termination}$ is to compare between the local SSDMPC solution and the global ERPC, *for a given* unstructured, optimal M^* matrix. A detailed discussion is given in section 4.9.2.

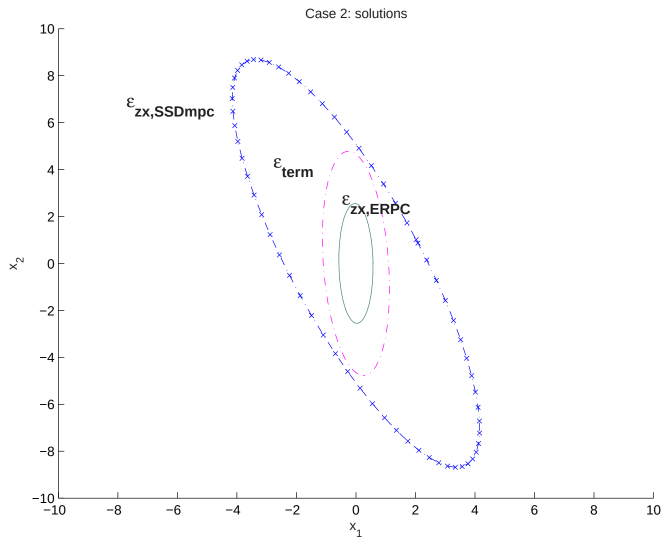


Fig. 4.4: The ellipsoid $\mathcal{E}_{zx,ERPC}$ (solid), $\mathcal{E}_{term}$ is the solution found by SSDMPC at termination phase, before solving the ERPC with M^*(-.-) and the final SSDMPC solution $\mathcal{E}_{zx,SSDMPC}$ (blue –x–)

Table 4.1: Properties of different ellipsoids, example 1

Ellipsoid	μ_k **update**	N_c	**# steps**	**Time [sec]**	**Volume factor**	Cond(P_{zx})
SSDMPC	mon.[a]	1	89	13.48	32.23	11.09
SSDMPC	Aug.[b]	1	36	5.2	17.53	20.71
SSDMPC	mon.	2	100	89.1	68.7	5.62
SSDMPC	Aug.	2	22	25	7.85	16.58
SSDMPC	mon.	3	49	250	62.0	7.17
SSDMPC	Aug.	3	43	268	36.64	11.4
ERPC	-	1	-	0.56	0.90	15.18
ERPC	-	2	-	0.7	1.46	19.75
ERPC	-	3	-	1.47	2.18	22.2
ERPC	-	5	-	5.34	4.3	22.21

[a] Monotonic rate
[b] Augmented at $k = 11$ to $\mu_k = 1000$

Table 4.1 summarizes the results for Example 2. Generally, the SSDMPC gives larger ROA than the ERPC. For $N_c = 2$, the size of the SSDMPC update at monotonic rate is 47 times larger than the ERPC(2). For a given horizon length N_c, the SSDMPC was executed twice: Updating the penalty μ_k at a monotonic rate, denoted by *mon.*, and augmenting μ_k at $k = 11$ to $\mu_k = 1000$ in order to penalize the residual term in the Lagrangian (4.9), thus converge faster to the solution. Details about this strategy is given in section 4.4.2, and the simulations verifies the theory that penalizing heavily the residual term at early steps, forces the Lagrangian to iterate towards a local solution. As the table indicates, this local solution is not global, since for the same horizon length, a better solution is found by increasing μ_k at a monotonic rate. The number of iterations is significantly smaller for augmented μ_k than for increase at monotonic rate, but notice that the total computation time for the case $N_c = 3$ is larger when using augmented update. Refer to the discussion at section 4.9.2 for further analysis for this matter.

SSDMPC algorithm usually produced ellipsoid with smaller condition number than the equivalent ERPC. With unstructured M, the ellipsoid can be expended in different directions, making it more circular around the origin, thus reducing the condition number. The monotonic increase rate of μ_k results in better condition number than the augmented. To see how the condition number is low for the SSDMPC mon. case $N_c = 2$, consider P_{zx} with eigenvalues

$$P_{zx} = \begin{pmatrix} 0.03 & 0.01 \\ 0.01 & 0.01 \end{pmatrix}, \quad \lambda = \begin{pmatrix} 0.006 \\ 0.034 \end{pmatrix} \tag{4.59}$$

and $\lambda_{max}/\lambda_{min} = 5.6 = \kappa(P_{zx})$.

Figure 4.5 illustrates some of the convergence properties of the SSDMPC algorithm. Two cases are presented: the monotonic increase rate (solid lines) and the augmented μ_k (dotted). It only takes 22 iteration when μ_k is augmented to 1000 at iteration $k = 11$, while 100 iterations are needed to converge when μ_k increase monotonically. The norm $\|Q_z P_z - I\|$ is decrease immediately after μ_k is augmented, allowing faster convergence. The SSDMPC algorithm was terminated when the Invariance LMI was satisfied and the norm $\|P_z Q_z - I\|$ was less than a threshold $\epsilon = 10^{-2}$

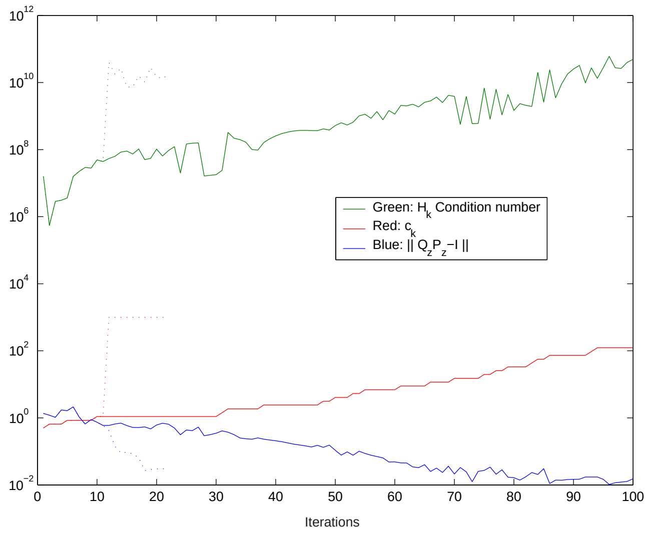


Fig. 4.5: SSDMPC convergence for two cases: monotonic increase rate of μ_k (solid lines) and augmented (dotted lines). The Hessian condition number(green), the penalty μ (Red) and the norm $\|Q_z P_z - I\|_\infty$ (Blue). Notice how fast it converge when μ_k is augmented at $k = 11$.

4.8.2 Example 2: Passenger aircraft longitudinal control

This example investigates an implementation of a three states system with a single control input. Modelling an aircraft is usually done by six non-linear equations to describe the longitudinal and lateral motion, but this example assumes constant speed at altitude $5000ft$. Using figure 4.6, the nonlinear equations can be described as

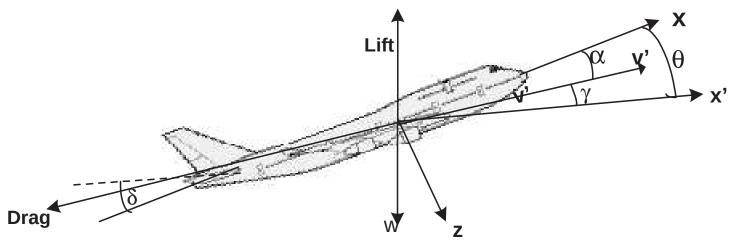


Fig. 4.6: The forces diagram of the passenger aircraft

$$\dot{\alpha} = \mu\Omega\sigma\left[-(N_L + D)\alpha + (\frac{1}{\mu} - N_L)q - (M_W \sin\gamma)\theta + N_L\right]$$

$$\dot{q} = \frac{\mu\Omega}{2I_{yy}}\left[(Y_M - \eta(N_L + D))\alpha + (Y_M + \sigma Y_M(1 - \mu N_L))q + (\eta N_W \sin\gamma)\delta_e\right] \qquad (4.60)$$

$$\dot{\theta} = \Omega q$$

$$(4.61)$$

where the states are α Angle Of Attack (AoA), q is the pitch rate (rad/sec), θ is the pitch angle (rad) and the control input δ_e is the elevator deflection angle (rad) The coefficients are:

$\mu = \rho_a S\bar{c}/(4m)$		$\Omega = 2U/\bar{c}$	
ρ	Density of surrounding air	S	Planform area of the wing
$\bar{c}$	Average chord length	m	Mass of the aircraft
U	Flight velocity	C_T	Coefficient of thrust
D	Drag coefficient	N_L	Lift coefficient
N_W	Weight coefficient	N_M	Pitch moment
γ	Flight path angle	$\sigma = 1/(1 + \mu C)$	Constant
I_{yy}	Moment of inertia about y axis	$\eta = \mu\sigma C$	Constant

This notation is compatible with Fossen (2002). The nonlinear model is linearized around

small angles at constant cruise speed and then converted to a discrete system using a sampling-rate $T_s = 0.01$. The model can now be written in the state space form

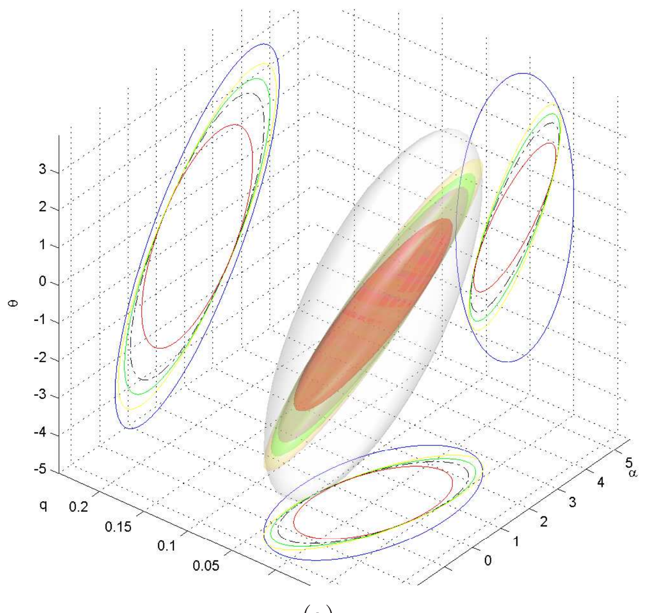
$$\begin{bmatrix}\alpha_{k+1}\\ q_{k+1}\\ \theta_{k+1}\end{bmatrix} = \begin{bmatrix}0.9968 & 0.05649 & 0\\ -0.0001 & 0.9957 & 0\\ 0 & 0.5658 & 1\end{bmatrix}\begin{bmatrix}\alpha_k\\ q_k\\ \theta_k\end{bmatrix} + \begin{bmatrix}0.0024\\ 0.0002\\ 0.0001\end{bmatrix}\delta_e(k) \tag{4.62}$$

$$= A\mathbf{x}_k + B\delta_e(k) \tag{4.63}$$

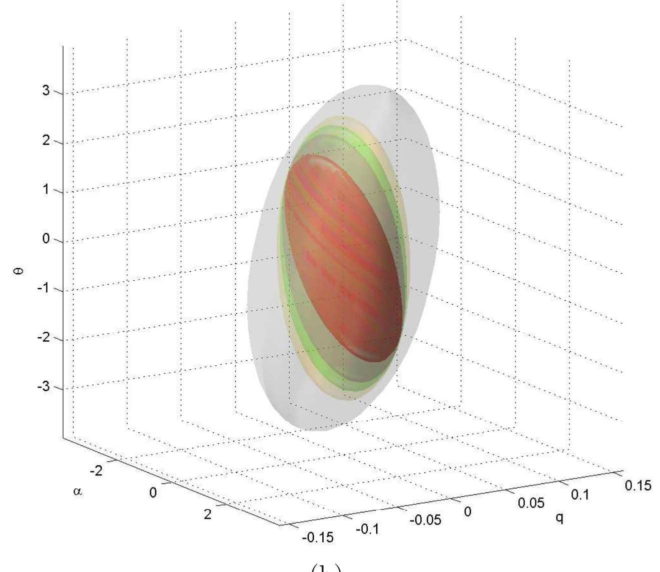
$$y_k = (0\ \ 0\ \ 1)\mathbf{x}_k = C\mathbf{x}_k \tag{4.64}$$

$$\tag{4.65}$$

The system (4.65) has a pole in the unit circle $z = 1$ which makes it marginally stable. Moreover, the aileron deflection has a physical limitation of $|\delta_e| < 60deg$ which presents a hard constraint on the input. The states are penalized using the matrix $Q = diag(1, 100, 1)$ and the weight on the input is $R = 1$. Using the offline SSDMPC algorithm to calculate the invariant ellipsoid $\mathcal{E}_{zx}$ a solution was found after 300 iterations, with an average of $10^{-4}sec$ per iteration (total simulation time approx. 60 seconds). The ERPC algorithm was also tested using different N_c values. The results are displayed in figure 4.7.



(a)



(b)

Fig. 4.7: Four different ellipsoids: Innermost green is the ERPC solution for $N_c = 2$,next $N_c = 25$ blue, the purple is $N_c = 35$ while the outmost, grey, is the solution for SSDMPC $N_c = 2$

The solution given by the ERPC algorithm with horizon $N_c = 2$ is not especially useful, as the ellipsoid becomes a thin disk, thus covering only a small region around the origin. It will be shown later (online section) that many initial points around the origin are outside the ERPC ellipsoids, thus not feasible and so cannot be used to calculate perturbation vector $\mathbf{c}_k$. Moreover, the SSDMPC ellipsoid is 6.2 times larger than the

Table 4.2: Offline properties for ellipsoids, example 2

Ellipsoid	N_c	**Time [sec]**	**Volume factor**	Cond(P_{zx})
SSDMPC	2	60	0.5522	5095.3
ERPC	2	0.55	0.0889	16461
ERPC	25	233	0.128	22482
ERPC	35	1418	0.1481	25116
ERPC	45	6597	0.1702	27706

ERPC ellipsoid (see the volume factor, table 4.2), even though the control horizon is the same. Using $N_c = 25$ the ERPC gives a solution after 233 seconds, with the square matrix P_z of size $m_x + m_u N_c = 3 + 25 = 28$. Increasing N_c to 35 still gives a solution much smaller in size than the SSDMPC ellipsoid (with $N_c = 2$), after 23.6 minutes, significantly longer time than taken to calculate the SSDMPC solution. This ellipsoid is the purple ellipsoid in figure 4.7. The largest ERPC ellipsoid which was calculated with $N_c = 45$ is only 1.15 times larger than the size with $N_c = 35$, but it was however, far more costly, as it took 1 hour and 49 minutes to calculate, with solution P_z of size 48×48, fairly large matrix for online application. The calculation time for each ellipsoid and the costs are given in table 4.2

Notice that the projections presented in the figure to the left can be deceptive, as they do not represent the actual volume of the ellipsoids, just the projection in the two dimensional plane. It may seem that the ERPC ellipsoid for $N_c = 2$ covers a large area in the $\alpha - q$ plane, but since the ERPC ellipsoid is a thin disk (observe from the 3D view), it covers a small volume. The volumes of the ellipsoids can be compared using the volume factor given in table 4.2. It is easily seen that the SSDMPC ellipsoid is the largest of all solutions, despite the short horizon N_c and the small size of the solution Q_z.

Table 4.2 also shows the condition number of P_{zx}. Recall from section 2.2.2 that the condition number of P_{zx} can be a measure for the ratio between the long and short axes of the ellipsoid. This quantity is $4 - 6$ times larger for ERPC than for SSDMPC. Low values indicate 'spherical' ellipsoids, circular in shape, while high values indicate long thin ellipsoids. The SSDMPC solution gives the preferable rounded ellipsoid, with a large diameter around the origin. Using (2.25), we can find the direction in which the SSDMPC ellipsoid has expended most. The size of the ellipsoid axes found by SSDMPC, ERPC $N_c = 2$ and ERPC, $N_c = 45$ are given as

$$a_{SSDMPC} = \begin{pmatrix} 0.06 \\ 1.89 \\ 4.55 \end{pmatrix}, \quad a_{ERPC(2)} = \begin{pmatrix} 0.02 \\ 1.63 \\ 2.64 \end{pmatrix}, \qquad a_{ERPC(45)} = \begin{pmatrix} 0.02 \\ 1.66 \\ 4.13 \end{pmatrix} \tag{4.66}$$

It is then possible to calculate the axes ratio as

$$r_i = \frac{a_i(SSDMPC)}{a_i(ERPC(2))}, \quad i \in \{1, 2, 3\} \tag{4.67}$$

While $a_3\,[ERPC(45)]$ is almost twice as large as $a_3\,[ERPC(2)]$ (i.e. 4.12/2.6), the first axis $a_1\,[ERPC(45)]$ is nearly unchanged (0.02/0.02). This causes the condition number to increase. Figure 4.8 presents these ratios in the three directions for two cases: (1) How many times the SSDMPC ellipsoid is bigger than the ERPC for N_c (blue columns) and (2) how many times the ERPC ellipsoid for $N_c = 45$ is bigger than the ERPC ellipsoid for $N_c = 2$ (red columns). For example, the SSDMPC ellipsoid is 3.1 times larger than the ERPC, $N_c = 2$ ellipsoid in the direction of state α. In comparison, the ERPC ellipsoid for $N_c = 45$ is only 1.2 times larger than $N_c = 2$ in the same direction. This figure emphasizes the superiority of the SSDMPC solution over the ERPC in this case. Even when increasing the horizon length of the ERPC from $N_c = 2$ to $N_c = 45$, its ellipsoid remains much smaller than that of the SSDMPC.

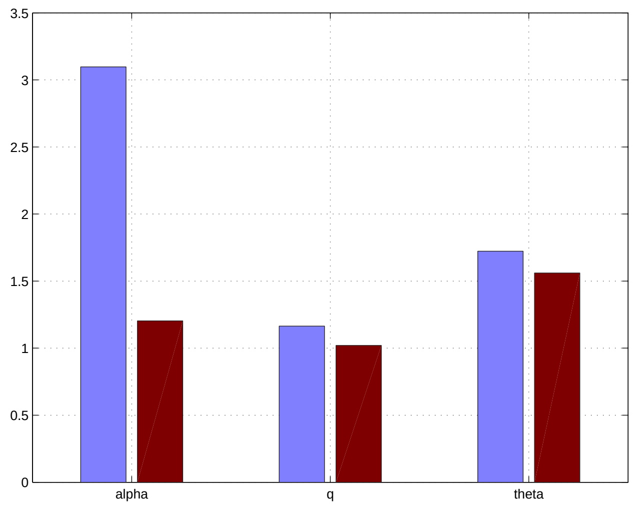


Fig. 4.8: The ellipsoid axes ratio SSDMPC-ERPC(2) (blue columns) and ERPC(45)-ERPC(2)(red columns).

4.8.3 Example 3: B-27 Aircraft Lateral motion control

Next, the control of lateral motion problem of the B-27 aircraft, based on Anderson and Moore (1989) shall be considered. The general equations governing the aircraft motion are the linear set

$$\begin{bmatrix} \dot{\phi} \\ \ddot{\phi} \\ \dot{\beta} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & L_p & L_\beta & L_r \\ \frac{g}{V} & 0 & Y_\beta & -1 \\ \frac{N_{\dot{\beta}} g}{V} & N_p & N_\beta + N_{\dot{\beta}} Y_\beta & N_r - N_{\ddot{\beta}} \end{bmatrix} \begin{bmatrix} \phi \\ \dot{\phi} \\ \beta \\ r \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ L_{\delta r} & L_{\delta a} \\ Y_{\delta r} & 0 \\ N_{\delta r} + N_{\dot{\beta}} Y_{\delta r} & N_{\delta a} \end{bmatrix} \begin{bmatrix} \delta_r \\ \delta a \end{bmatrix}$$

where ϕ denotes the bank angle (rad), β is the sideslip angle(rad), r is the yaw rate (in rad/sec), δ_r the rudder deflection and δ_a is the aileron deflection (both in rad). L_p, L_β, L_r and Y_r are fixed parameters associated with the aircraft, and N_i represents the forces on the aircraft. Linearizing the model around $1000m$ height with constant speed gives the next numerical values, using Euler discretization:

$$\mathbf{x}_{k+1} = A_d \mathbf{x}_k + B_d \mathbf{u}_k$$

the states vector is $\mathbf{x} = [\phi \; \dot{\phi} \beta \; r]^T$, $u_k = [\delta_r \; \delta_a]^T$ are the control inputs, and the matrices

$$A = \begin{bmatrix} 1 & 0.01 & 0 & 0 \\ 0 & 0.99 & -0.73 & 0.03 \\ 0.0009 & 0 & 0.9989 & -0.01 \\ 0.0001 & 0.0009 & 0.0895 & 0.99 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 & 0.0100 \\ 0 & -0.0391 \\ 0.0004 & 0 \\ -0.0253 & 0.3100 \end{bmatrix}$$

The poles of this system lie in $.9971 \pm 0.0294i$, 0.9898, 1.0. Two poles lie outside the unit circle, which makes the system unstable. Furthermore, physical constraints on the rudder and aileron deflections are presented, i.e. $|\delta_r| < 55deg$ and $|\delta_a| < 55deg$, which makes it harder to stabilize the aircraft. Thus a linear quadratic feedback controller is implemented where $Q = diag(1, 1, 1, 1)$ and $R = diag(0.1, 0.1)$. Since the system has four states, the ellipsoid $\mathcal{E}_{zx}$ is four dimensional, and must be projected in the $\mathbb{R}^3$ or $\mathbb{R}^2$ in order to be visualized.

The solution for the SSDMPC offline problem is found after 62 iterations. This and a few selected ERPC solutions are presented in figure 4.9. The largest ellipsoid is the SSDMPC solution, where $Q^*_{z,SSDMPC}$ is a square matrix of size $m_x + m_u N_c = 8$. The

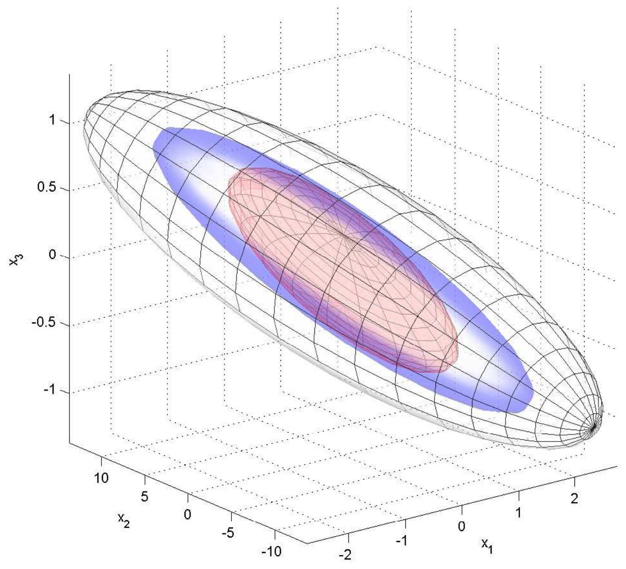
Table 4.3: Offline properties of different ellipsoids, example 3

Ellipsoid	N_c	**Time [sec]**	**Volume factor**	Cond(P_{zx}) [$\times 10^3$]
SSDMPC	2	101	20.16	6.07
ERPC	2	0.455	0.84	8.254
ERPC	5	4.0	1.8594	5.923
ERPC	10	86.50	4.9507	5.75
ERPC	15	683.798	11.0857	6.138

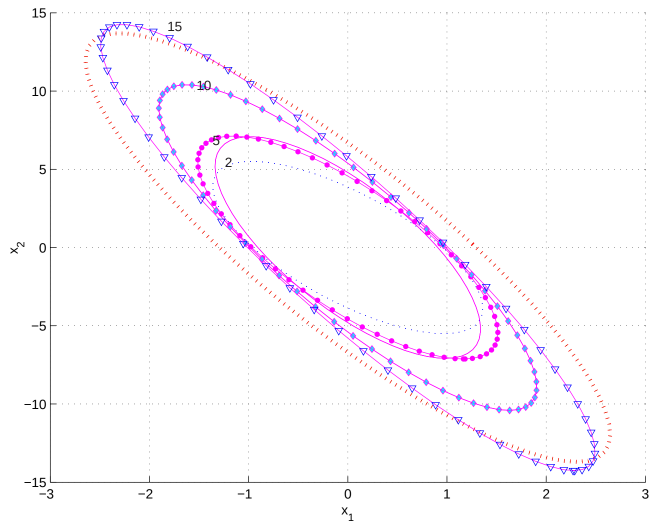
innermost ellipsoid is the ERPC solution with the same control horizon $N_c = 2$ and the middle size ellipsoid resulted by increasing horizon N_c for the same ERPC algorithm, i.e. $N_c = 10$. In order to achieve an ERPC solution of the same size as the SSDMPC, the size of the horizon was increased to $N_c = 15$ (see the right figure 4.9). This results in a $P_{z,ERPC}$ matrix of size 34, four times bigger than the SSDMPC solution! This will have a serious implications for the online simulations, as the number of operations performed in the simulation is significantly increased.

Table 4.3 displays the properties of a few selected solutions. The volume factor clearly indicates the ellipsoid volume for the SSDMPC solution is larger than any of the ellipsoids created by the ERPC algorithm. It is almost twice as large as the ERPC with $N_c = 15$, but with much smaller matrix dimensions, and 22 times larger than the ERPC with the same control horizon! The general shape of the ellipsoids (presented by the large condition numbers of the matrix P_{zx}) is a long ellipsoid. This is due to the system dynamics, the system becomes saturated fast for large sideslip angle values β. This is due to the linearization of the model (4.60). Although the SSDMPC solution is large, its condition number is lower, which indicates that the ellipsoid shape is more circular than the ERPC's.

The Modified Cholesky algorithm was applied in this example, as figure 4.10 indicates, where the condition number of the Hessian is maintained relatively low, but since it was applied only occasionally, the algorithm still converged to a desired low value of the norm $\|Q_z P_z - I\|$.



(a) 3D projection



(b) 2D projection

Fig. 4.9: Projection of the ellipsoids in the $\mathbb{R}^3$ space (a) and in $\mathbb{R}^2$ (b). The red inner ellipsoid is the ERPC solution with horizon $N_c = 2$, the blue (smooth) is ERPC solution using $N_c = 10$ while the outer(wireframe) is the SSDMPC solution $N_c = 2$ (in the left figure)

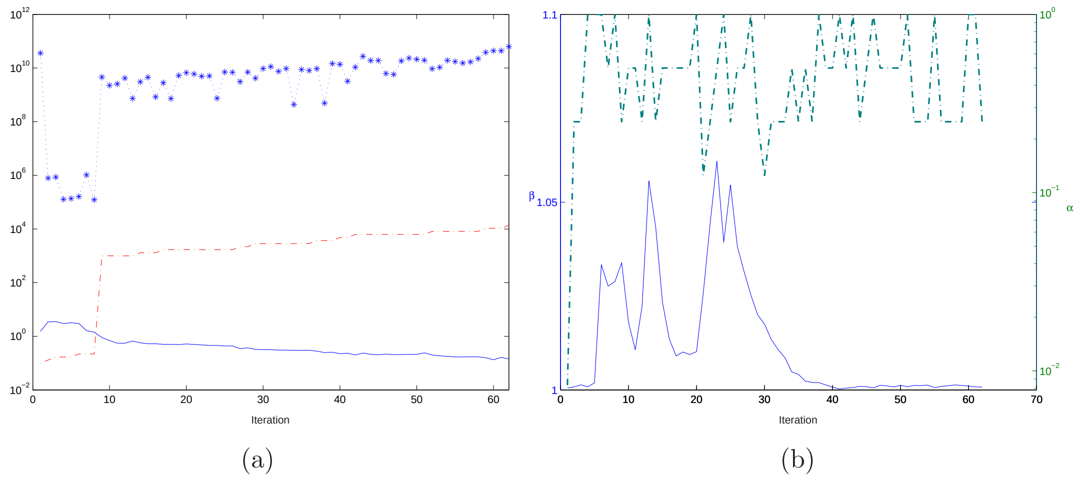


Fig. 4.10: Left: The condition number $(\cdots * \cdots)$, Penalty μ_k (-.-) and $\|Q_zP_z - I\|$. Right: Step Length α_k.

4.8.4 Example 4: F404 Aircraft Engine

The next example is adapted from Ennix et al. (1993). The $F404 - GE - 400$ is a two-spool, low-bypass, axial-flow turbofan with afterburner. The engine consists of a three-stage fan driven by a single-stage, low-pressure turbine and a seven-stage, high-pressure compressor driven by a single-stage, high-pressure turbine. Variable geometry is incorporated into the fan and high-pressure compressor and there is a convergent-divergent nozzle. It is equipped with an Engine Control Unit (ECU) where idle power is 35^o Power Level Angle (PLA) and intermediate (maximum nonafterburning) power is 102^o PLA. Figure (4.11) is a sketch of the engine, with the different phases.

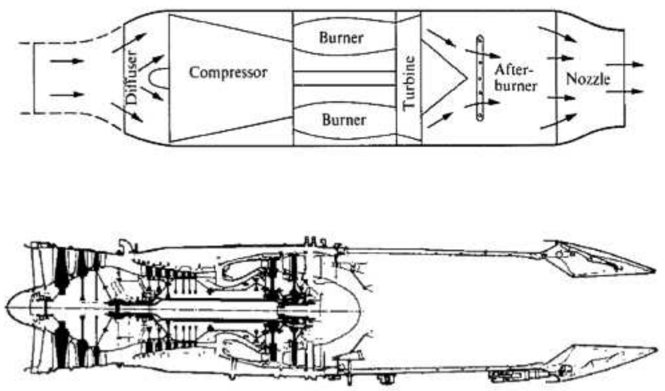


Fig. 4.11: Sketch of the F404 Engine

The linear discrete model is given below

$$\mathbf{x}(k+1) = \begin{pmatrix} -0.93 & 0 & 0.11 \\ 0.007 & 0.98 & -0.02 \\ 0.01 & 0 & 0.85 \end{pmatrix} \mathbf{x}(k) + \begin{pmatrix} 0.021 & 0.25 \\ 0.019 & -0.005 \\ 0.024 & -0.003 \end{pmatrix} \mathbf{u} \tag{4.68}$$

where

$\mathbf{u} = [\ W_f(pph)$ fuel injection, $\quad A_8(sq\ in)$ areal nozzle $]^T$

$\mathbf{x} = [\ N_2(rpm), \quad N_{25}(rpm), \quad T_{45}(deg\ Fah)\ ]^T$

The system is normalized, and the control inputs have the constraints

$$|\mathbf{u}| < I$$

and the penalty on the states and inputs are

$$Q = \text{diag}\ (10, \ \ 10, \ \ 10), \qquad R = \text{diag}\ (0.05, \ \ 0.05)$$

A solution to the SSDMPC offline was found after 128 iterations. The SDP calculation time was high only for the first iteration, since an arbitrary initial condition t_0 was

Table 4.4: Properties of different offline solutions, example 4

Ellipsoid	N_c	**Time [sec]**	$\mathbf{V_f}$	Cond(P_{zx})
SSDMPC	2	95	0.36	55.4
ERPC	2	0.55	0.06	31.9
ERPC	5	2.66	0.3	30.7

set, while the rest of the simulation used initial conditions t_{k-1} found at the preceding iterations. Thus a significant reduction in calculation time was observed, from $0.01sec$ to $1sec$.

Table 4.4 summarize the size and form of each ellipsoid. The volume factor of the SSDMPC ellipsoid is 6 times larger than an equal horizon length ERPC ellipsoid.

Comparison of the length of each axis reveals that $\mathcal{E}_{zx,SSDMPC}$ is larger than $\mathcal{E}_{zx,ERPC}$ for $N_c = 2$ in x_2 axis direction by a factor of 2.8, while it has the same length in the other axes. The ERPC(5) ellipsoid has almost the same volume as the SSDMPC(2), but is more circular (with a lower condition number).

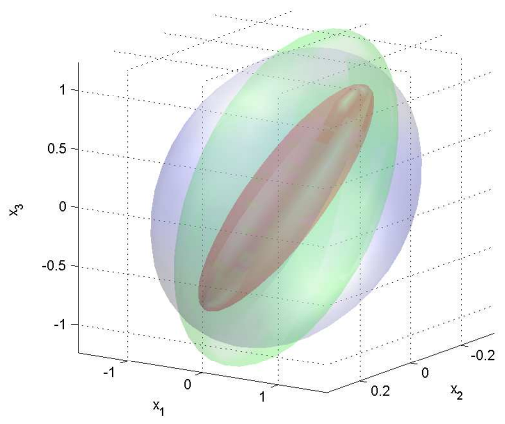


Fig. 4.12: Ellipsoids example 4: innermost red is the ERPC ellipsoid for $N_c = 2$. Green: ERPC ellipsoid $N_c = 5$. Grey: SSDMPC ellipsoid.

4.8.5 Example 5: DSRV

Deep Submerged Rescue Vehicles are autonomous submarines designed for quick deployment in the event of a submarine accident. At the accident site, the DSRV works with either a 'mother' ship or 'mother' submarine. The DSRV dives, conducts a sonar search, and attaches to the disabled submarine's hatch. DSRV's can embark up to 24 personnel for transfer to the 'mother' vessel. A simplified model of such DSRV is given below.

Table 4.5: DSRV solutions computed by offline algorithms, example 5

Ellipsoid	N_c	**Time [sec]**	$\mathbf{V_f} \times 10^{-3}$	Cond(P_{xz})
SSDMPC	1	47.46	5.5	47.4
SSDMPC	2	150.4	3.6	88.28
SSDMPC	3		4.7	58.6
ERPC	1	1.25	2.1	206.9
ERPC	2	1.95	2.2	183.6
ERPC	3		2.4	164.9

$$\begin{pmatrix} w(k+1) \\ q(k+1) \\ x(k+1) \\ z(k+1) \end{pmatrix} = \begin{pmatrix} 0.99 & -0.006 & 0 & 0.0005 \\ -0.032 & 1.032 & 0 & -0.096 \\ 0 & -0.0001 & 1.0 & -0.021 \\ -0.0002 & 0.01 & 0 & 0.99 \end{pmatrix} \begin{pmatrix} w(k) \\ q(k) \\ x(k) \\ z(k) \end{pmatrix} + \begin{pmatrix} 0.004 & 0.0001 \\ -0.037 & -0.0102 \\ 0 & 0 \\ -0.0002 & -0.0001 \end{pmatrix} \begin{pmatrix} \delta_e \\ u_i \end{pmatrix} \tag{4.69}$$

where

w	= heave velocity	(m/s)
q	= pitch velocity	(rad/s)
x	= x-position	(m)
z	= z-position, positive downwards	(m)

and the inputs are δ_e(rad) is the stern plane and u_i is an auxiliary propel speed (rad/sec). The model 4.69 is based on a nonlinear model given by A.J. Healey (1992), and is linearized around constant speed $U_0 = 4.11 m/s$, and descritized using $T_s = 0.001 sec$. The nonlinear model is presented in appendix C.1.1.

Table 4.5 presents the offline SSDMPC and ERPC results. Since the SSDMPC uses augmented increase rate of μ_k for this model, the solution obtained for $N_c = 2$ and $N_c = 3$ has smaller ROA than for $N_c = 1$. It is due to the fact augmenting the penalty to a high value ($\mu_k = 1000$ when $k = 11$) forces the algorithm to converge to a local solution. Although the volume factor is higher for the SSDMPC case, the condition number is considerable lower than the ERPC case. This means that the ROA is more circular around the origin for the SSDMPC solutions. The ROA obtained by ERPC increases only about $4\% - 6\%$ for increase in horizon length, a very slow increase rate.

4.9 Summary and Discussion

4.9.1 Review of Chapter 4

This chapter presented a new method for expending the invariant ellipsoid introduced in chapter 3. Instead of using a constant matrix M, a variable full matrix is used to achieve the effect of infinite control horizons. Since the matrix Ψ_k is a variable containing M_k, it is necessary convert the BMI (4.4) to an LMI form, using Schur complement. This innovation introduces a new difficulty, namely that equation $Q_z P_z - I = 0$ must be satisfied. Including this nonlinear equation as a constraint to the original $max\,\det$ problem (3.28), it was no longer possible to solve the problem using the standard LMI solvers.

This chapter presented a strategy to solve the new non-linear non-convex problem namely to include the nonlinear equation in a Lagrange function (4.9), approximating it quadratically and then introducing a slack variable t. This makes it possible to define a new convex problem, where t is minimized, and subjected to the inequality $t \geq \nabla\mathcal{L}_A^T p_k + \frac{1}{2} p_k^T \nabla^2 \mathcal{L}_A p_k$ and the rest of the LMI set, where the search direction is defined as $\mathbf{p}_k = [Q_z(k) - Q_z(k-1), P_z(k) - P_z(k-1)]^T$. This way, if a minimum solution to t is found, it is equal to the quadratic term given above, thus finding a local solution v_k.

After finding a local solution to the SDP (4.55), backtracking is performed, in order to find the step size α along the search direction p_k, according to the first and the second Wolfe conditions (Nocedal and Wright; 1999). This step length α is used to calculate the new iterate decision variable v_{k+1} , i.e. $v_{k+1} = v_{k-1} + \alpha_k [v_k - v_{k-1}]$. The termination criteria are then checked, and if not satisfied, the SDP (4.55) is solved again, using the new decision variable v_{k+1} (the origin of the name *Sequential* SDP).

If the termination criteria are satisfied, the local optimal solution is $M_k = M^*$ and the convex ERPC problem (3.28) is solved using this full matrix M^*, to obtain the final solution Q_z^*. The inverse of Q_z^*, namely P_z^* is then used to solve the efficient online MPC.

4.9.2 SSDMPC Offline Discussion

The results from the examples presented in section 4.8 are summarized in table 4.6 below. The purpose of the table is to compare the SSDMPC algorithm with the ERPC algorithm with regard to the size of the solution, the volume of the ellipsoids, computation time, and so on. The benefits and drawbacks of each algorithm will be presented here, as well as certain SSDMPC algorithm issues.

Table 4.6: Comparison between ellipsoids

Example	**Ellipsoid**	N_c	**Time [sec]**	**Volume factor**	Cond(P_{zx})
1. Angular pos.*	SSDMPC	1	13.5	32.23	11.09
	SSDMPC	2	89.1	68.7	5.62
	SSDMPC	3	250	62.0	7.17
	ERPC	1	0.56	0.90	15.18
	ERPC	2	0.7	1.46	19.75
	ERPC	3	1.47	2.18	22.2
	ERPC	5	5.34	4.3	22.21
2. Passenger Aircraft	SSDMPC	2	60	0.55	5095.3
	ERPC	2	0.55	0.09	16461
	ERPC	25	233	0.13	22482
	ERPC	35	1418	0.15	25116
	ERPC	45	6597	0.17	27706
3. B-27 aircraft	SSDMPC	2	101	20.16	6070
	ERPC	2	0.455	0.84	8254
	ERPC	5	4.0	1.86	5923
	ERPC	10	86.50	4.95	5750
	ERPC	15	683.798	11.08	6138
4. F-404 Engine	SSDMPC	2	60	0.33	60.2
	ERPC	2	0.55	0.06	31.9
	ERPC	5	2.66	0.30	30.7
5. DSRV model	SSDMPC	1	47.46	0.0055	47.43
	SSDMPC	2	150.4	0.0036	88.28
	SSDMPC	3		0.0047	58.66
	ERPC	1	1.25	0.0021	206.9
	ERPC	2	1.95	0.0022	183.6
	ERPC	3		0.0024	164.9

* Only results for monotonic increase rate of μ_k are given here, since they have larger ROA sizes

The Size of Ellipsoids

As the examples indicate, the size and shape of the region of attraction (the ellipsoid contains the origin) is affected by the horizon length, both for the ERPC and the SS-DMPC case. Increasing the horizon length does not always result in large growth of the ellipsoids, sometimes the growth rate is very low, as in the case of the passenger aircraft, example 2. Increasing the horizon length by a factor of 22.5 (from $N_c = 2$ to $N_c = 45$) only increases the volume of the ellipsoid by factor of 1.91. This slow expansion of the volume is impractical for most cases since the size of the matrix P_z, used in the online algorithm, increases with growing horizon length. This will cause high computation time and will consume large amounts of resources, such as CPU and memory, which are not always available for embedded systems (due to limitations such as weight and space). For some cases, the computational time must not exceed a certain limit, as for systems with high sampling-rates. This will be discussed in detail in chapter 5. The solution of ERPC algorithm using a non-structured M^* matrix (computed by SSDMPC) generates ellipsoids *at least* as large as the structured M matrix. The question is whether finding such non-structured M^* matrices is worth the effort. For some cases, e.g. examples 1 and 3, the regions of attraction generated by ERPC using non-structured M^* were 20 times larger than those generated by structured M ERPC. In these cases, the answer is definitely yes, since large ellipsoid sizes, together with small P_z matrix sizes, are crucial factors in real-time embedded systems with tight dynamic requirements. Notice that for some cases, the SSDMPC algorithm generates larger ROA for $N_c = 1$ than for $N_c = 2$. This is due to the fact the SSDMPC gives only a local solution M^*. The algorithm using N_c may have crossed upon a local solution M^*, which generates smaller ROA than the one found by $N_c = 1$. Hence a disadvantage of the local convergence.

The size of the region of attraction is greatly affected by the LQ gain matrix K. The Newton Raphson algorithm is applicable only if the initial condition lies inside the region, i.e. $\mathbf{x}(0) \in \mathcal{E}_{zx}$. If, for a certain application, the initial point is outside this region, it is possible to enlarge the invariant set by using low-tuned K. Low-tuned gain K is achieved by increasing the weights on the inputs (R values), thus reducing the effect that the input admission has on the system. It is possible to switch between two different solutions, based on a detuned K and a highly-tuned K when needed. If the start point $\mathbf{x}_0$ is outside the ellipsoid, use a large ellipsoid, found by the less tuned gain K. Given the stabilizing nature of K and the monotonic decrease of the cost, it is apparent that $\mathbf{x}$ is progressively steered toward the origin (Kouvaritakis et al.; 2000). Thus there is likely to exists a finite time where $\mathbf{x}$ is feasible for $\mathcal{E}_{zx}$ generated by the highly-tuned K. It is then possible to 'switch' online between the solutions, thus guaranteeing the feasibility of $\mathbf{x}$. This will be discussed with further details section 7.4.1.

Shape of Ellipsoids

It is desired in many cases for the region of attraction to have a circular shape around the origin. It is not always possible to achieve this physically, since the states may have different physical meanings, or may be scaled differently. Consider, for instance, a simple model of DC motor, where the states are induced current i_a, given by units of Amperes, and angular velocity ω_m, given by rad/sec. Usually, such motors work with very low Amperes, but a range of a few thousands rounds per second. Obviously, without proper scaling of such systems, the region of attraction will be very long and narrow. As has been proven in section 2.2.2, the condition number is a very good tool for measuring the shape of the ellipsoid. A few of the examples given by the table above are scaled (such as the $F404$-engine and the DSRV model), so the condition number is low. Examples which were not not scaled exhibit a large condition number (especially the passengers aircraft example). Even in these examples, it is useful as a tool for comparison of algorithms and horizon length.

Table 4.6 indicates that the condition number of the SSDMPC ellipsoids is generally lower than the ERPC's, despite the fact that these ellipsoids are significantly larger.

This is not surprising since extra degrees of freedom are applied in certain directions, states, and the expansion of the ellipsoid is not uniform. For some cases (see example 2), the ERPC ellipsoid has a large difference between the short axis and the long axis, resulting in a large condition number. While simply increasing the N_c only increases this difference, the un-structured M^* with extra degrees of freedom gives expansion in the short axis direction, thus reducing the condition number.

It is possible to reduce the condition number of the ERPC and SSDMPC, by forcing the ellipsoid to be larger than its original size, in the direction of certain axes. This can be done by enforcing a polytope inside the ellipsoid, for example, or when using SSDMPC, it is possible to compute an ellipsoid which contains another ellipsoid by the $\mathcal{S}$-procedure. Note that introducing additional constraints to the ERPC or SSDMPC will most likely reduce the size of the ellipsoid, so a trade-off should be considered.

Offline Initial condition

The initial variables are crucial in finding the desired solution. By using a 'Hot start' (for instance, ERPC solution with the same horizon) the SSDMPC algorithm has a better chance of converging to a local solution. A few experiments were made using an arbitrary initial point but none of these yielded satisfactory results. This is due to the large number of iterations required to converge from a remote starting point. The penalty parameter then is increased to high values, such that the Hessian gets ill-conditioned and numerical inaccuracies (as well as frequent modification of the Hessian) cause loss of second-order information. Thus, due to the nature of the Lagrange multiplier method, the initial variables have to lie in the vicinity of a local solution if convergence is to be achieved.

The SSDMPC algorithm experiences numerical problems if the initial condition P_z, computed by the ERPC, has a large condition number. P_z may not be symmetric and the SSDMPC will not have feasible LMIs. In these cases, consider re-scaling the system or, if possible, tune the LQ weights.

Hessian Modification

While solving the sequential SDP problem, it was necessary to modify the Hessian more than once, making it positive definite. A few strategies are given in section 4.3.4. The method of eigenvalue modification shows good convergence properties, but only when the initial condition is not too far from some local minima. If this is not the case, then it might take many iterations to get closer to a local minimum point. Alas, the Hessian may become ill-conditioned after many iterations, since the penalty parameter increases significantly. The modified Cholesky algorithm can be a good solution, since it reduces the condition number of the Hessian. This is not without limitation: repeatedly using the modified Cholesky strategy causes problems in convergence, since the Hessian is altered to a great extent, losing second-order information, and eventually diverging from the local solution. It is possible to tune the level of modification, so if convergence is intractable, it is preferred to allow only small Hessian modifications, just enough to make it positive definite, with little regard to the condition number. Tuning is done by choosing suitable parameters (see section 4.3.4 and the algorithm in appendix A).

Nocedal and Wright (1999) discusses the algorithm and the choice of parameters further. Fares et al. (2000) suggest using the modified Cholesky while neglecting the term $Q_z P_z - I$ in the Hessian matrix. This is fairly reasonable as the modified matrix is approaching the true Hessian when the term $Q_z P_z - I$ is approaching zero, that is, when the problem converges to a local minima. Thus, if starting close enough to the solution, we can guarantee that the condition number will be preserved as low as possible, while converging to the solution. It is worth mentioning that the initial position of the SSDMPC is chosen as the appropriate (that is, same system and control horizon) solution of the ERPC algorithm, rather then some arbitrary solution. This increases the chances of finding a (preferably larger) local solution, if it exists.

The Choice penalty parameter

The penalty parameter is very important for the behavior of the algorithm. If it is left to increase dynamically, as explained in section 4.4.2, it converge fast, but at the expense of the size of ROA. If μ_k is increased at a monotonic rate, the size is indeed larger, but with many iterations. Ill-conditioning can occur if too many iterations are made, and for large systems the SDP (4.55) can even be unbounded, failing to converge. As an example with a sixth states with two control inputs (Helicopter model), for a constant increase rate, a solution was found after 600 iterations, compared to 72 iterations using augmented μ_k. It is recommended to start with a dynamic increase of μ_k, and then to let it increase at a constant rate if only a small number of steps are required for it to converge.

Robust MPC

The *Robust Feasibility* term was introduced in section 1.3.1: designing the predictive controller so that it maintains feasibility despite the present of uncertainty. Examples two and three in this chapter were linearized around small angles. Operating around large angles can result in large model mismatch, and using this model can give bad performance, or even instability. Both the ERPC and the SSDMPC presented a strategy to deal with such uncertainty. This is done by adding further LMIs to the problem, increasing the complexity. The nature of the algorithms does not change, but it was observed that the regions of attraction were somewhat smaller. This is expected since additional LMIs make the solution more conservative.

Does SSDMPC worth the effort?

The question remains as to whether it is worthwhile using the non-convex SSDMPC problem, considering it finds only local solutions, and is strongly dependent on the initial point and the penalty μ_k. The results from table 4.6 indicate that the regions of attraction are both larger and usually better conditioned than the equivalent ERPC solutions. Regarding the computation time for low N_c values, the ERPC has definite superiority here. However, increasing the control horizon causes the ERPC computation time to increase exponentially. As the table indicates, for large horizons, the ERPC loses its advantage to the SSDMPC algorithm (see figure 4.13 for comparison). Number of iterations and total computation time for SSDMPC can be decreased by a proper μ_k update, on the expanse of the size of ROA.

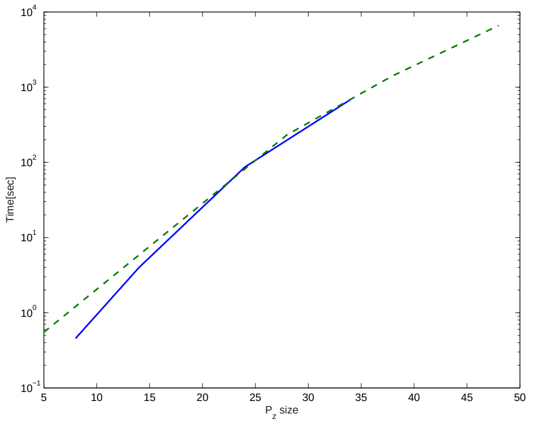


Fig. 4.13: ERPC offline computation time for Example 2 (green dashed) and Example 3 (blue solid).

Again it is emphasized that the SSDMPC algorithm is not convex and thus finds only a *local* solution, i.e. the solution M^* is not global. This means that Q_z^*, which is found upon termination, is global only to the specific M^* matrix and not the global Q_z^* exists. Simulations (see Examples 1-4 in section 4.8) show that it creates larger ellipsoids than the ERPC solution. It creates also larger ellipsoids than the $Q_z(k)$ at the final SSDMPC iteration k, i.e. before solving ERPC with full matrix M^*. This comes from the fact that ERPC gives a global solution to a given M while SSDMPC gives only a local one (for the same M).

Computation time is hardly an issue in the offline algorithm. It is worth mentioning that for small N_c, the ERPC is very fast. However, ERPC computation time grows exponentially with N_c (also see table 4.6).

Chapter 5

Online Controller

Recall that classical MPC algorithms minimize online a quadratic cost function by solving a QP problem. This consumes time and resources and usually cannot be done for system with high sampling-rate. Another approach, less time-consuming will be presented in this chapter. The offline solution P_z^* is used to solve an efficient Newton Raphson, an alternative the QP-MPC. The online guarantees the feasibility and invariance of the system.

Section 5.1 presents the theory behind the online approach. Stability is proven in section 5.4, followed by a short description of the Newton Raphson algorithm and feasibility issues. The performance of NR online controller is demonstrated by a few numerical examples in section 5.6.

5.1 Online Strategy

Given a system model, control law and the system constraints

$$\mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k \tag{5.1}$$
$$\mathbf{u}_k = -K\mathbf{x}_k + \mathbf{c}_k \tag{5.2}$$
$$|u_i| \leq \bar{u}_i, \qquad i = 1, ..., m_u \tag{5.3}$$
$$|x_i| \leq \bar{x}_i, \qquad i = 1, ..., m_x \tag{5.4}$$

where the future control perturbations vector $\mathbf{c}_k$ replaces the classical LQ control law $\mathbf{u} = -K\mathbf{x}$. For $\mathbf{c}_k = 0$ the closed loop system is optimal in the unconstrained LQ sense, but in order to prevent violation of the constraints, feasibility is achieved by using proper $\mathbf{c}_k$ value. Otherwise, it should be maintained as small as possible.

This reduces the online optimization to minimize a performance index, J_f subject to ellipsoid

$$\min_f J_f \quad \text{subjected to } \mathbf{z}^T P_z \mathbf{z} \leq 1 \tag{5.5}$$

where the P_z matrix was pre-computed by one of the offline algorithms ERPC or SS-DMPC.

For the case of the ERPC, the cost function J_f penalize the future control perturbations by

$$J_f = \sum_{i=0}^{N_c-1} c_i^T W c_i \tag{5.6}$$

where W is found from the equation

$$\begin{aligned} W &= B^T P_{LYAP} B + R &(5.7)\\ P_{LYAP} &= Q + K^T R K + \Phi^T P_{LYAP} \Phi &(5.8)\end{aligned}$$

P_{LYAP} is the solution of Lyapunov equation. Kouvaritakis et al. (2002) shows that J_{LQ} and J_f differ by a constant term and thus minimizing the two cost functions is equivalent. The online minimization is reduced then to norm minimization, subject to a quadratic constraint, which can be solved very efficiently. When minimizing the cost function (5.6) online, the special structure of M given by (4.1) ensures that all the perturbation values $c_{k+N_c+i} = 0$ for all $i \geq 0$.

When using the offline solution given by SSDMPC, the M matrix no longer has the form (4.1), but extra degrees of freedom are achieved by using full unstructured matrix (4.3).

The infinite sum can be expressed as a quadratic function (Drageset et al.; 2003)

$$J_M = \sum_{i=0}^{\infty} c_{k+i}^T W c_{k+i} = f_k^T \Gamma f_k \tag{5.9}$$

where Γ is a positive definite matrix given by the Lyapunov equation

$$M^T \Gamma M - \Gamma = -D^T W D \tag{5.10}$$

Then J_M *replaces* the cost function J_f of the minimization problem (5.5).

5.2 Newton Raphson Algorithm

A comprehensive description of the Newton Raphson is given by Drageset (2002), Kouvaritakis et al. (2000), Kouvaritakis et al. (2002) so only the main topics are repeated here.

Lagrange multiplier can be used in order to minimize 5.5. It can be written as

$$\begin{aligned} L &= \mathbf{f}^T\mathbf{f} + \lambda\left(\mathbf{z}_k^T P_z \mathbf{z}_k - 1\right) &(5.11)\\ &= \mathbf{f}^T\mathbf{f} + \lambda\left(\mathbf{x}_k^T P_{11}\mathbf{x}_k + 2\mathbf{f}^T P_{12}\mathbf{x}_k + \mathbf{f}^T P_{22}\mathbf{f} - 1\right) &(5.12)\end{aligned}$$

Differentiating 5.11 with respect to $\mathbf{f}$, and equating to zero, we get

$$\mathbf{f} = G(\lambda)\mathbf{x}_k \quad \text{where } G(\lambda) = -\lambda(I_{\mathbf{f}} + \lambda P_{22})^{-1} P_{21} \tag{5.13}$$

where $I_{\mathbf{f}}$ is the identity matrix size $m_u N_c \times m_u N_c$. Denoting $F(\lambda) = \nabla_\lambda L = 0$

$$F(\lambda) = \mathbf{x}_k^T \left[P_{11} + 2P_{12}G(\lambda) + G(\lambda)^T P_{22} G(\lambda)\right] \mathbf{x}_k - 1 = 0 \tag{5.14}$$

Newton Raphson algorithm solves 5.14 to find roots of $F(\lambda)$ iteratively, and full description can be found in any numerical analysis book, e.g. Kincad and Cheney (2002). The roots λ is then

$$\lambda_{n+1} = \lambda_n - \frac{F(\lambda_n)}{F'(\lambda_n)} \tag{5.15a}$$

$$F'(\lambda_n) = \mathbf{x}^T \left[2P_{12}G'(\lambda_n) + G'(\lambda_n)^T P_{22} G(\lambda_n) + G(\lambda_n)^T P_{22} G'(\lambda_n)\right] \mathbf{x} \tag{5.15b}$$

$$G'(\lambda_n) = (I_\mathbf{f} + \lambda P_{22})^{-1}[\lambda P_{22}(I_\mathbf{f} + \lambda P_{22})^{-1} - I_\mathbf{f}]P_{21} \tag{5.15c}$$

Equation 5.15 is solved iteratively.

5.3 Feasibility verification

If the state $\mathbf{x}_k$ is inside the region defined by E_{zx}, then it is feasible and NR will find a solution to (5.5) at the iteration k. If it is not inside E_{zx} then we don't have any guarantee that it is feasible. Solution to (5.5) might be found, but it is not guaranteed. If $\mathbf{x}_k$ is not feasible, then $F'(\lambda_n)$ is getting arbitrary small, making (5.15a) impossible to solve due to division by number close to zero. Reduction of $F'(\lambda_n)$ can take several iteration, up to few dozens. If P_z is a large matrix, number of operations that has to be computed before error occurs, is large. This time consuming process can be avoided if one set a lower bound on F', say $F'(\lambda_n) > \epsilon$, and when this bound is reaches, one aborts the NR algorithm at this iteration, with $\mathbf{c}_k = 0$.

An alternative approach is suggested here, by using a simple feasibility verification. Given $\mathbf{x}$ and P_z. It is possible to check whether the constraint (5.5) is feasible, i.e. there exist f that satisfying

$$z_k^T P_z z_k \leq 1 \tag{5.16}$$

This is equivalent to finding f which satisfies

$$\mathbf{x}_k^T P_{11}\mathbf{x}_k + 2\mathbf{f}_k^T P_{21}\mathbf{x}_k + \mathbf{f}_k^2 P_{22}\mathbf{f}_k \leq 1 \tag{5.17}$$

The last equation can be written as a LMI using the Schur complement (2.7):

$$\begin{pmatrix} 1 - 2\mathbf{f}_k^T P_{21}\mathbf{x}_k - \mathbf{x}_k^T P_{11}\mathbf{x}_k & \mathbf{f}^T \\ \mathbf{f} & P_{22}^{-1} \end{pmatrix} \geq 0 \tag{5.18}$$

The feasibility of the LMI can be check using the MATLAB LMI toolbox, for instance, *feasb.m*. The feasibility algorithm can be accelerated by using the '*option*' properties, described by Gahinet et al. (1995).

5.4 Stability

Consider again the state space presentation

$$\begin{aligned} &\mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k \\ &|u_i| \leq \bar{u}_i, \qquad i = 1, ..., m_u \\ &|x_i| \leq \bar{x}_i, \qquad i = 1, ..., m_x \end{aligned} \tag{5.19}$$

Applying a state feedback to the system will move the eigenvalues inside the unit circle (assuming the system is controllable). This is not enough to guarantee stability of (5.19) since the system is non linear when the constraints are active. This section will prove the stability using Lyapunov stability theory inside the region $\Omega = \{\mathbf{x} \in \mathbb{R}^{m_x} | \ \mathcal{E}_{zx}\}$. This region is shown in figure 5.1 as the outer ellipsoid. Before moving to the proof define the LQ ellipsoid $\mathcal{E}_x = \{\mathbf{x}_k \in \mathbb{R}^{m_x} | \ \mathbf{x}_k^T P_x \mathbf{x}_k < 1\}$ in the figure is an invariant ellipsoid, by the LQ definition.

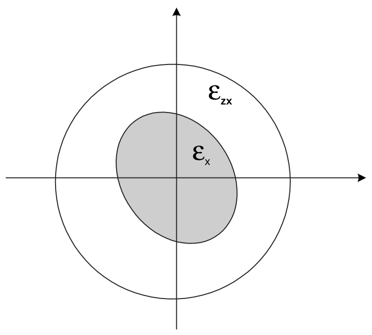


Fig. 5.1: The regions $\mathcal{E}_{zx}$ and $\mathcal{E}_x$

Theorem 5.1 (Closed loop stability)
If for the system (5.19) there exist K, P_z, N_c and M such that $\mathbf{x}_k \in \mathcal{E}_{zx}$, then the closed loop application of SSDMPC is feasible and robustly asymptotically stabilizing.

Proof. *Consider the case where $\mathbf{f}_k = 0$. This occur when the initial condition is inside the ellipsoid $\mathcal{E}_x$, as shown in figure 5.1. Let $V(\mathbf{x}_k) = \mathbf{x}_k^T P_x \mathbf{x}_k$ where P_x defines the LQ ellipsoid $\mathcal{E}_x$. The function $V(\mathbf{x}_k)$ is positive definite since P_x is positive definite. Moreover, if $\mathbf{x}(k) = 0$ then $V(0) = 0$. Next, the forward difference of $V(\mathbf{x}_k)$ is*

$$\begin{aligned}
\Delta V(\mathbf{x}) &= V(\mathbf{x}(k+1)) - V(\mathbf{x}(k)) \\
&= \mathbf{x}^T(k+1) P_x \mathbf{x}(k+1) - \mathbf{x}^T(k) P_x \mathbf{x}(k) \\
&= (\Phi \cdot \mathbf{x}(k))^T P_x (\Phi \cdot \mathbf{x}(k)) - \mathbf{x}^T(k) P_x \mathbf{x}(k) \\
&= x^T(k) \Phi^T P_x \Phi \mathbf{x}(k) - x^T(k) P_x \mathbf{x}(k) \\
&= \mathbf{x}^T(k) \left(\Phi^T P_x \Phi - P_x \right) \mathbf{x}(k)
\end{aligned}$$

which is negative definite if

$$\Phi^T P_x \Phi - P_x < 0 \tag{5.20}$$

This inequality holds by assumption on K.

Now consider the case where the initial condition lays inside the ellipsoid $\mathcal{E}_{zx}$ but outside or in the boundary of $\mathcal{E}_x$. This is shown in figure 5.1 as the area between $\mathcal{E}_{zx}$ and $\mathcal{E}_x$. In that case, the perturbation vector $\mathbf{f}_k = \left(\mathbf{c}_k^T, \ \mathbf{c}_{k+1}^T, \ \ldots, \ \mathbf{c}_{k+N_c-1}^T \right)^T \neq \mathbf{0}$

Note that $\tilde{\mathbf{f}}_{k+1} = M\mathbf{f}_k$ is feasible. Since $M - I \leq 0$, we can conclude that

$$\|\tilde{\mathbf{f}}_{k+1}\|_\Gamma^2 = \|M\mathbf{f}_k\|_\Gamma^2 \leq \|\mathbf{f}_k\|_\Gamma^2 \tag{5.21}$$

From the cost function (5.9) it is easily seen that $\|\tilde{\mathbf{f}}_{k+1}\|_\Gamma^2 = \|\mathbf{f}_k\|_\Gamma^2 - \mathbf{c}_k^T W \mathbf{c}_k$*. The minimization of* $\|\mathbf{f}_{k+1}\|_\Gamma$ *assures that it will be equal to or lower than the former value, i.e*

$$\|\mathbf{f}_{k+1}\|_\Gamma^2 \leq \|\mathbf{f}_k\|_\Gamma^2 - \mathbf{c}_k^T W \mathbf{c}_k \tag{5.22}$$

where W *is positive definite. This means that* $\|\mathbf{f}_{k+1}\|_\Gamma$ *is monotonically decreasing. Since every infinite series which is monotonically decreasing and bounded from below, converge to some value, we conclude that* $\|\mathbf{f}_k\|_\Gamma^2$ *converge to some positive value. By doing so,*

$$\|\mathbf{f}_{k+1}\|_\Gamma^2 - \|\mathbf{f}_k\|_\Gamma^2 \to 0 \quad when \quad i \to \infty \tag{5.23}$$

$$\Rightarrow \quad \mathbf{c}_k^T W \mathbf{c}_k \leq 0 \tag{5.24}$$

which means that $\mathbf{c}_k$ *converge to zero, since* W *is positive definite. When that happens, the control low is* $\mathbf{u} = K\mathbf{x}$ *and the trajectories are inside the ellipsoid* $\mathcal{E}_x$*.*

Hence the system is asymptotically stable with a region of attraction $\mathcal{E}_{zx}$ ■

An alternative way is to use the Lyapunov function $V(\mathbf{z})$ and show that $\Delta V(\mathbf{z}) < 0$.

5.5 Offline-Online control strategy

After an extensive discussion about the offline solution, followed by the online theory, the algorithm which combines the two parts can be given.

Algorithm 5.1 (Offline-Online control strategy)

Step 1 (*Offline*): *For a given linear system (3.1) or a polytope (3.37), from which LDI is determined, computed the unconstrained LQ-optimal gain* K*. Choose* P_{zx} *to maximize the ellipsoid* $\mathcal{E}_{zx} = \{\mathbf{x} | \ \mathbf{x}^T P_{zx} \mathbf{x}\}$*, subjected to one of the ERPC or SSDMPC LMI sets. This is done by solving ERPC or SSDMPC offline algorithms, and find a solution* $P_z = Q_z^{-1}$*.*

Step 2 (*Online*): *Use* P_z *from step 1 to solve the optimization problem 5.5, using NR, and find the perturbation vector* $\mathbf{c}_k$*. Calculate the input* $\mathbf{u}_k = -K\mathbf{x}_k + \mathbf{c}_k$

step 3 (*Online*): *Implement* $\mathbf{u}_k$ *and return to step 2 at time step* $k+1$*.*

5.6 Examples

This section presents some numerical examples for the online NR controller, using different offline solutions P_z, found in chapter 4. Description of the models for each example were discussed in the offline context, so they will not be repeated here.

5.6.1 Example 1: Passenger Aircraft Longitudinal motion control

Consider the same system in example 2 section 4.8.2. The linearized state space equations are given below:

$$\begin{aligned}\begin{bmatrix}\alpha_{k+1}\\ q_{k+1}\\ \theta_{k+1}\end{bmatrix} &= \begin{bmatrix}0.9968 & 0.05649 & 0\\ -0.0001 & 0.9957 & 0\\ 0 & 0.5658 & 1\end{bmatrix}\begin{bmatrix}\alpha_k\\ q_k\\ \theta_k\end{bmatrix} + \begin{bmatrix}0.0024\\ 0.0002\\ 0.0001\end{bmatrix}\delta_e(k)\\ &= A\mathbf{x}_k + B\delta_e(k) \qquad (5.25)\\ y_k &= (0 \;\; 0 \;\; 1)\mathbf{x}_k = C\mathbf{x}_k\end{aligned}$$

with samplings-period of $T_s = 0.01sec$. This system is marginally stable due to a pole on the unit circle. Feedback control is applied to (5.25) using the gain found by Linear Quadratic control

$$\mathbf{u}_k = -K\mathbf{x}_k + \mathbf{c}_k \qquad (5.26)$$

$$-60^0 \leq \mathbf{u}_k \leq 60^0 \qquad (5.27)$$

the constraints on the input are hard constrains, due to a physical limitation of the elevator angle. Recall from (Example 2) section 4.8.2 that few ellipsoids were found, there the largest was the ellipsoid using P_z solution from SSDMPC offline algorithm, with the size of 5×5. For comparison, the largest ERPC ellipsoid was calculated using P_z with the size of 38. It will be shown here that this large difference will have strong impact on the performances of the online simulation.

Two different feasible initial points will be examined. The first feasible initial point

$$x_{01} = [-1, \;\; -0.007, \;\; -1]^T$$

which lies inside the ellipsoid obtained using SSDMPC $N_c = 2$ and the ERPC ellipsoid for $N_c = 35$. Hence it is feasible for both solutions. The minimization problem (5.5) is solved in order to find the perturbation c_k at each iteration k, and to apply it to the feedback control law (5.26). This allows us to maintain feasibility and prevent undesired saturation of u. Table 5.1 summarizes the parameters.

The results are presented in figures 5.2 and 5.3.

The bottom left of figure 5.2 shows the control input for the first 2 seconds, with three different control feedback: LQ control without constraints (blue dashed), ERPC solution for $N_c = 35$ and SSDMPC $N_c = 2$. The 'pure' LQ controller, i.e. is when $c_k = 0 \; \forall k$, has saturation of the input for the first 30 iterations (i.e. 0.3 seconds).

Table 5.1: The two solution that are used in the online algorithm

Algorithm	N_c	**size** P_z	**Initial pos.**	x_0 **Feasb.**	**Constr. violation**	**Max. time** [sec]
None	0	3×3	$\mathbf{x}_{01}$	-	Yes	-
SSDMPC	2	5×5	$\mathbf{x}_{01}$	Yes	No	0.01
ERPC	35	38×38	$\mathbf{x}_{01}$	Yes	No	0.09
SSDMPC	2	5×5	$\mathbf{x}_{02}$	Yes	No	0.01
ERPC	35	38×38	$\mathbf{x}_{02}$	No	Yes	-

When solving the online minimization problem (5.5) using the ERPC solution with $N_c = 35$ or the SSDMPC, and using c_k in the feedback, we avoid this saturation, though the input feedback of SSDMPC (red) is slightly lower than the ERPC (solid blue). The calculation time of the perturbation c_k is given in the bottom left corner. For the ERPC case, since the matrix P_z is large (38×38), the calculation time is considerably higher than the sampling-period $T_s = 0.01$. The first two calculations of c_k takes almost 0.1 seconds, ten times more than the sampling-time. It is then reduced to $0.05 - 0.04$ seconds during the rest time that the states are out of the ellipsoid $\mathcal{E}_x$. When calculating c_k using the SSDMPC P_z solution, the time exceeds 0.01 only once. The trajectory of is shown in figure 5.3 (blue dashed).

The second initial point

$$\mathbf{x}_{02} = \frac{\pi}{180}\left[-10\ , 2\ , 30\ \right]^T$$

lies outside any of the ERPC ellipsoids that had been calculated. It may possible to

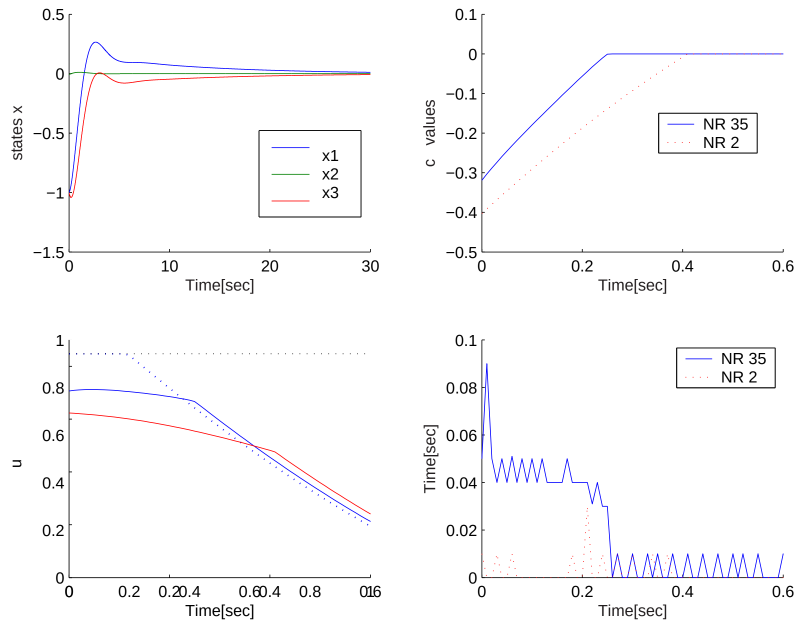


Fig. 5.2: Initial point $\mathbf{x}_{01}$. Top left: The three states. Bottom left: The input u. Top right: the c_k perturbation value. Bottom right: The calculation time of c_k at time point t.

find a suitable ERPC solution which will enclose $\mathbf{x}_{01}$ inside, but if this ellipsoid exists, then the control horizon N_c has to be so large that it will not be effective for systems with short sampling-period, e.g. less than 0.1 sec. While the input using LQ controller with $c_k = 0$ causes saturation of the elevator during the first 1.5 seconds, the input $u_k = -K\mathbf{x} + c_k$ where c_k is calculated using the SSDMPC solution prevents saturation of the elevator during the whole simulation time. The c_k calculation time using Newton Raphson algorithm never exceeds the sampling-time T_s, it is actually much lower than that. The state trajectory is shown in figure 5.3 (red solid).

The solution given by the ERPC algorithm with horizon $N_c = 30$ is not useful, as the ellipsoid becomes a thin disk, there it covers only $\pm 2.3deg$ of the pitch angle. If the plane is to be stabilized from a starting pitch point of more than $3deg$, the ellipsoid will not be useful for calculation of the perturbation vector c_k, and saturation will occur.

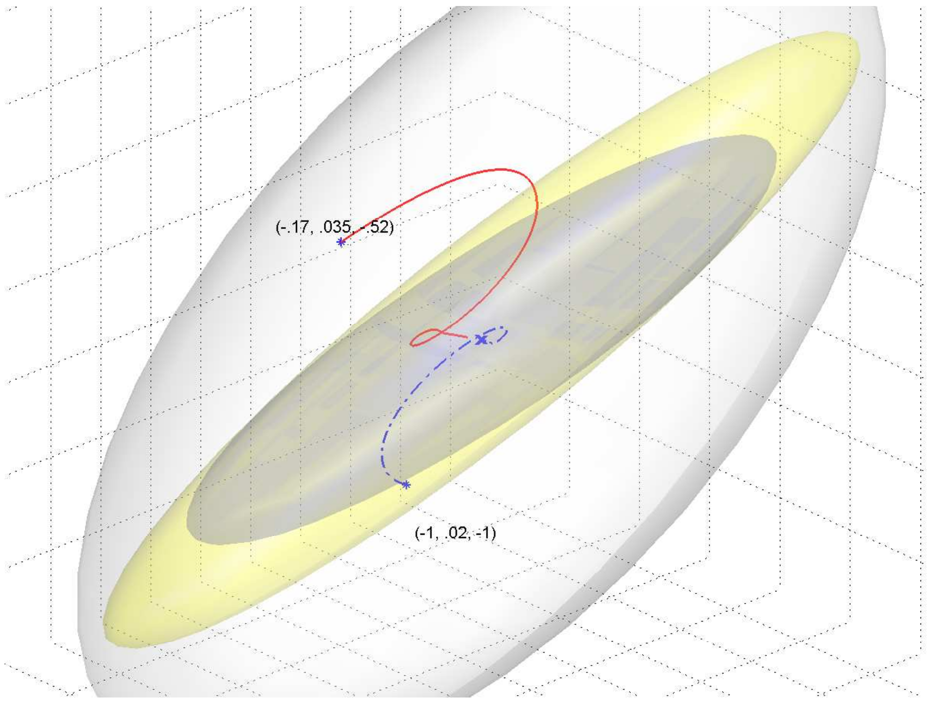


Fig. 5.3: The 3 dimensional trajectories from start point $\mathbf{x}_{01}$ and $\mathbf{x}_{02}$ to $\mathbf{0}$. The inner blue ellipsoid is the ERPC $N_c = 2$, the yellow is ERPC $N_c = 35$ and the outer grey is the SSDMPC $N_c = 2$

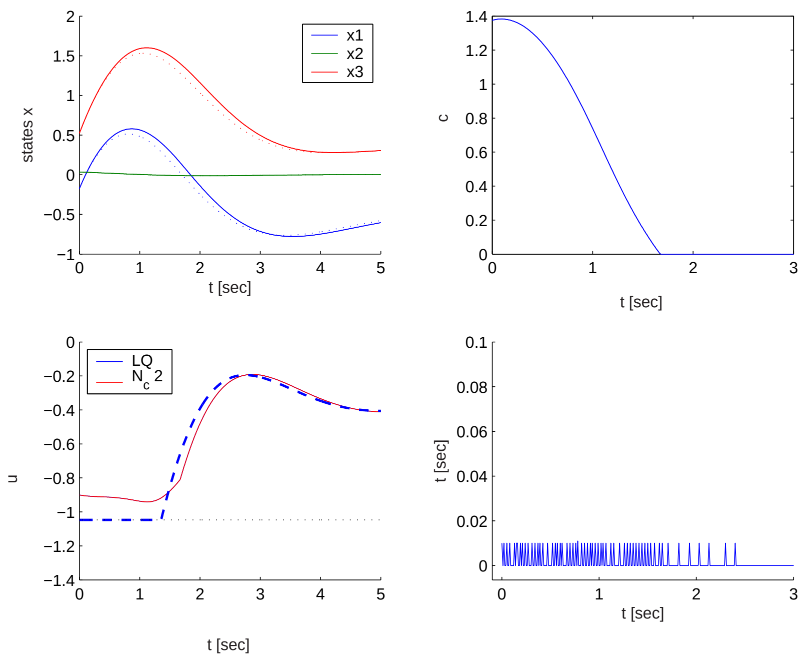


Fig. 5.4: Initial point $\mathbf{x}_{02}$. Top left: The three states. Bottom left: The input u. Top right: the c_k perturbation value. Bottom right: The calculation time of c_k at time point t.

5.6.2 Example 2: B-27 aircraft model

Consider the same system described in the offline part, section 4.8.3 on page 53. This example will simulate the same model, using a linear quadratic feedback control gain matrix K, that is $\mathbf{u}_k = -K\mathbf{x}_k + \mathbf{c}_k$ where $\mathbf{c}_k$ is the perturbation vector computed by the NR algorithm at each iteration. The system has 'hard' input constraints $\pm 1.04rad$ on both rudder and ailerons deflection. The controller will try to stabilize the plane from initial position given by

$$\mathbf{x}(0) = (0.52,\ 0.6,\ 0.2,\ 0.4)^T$$

Recall that the SSDMPC ellipsoid was much larger than the ellipsoid found by the ERPC algorithm, for the same N_c. It was possible however, to increase the horizon to $N_c = 15$ in order to get an ellipsoid which will cover an acceptable volume around the origin. Increasing the horizon affects the size of the matrix P_z and thus affects the performances of the online part, as will be demonstrated here. The longest horizon that was tested was using $N_c = 15$. This is due to the fact that the size of the solution P_z is growing with rate $N_c \cdot m_u$, and matrices which are larger than 34×34 will probably not be suitable for real-time application anyway. Table 5.2 summarize the offline results including the sizes of the ellipsoids.

Table 5.2: Different solutions used online.

Algorithm	N_c	**size** P_z	$\mathbf{x}_0$ **Feas.**	**Constr. violation**	**Max. time [sec]**
None	0	4×4	-	yes	0
SSDMPC	2	8×8	yes	no	0.01
ERPC	10	24×24	no	yes	0.2
ERPC	15	34×34	no*	yes	0.2

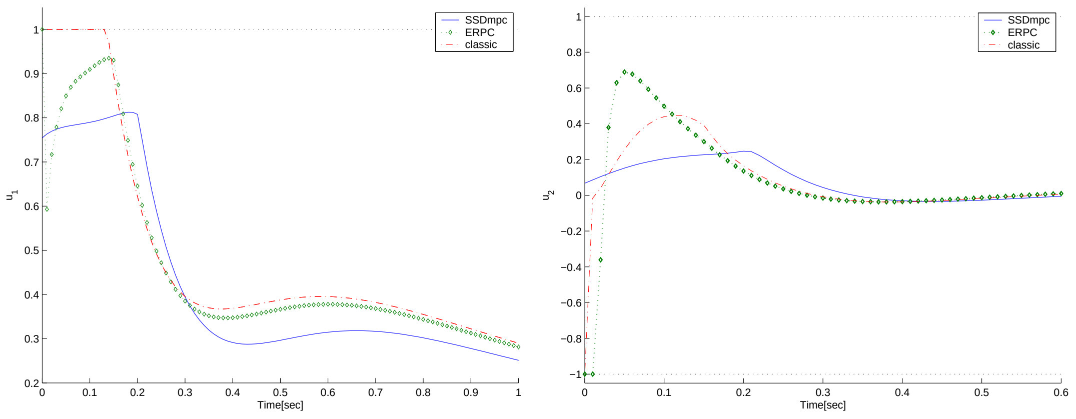
The purpose of this example is to demonstrate the differences in performance of the online algorithm between the solution achieved by the ERPC, SSDMPC and the classical control.

Figure 5.5 shows the control inputs u_1 (left) and u_2 (right) resulted in online simulations. Using The solution P_z found by the SSDMPC algorithm (solid blue), the input control never reaches saturation, as a suitable perturbation vector $\mathbf{c}_k$ is computed at each iteration. The $\mathbf{c}_k$ calculation time never exceeds the sampling-time limit $0.01sec$ (see figure 5.6 right).

ERPC solution for $N_c = 15$ will be the only one considered (of all the ERPC's) since the initial position lies in its vicinity. It is not feasible at the first iteration, but gets feasible right after, for $k \geq 2$, allowing the NR algorithm to compute a suitable $\mathbf{c}_k$ vector. The Newton-Raphson calculation time is very high with a peak of $0.2sec$. This indicates once more that the online Newton Raphson algorithm results in high calculation time for large P_z matrices.

Using classic feedback control $\mathbf{u} = -K\mathbf{x}$, the input u_1 is saturated for the first 0.18 seconds. This causes the system to behave unexpectedly, as figure 5.7 indicates. For the

*Infeasibility occurs only in the first iteration classic control case, the state x_4 has an inverse response. Physically, this means that the yaw rate first gets some negative values, follows by high acceleration, and settles at zero. However, for $\mathbf{u} = -\mathbf{K}\mathbf{x} + \mathbf{c}$ where $\mathbf{c}_k$ is found using the SSDMPC solution, the x_4 response is normal, with low overshoot.



(a) u_1 (b) u_2

Fig. 5.5: Input controls for B-27 aircraft. Using the SSDMPC solution (blue solid), ERPC solution $(-\diamond-)$ and using classic control (red $-.-$).

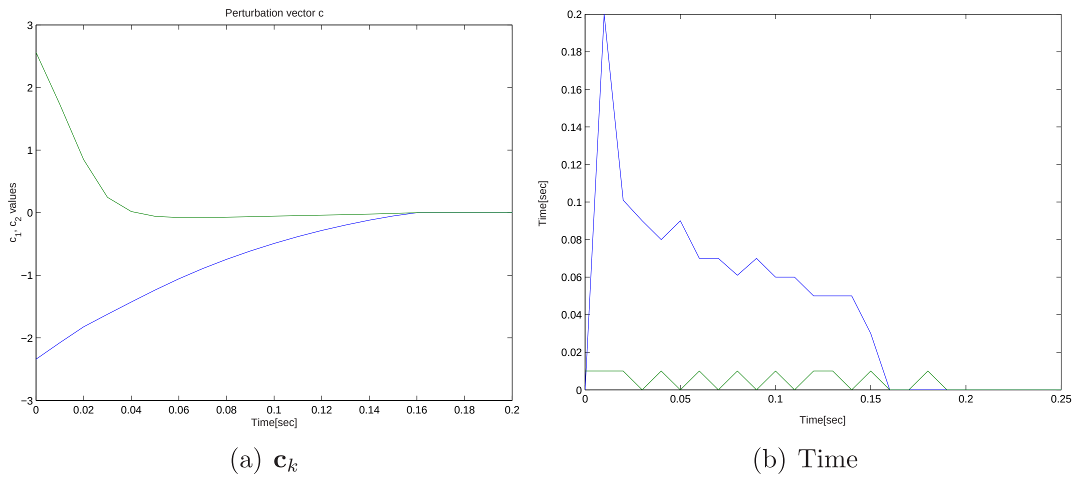


Fig. 5.6: Online simulation, B-27 aircraft. c_1 and c_2 values are presented for the SSDMPC only. NR calculation time is given the right side. SSDMPC time(blue solid) and ERPC time (red $-.-$).

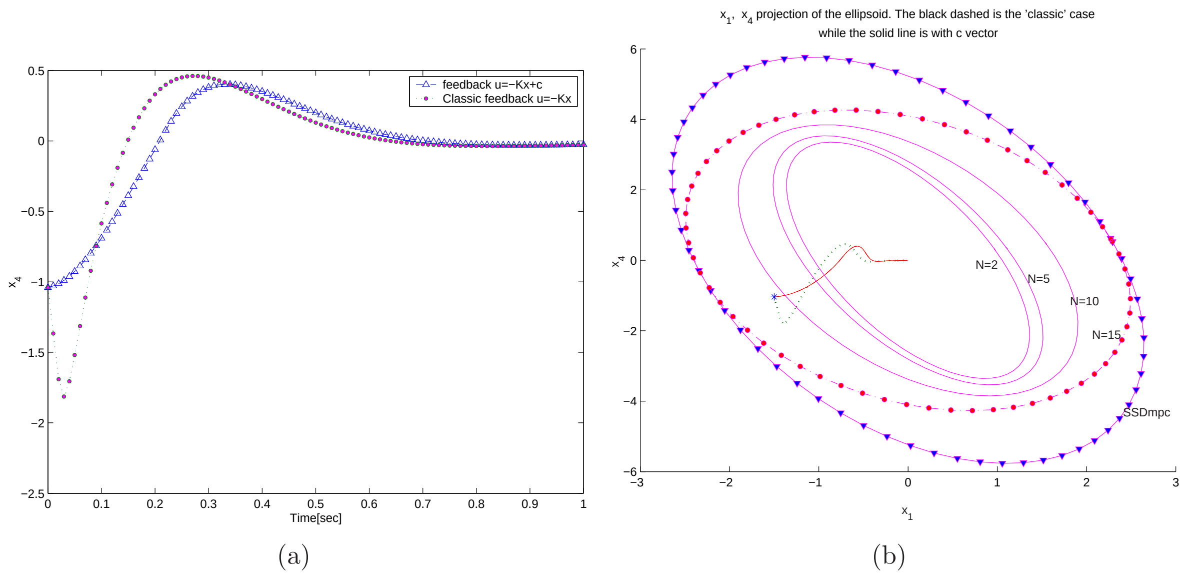


Fig. 5.7: Online simulation, B-27 aircraft. (a) The time response of x_4 and (b) The trajectory x_1 - x_4.

5.7 Online Summary and Discussion

5.7.1 Simulations

The results from simulations in the Online chapter are summarized here. Two of the examples from chapter 4 are summarized in table 5.3 below, and in addition the online simulation results of the Helicopter model is considered. Chapter 6 discussed extensively about the mathematical model of the system, the offline and online results from both simulations and real-time experiments of the helicopter.

5.7.2 Online Computation Time

The experiments indicate that NR computation time for low N_c solutions does not exceed the sampling-rates. It is much lower in practice but since all the simulations were performed using a fixed step-size, the lowest computation time presented is these step-sizes. The results presented in table 5.3 were found using a $900MHZ$ CPU, $256MB$ memory, Windows NT operation system, with no other visible applications running. Lower computation times for large N_c values can be achieved with better CPU and no background applications executed by the operation system.

As was described in the theory section 1.2.4, in order to ensure the stability of the system when using QP MPC, the control horizon N_c should be large enough. In robust model predictive control algorithms (e.g. ERPC, SSDMPC) the control horizon is chosen so the states will be feasible, i.e. inside a given ellipsoid (invariant set). If a start position is not inside the region of attraction, then it is possible to increase the control horizon N_c until $\mathbf{x}_0 \in \mathcal{E}_{zx}$. This may impose problems since the computational

Table 5.3: Simulations results, online

Example	**Algorithm**	N_c	**size** P_z	$\mathbf{x}_0$ **Feasb.**	**Constr. violation**	**Max. time [sec]**	**Sampling time**
1. $\mathbf{x}_{01}$	None	0	3×3	-	Yes	-	0.01
	SSDMPC	2	5×5	Yes	No	0.01	
	ERPC	35	38×38	Yes	No	0.09	
1. $\mathbf{x}_{02}$	SSDMPC	2	5×5	Yes	No	0.01	
	ERPC	35	38×38	No	Yes	-	
2.	None	0	4×4	-	Yes	-	0.01
	SSDMPC	2	8×8	Yes	No	0.01	
	ERPC	10	24×24	no	yes	0.2	
	ERPC	15	34×34	No	yes	0.2	
3. Helicopter	None	0	6×6	-	Yes	-	0.001
	SSDMPC	2	10×10	Yes	No	0.001	
	ERPC	13	32×32	Yes	No	0.04	

time for solving the Newton Raphson algorithm depends on the size of P_z. Kouvaritakis et al. (2002) state that the computational burden of solving the NR increases linearly with N_c. It is especially important for large systems with many control inputs, since P_z increases with rate of $N_c \cdot m_u$. P_z will also be large for systems with many states. For real time applications, it is desired therefore to keep N_c as small as possible. The SSDMPC solution P_z allows expansion of the ellipsoid without necessarily increasing N_c. Simulation results and real-time experiments shows that the NR computational time for large P_z matrices is not suitable for high dynamic real-time applications. For matrix size 38×38 the NR found a solution f after 0.09 seconds. If the sampling time is lower than that, the real time application will not be able to update the system in time, resulting in online-failure.

5.7.3 Hard Constrained are avoided

Section 1.2.3 discusses the problem of soft and hard constraints. In QP MPC, it is possible to soften certain constraints, allowing them to be violated occasionally. This can not always be done, especially for input constraints which represent usually physical limitations. The predictive controller can decide that it does not have enough control authority to keep the plant within the constraints. A standard QP problem which faces infeasibility simply stop functioning. It is unacceptable behavior for online controllers, and different strategies are given in the literature (see section 1.2.3). It was shown by the examples in this chapter that saturation of the control inputs can be avoided if using the online strategy presented in this chapter. Moreover the problem is always feasible once the states are inside the region of attraction, found by the offline strategy.

5.7.4 States behavior

Example two gives a good demonstration how the time response of the system can be improved if using the online strategy presented in this chapter. By avoiding saturation of the input control, the states performance was excellent. This is not surprising since the plant in Example 2 is unstable and saturation of the controller usually leads the states away from the equilibrium point.

Chapter 6

Real-Time Experiments

6.1 Introduction

Nominal stability is hardly an issue when designing and simulating a model predictive controller. It is easy to obtain nominal stability and feasibility by tuning of parameters in the problem formulation. Unfortunately, the situation is very different when the system is subjected to disturbances. This chapter explores the behavior of a system with tight dynamic specifications in a real-time environment. Both the offline and the online algorithms will be tested and analyzed. A mathematical model of the system is developed, followed by a brief description of the WinCon 3.3 and RTX 5.5 real time environment. Solution for both the ERPC and the SSDMPC offline algorithms is computed and compared in section 6.4. The online section gives results from both theoretical simulations and real-time experiments. A discussion about the experiments concludes the chapter.

The objectives of this chapter are as follow

- Demonstrate the effectiveness of the SSDMPC solution online in a plant subjected to disturbances.
- Investigate the affect of deviation between the model and the real plant.
- Comparison between the SSDMPC and the ERPC solutions online.
- Investigate system performances using online controllers in real-tine environment.
- Investigate the behavior of the system at the boundary of the region of attraction.

We shall distinguish between two different terms in this chapter: The first one is the 'model' of the helicopter, that is, the mathematic representation (in state space) of the actual model. The term 'Helicopter' will be referred to as the actual plant itself.

6.2 System Description

6.2.1 Mathematical description

Elevation Axis

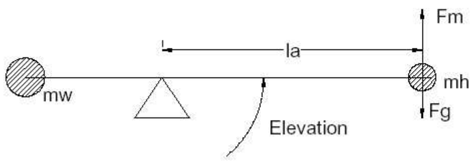


Fig. 6.1: Elevation free body diagram.

Consider the diagram in figure 6.1. Assuming the pitch is zero, then the elevation torque is controlled by the forces generated by the two propellers $F_f + F_b$. The differential equation for the forces are

$$J_e\ddot{e} = l_a F_m - l_a G_g = l_a(F_f + F_b) - l_a F_g \quad (6.1)$$

$$J_e\ddot{e} = l_a F_f + l_a F_b - l_a F_g \quad (6.2)$$

$$J_e\ddot{e} = K_f l_a(V_f + V_b) - T_g = K_f l_a V_s - T_g \quad (6.3)$$

where J_e is the moment of inertia of the system about the elevation axis, V_f and V_b are the voltages applied to the front and the rear motors resulting in forces F_f and F_b, K_f is the force constant of the motor/propeller combination, l_a is the distance from the pivot point to the helicopter body and T_g is the effective gravitational torque.

Pitch Axis

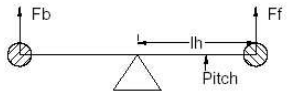


Fig. 6.2: Pitch free body diagram.

Consider the free body diagram in figure 6.2. The pitch axis is controlled by the difference of the forces generated by the propellers. if the force generated by the front motor is higher than the force generated by the back motor, the helicopter body will pitch up.

$$J_p\ddot{p} = F_f l_h - F_b l_h \quad (6.4)$$

$$J_p\ddot{p} = K_f l_h(V_f - V_b) = K_f l_h V_d \quad (6.5)$$

where J_p is the moment of inertia of the helicopter body about the pitch axis and i_h is the distance from the pitch axis to either motor.

Travel Axis

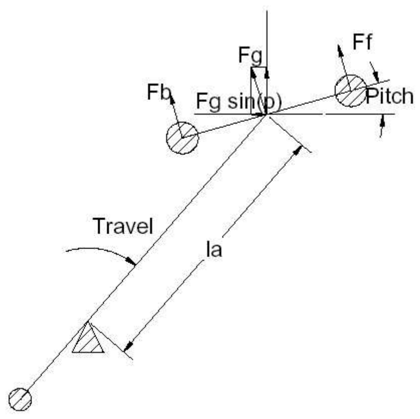


Fig. 6.3: Travel free body diagram.

The only way to apply a force in the travel direction is to pitch the body of the helicopter. Figure 6.3 presents the free body diagram of the travel direction. For small angles, the force requires to keep the body in the air is approximately Fg. The horizontal component of the Fg will cause a torque about the travel axis, which results in an acceleration about the travel axis

$$\begin{aligned} J_t\dot{r} &= -F_g\sin(p)l_a && (6.6)\\ J_t\dot{r} &= -K_p\sin(p)l_a && (6.7)\end{aligned}$$

where r is the travel rate in rad/sec and K_p is the force required to maintain the body in flight.

6.2.2 Nominal Model

The continuous nominal plant model is given by the next equation

$$\begin{pmatrix}\dot{\lambda}\\ \dot{r}\\ \dot{p}\\ \ddot{p}\\ \dot{e}\\ \ddot{e}\end{pmatrix} = \begin{pmatrix}0&1&0&0&0&0\\ 0&0&-K_2&0&0&0\\ 0&0&0&1&0&0\\ 0&0&-K_1K_{pp}&-K_1K_{pd}&0&0\\ 0&0&0&0&0&1\\ 0&0&0&0&-K_3K_{ed}&-K_3K_{ep}\end{pmatrix}\begin{pmatrix}\lambda\\ r\\ p\\ \dot{p}\\ e\\ \dot{e}\end{pmatrix} + \begin{pmatrix}0&0\\ 0&0\\ 0&0\\ K_1K_{pp}&0\\ 0&0\\ 0&K_3K_{ep}\end{pmatrix}\begin{pmatrix}p_c\\ e_c\end{pmatrix} \quad (6.8)$$

and the output is described by

$$y(k) = I_{6\times 6}\mathbf{x} = C\mathbf{x} \quad (6.9)$$

where $K_{ed} = -0.2$, $K_{ep} = -0.43$, $K_{pp} = -0.14$, $K_{pd} = -0.045$ and we assume that all the states are available for feedback. In practice, the only measurements are λ, p and e.

The rest of the states are derived using low pass filters (see Simulink diagram appendix B.1).

Next, the model is discretized using the first order Euler approximation method with sampling-time $T_s = 0.001sec$:

$$\begin{aligned} \mathbf{x}(k+1) &= (I + T_s A_c)\mathbf{x}(k) + (T_s B_c)\mathbf{u}(k) \\ &= A\mathbf{x}(k) + B\mathbf{u}(k) \\ \mathbf{y}(k) &= (T_s C)\mathbf{x}(k) \end{aligned} \tag{6.10}$$

6.2.3 Model Uncertainty

The plant of the helicopter is nonlinear, where travel rate r can be described by the differential equation

$$J_t \dot{r} = -F_g \sin(p) l_a \tag{6.11}$$

where p is the pitch angle, F_g is the force require to maintain the helicopter in flight (see figure 6.3). If the pitch operates at angles larger than ± 5, then $sin(p) \not\cong p$ and the linear model (6.8) is not accurate.

Consider a polytope

$$\Omega = \mathbf{Co}\{[A_1\ B_1],\ [A_2\ B_2]\} \tag{6.12}$$

where

$$A_{1,2} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -K_2 \pm \delta & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -K_1 K_{pp} & -K_1 K_{pd} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -K_3 K_{ed} & -K_3 K_{ep} \end{pmatrix} \tag{6.13}$$

and δ is chosen to model angle errors between 5^o to 9^o, that is $\delta = 0.07rad$. Non of the inputs directly affect the travel rate, so B has no uncertainties and $B = B_1 = B_2$.

6.3 The Real-time Workshop

Table 6.1 summarizes the hardware and software used in the real-time experiments.

6.3.1 Restriction on the sampling-rate

The time period it takes the controller to performs its computations is T_c. After the controller completes its calculations and I/O, the processor resumes executing the foreground tasks, such as Simulink, until an interrupt occurs. Tests indicate a latency time in the tens of microseconds under Windows NT/2000/XP, that is to say within the RTX real-time environment. This value depends on the foreground tasks that are running, their use of hardware resources, and the speed of the system. The computational delay,

Table 6.1: Helicopter experiments: Hardware and software used

Hardware:	CPU	$300MHZ$
	Memory	$256MB$
	I/O Board	Quanser MultiQ-3
Software:	Operation system:	Windows XP
	Real time environment:	RTX 5.5
	Real-time Server:	WinCon 3.3
	Matlab:	6.1
	Programming :	Visual C++ 6.0

T_c, depends mostly on the complexity of the real-time controller that is running. The sampling-rate can be chosen by the next equation

$$T_s = 20e - 6 + T_c[sec] \tag{6.14}$$

to be used as an indication of the fastest possible sampling period applicable for a given controller. The sampling period can be decreased for faster CPU. The lowest bound on the sampling period is fixed by the RTX server and can not be change in this experiment.

6.3.2 Restriction on the Real-Time Code

On all platforms, WinCon supports user-defined S-functions written in C. The real-time code is executed in a real-time kernel environment (essentially an operating system buried underneath Windows). Thus Windows operating system functions cannot be called since the S-function code is running in a different (hidden) operating system. There is also only a limited subset of the standard C library functions available under NT/2000/XP. Some functions only have a limited subset of their full functionality. The complete list of functions is available at the WinCon 3.3 manual. Standard Linear Algebra packages (*Lapack* for example) use memory allocations to perform tasks as inverse and matrix multiplications or transpose. Since memory allocation is not allowed in WinCon, these packages cannot be used.

While MATLAB scripts may be used to control WinCon, they cannot be used in real-time code. MATLAB is an interpretive language and cannot run in real-time. Furthermore, the MATLAB engine and numerical libraries have not been compiled for the real-time environment, thus it is not possible to call MATLAB from real-time code.

6.4 The 3-DOF Offline Solution

6.4.1 The Choice of Parameters

In order to get good performance of the online real-time experiments, a tuning of the unconstrained case was made, that is, $\mathbf{c}_k = \mathbf{0}$. The control feedback is then $\mathbf{u}_k = -K\mathbf{x}$ where K is found by optimizing the Linear Quadratic cost function $\sum_{i=k}^{k+N-1} \mathbf{x}_{i+1}^T Q \mathbf{x}_{i+1} +$

$\mathbf{u}_i^T R \mathbf{u}_i$. Experiments indicate that the helicopter is unstable for low R values, as the control input is too active, the pitch and the pitch velocity are too excited. The Q and R matrix were chosen such that the best performance (regardless constraints) achieved:

$$Q = diag\{1\ ,1\ ,1\ ,10,\ 1\ ,10\}, \qquad R = diag\{80,\ 80\}$$

The constraints chosen on control inputs p_c and e_c are low, as low as 20^o for both signals. It will be easy this way to demonstrate the behavior of the system at the boundary of the region of attraction and the states trajectories. These constraints are not physical constraints, as the p_c angle can be higher than $\pm 20^o$ (actually, the physical pitch angle limitation is $\pm 90^0$, but this limitation is rarely achieved while the helicopter is stable). No constraints were imposed on the states.

6.4.2 Solution Analysis

Nominal case

A few different offline solutions are presented below. The SSDMPC algorithm using control horizon $N_c = 2$ has found a solution after 72 iterations, with very promising results, where the ellipsoid is considerably larger than the one found by ERPC algorithm using $N_c = 2, 5, 8, 10$. ERPC solution using $N_c = 13$ might have an acceptable size of ROA, but still much smaller than the SSDMPC one. This can be verified both from the projections, figures (6.6-6.7) and from the volume factor. Table 6.2 summarizes the parameters and results for each solution.

The volume factor in table 6.2 is given for the six dimensional $P_{zx} \in \mathbb{R}^6$ and thus is a better comparison tool for the difference in ROA sizes than the graphical representations (recall that the projections can be misleading, as they represent only $3D$ of the real n-dimensional volume, see section 2.2.1). V_f for the SSDMPC is 96 times larger than ERPC solution with $N_c = 2$! Another thing to notice from the table is that increase of N_c for the ERPC causes a (nearly) linear increase of V_f (see figure 6.4). This is a major drawback of ERPC algorithm as increase in N_c enlarges the size of the matrix P_z by $2N_c \times 2N_c$, but does not enlarge the volume factor by a significant size.

The condition number of the SSDMPC solution is 17 times bigger than the ERPC solution for $N_c = 2$. It does not mean that the ellipsoid is thinner than the ERPC, it is

Table 6.2: Offline Helicopter solution properties

Solution	**Algorithm**	$\mathbf{N_c}$	**Calculation Time**	**Size** $\mathbf{Q_z}$	$\mathbf{V_f}[\times 10^3]$	$\mathbf{Cond(P_{zx})}$
$\mathbf{Q_{z21}}$	SSDMPC	2	600sec	10x10	55.3	17.07×10^3
$\mathbf{Q_{z22}}$	ERPC	2	0.922sec	10x10	0.572	1.1×10^3
$\mathbf{Q_{z5}}$	ERPC	5	8.968sec	16x16	1.023	889
$\mathbf{Q_{z8}}$	ERPC	8	48.699sec	22x22	1.356	982
$\mathbf{Q_{z10}}$	ERPC	10	134.45sec	26x26	1.7836	965
$\mathbf{Q_{z13}}$	ERPC	13	560.3sec	32x32	2.667	949.5

simply much bigger in certain directions (specially in $\dot{e}$ direction). It can be verified by dividing the eigenvalues of P_x from the two solutions

$$\begin{aligned}\frac{a_i(SSDMPC)}{a_i(ERPC)} &= \frac{(\lambda_i(P_{zx,SSDMPC}))^{1/2}}{(\lambda_i(P_{zx,ERPC}))^{1/2}}, i = (1, ..., 6) \\ &= (1.24,\ 2.63,\ \mathbf{8.81},\ 1.33,\ 1.88,\ \mathbf{128.02})\end{aligned}$$

The SSDMPC solution is larger than the ERPC in *all* directions, particulary in p and $\dot{e}$ axes, which causes the condition number to increase (it is clearly seen in figures 6.6 and 6.7, the ellipsoids are larger in these directions).

The top left of figure 6.5 shows the calculation time of the SDP problem (4.55), during the iterations. Since a feasible start position needs to be calculated at the first iteration, this iteration is the longest- $30sec$, while it drops sharply to 6 seconds at the second iteration. The average calculation time of the SDP problem is about 8.1 seconds. Total calculation time stands at 600 seconds, longer than the ERPC solution with $N = 13$.

The norm $\|Q_z - P_z^{-1}\|_\infty$ (top right) is as large as 10^5 at the first few iterations, despite the relatively low value of $\|Q_z P_z^{-1} - I\|_\infty$. This is when the initial variable is heavily augmented in order to find a larger solution. The SSDMPC algorithm increases the penalty parameter to $\mu_k = 1000$ at iterations $k \geq 11$. It causes the norm to drop sharply from 10^5 to 1. This enables faster convergence and avoids the problem of ill-conditioning. The bottom left of figure 6.5 shows the condition number of the hessian and the penalty parameter. The increase in the penalty parameter after 11 iterations causes the Hessian to be ill-conditioned almost instantly, as was proven analytically in section 4.4.1. α_k sizes (bottom right) indicate that the Newton direction p_k is a descend direction where the line search backtracking algorithm reduces v_k in a satisfactory manner (i.e. the directions are not blocked as the α_k sizes are near value one).

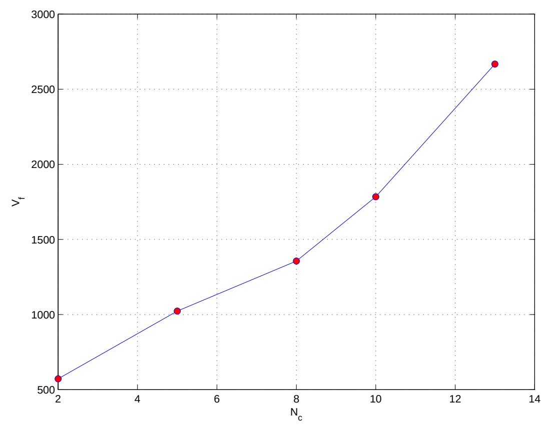


Fig. 6.4: Offline ERPC, N_c vs. the volume factor.

Uncertain model

The helicopter LDI (6.13) was simulated using ERPC algorithm only. The ellipsoid $\mathcal{E}_{zx}$ is very conservative as expected, with $V_f = 4.1$. This is a region which is 130 times smaller than the equivalent ERPC(2)! However, increasing the penalty on the state p to $q_3 = 1000$ causes a significant increase in $\mathcal{E}_{zx}$ size, up to $V_f = 160$, a region with a reasonable operating range. A similar test was made for the nominal case, with high penalty $q_3 = 1000$, but there was only minor increase in $\mathcal{E}_{zx}$. Penalizing the states where the uncertainties appear helps to reduce their effect, thus increasing the size of $\mathcal{E}_{zx}$, that is, if the system permits such penalties. It is recommended therefore to use reasonable Q, R values when uncertainties are presented, rather than 'blind' tuning. The results are summarized in table 6.3.

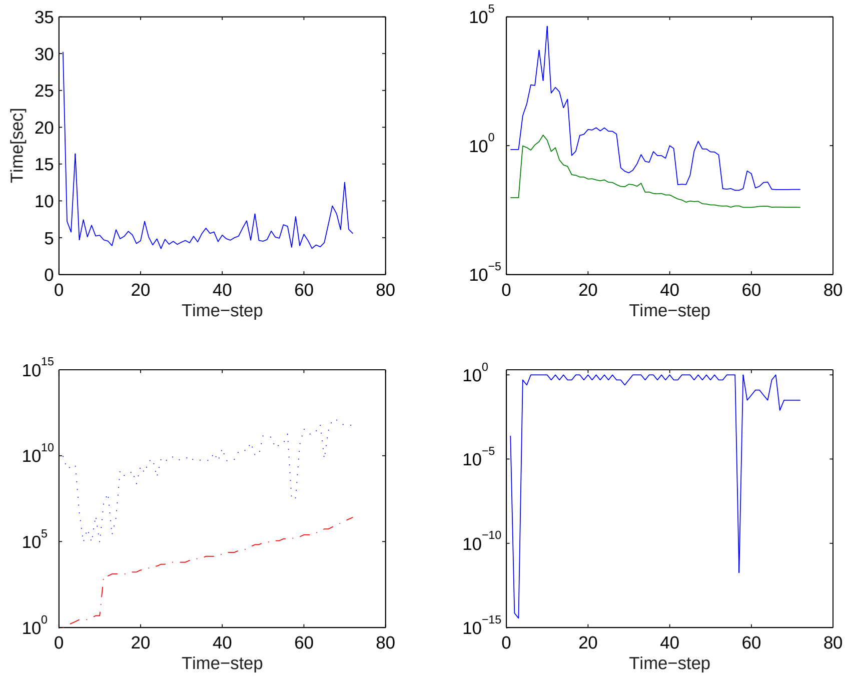


Fig. 6.5: Offline SSDMPC algorithm: Top left: Optimization time. Top right: $\|Q_z - P_z^{-1}\|$ blue solid and $\|Q_z P_z - I\|$ green (–) Bottom right: Hessian condition number (blue dotted) and penalty parameter μ_k in red (–). Bottom left: Step size α_k

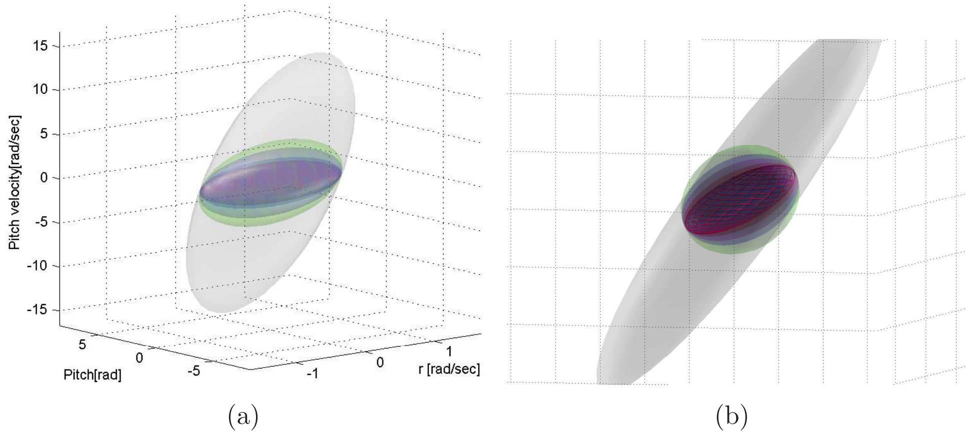


Fig. 6.6: r, p, $\dot{p}$ ellipsoid projections for SSDMPC (outer grey) and ERPC with $N_c = 2,\ 5,\ 8,\ 10,\ 13$. The ERPC ROA sizes are hardly increased.

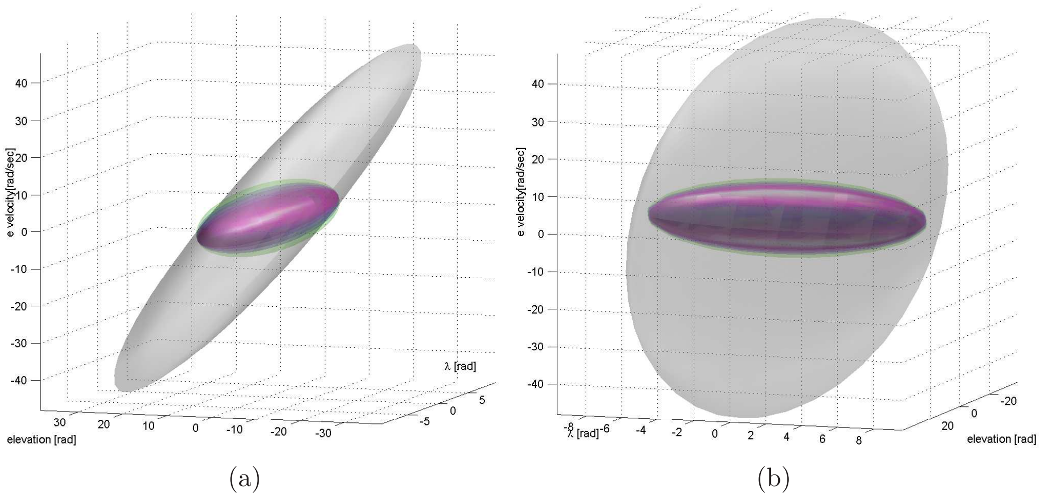


Fig. 6.7: $\lambda, e, \dot{e}$ ellipsoid projections for SSDMPC (outer grey) and ERPC with $N_c = 2,\ 5,\ 8,\ 10,\ 13$.

Table 6.3: Helicopter LDI ERPC offline results.

Simulation	$\mathbf{q_3}$	$\mathbf{V_f}$
LDI	1	4.1
	1000	160
Nominal	1	572
	1000	678

6.5 The 3-DOF online

6.5.1 Online Model Simulation

This section will deal with simulations only, that is, not real time experiment with hardware, only online results in Simulink. The motivation is to examine the time necessary to obtain the NR solution $\mathbf{c}_k$ and the effect of sampling-rate on the performance. The conclusions which are drawn in this section will be demonstrated in the real-time environment with a very high sampling-time.

The simulations of the 6-states model (6.10) is performed in Simulink with parameters similar to the real-time experiment, with less conservative sampling-time of 0.01 seconds (ten times slower than the real-time experiment). If it takes longer than $0.01sec$ to solve the Newton Raphson algorithm, It will definitely not be suitable for the real-time application with sampling-rates of 10^{-3}. The Simulink diagram is given in figure 6.8.

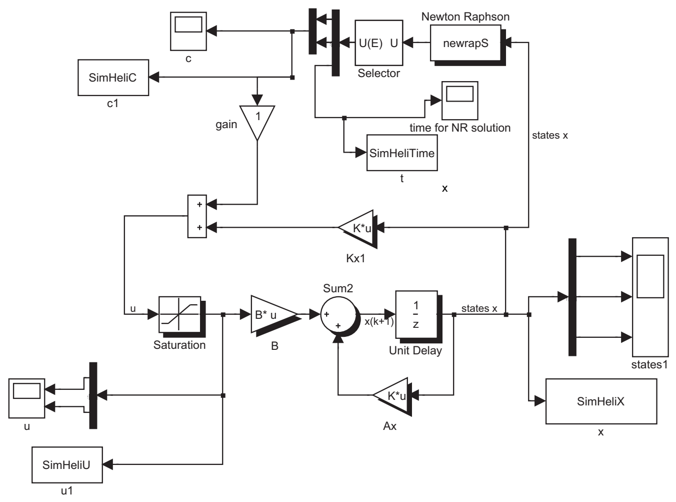


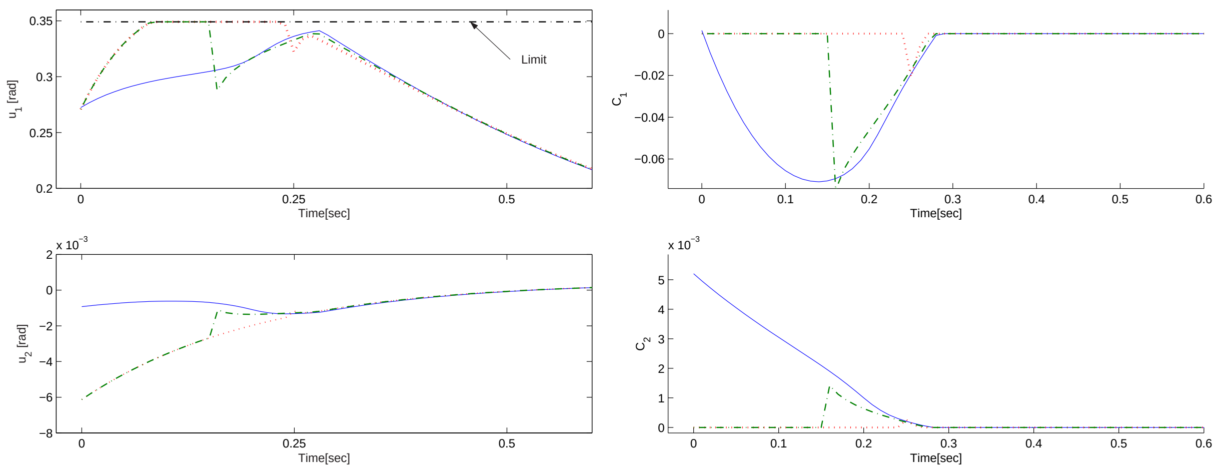
Fig. 6.8: The Simulink diagram

The sampling-time was chosen to be fixed (as is usually the case for real-time applications). The Newton Raphson is solved using the S-function 'newrapS' in the figure (the file is given in appendix C.2.3).

Consider the next initial position

$$\mathbf{x}_0 = (0, 0.4, 1.2, -1.0, 0, 0.1)^T \tag{6.15}$$

The initial point was chosen such that the position is realistic, with travel rate of $0.4\frac{rad}{sec}$ and pitch angle of 68^o, value which is higher than the maximum input $\bar{p}_c = 20^o$. This should cause saturation of the control input. The pitch velocity is chosen to be $-1\frac{rad}{sec}$. This initial position is feasible for the SSDMPC solution P_z (feasibility can be verified using the procedure discussed in section 5.3), as it lies inside the ellipsoid $\mathcal{E}_{zx,SSDMPC} \in \mathbb{R}^6$. This initial point is not feasible, however, for the any of the ERPC solutions, as will be demonstrated by the next results. Figure 6.9 left shows the control input for three different cases: The blue solid line is the control input for the SSDMPC solution. As clearly seen in the figure, it never reaches the saturation value of $0.35rad = 20^o$, but the value $\mathbf{c}(k)$ provides the correction needed to avoid this saturation. In contrast to the behavior of the ERPC solution for $N_c = 2$ (red dotted), the control input $u_1(k)$ reaches saturation after about $0.075sec$ and the system becomes non-linear for about 0.23 seconds (during the saturation). During this time, the trajectories are getting closer to the origin, eventually getting inside the ERPC ellipsoid after 0.24 seconds. Then $\mathbf{x}$ becomes feasible, allowing the Newton Raphson algorithm to find a solution $\mathbf{c}(k)$ (this is seen as the breaking point of the control input at time 0.24 seconds). The ERPC solution for $N_c = 13$ (green -.-) is getting feasible after 0.15 seconds, with a drastic perturbation of the control input at that time instance (see the top right side of the figure). $\mathbf{c}(k)$ computation time for the two solutions SSDMPC $N_c = 2$ and the ERPC $N_c = 13$ can be read in figure 6.10. It never exceeds the sampling-time limit of 0.01 seconds for the SSDMPC (it is lower in practice, but the simulation's time-step is fixed). Yet, using the ERPC solution, calculation time is as high as 0.04 seconds, due to the size of P_z. This is clearly not acceptable in real-time applications since update time must lie within the sampling-period (this case not exceed 0.01 seconds). This section has demonstrated the importance of a solution with low size matrix Pz, but in the same time ellipsoid which covers a large region around the origin.



(a) (b)

Fig. 6.9: Helicopter simulation $SSDMPC$ (blue), $ERPC$ with $N_c = 2$ (red dotted) and $ERPC$ with $N_c = 13$ (green -.-). Left: The control input. Right: The value $\mathbf{c}_k$.

6.5.2 Online Lab Experiment

This section will summarize the results from the real-time experiment of the Quanser helicopter dynamic system. The state space model and the mathematic description of the system is given in section 6.2.2 and the Simulink diagrams of the Helicopter model are given by appendix B. As will be shown here, not only can the online Newton Raphson be used in high sampling-rates, real-time systems, but it produces better results than the unconstrained LQ control scheme. Constraints violation is prevented by using the efficient online MPC, and good performance is achieved.

As the previous section suggests, perturbing the pitch and pitch-velocity is the main reason for instability, and the state trajectory can easily get outside the boundaries of the ellipsoid $\mathcal{E}_x = \{\mathbf{x}|\ \mathbf{x}^T P_x \mathbf{x} \leq 1\}$, there $\mathbf{f}_k \neq \mathbf{0}$. The Newton Raphson algorithm should then calculate a solution $\mathbf{c}_k$ which will perturb the input signal such that saturation will be avoid. Figures 6.11 presents the experiment results. The control input is perturb to a value of (-15^o). The left side (a) shows both the control signal and $\mathbf{c}_k$ value. During the time interval $52.5 - 53.2sec$, the input signal sinks to minimum value $-18deg$. At time point $53.2sec$, The state trajectory $\mathbf{x}$ is leaving the boundary of the ellipsoid $\mathcal{E}_x$ (shown in red, figure 6.12). This makes the Newton Raphson algorithm active, and a perturbation value c is calculated (presented in green, figure 6.11(a)). The positive c values prevent further decrease of the input signal, and p_c never violates the -20^o constraint. After about $5sec$, the input signal stabilizes on a stationary value within the constraints. The performance of unconstraint LQ controller is shown in figure 6.11(b). The NR algorithm was disconnected and the input signal is calculated by $u = -Kx$. The control input sinks at time $t = 131sec$ but since no consideration to the constraint $u = -20deg$ is taken, it is violated and the input signal decreases to a minimum value of -23^o.

Figure 6.12 presents two views of the trajectory in the $(r,\ p,\ \dot{p})$-space. The inner red ellipsoid is $\mathcal{E}_x$ where $\mathbf{f}_k = 0$. The volume between the outer ellipsoid, $\mathcal{E}_{zx}$ and the inner one is where the Newton Raphson algorithm performs optimization, i.e. $\mathbf{f}_k \neq 0$. The right figure shows clearly when the state trajectory leaves $\mathcal{E}_x$ and enters it again. The ellipsoid $\mathcal{E}_{zx}$ created by the offline algorithm SSDMPC is large so the trajectory never leaves its boundary.

Experiment results for long control horizons ($N_c = 10$) are shown in Figure 6.13.

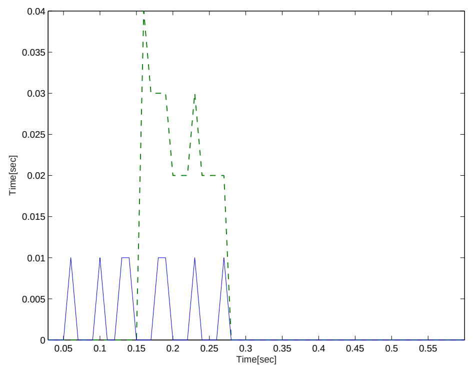


Fig. 6.10: Calculation time of Newton Raphson. *SSDMPC* (blue solid) and *ERPC* with $N_c = 13$ (green - -).

Clearly, the algorithm functions properly for small angles since the NR is not activated. At time point $t = 83.3$, the trajectory crossed the ellipsoid $\mathcal{E}_x$ boundary, and the first NR iteration for solving (5.15) demanded long calculation time, which resulted in RTX exception, and system crush.

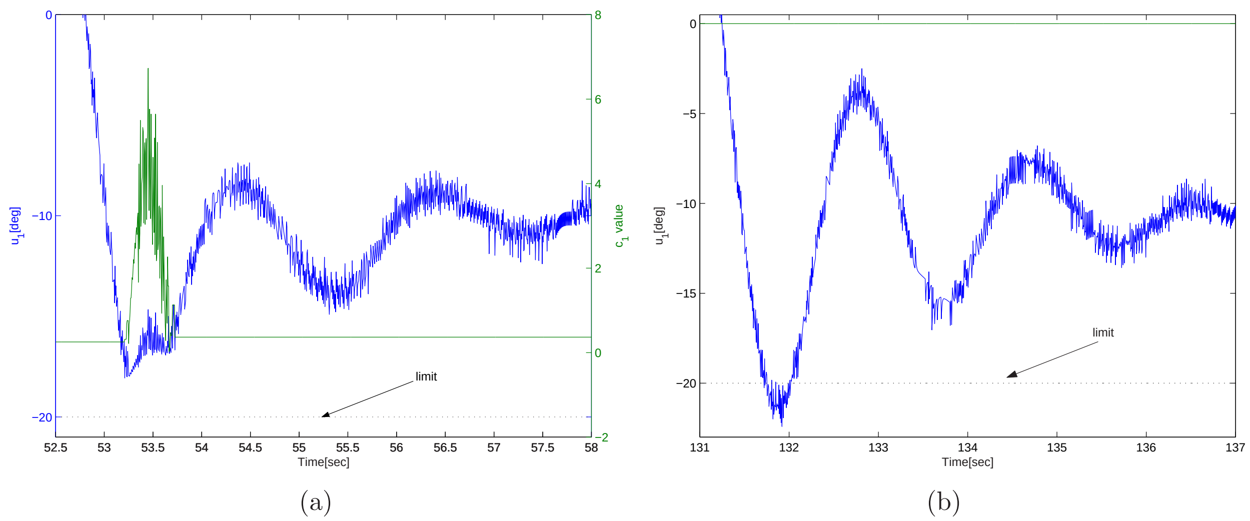
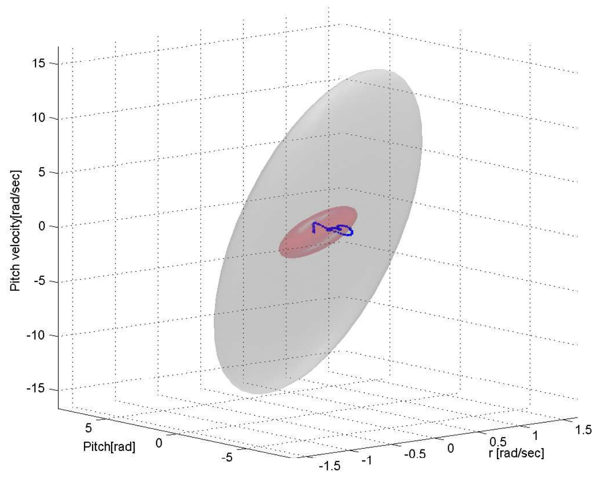
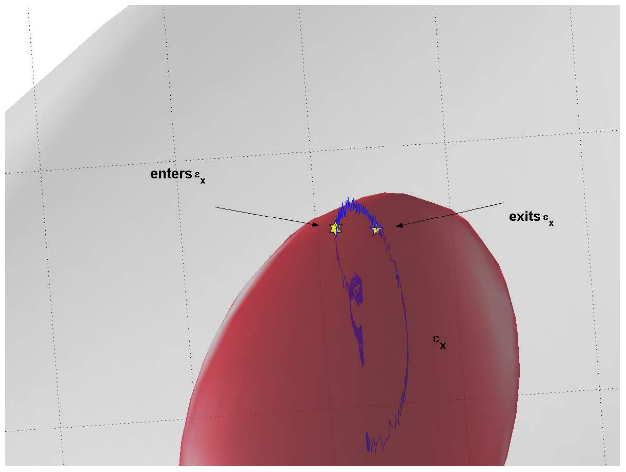


Fig. 6.11: Helicopter experiment: (a) The input signal (blue) and the perturbation $\mathbf{c}_k$ (b) The input signal for unconstraint LQ control, i.e. $\mathbf{c}_k = 0$ (green).

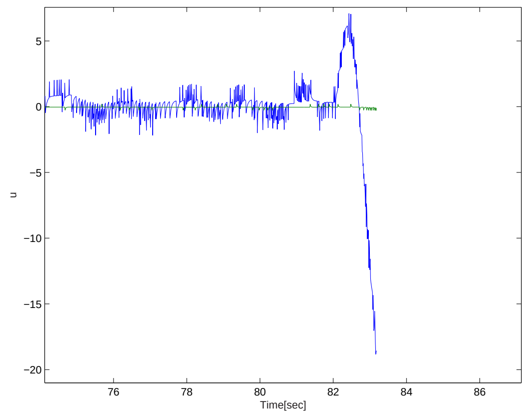


(a)

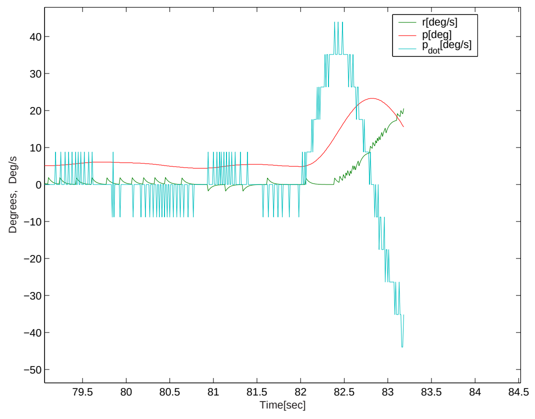


(b)

Fig. 6.12: The real-time experiment trajectory and the ellipsoids $\mathcal{E}_x$ (innermost) and $\mathcal{E}_{zx}$ (outermost). (b) is a magnification of (a). The exit point from $\mathcal{E}_x$ is where $\mathbf{c}_k \neq 0$, and enter point is where $\mathbf{c}_k = 0$ again.



(a)



(b)

Fig. 6.13: The real-time experiment for $N_c = 10$. An exception occurred at $t = 83.2$, when NR computation time exceeded sampling-rate 0.001. Right: The control input. Left: Time response of the states r, p, and $\dot{p}$.

6.6 Experiment discussion

6.6.1 Online Effective MPC performance

Implementing the Online effective MPC scheme in a real-time environment was very successful. The experiments indicate that the Newton Raphson algorithm was able to cope with high update rates. Experiments were taken for sampling-times range $0.02 - 0.1 \cdot 10^{-3}$, as it was not possible to use higher update rates than $0.02sec$ due to instability of the model. This fact excludes all solutions which demand long computation times (such as $N_c = 10$ and $N_c = 13$). Both feasibility and invariance were preserved during the entire experiment.

As the figures from the experiment suggest, the mathematical model given by (6.10) is very different from the actual plant. Although the closed loop system is stable, noise can be clearly seen, especially in the pitch velocity. Even if the model was close to the real plant, the sensors which measure the angles are not accurate. Accurate calibration requires sending the sensors, motors and other instruments to the manufacturer, and this has not been done due to time limitation of this thesis. Despite these inaccuracies, the closed loop system performances is satisfactory, and the helicopter experiment is a reliable tool for evaluation of the offline-online scheme.

Notice that no actual saturation of the input occurs at $u = -20^o$ during the experiment. This constraint is not a physical limitation and *can* be crossed. Setting this constraint value for the input was meant to illustrate the advantage of using the online method.

6.6.2 Improving Real-Time Performance

The Helicopter experiment was conducted using the Simulink model given by appendix B. The NR algorithm was implemented using C-code based S-function. The C-code was not optimized for best performance, and improvement of the code may yield fast calculation time, although the expected difference should not be large. The code itself is not long, so code optimization should not increase the efficiency enough to achieve faster NR calculation times. There is a slight difference between the block model-based NR (see appendix) and the C-code based NR, but since the number of operations in the NR is relatively small, this difference is not significant, and both schemes (C-code and model-based NR) can be implemented. The Newton Raphson inverse operation $(I_f + \lambda P_{22})^{-1}$ from (5.13) is the most time consuming operation and can be accelerated using eigenvalue decomposition of P_{22}.

Chapter 7

Discussion and Evaluation

This chapter discusses some important topics of the offline-online strategy and presents extensions and future work which can be done. Detailed discussion about the ERPC and SSDMPC offline algorithms is given in the end of chapter 4. Online and experiment discussions are given at the end of each chapter, and generally not repeated here.

7.1 Robust feasibility

Explicit limits on the inputs and a polytope region (3.37) can be used to compute an LDI. This LDI results in a conservative offline solution, and the regions of attraction can be reduced to unusable dimensions. Simulations reveal that the region diminishes as the size of the polytope Ω increases. Cannon et al. (2003) present simulation results of an LDI representation of a closed loop MIMO system, where the degree of conservatism of standard online algorithms (e.g. QP-MPC) increases with the size of the inclusion polytope, and they become impractical. The online NR controller, on the other hand, is extremely fast and can make a good replacement. An important drawback however, is that the offline ellipsoid can be very conservative for such an LDI, limiting the operating range of the system. A fast online NR controller has little use if the operation range is very small. It is important, therefore, to attempt to increase the region of attraction.

if there are uncertain parameters in the plant, however, the region of attraction can be increased with proper tuning of the penalty matrices Q and R. Increasing the penalty on the term providing the input to the uncertain parameter will reduce the effect of this parameter on the closed-loop system. This will most likely increase the size of the ellipsoid and R should then be large. If a particular entry of the A matrix is likely to be highly variable (i.e. $a_{ij} + \delta$ where δ varies largely), then increasing the penalty on that state is worthwhile. This should result in reduction of the magnitude of the signal $a_{ij}x_j(k)$, and the perturbations introduced by varying δ will also be smaller. This is likely to increase the size of the ellipsoid. These conclusions were verified by the helicopter uncertainty model (6.13). Heavily penalizing x_3^2 results in an ellipsoid 130 times larger. This is not true in the nominal case, where penalizing x_3^2 by the same factor actually *reduced* the size of the ellipsoid. See section 6.4.2.

Trading Q and R values should be done with care in order to maximize the size of the resulting ellipsoid. Of course, the properties of the plant must be taken into consideration. For instance, the dynamics of the helicopter system require that R will not be too low, or instability occurs (due to model inaccuracy the plant). This was verified by experiments.

7.2 Weaknesses

7.2.1 Static method

Feasibility, Invariance and closed loop stability of the system are not assured if the ERPC or SSDMPC controllers are used in a manner requiring any online re-design. This includes any adaptive control in the traditional set-up, where re-estimation of the plant model parameters and consequent redesign of the controller is assumed to occur continuously. This imposes a serious limitation since a large variety of systems are built in this way.

By its very nature, the offline-online method cannot be used for tracking control, at least not in its current state. It is possible to achieve similar effect by transforming the center point of the ellipsoid to a sequence of equilibrium points, which creates the path. This is suitable for nominal linear models only, and the online controller loses the optimality.

7.2.2 Conservatism

The ellipsoid can be very conservative compared to the maximal admissible set, since it is based on a mathematical framework (namely the ellipsoid representation). Thus QP-based MPC can be feasible in regions where the online NR controller is not feasible. The difference can be significant if a large number of LMI constraints are used. Conservatism was especially observed in uncertain systems, using inclusion polytopes.

7.2.3 Symmetric constraints

By the structure of the offline algorithms, the inputs and states constraints can only be defined symmetrically, i.e. $\underline{\mathbf{u}} \leq \mathbf{u} \leq \overline{\mathbf{u}}$ where $\underline{\mathbf{u}} = -\overline{\mathbf{u}}$. The reason for this limitation comes from the symmetrical definition of the ellipsoid, which is centered at the origin, symmetric to its main axes. This imposes a limitation which a standard QP-MPC algorithm handles with ease, since it is not constrained to the mathematical description of the ellipsoid.

At the time of writing, no solution to this problem has yet been suggested in the literature. The concept of the offline will probably have to be rendered from ellipsoid to other geometrical shapes.

7.2.4 Complexity

The complexity level of the SSDMPC offline and the NR online algorithms increases dramatically with increase in the size of the system. The largest system which was tested in a real-time environment was the helicopter model with size of 6 states and 2 control inputs. Larger systems exhibits long calculation times and this may prevent the NR from calculating c_k in a satisfactory time (by satisfactory means shorter time than the sampling-rate T_s). Fortunately, systems with high sampling-rates such as aircrafts, underwater vehicles, engines, etc., have usually small models, while process control systems have very large models, but also sampling-periods up to few hours, and thus applicable by the SSDMPC strategy. Obtaining the offline solution for large number of variables is a time consuming process, but since it is performed *offline*, it should not present any difficulties. Ergo, complexity is only a minor drawback.

7.3 Strengths

7.3.1 Low computation time

The Newton Raphson online algorithm has been shown to be extremely fast.

Online NR performs best when the matrix P_z is small. If the offline ERPC produces small ellipsoids, one can either increase the control horizon N_c until satisfactory size achieved, or solve the SSDMPC for low N_c values. The latter is clearly a better solution as short horizons yield smaller P_z matrix sizes, which are less expensive to the NR to solve, resulting in a faster convergence time.

The helicopter experiment has demonstrated feasibility and stability for a very high update rate of $10^{-3}sec$, a rate which cannot be used for QP based MPC methods today. Moreover, the ability to handle constraints where the unconstrained LQ method fails was demonstrated.

7.3.2 Demand for minimum resources

Not only is the Newton Raphson extremely fast, it has a very low demand for resources, especially when using small P_z sizes, due to a low number of operations required. This is encouraging since many embedded systems place a heavy demand on resources. Weight, space and costs are the prime factors (as well as limitations) for almost every embedded system, whether a missile with a serious weight-space limitation or medical systems with cost limitations. The superiority of the Newton Raphson algorithm upon QP-MPC is clear: complexity and resource requirements are incomparable between the two. The helicopter experiment has demonstrated good performances with low resources (a weak Pentium II 300MHz CPU).

7.3.3 Constant Reference Set Point

As mentioned above, SSDMPC and ERPC are not suitable for tracking. They are suitable, however, for constant reference set points $\mathbf{x}_0$. The offline algorithm is calculated as usual for an ellipsoid centered at the origin. Before using the controller online, a new equilibrium point is calculated by using $f(x_0,\ u_0) = 0$ to calculate $\mathbf{u}_0$ for the given set point $\mathbf{x}_0$. Next, the center of the ellipsoid is transformed to the new equilibrium point using $\tilde{\mathbf{x}} = (\mathbf{x} - \mathbf{x}_0)$ and $\tilde{\mathbf{u}} = \mathbf{u} - \mathbf{u}_0$.

7.3.4 Flexibility

Since the SSDMPC offline algorithm has both P_z and Q_z as variables, it is possible to use utilities such as the $\mathcal{S}$-procedure, where an initial ellipsoid is confined inside the ROA. This gives the SSDMPC a certain flexibility in the definition of additional LMIs, as any of the decision variables defined by the SSDMPC can be used by such utilities. It is not case for the ERPC, since it does not contain P_z as a variable, and cannot use the $\mathcal{S}$-procedure in its current state. In fact, ERPC algorithm cannot use any LMI constraint which contains other variables than Q_z.

7.4 Possible Extensions

7.4.1 Switching Modes

As discussed in Section 4.9.2, tuning the weights on the control input affects the size of the region, both for ERPC and SSDMPC. For the unconstrained case, as the control weighting tends to zero, $\mathbf{x}(t)$ signals may becomes arbitrarily large, especially near $t = 0$ [*]. This reduces the probability that a $\mathbf{c}_k$ value exists to maintain the system's feasibility, and the region of attraction shrinks. This is true for the opposite case: If R increases, the control signals are low, and the chances for infeasibility are reduced. This increases the size of the ROA.

Now consider the situation where the initial condition $\mathbf{x}_0$ is not in the set $\mathcal{E}_{zx}$, rendering the online NR controller ineffective. A remedy would be to generate *another* region, by increasing N_c, or using a less tuned controller by setting higher R values. R or N_c can be increased repeatedly until $\mathbf{x}_0 \in \mathcal{E}_{zx}$. This new region will be larger, enlarging the operating range of the system, but at the expense of less tuned controller, or even a large P_z matrix size. It is then possible to operate with a few different ellipsoids and *switch* between them as the state trajectory tends to zero. This creates a set of ℓ ellipsoids $\Upsilon = \{\mathcal{E}_{zx}^1\ \mathcal{E}_{zx}^2\ ...\ \mathcal{E}_{zx}^\ell\}$ where $\mathbf{x}_0 \in \Upsilon$. This may enable improvement of the performance while maintaining feasibility. Switching between ellipsoids can be done in the transition of the state trajectory from one region to another. A simple feasibility check can be done, in order to verify that $\mathbf{x}_k$ is feasible for the new ellipsoid (see section 5.3). The set of regions Υ can be chosen by different categories, depending on the priorities of the

[*]At the few first iterations for the discrete case.

user. For limited resources (CPU, memory) and/or high sampling-rate, the R value can be the augmentation element. If time and resources are less significant, the operating range can be increased by allowing the increase in N_c values to generate the set Υ.

The switching mode strategy can be applied to both ERPC and SSDMPC, or any combination of the two.

This strategy has not yet been investigated closely, and is, for the time being, simply a theory.

7.4.2 State Feedback Observer

The efficient online NR controller requires a full state information. An observer such as Kalman Filter can be used to estimate unavailable states, assuming the system is observable. It is interesting to investigate whether the system performance can be improved by using a state observer. Kalman Filter can be very useful for the helicopter model since it can eliminate noise from the sensors and expected noise of the process (e.g. $\pm 5^0$ pitch noise).

7.4.3 Symmetric constraints

As was mentioned above, the constraints must be symmetric since all axes of the ellipsoid are symmetrical. If input or state constraints is to be considered, then another geometrical shape should be considered, which is non-symmetrical in at least one axis. In the two and three dimensional case, this can be achieved by define an oval shape, which is symmetric only about one axis, as the geometric shape of an egg. The difficulty lies in describing this shape mathematically using linear algebra tools, and even if such a description exists, the problem formulation and the approach will most likely have to be rended since the LMI framework requires that the variable matrices will be symmetric positive definite.

Chapter 8

Conclusions and recommendations

The SSDMPC algorithm was analyzed in this thesis. The most important elements which affect the size of the region of attraction are discussed and extensive comparison to the offline ERPC algorithm is given. SSDMPC generates larger ellipsoids than the ERPC for the same horizon length. Large ROA which has a small P_z matrix size makes the SSDMPC applicable for systems with high sampling-rates and embedded systems with resources and/or cost limitations. The SSDMPC gives possibility to handle constraints in such systems, and ensures stability. This strategy can also maintain feasibility despite uncertainty about the plant, by generating a suitable region of attraction, thus ensuring robust feasibility of the system. A successful implementation of the SSDMPC combined with the NR-online demonstrates the effectiveness of the strategy. It is thus recommended to use the SSDMPC to generate an ROA with small horizon lengths, first using a monotonic increase rate of the penalty parameter. If many iterations are needed, or if the algorithm fails to obtain a solution due to numerical inaccuracies, an augmented increase of penalty parameter is advised. Then the solution obtained can be used by the fast NR algorithm to achieve good performance and stability for constrained systems.

Bibliography

Alizadeh, F., Haeberly, J.-P. A. and Overton, M. L. (1998). Primal-dual interior-point methods for semidefinite programming: convergence rates, stability and numerical results, *SIAM J. Optim.* **8**(3): 746–768.

Anderson, B. and Moore, J. (1989). *Optimal Control, Linear Quadratic methods*, Prantice-Hall international, Englewood Cliffs, NJ 07632.

Bertsekas, D. P. (1996). *Constrained optimization and Lagrange Multiplier methods*, Athena Scientific, PO box 391, Belmont, Mass. 02178-9998, USA.

Boyd, S., El Ghaoui, L., Feron, E. and Balakrishnan, V. (1994). *Linear Matrix Inequalities in System and Control Theory*, number 15 in *SIAM Studies in Applied Mathematics*, SIAM.

Cannon, M., Kouvartakis, M. and Bulut, B. (2003). Nonlinear predictive control of hot strip rolling mill, *Robust and Nonlinear Control* (13): 365–380.

Cheng, S. H. and Higham, N. J. (1998). A modified cholesky algorithm based on a symmetric indefinite factorization, *Society for industrial and applied mathematics*.

Drageset, S. (2002). *A bilinear matrix inequalities approach to efficient model predictive control*, Master's thesis, Department of Engineering Cybernetics, NTNU. http://www.itk.ntnu.no/ansatte/Imsland_Lars_Struen/BMIEMPC.pdf.

Drageset, S., Imsland, L. and Foss, B. (2003). Efficient model predictive control with prediction dynamics, *IEEE Transactions on Automatic control*. Accepted as a regular paper.

Ennix, K. A., Burcham, F. W. J. and D., W. L. (1993). Flight-determined engine exhaust characteristics of an f404 engine in an f-18 airplane, *Technical Memorandum 4538*, NASA, Edwards, California.

Fares, B., Apkarian, P. and Noll, D. (2000). An Augmented Lagrangian Method for a class of LMI-Constrained Problems in Robust Control Theory, *Submitted to Int. Journal of Control*.

Fletcher, R. (1987). *Practical Methods of Optimization*, 2th edn, John Wisely and Sons.

Fossen, T. I. (2002). *Marine Control Systems*, Marine Cybernetics, Trondheim, Norway.

Gahinet, P., Nemirovski, A. and Laub, A. (1995). *LMI Control Toolbox, For use with Matlab*, 1st. edn, MathWorks, 24 Prime Park Way, Natick, Mass. 01760-1500.

Gilbert, E. G. and Tan, K. T. (1991). Linear systems with state and control constraints: the theory and application of maximal output admissible sets, *IEEE Trans. Aut. Control* **3**6(9): 1008–1020.

Horn, A. R. and Johnson, C. R. (1985). *Matrix Analysis*, reprint 1999 edn, Cambridge University Press, The Edingburgh Building, Cambridge CB2 2RU, UK.

Imsland, L. S. (2002). *Topics in Nonlinear Control - Output Feedback Stabilization and Control of Positive Systems*, PhD thesis, Norwegian University of Science and Technology, Department of Engineering Cybernetics.

Kincad, D. and Cheney, W. (2002). *Numerical Analysis, Mathematics of sceintific computing*, third edn, Wadsworth group, 511 Forest Lodge Road, Pacific Grove, CA 93950 USA.

Kothare, M. V., Balakrishnan, V. and Morari, M. (1996). Robust constrained model predictive control using linear matrix inequalities, *Automatica* **3**2(10): 1361–79.

Kouvaritakis, B., Cannon, M. and Rossiter, J. A. (2002). Who needs QP for MPC anyway?, *Automatica* **3**8(5): 879–884.

Kouvaritakis, B., Rossiter, J. A. and Schuurmans, J. (2000). Efficient robust predictive control, *IEEE Trans. Aut. Control* **4**5(8): 1545–1549.

Liu, R. (1968). Convergent systems, *IEEE Transactions on automatic control* (AC-13): 384–391.

Maciejowski, J. (2002). *Predictive Control with Constraints*, 1st. edn, Pearson Education, Edinburgh Gate, Harlow, Essex, CM20 2JE, England.

Nocedal, J. and Wright, S. J. (1999). *Numerical Optimization*, Springer-Verlag, New York.

Ogata, K. (1995). *Discrete-Time Control Systems*, 2. edition edn, Prentice Hall, Upper Saddle River, New Jersey 07458.

Qin, S. J. and Badgwell, T. A. (2002). A survey of industrial model predictive control technology, *Contorl engineering practice* (11): 733–764. Availabe online at www.sciencedirect.com.

Rawlings, J. and Muske, K. (1993). The stability of constrained recedinghorizon control, *IEEE Transactions on automatic control* (38): 1512–1516.

Scherer, C. and Weiland, S. (1999). *Lecture Notes DISC Course on Linear Matrix Inequalities in Control*. *http://www.er.ele.tue.nl/sweiland/lmi99.htm.

Slupphaug, O. (1998). *On Robust Constrained Nonlinear Control and Hybrid Control: BMI- and MPC-based State-Feedback Schemes*, PhD thesis, Norwegian University of Science and Technology, Department of Engineering Cybernetics.

Wolkowicz, H., Saigal, R. and Vandenberghe, L. (eds) (2000). *Handbook of Semidefinite Programming*, Vol. 27 of *International series in operations research and managament science*, Kluwer, Boston, MA.

Appendix A

Modified Cholesky algorithm

The Modified Cholesky algorithm given here is taken from Nocedal and Wright (1999).

Algorithm A.1 (Modified Cholesky)

```
*choose* $\lambda > 0, \quad \beta > 0$
**for** $k = 1, 2, ..., n$
  $c_{kk} = a_{kk}$
**end**
**for** $j = 1, 2, ..., n$ *(compute the jth column of L)*
   **for** $s = 1, 2, ..., j-1$
        $l_{js} := c_{js}/d_s$ ;
   **end**
   **for** $i = j+1, ..., n$
        $c_{ij} = a_{ij} - \sum_{s=1}^{j-1} l_{js} c_{is}$;
   **end**
   $\theta_j = 0$;
   **if** $j < n$
        $\theta = max_{j<i\leq n} |c_{ij}|$;
   **end** $d_j = max\{|c_{jj}|, \ (\frac{\theta_j}{\beta})^2, \ \delta\}$;
   **if** $j < n$
        **for** $i = j+1, ..., n$
             $c_{ii} = c_{ii} - c_{ii}^2/d_j$;
        **end**
   **end**
**end.**
```

Appendix B

Simulink Diagrams

B.1 Helicopter model

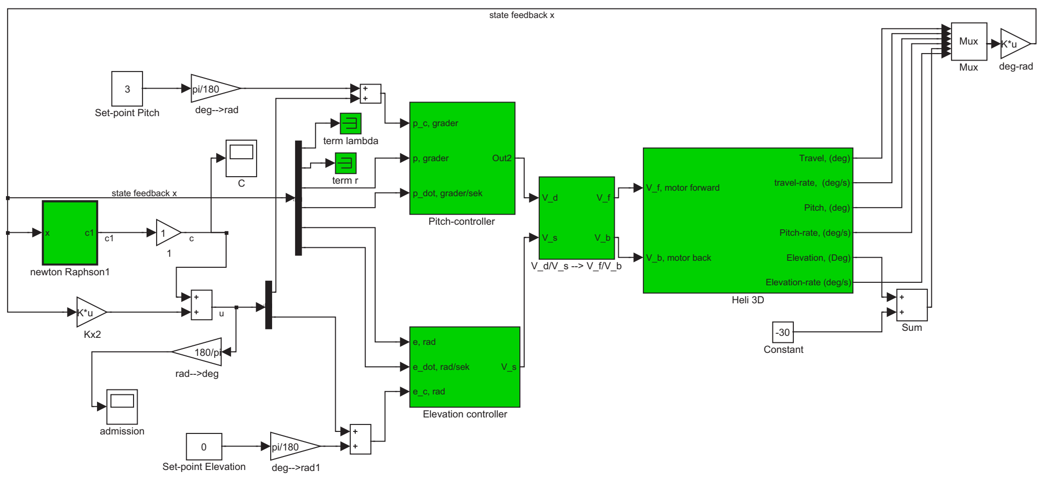
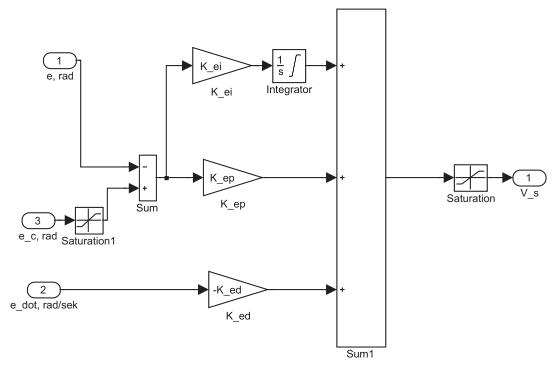
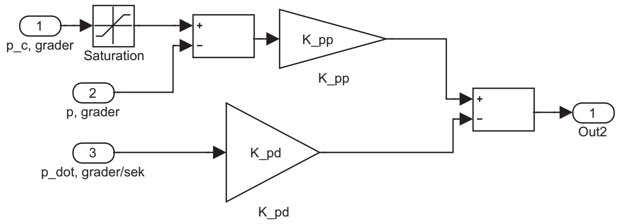


Fig. B.1: Simulink model Helicopter



(a) Elevation control



(b) Pitch control

Fig. B.2: Elevation and pitch controllers

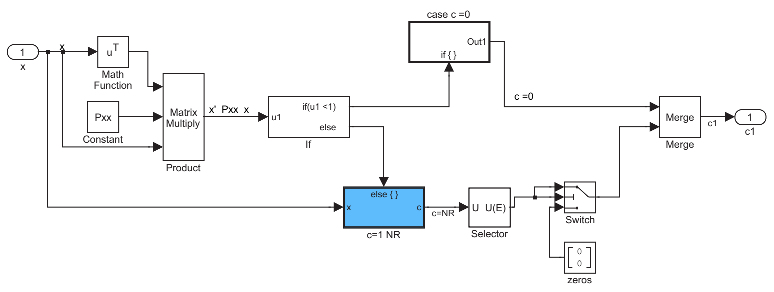


Fig. B.3: Newton Raphson, top level

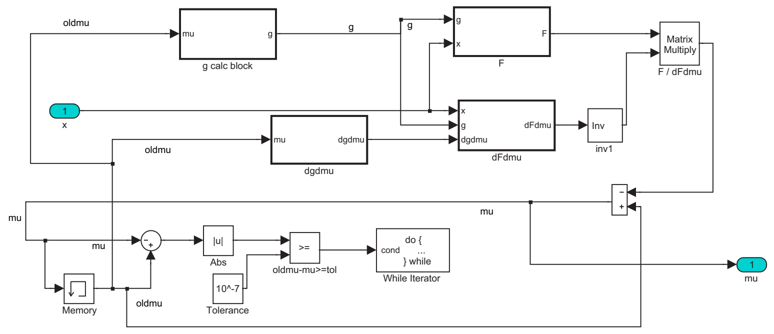


Fig. B.4: Newton Raphson, mid-level

B.2 Newton Raphson Simulink

Appendix C

Offline-Online algorithms

C.1 Offline algorithms

Note that SSDMPC and ERPC offline algorithms requires the LMI toolbox, can be downloaded free from the world wide web. All the required files can be found in the CD attached to the thesis. Please open *readme.txt* first.

C.1.1 SSDMPC offline

```
%SSDMPC ver 6
% Script for optimizing ellipsoidal volumes s.t. LMIs
% and Q = P^{-1}. A sequential semi-definite programming
%method is used, making a quadratic approximation of the
%objective at each iteration.
% ***********************************************************
% Main program, only Offline
% Call Online.m to execute the online part.
% start.
% Functioning:  - properly 23.03 2003
% Problems known:  B matrix with certain values produces
%                  Qz with high condition numbers.=> problems with inv
%  Version 6       Just like the main_multi2 but with
%                  another stop criteria (Feasibility + min normen )
%  Function very good with cmult(1) = 0.5 rho = 1.3;  mu = .996 and high value
% of c_linesearch = 0.3 after 10 iterations. (before 10 iterations c_linesearch=0.001)
% Nadi Bar
% files needed:
% init2.m
% nrf_multi.m, newrap.m, maxdet.mex
% funvalue2.m,
% svec, sgrad.m , sHessian.m, skron.m, T.m, invsvec.m, grad.m, hessian.m

% Updated: 19.05

clc; clear all; datetime1=clock; datetime=[['simulation date:
',int2str(datetime1(3))] ['/',int2str(datetime1(2))] [' , time:
',int2str(datetime1(4))] [':',int2str(datetime1(5))]]; clear
datetime1;
%---------INITIALISE-----
%solve prob with ERPC algorithm by Kouvaritakis
%initialises the following variables:
```

```
% N (N_c), horizon
% B, from system-model:  x_k+1=A*x_k+B*u_k.
% mx, length(A), dimension of system
% umax, upper input constraint
% phi, transition matrix x_k+1=phi*x_k (phi=A-B*K)
% KK, augmented feedback gain, u=KK*z
% PSIz, augmented transition matrix z_k+1=PSIz*z_k
% M block matrix in PSIz, treated as a general variable
% Pz, defines invariant ellipsoid z'*Pz*z < 1
% Qz=Pz^-1
% Tx, x=Tx*z
M=[];%nrf generates standard M
[A,B,C,xk,Q,R]=init2;
[N,B,phi,mx,dimu,umax,KK,PSIz0,M0,Qz0,Tx]=nrf_multi2([])
ConvCount=0
%load('simulations/Heli.mat')
Pz0=inv(Qz0); cond(Qz0) warning off;
sprintf('N=%d, mx=%d, dimu=%d  CASE= 3  Confirm: (press key)',N, mx,dimu)
 pause
% if mx<3
%     plotellipse(Px0,1,'r-');legend('Nom (r)');
%     hold on
% end

%------------BASIS LMIs------------
%Define LMIs for Matlab's LMI toolbox
setlmis([])
%n is the number of decision variables used so far.
[Q_id,n,Q_dec]=lmivar(1,[N*dimu+mx 1]);
[P_id,n,P_dec]=lmivar(1,[N*dimu+mx 1]);
[M_id,n,M_dec]=lmivar(2,[N*dimu N*dimu]);

[PSIvar_id,n,PSIvar_dec]=lmivar(3,[zeros(N*dimu+mx,mx)
[zeros(mx,N*dimu);M_dec]]);
%[PSIvar_id,n,PSIvar_dec]=lmivar(3,[Phi_dec B_dec zeros(mx,dimu*(N-1)); zeros(dimu*N,mx) M_dec]);

Te=zeros(N*dimu+mx,N*dimu+mx); Te(1:mx,1:mx+dimu)=[phi B];

%1st LMI
lmi1 = newlmi; lmiterm([-lmi1 1 1 Q_id],1,1) lmiterm([-lmi1 1 2
PSIvar_id],1,1) lmiterm([-lmi1 1 2 0],Te) lmiterm([-lmi1 2 2
P_id],1,1)

%2.1 LMI pådrag 1
lmi2 = newlmi; lmiterm([-lmi2 1 1 0],umax(1)^2) lmiterm([-lmi2 1 1
Q_id],-KK(1,:),KK(1,:)')

if dimu>1
    %3 LMI pådrag 2
    lmi3 = newlmi;
    lmiterm([-lmi3 1 1 0],umax(2)^2)
    lmiterm([-lmi3 1 1 Q_id],-KK(2,:),KK(2,:)')
end

%4 LMI  ( M-I<0 )
lmi4 = newlmi; lmiterm([-lmi4 1 1 M_id],-1,1) lmiterm([-lmi4 1 1
0],1)

basisLMI = getlmis; n_basisLMI = decnbr(basisLMI);

% Check if the LMI part is feasible
[lmi_min,xbeg]=feasp(basisLMI); if lmi_min>0
  display('BMI infeasible since LMI part is infeasible');
  break
end

Qm=Qz0; Pm=Pz0; PSIm=PSIz0; %convert initial results to lmitoolbox custom notation
PSIvar=PSIz0; PSIvar(1:mx, 1:mx+dimu)=zeros(mx,mx+dimu);

zk=[mat2dec(basisLMI,Qz0,Pz0,M0,PSIvar);0];
%Other initial values


%---------INITIALISE LOOP------
clear FWLMIs; clear objfun; pack tmm = 0;
Tm = T(Pm); dimP=length(Pm);%N*dimu+nx
I = eye(dimP);
%Tuning

kappa = 1;
cmult(1) = .5; %initial penalty weight
rho = 1.3; %rate at which penalty weight increases (the higher - the lower changes will be excepted)
mu = .996

norm_old= norm(Pz0-Qz0^-1,inf); Phi_old =
kappa*log(det(inv(Tx*Qm*Tx'))) + trace((Qm*Pm-I)'*(Qm*Pm-I));
optERPC = log(det(inv(Tx*Qz0*Tx')));
Lambda(:,:,1) = (Qm*Pm-I); %100*ones(dimP);
%Lambda(:,:,1) = zeros(dimP);
iter = 0;
iter_k=0; % for stop kriteria

tminAll=0
%option=[accuracy, max # iter, feasb radius, stop after J accuracy iter, trace execution 1=off]
options=[0.01,150,10^12,30,1]; % for mincx

%---------------LOOP, finds pk-------
STOP=0; while (not(STOP))
  tic
  iter = iter + 1
  %tic;
  xold = [svec(Qm);svec(Pm);];
  zold = zk; % z(k+1)=z(k)

  % --------Find Hessian and Gradient, Hk and gk
  %create perturbation matrix, pp, that transforms g and H from
  %being compatible with svec(Qx)=svec(TQzT') to being compatible
  %with svec(Qz)
teller=0; % generates pp
  for i=1:mx
      teller=teller+i;
  end
  posy=1;posx=1;
  pp=zeros(teller,length(xold))';
  for i=0:mx-1
      pp(posy:posy+mx-1-i,posx:posx+mx-1-i)=eye(mx-i);
      posy=posy+mx-i+dimu*N;
      posx=posx+mx-i;
  end
  pp=pp';
  clear posy posx teller % unessesary variables

  gQx=sgrad(Tx*Qm*Tx');
  gkte=pp'*gQx;
  HQx=sHessian(Tx*Qm*Tx');
  Hkte=pp'*HQx*pp;
  %Find g and H including rest of objective function
  Lk = Lambda(:,:,iter); % Lk is penalty parameter from 4.28
  gk = kappa*gkte + [Tm^2*( cmult(iter)*svec(Pm^2*Qm+Qm*Pm^2-2*Pm) + svec(Pm*Lk+Lk'*Pm) ); % Q
                     Tm^2*( cmult(iter)*svec(Qm^2*Pm+Pm*Qm^2-2*Qm) + svec(Qm*Lk'+Lk*Qm) )];% P

  Hk11 = cmult(iter)*skron(Pm^2,I,dimP);
%  Hk21 = skron(Lk,I,dimP) + cmult(iter)*(skron((Qm*Pm-I),I,dimP)+skron(Qm,Pm,dimP));
  Hk21 = skron(Lk,I,dimP) + cmult(iter)*(skron(Qm,Pm,dimP));
  Hk22 = cmult(iter)*skron(Qm^2,I,dimP);

  Hk = kappa*Hkte + blkdiag(Tm,Tm)*[Hk11,Hk21';Hk21,Hk22]*blkdiag(Tm,Tm);
  Hk=Hk*0.5;

  %  Modify Hk so that is is positive definite
  [V,D] =  eig(Hk);
  %eig(Hk)
  d = diag(D);

  d(find(d<1e-4))=10^-3;
  Hkm =  V*diag(d)*V';
  pow=2;

  while Hkm(isreal(Hkm)==0) & pow>=-4
      d = diag(D);
      d(find(d<1e-1))=10^-pow;
      Hkm =  V*diag(d)*V';
      pow=pow-1;
  end

  if cond(Hkm)>10^17 | Hkm(isreal(Hkm)==0)
      [Hkm,condition,Hkm_E]=mod_cholesky(Hk);
      disp('modified Cholesky')
      end
  condHkm(iter)=cond(Hkm);
%  eig(Hkm)
  %disp('cond(Hkm)=');cond(Hkm)
  %disp('cond(Hk)=');cond(Hk)

  %-------Add extra LMI (QP)--------
  setlmis(basisLMI);

  [tn_id, n_tn, tn_dec] = lmivar(1,[1 1]);
  [x_id,n_x,x_dec] = lmivar(3,[svec(Q_dec); svec(P_dec)]);

  minlmi = newlmi;
  lmiterm([-minlmi 1 1 tn_id],1,1);
  lmiterm([-minlmi 1 1 x_id],-gk',1);
  lmiterm([-minlmi 1 1 0],gk'*xold);% fordi x(k+1)=x(k)+P ==> P=x(k+1)-x(k)
  lmiterm([-minlmi 2 1 x_id],1,1);
  lmiterm([-minlmi 2 1 0],-xold);%P=x_id-xold
  invHkm = inv(Hkm);
  lmiterm([-minlmi 2 2 0],.5*(invHkm+invHkm'));
   allLMIs = getlmis;
   n_tn=decnbr(allLMIs); %
  %---------find pk---------
  c = zeros(decnbr(allLMIs),1); c(n_tn) = 1; %linear objective
  zk(n_tn) = 1e6;%initial values

  [lmi_min, zk] = mincx(allLMIs,c,options,zk);
  if isempty(zk)
      disp('mincx could not find t which is bigger than the Lagrange approx')
      disp('  while satisfying the LMI constraint. Check whether allLMIs is a feasible')
      disp('  set. if it is, then no solution is found.')
```

```
    disp('zk1=[] , aborting the program................')
    break
end

if (lmi_min<-1e12),
  disp('The algorithm has diverged!')
  break;
end

Pm = dec2mat(allLMIs,zk,P_id);
Qm = dec2mat(allLMIs,zk,Q_id);
Mm = dec2mat(allLMIs,zk,M_id);

xnew = [svec(Qm); svec(Pm)];

% Perform linesearch

% Backtracking linesearch (Nocedal & Wright, p.42)
alpa=1;
rho_linesearch = .50;
if iter>11
    c_linesearch(iter) = 0.1;
else
    c_linesearch(iter) = 0.0001 %
end
% grad f_k is gk

% descent direction
pk = xnew - xold;
pkAll(iter) = norm(pk,1);

while not(funvalue2(xold + alpa*pk,Lambda(:,:,iter),cmult(iter),Tx,kappa) ...
      <= funvalue2(xold,Lambda(:,:,iter),cmult(iter),Tx,kappa)+ ...
      c_linesearch(iter)*alpa*(gk'*pk))
  alpa = rho_linesearch*alpa;
end
[F(iter), logDetInvAll(iter), lagrF(iter), penaltyF(iter) ] =
              funvalue2(xold + alpa*pk,Lambda(:,:,iter),cmult(iter),Tx,kappa);
alpaAll(iter)=alpa;

zk = zold + alpa*(zk-zold);
Qm = dec2mat(allLMIs,zk,Q_id);
Pm = dec2mat(allLMIs,zk,P_id);
Mm = dec2mat(allLMIs,zk,M_id); MmAll(:,:,iter)=Mm;
%--------plotting--------
% Constraint deviation in Frobenius norm
Phi(iter) = norm((Qm*Pm-I),inf);% Nadi (Fra Fares lig 24)
objfun(iter) = funvalue2(xold+alpa*pk,Lambda(:,:,iter),cmult(iter),Tx,kappa); %max(eig(Qm-inv(Pm)));
logdet(iter) = log(det(inv(Tx*Qm*Tx')));

normen(iter)=norm(Pm-Qm^-1,inf); % termination criteria
% Update penalty and multiplier
Lambda(:,:,iter+1) = Lambda(:,:,iter) + cmult(iter)*(Qm*Pm-I);
if (iter > 1)
    Phi_old = Phi(iter-1);
    norm_old= normen(iter-1);
end
if (Phi(iter) > mu*Phi_old) & norm(Pm-Qm^-1,inf)> mu*norm_old
                      %if the Fob norm does not improve (reduced) by itself,
    if cmult(iter)>10^7 % to prevent that cmult will be to large.(ill conditioning)
        rho=1.05         % will slow down the growth of cmult
    end
    cmult(iter+1) = rho*cmult(iter);
```

```
            disp('increasing penalty ')%increase penalty weight
        else
           cmult(iter+1) = cmult(iter);
        end
      %   if iter==10 % Augmented increase of penalty parameter \mu_k
      %       cmult(iter+1)=1000
      %   end

        Phi_old = Phi(iter);

        tol=1e-2;
      l=logDetInvAll(iter); lAll=logDetInvAll;
      %if tmin<0
      %debugging :
          %iter_k=iter_k+1;
          QmAll(:,:,iter)=Qm;
          PmAll(:,:,iter)=Pm;
          disp('norm(Pm-Qm^-1,inf)=');
          if iter>11 % for debugging
              normen(iter-10:iter)
          else
              normen
          end
          Phi_old %show the normen of frobenius
          %end
          if norm(Qm)==0
              disp('Norm Qm is zero!')
              break
          end
          %to measure the time it takes to calculate one iteration
          if iter==1
              IterTid=[toc];
          else
              IterTid=[IterTid;toc];
          end

          if norm(Pm-Qm^-1,inf) < 5e-2 & iter > 10
              %feasibility test
              setlmis([])
              %n is the number of decision variables used so far.
              [M_idtest,ntest,M_dectest]=lmivar(2,[N*dimu N*dimu]);
              [PSIvar_idtest,ntest,PSIvar_dectest]=lmivar(3,[zeros(N*dimu+mx,mx) [zeros(mx,N*dimu);M_dectest]]);
              %1st LMI, invariance
              lmiT = newlmi;
              %ERPC formel 1 2 element transposed
              lmiterm([-lmiT 1 1 0],Qm);
              lmiterm([-lmiT 1 2 PSIvar_idtest],1,Qm);
              lmiterm([-lmiT 1 2 0],Te*Qm);
              lmiterm([-lmiT 2 2 0],Qm);%Pm

              testLMI = getlmis;
              [tmin,xopt] = feasp(testLMI);
              if tmin<0
                  STOP = 1;
              else
                  iter_k=iter_k+1;
                  Qm_ferdig(:,:,iter_k)=Qm;
                  norm_ferdig(iter_k)=normen(iter);
                  iter_ferdig(iter_k)=iter;
              end
          end

          if not(STOP),
          clear testlmi;
          end
      %***************** End STOP criteria **********
      end if STOP==1
          disp(' End of program...........................')
      %     plotellipse(Px0,1,'b:');hold on % ERPC ellipse
      %     plotellipse(inv(Tx*Qm*Tx'),1,'m--'); % BMI ellipse

      end fig=9; tic;
      [N,B,phi,mx,dimu,umax,KK,PSIz,Mm2,Qz2,Tx2,logDetInv]=nrf_multi2([Mm]);
      TimeNrf2Qz2=toc

      fig=100 Tx2=zeros(2,length(Qz2)); Tx2(1,1)=1;Tx2(2,2)=1; Pmx =
      inv(Tx2*Qz2*Tx2'); plotellipse(Pmx,fig,'g'); hold on
      Tx2=zeros(2,length(Qz0)); Tx2(1,1)=1;Tx2(2,2)=1; Pmx =
      inv(Tx2*Qz0*Tx2'); plotellipse(Pmx,fig,'k:'); hold on
      legend('SSDMPC','ERPC')
```

C.1.2 SSDMPC for LDI

```
clear datetime1;
% SSDMPC for Robust Control
%---------INITIALISE-----
%solve prob with ERPC algorithm by Kouvaritakis
%initialises the following variables:
% N, horizon
% B, from system-model:  x_k+1=A*x_k+B*u_k.
% mx, length(A), dimension of system
% umax, upper input constraint
% phi, transition matrix x_k+1=phi*x_k (phi=A-B*K)
% KK, augmented feedback gain, u=KK*z
% PSIz, augmented transition matrix z_k+1=PSIz*z_k
% M block matrix in PSIz, treated as a general variable
% Pz, defines invariant ellipsoid z'*Pz*z < 1
% Qz=Pz^-1
% Tx, x=Tx*z
%*******************************
% Nadi Bar
% *******************************
M=[];%nrf generates standard M
[A,A1,A2,B,C,xk,Q,R,N,VLB,VUB,phi,K]=init3;
[N,B,phi1,phi2,mx,dimu,umax,KK,M0,Qz0,Tx]=nrf_multi3([])
ConvCount=0
%load('simulations/Heli.mat')
Pz0=inv(Qz0); cond(Qz0) warning off;
sprintf('N=%d, mx=%d, dimu=%d  CASE= 3  Confirm: (press key)',N, mx,dimu)
 pause

phi1=A1-B*K; phi2=A2-B*K; break;
%------------BASIS LMIs------------
%Define LMIs for Matlab's LMI toolbox
setlmis([])
%n is the number of decision variables used so far.
[Q_id,n,Q_dec]=lmivar(1,[N*dimu+mx 1]);
[P_id,n,P_dec]=lmivar(1,[N*dimu+mx 1]);
[M_id,n,M_dec]=lmivar(2,[N*dimu N*dimu]);

[PSIvar_id,n,PSIvar_dec]=lmivar(3,[zeros(N*dimu+mx,mx)
[zeros(mx,N*dimu);M_dec]]);
```

```
%constants on matrix form
Te=zeros(N*dimu+mx,N*dimu+mx); Te(1:mx,1:mx+dimu)=[phi B];
Te1=zeros(N*dimu+mx,N*dimu+mx); Te1(1:mx,1:mx+dimu)=[phi1 B];
Te2=zeros(N*dimu+mx,N*dimu+mx); Te2(1:mx,1:mx+dimu)=[phi2 B];

%1st LMI
lmi11 = newlmi; lmiterm([-lmi11 1 1 Q_id],1,1) lmiterm([-lmi11 1 2
PSIvar_id],1,1) lmiterm([-lmi11 1 2 0],Te1) lmiterm([-lmi11 2 2
P_id],1,1)

lmi12 = newlmi; lmiterm([-lmi12 1 1 Q_id],1,1) lmiterm([-lmi12 1 2
PSIvar_id],1,1) lmiterm([-lmi12 1 2 0],Te2) lmiterm([-lmi12 2 2
P_id],1,1)


%2.1 LMI pådrag 1
lmi3 = newlmi; lmiterm([-lmi3 1 1 0],umax(1)^2) lmiterm([-lmi3 1 1
Q_id],-KK(1,:),KK(1,:)')

if dimu>1
    %3 LMI pådrag 2
    lmi4 = newlmi;
    lmiterm([-lmi4 1 1 0],umax(2)^2)
    lmiterm([-lmi4 1 1 Q_id],-KK(2,:),KK(2,:)')
end

%4 LMI  ( M-I<0 )
lmi5 = newlmi; lmiterm([-lmi5 1 1 M_id],-1,1) lmiterm([-lmi5 1 1
0],1)


basisLMI = getlmis; n_basisLMI = decnbr(basisLMI);

% Check if the LMI part is feasible
[lmi_min,xbeg]=feasp(basisLMI); if lmi_min>0
  display('BMI infeasible since LMI part is infeasible');
  break
end
%   zk contains all LMI variables
% Store information from preliminary optimization - add 0 for dummy variable 't'
Qm=Qz0; Pm=Pz0; PSIm=PSIz0; %convert initial results to lmitoolbox custom notation
PSIvar=PSIz0; PSIvar(1:mx, 1:mx+dimu)=zeros(mx,mx+dimu);

zk=[mat2dec(basisLMI,Qz0,Pz0,M0,PSIvar);0];
%Other initial values

Tm = T(Pm); dimP=length(Pm);%N*dimu+nx
I = eye(dimP);
%Tuning

kappa = 1; % unessecery
cmult(1) = .51; %initial penalty weight
rho = 1.3; %rate at which penalty weight increases (the higher - the lower changes will be excepted)
mu = .996

norm_old= norm(Pz0-Qz0^-1,inf); Phi_old =
kappa*log(det(inv(Tx*Qm*Tx'))) + trace((Qm*Pm-I)'*(Qm*Pm-I));
optERPC = log(det(inv(Tx*Qz0*Tx')));
Lambda(:,:,1) = (Qm*Pm-I); %100*ones(dimP);
%Lambda(:,:,1) = zeros(dimP);
iter = 0;
iter_k=0; % for stop criteria
```

```
tminAll=0
%option=[accuracy, max # iter, feasb radius, stop after J accuracy iter, trace execution 1=off]
options=[0.01,150,10^12,30,1]; % for mincx

%---------------LOOP, finds pk-------
STOP=0; while (not(STOP))
  tic
  iter = iter + 1
  %tic;
  % Present value of "non-convex" variables - note different ordering than in LMI variable!
  xold = [svec(Qm);svec(Pm);];
  zold = zk; % z(k+1)=z(k)

  % --------Find Hessian and Gradient, Hk and gk
  %create perturbation matrix, pp, that transforms g and H from
  %being compatible with svec(Qx)=svec(TQzT') to being compatible
  %with svec(Qz)
teller=0; % generates pp
  for i=1:mx
      teller=teller+i;
  end
  posy=1;posx=1;
  pp=zeros(teller,length(xold))';
  for i=0:mx-1
      pp(posy:posy+mx-1-i,posx:posx+mx-1-i)=eye(mx-i);
      posy=posy+mx-i+dimu*N;
      posx=posx+mx-i;
  end
  pp=pp';
  clear posy posx teller % unessesary variables

  gQx=sgrad(Tx*Qm*Tx');
  gkte=pp'*gQx;
  HQx=sHessian(Tx*Qm*Tx');
  Hkte=pp'*HQx*pp;
  %Find g and H including rest of objective function
  Lk = Lambda(:,:,iter); % Lk is penalty parameter from 4.28
  gk = kappa*gkte + [Tm^2*( cmult(iter)*svec(Pm^2*Qm+Qm*Pm^2-2*Pm) + svec(Pm*Lk+Lk'*Pm) ); % Q
                     Tm^2*( cmult(iter)*svec(Qm^2*Pm+Pm*Qm^2-2*Qm) + svec(Qm*Lk'+Lk*Qm) )];% P

  Hk11 = cmult(iter)*skron(Pm^2,I,dimP);
%  Hk21 = skron(Lk,I,dimP) + cmult(iter)*(skron((Qm*Pm-I),I,dimP)+skron(Qm,Pm,dimP));
  Hk21 = skron(Lk,I,dimP) + cmult(iter)*(skron(Qm,Pm,dimP));
  Hk22 = cmult(iter)*skron(Qm^2,I,dimP);

  Hk = kappa*Hkte + blkdiag(Tm,Tm)*[Hk11,Hk21';Hk21,Hk22]*blkdiag(Tm,Tm);
  Hk=Hk*0.5;

  %  Modify Hk so that is is positive definite
  [V,D] =  eig(Hk);
  %eig(Hk)
  d = diag(D);

  d(find(d<1e-4))=10^-3;
  Hkm =  V*diag(d)*V';
  pow=2;

  while Hkm(min(find(imag(eig(Hkm))~=0))) & pow>=-4
      d = diag(D);
      d(find(d<1e-1))=10^-pow;
      Hkm =  V*diag(d)*V';
      pow=pow-1;
  end
  if cond(Hkm)>10^18 | Hkm(min(find(imag(eig(Hkm))~=0)))
      [Hkm,condition,Hkm_E]=mod_cholesky(Hk);
      disp('modified Cholesky')
      end
  condHkm(iter)=cond(Hkm);
%  eig(Hkm)
  %disp('cond(Hkm)=');cond(Hkm)
  %disp('cond(Hk)=');cond(Hk)

  %-------Add extra LMI (QP)--------
  setlmis(basisLMI);

  [tn_id, n_tn, tn_dec] = lmivar(1,[1 1]);
  [x_id,n_x,x_dec] = lmivar(3,[svec(Q_dec); svec(P_dec)]);

  minlmi = newlmi;
  lmiterm([-minlmi 1 1 tn_id],1,1);
  lmiterm([-minlmi 1 1 x_id],-gk',1);
  lmiterm([-minlmi 1 1 0],gk'*xold);% fordi x(k+1)=x(k)+P ==> P=x(k+1)-x(k)
  lmiterm([-minlmi 2 1 x_id],1,1);
  lmiterm([-minlmi 2 1 0],-xold);%P=x_id-xold
  invHkm = inv(Hkm);
  lmiterm([-minlmi 2 2 0],.5*(invHkm+invHkm'));
   allLMIs = getlmis;
   n_tn=decnbr(allLMIs); %
  %---------find pk---------
  c = zeros(decnbr(allLMIs),1); c(n_tn) = 1; %linear objective
  zk(n_tn) = 1e6;%initial values


  [lmi_min, zk] = mincx(allLMIs,c,options,zk);
  if zk==[]
      disp('mincx could not find t which is bigger than the Lagrange approx')
      disp('  while satisfying the LMI constraint. Check whether allLMIs is a feasible')
      disp('  set. if it is, then no solution is found.')
      disp('zk1=[] , aborting the program................')
      break
  end

  if (lmi_min<-1e12),
    disp('The algorithm has diverged!')
    break;
  end

  Pm = dec2mat(allLMIs,zk,P_id);
  Qm = dec2mat(allLMIs,zk,Q_id);
  Mm = dec2mat(allLMIs,zk,M_id);

  xnew = [svec(Qm); svec(Pm)];

  % Perform linesearch

  % Backtracking linesearch (Nocedal & Wright, p.42)
  alpa=1;
  rho_linesearch = .20;
  if iter>11
      c_linesearch = 0.13; %
  else
      c_linesearch = 0.0001 %
  end
  % grad f_k is gk
  % descent direction
  pk = xnew - xold;
  pkAll(iter) = norm(pk,1);

  while not(funvalue2(xold + alpa*pk,Lambda(:,:,iter),cmult(iter),Tx,kappa) ...
        <= funvalue2(xold,Lambda(:,:,iter),cmult(iter),Tx,kappa)+ ...
        c_linesearch*alpa*(gk'*pk))
    alpa = rho_linesearch*alpa;
  end
  [F(iter), logDetInvAll(iter), lagrF(iter), penaltyF(iter) ] =
                    funvalue2(xold + alpa*pk,Lambda(:,:,iter),cmult(iter),Tx,kappa);
  alpaAll(iter)=alpa;

  zk = zold + alpa*(zk-zold);
  Qm = dec2mat(allLMIs,zk,Q_id);
  Pm = dec2mat(allLMIs,zk,P_id);
  Mm = dec2mat(allLMIs,zk,M_id); MmAll(:,:,iter)=Mm;
  %--------plotting--------
  % Constraint deviation in Frobenius norm
  %Phi(iter) = trace((Pm*Qm-I)'*(Pm*Qm-I)); %Originally Stian
  Phi(iter) = norm((Qm*Pm-I),'fro');% Nadi (Fra Fares lig 24)
  objfun(iter) = funvalue2(xold+alpa*pk,Lambda(:,:,iter),cmult(iter),Tx,kappa); %max(eig(Qm-inv(Pm)));
  logdet(iter) = log(det(inv(Tx*Qm*Tx')));

  normen(iter)=norm(Pm-Qm^-1,inf); % termination criteria
  % Update penalty and multiplier
  Lambda(:,:,iter+1) = Lambda(:,:,iter) + cmult(iter)*(Qm*Pm-I);
  if (iter > 1)
      Phi_old = Phi(iter-1);
      norm_old= normen(iter-1);
  end
  if (Phi(iter) > mu*Phi_old) & norm(Pm-Qm^-1,inf)> mu*norm_old
                              %if the Fob norm does not improve (reduced) by itself,
      if cmult(iter)>10^7 % to prevent that cmult will be to large.(ill conditioning)
          rho=1.5         % will slow down the growth of cmult
      end
      cmult(iter+1) = rho*cmult(iter);
      disp('increasing penalty ')%increase penalty weight
  else
     cmult(iter+1) = cmult(iter);
  end
  if iter==10
      cmult(iter+1)=1000
  end
  Phi_old = Phi(iter);

  tol=1e-2;
l=logDetInvAll(iter); lAll=logDetInvAll;
%if tmin<0
%debugging :
    %iter_k=iter_k+1;
    QmAll(:,:,iter)=Qm;
    PmAll(:,:,iter)=Pm;
    disp('norm(Pm-Qm^-1,inf)=');
    if iter>11 % for debugging
        normen(iter-10:iter)
    else
        normen
    end
    Phi_old %show the normen of frobenius
    %end
    if norm(Qm)==0
```

```
            disp('Norm Qm is zero!')
            break
        end
        %to measure the time it takes to calculate one iteration
        if iter==1
            IterTid=[toc];
        else
            IterTid=[IterTid;toc];
        end

        if norm(Pm-Qm^-1,inf) < 5e-2 & iter > 10
            %feasibility test
            setlmis([])
            %n is the number of decision variables used so far.
            [M_idtest,ntest,M_dectest]=lmivar(2,[N*dimu N*dimu]);
            [PSIvar_idtest,ntest,PSIvar_dectest]=lmivar(3,[zeros(N*dimu+mx,mx) [zeros(mx,N*dimu);M_dectest]]);
            %1st LMI, invariance
            lmiT = newlmi;
            %ERPC formel 1 2 element transposed
            lmiterm([-lmiT 1 1 0],Qm);
            lmiterm([-lmiT 1 2 PSIvar_idtest],1,Qm);
            lmiterm([-lmiT 1 2 0],Te2*Qm);
            lmiterm([-lmiT 2 2 0],Qm);%Pm

            testLMI = getlmis;
            [tmin,xopt] = feasp(testLMI);
            if tmin<0
                STOP = 1;
            else
                iter_k=iter_k+1;
                Qm_ferdig(:,:,iter_k)=Qm;
                norm_ferdig(iter_k)=normen(iter);
                iter_ferdig(iter_k)=iter;
            end
        end

        if not(STOP),
        clear testlmi;
        end
%***************** End STOP criteria **********
%end
```

C.1.3 Nrf_multi2

```
function [N,B,phi,mx,dimu,umax,KK,PSIz,M,Qz,Tx]=nrf_multi2(M)
%nr.m (NRMPC) minimises costfunction Jinf=x'Qx+u'Ru, wrt predicted input, u
%for a constrained linear descrete SISO system, x_k+1=A*x_k+B*u_k. An LDI is
%solved offline, and online only a simple Newton-Rapshon solution is required.
%Initialise system with above mentioned parameters and:
%N, horizon
%umin,umax, constraints
%phi =A-B*K, transition matrix
%K, feedback, usually found by LQ (dlqr)
%mx, length(A), dimension of system
% modified Nadi Bar

addpath lmitool_multivar/ addpath MPClib_multivar/
%addpath C:\matlabR12\work\gilb
%clear all
closedLoop=1; PlotGraph=0;
```

```
computedCost=0;
 [A,B,C,xk,Q,R,N,umin,umax,phi,K,SS,mx,dimu]=init2;

%if M == -1, N=10,M=[];end
%[Ac,Bx,bb] = lin_constraintsOri(phi,K,N,umin,umax);
%plotROA(Bx,b,1)


iter=1;%max(12)%1+N);
%N=NN
%Generate Augmented system,
%z_k+1=PSIz*z_k and Tx defined by x=Tx*z
Tx=zeros(mx,mx+dimu*N); Tx([1:mx],[1:mx])=eye(mx);
%[PSIz,Tx,M]=genAug(phi,N,B,mx);
if ~isempty(M)
    PSIz = [[phi,B*eye(dimu,N*dimu)];[zeros(N*dimu,mx),M]];
else
    if N ~= 0
        %M=diag(ones(N-1,1),1)
        M=zeros(dimu*N,dimu*N);
%     M=[0 1 0 0 0
%        0 0 1 0 0
%        0 0 0 1 0
%        0 0 0 0 1
%        0 0 0 0 0];
        if dimu>1
            for ii=1:dimu:(N*dimu-2)
               M(ii:(ii+dimu-1),(ii+dimu):(ii+dimu+dimu-1))=eye(dimu,dimu);
            end
        else
            M=diag(ones(N-1,1),1) % case dimu==1
        end
        PSIz = [[phi,B*eye(dimu,N*dimu)];[zeros(N*dimu,mx),M]];
    else
        PSIz=phi;M=0
    end
end beta=1;
%Find optimal feasible invariant ellipsoid
t0 = cputime; sysdata = {PSIz,K,umax,Tx,N};
[Qz,info]=mdsolver('lmi_aug',sysdata);
if strcmp(info(1),'e')%error
      disp('-----ERROR: maxdet failed-----')
      eigM=eig(M)
  return
end
if strcmp(info(1),'i')%error
      disp('-----ERROR: Infeasible problem-----')
      eigM=eig(M)
  return
end

Qx = Tx*Qz*Tx'; Px = inv(Qx); Maxdettime = cputime-t0;
KK= [-K eye(dimu,dimu) zeros(dimu,dimu*(N-1))]; %used in bmi solver
%check Qz
%umax^2- [-K 1 0 0 0 0]*Qz*[-K 1 0 0 0 0]'
%eig([Qz Qz*PSIz'; PSIz*Qz Qz])
```

C.1.4 lmi_aug_uncert

```
%**********************************
%Nadi Bar
%***********************************************
 function [LME,LMI,empty] =lmi_aug(xdata,xvar)
% In order to generate Ellipsoide S and send it to mdsolver.
% Ex is actually Tx which is generated in genAug.m
% the rest of the cariables come from sysdata, which is defined in nrf.m
% S is Qz = Pz^-1

[Mz1,Mz2,K,umax,Ex,N] = deal(xdata{:}) ; % for ROBUST checking 1/3
%[Mz1,K,umax,Ex,N] = deal(xdata{:}) ; %vanlig
nz = length(Mz1); nx=length(Ex*Ex'); dimu=size(K,1); if N==0
    KK=-K
else
KK= [-K eye(dimu,dimu) zeros(dimu,dimu*(N-1))]; %KK* Zk
end Vek=ones(1,nz);
%Vek=[ 1 1 1 1 zeros(1,N*dimu)]; %for case with 4 variables

% Nadi, to define the norm bound to Qz
normax=1; % max norm_1 bound (tunable)
%Sx= Ex*S*Ex'

% LMEs  (symmetric)
% Objective (det_obj, lin_obj)
% LMIS a) invariance
%      b) feasibility
%      c) possitivity

%% INITIAL GUESS (only dimensions matter)
%P=Qz^-1, S=Qz
if(nargin == 1),
  % Sx=eye(length(MZ)-N)
   S   = eye(nz);

 %  LME = {S};
  LME = {eye(nz)};
   return;
end;

%% LME, LMI, and OBJECTIVE
S = deal(xvar{:});

%% LME's
% force P to be symmetric
LME = {S-S'};

%% OBJECTIVE
% min -logdet(det_obj) + lin_obj
det_obj = Ex*S*Ex'; %mx+mu*N
lin_obj = 0;

%% LMI's

% a) invariance
%lmi_inv = [S      S*Mz';
%           Mz*S      S];
lmi_inv1 = S- Mz1*S*Mz1';
lmi_inv2 = S- Mz2*S*Mz2'; %FOR ROBUST checking 2/3
tracemax=1;
% if tracemax==1
%     e=zeros(1,nz);
%     ii=0;
%     for i=1:nz:nz^2
%         e(i+ii)=1;
%      ii=ii+1;
%     end
%     QQ=zeros(nz^2,nz^2);
%     for i=1:nz:nz^2
%         QQ(i:i+nz-1,i:i+nz-1)=S;
%     end
% end
if tracemax==1
    enx=zeros(1,nz^2);
    enz=zeros(1,nz^2);
    ii=0;
    for i=1:nz:nz*nz
        if i<=nx*nz
            enx(i+ii)=1;
        else
            enz(i+ii)=1;
        end
     ii=ii+1;
    end

    QQ=zeros(nz^2,nz^2);
    for i=1:nz:nz^2
        QQ(i:i+nz-1,i:i+nz-1)=S;
    end
end tracemax=100; R=eye(nz,nz)*2;L=eye(nz,nz);
% b) feasibility
lmi_feas(1) = umax(1).^2 - KK(1,:)*S*KK(1,:)'; if dimu>1
    lmi_feas(2) = umax(2).^2 - KK(2,:)*S*KK(2,:)';
end

lmi_feas(3)= normax-Vek*S*Vek';
lmi_feas(4)= 0.01-(enx*QQ*enx')*(enz*QQ*enz');% num - stor*liten >0
lmi_feas(5)= -.1+(enz*QQ*enz')-(enx*QQ*enx');%  liten - stor > 0.1
% c) possitivity
lmi_pos = S;

% d) S- procedure
%lmi_Sproc=
%----------------------------------
%% Store OBJECTIVE and LMI's in 2nd return argument
LMI = {det_obj,lin_obj}; lmi_n=3; LMI{lmi_n} =
lmi_inv1;lmi_n=lmi_n+1;
LMI{lmi_n} = lmi_inv2;lmi_n=lmi_n+1; % for ROBUST checking 3/3
LMI{lmi_n} = lmi_feas(1); lmi_n=lmi_n+1;%pådrag u_1
if dimu>1
    LMI{lmi_n} = lmi_feas(2);lmi_n=lmi_n+1; % pådrag u_2
end

if nx>2
%   LMI{lmi_n}=lmi_feas(3); lmi_n=lmi_n+1;
   %LMI{lmi_n}=lmi_feas(5);lmi_n=lmi_n+1;
%  LMI{lmi_n}=lmi_feas(4);lmi_n=lmi_n+1;% possible not nessesary.
   LMI{lmi_n} = lmi_pos; lmi_n=lmi_n+1;
else
    %LMI{lmi_n}=lmi_feas(3);    lmi_n=lmi_n+1;
    LMI{lmi_n} = lmi_pos;    lmi_n=lmi_n+1;
end

% NB
% *** must have LMI{1} = det_obj, LMI{2} = lin_obj;
% *** define empty = [] for compatibility with lmiformat.m
empty =  [];
```

C.1.5 Nrf_multi3

```
function [N,B,phi1,phi2,mx,dimu,umax,KK,M,Qz,Tx]=nrf_multi3(M)
addpath lmitool_multivar/ addpath MPClib_multivar/
%addpath C:\matlabR12\work\gilb
[A,A1,A2,B,C,xk,Q,R,N,umin,umax,phi,K,SS,mx,dimu]=init3;

phi1=A1-B*K; phi2=A2-B*K;

iter=1; Tx=zeros(mx,mx+dimu*N); Tx([1:mx],[1:mx])=eye(mx); if
~isempty(M)
    PSIz1 = [[phi1,B*eye(dimu,N*dimu)];[zeros(N*dimu,mx),M]];
    PSIz2 = [[phi2,B*eye(dimu,N*dimu)];[zeros(N*dimu,mx),M]];
else
    if N ~= 0
        %M=diag(ones(N-1,1),1)
        M=zeros(dimu*N,dimu*N);
%     M=[0 1 0 0 0
%        0 0 1 0 0
%        0 0 0 1 0
%        0 0 0 0 1
%        0 0 0 0 0];
        if dimu>1
            for ii=1:dimu:(N*dimu-2)
               M(ii:(ii+dimu-1),(ii+dimu):(ii+dimu+dimu-1))=eye(dimu,dimu);
            end
        else
            M=diag(ones(N-1,1),1) % case dimu==1
        end
        PSIz = [[phi,B*eye(dimu,N*dimu)];[zeros(N*dimu,mx),M]];
        PSIz1 = [[phi1,B*eye(dimu,N*dimu)];[zeros(N*dimu,mx),M]];
        PSIz2 = [[phi2,B*eye(dimu,N*dimu)];[zeros(N*dimu,mx),M]];
    else
        PSIz=phi;M=0
    end
end
%Find optimal feasible invariant ellipsoid
t0 = cputime; sysdata = {PSIz1,PSIz2,K,umax,Tx,N};
[Qz,info]=mdsolver('lmi_aug_uncert',sysdata);
if strcmp(info(1),'e')%error
      disp('-----ERROR: maxdet failed-----')
      eigM=eig(M)
  return
end
if strcmp(info(1),'i')%error
      disp('-----ERROR: Infeasible problem-----')
      eigM=eig(M)
  return
end

Qx = Tx*Qz*Tx'; Px = inv(Qx); Maxdettime = cputime-t0;
KK= [-K eye(dimu,dimu) zeros(dimu,dimu*(N-1))]; %used in bmi solver
```

C.1.6 init2

```
%**************************************************
%nadi Bar
%***********************************************
function  [A,B,C,xk,Q,R,N,VLB,VUB,phi,K,SS,mx,dimu]=init
%Initialises the following parameters:
%Chosen:
%A,B, system-model  (x_k+1=A*x_k+B*u_k.)
%xk, state-vector
%Q,R, weighting matrixes (Jinf=x'Qx+u'Ru,)
%N, horizon
%VLB,VUB, constraints
switch 4
    case 1 % Nadi simulering SSDMPC (from QP anyway).
        xk=[ .1 .1 ]' %start value
        %xk=[-0.1  0.1 ]';%for N=0
        A=[1 0.1;0 1]; % marginal stable system
        B=[0 0.0787 ]';
        C=[1 0];
        %Q=C*C';
        Q=diag([1 1]);
        R=1
        N=5
        VUB=1;
        VLB=-VUB;
        D=[0];
        [K,SS]=dlqr(A,B,Q,R)
        %[K,SS]=dlqry(A,B,C,D,Q,R);
        %Q=C'*C;
    case 2  % ;Multi variables
         xk=[3 2]'; %start value
        A=[1 0.1;0 1]; % marginal stable system
        B=[0.2 0.1;1.1 0.1] % mx x mu
        C=[1 0]; % only one output y1
        Q=diag([1 1]);
        R=[.1 0;0 .1];% mu x mu
        N=2;
        VUB=[1;1];
        VLB=-VUB;
        D=[0 0];
        [K,SS]=dlqr(A,B,Q,R);
        Q=C'*C;
    case 3  % Helilab Tor Arne Johanssen
        xk=[-2 1 1 -1 1 1]'; %start value
        A=[1 0 0.01 0 0 0; 0 1 0 0.01 0 0;0 0 1 0 0 0 ;
            0 0 0 1 0 0;0.01 0 0 0 1 0;0 0.01 0 0 0 1]; % marginal stable system
        B=[0 0; 0.0001 -0.0001; 0.0019 0.0019;0.0132 -0.0132; 0 0;0 0];% mx x mu
        C=[1 0 0 0 0 0]; % only one output y1
        Q=diag([10,100,10,10,400,200]);
        R=[1 0;0 1];% mu x mu
        N=3;
        VUB=[pi/2;pi/2.5];
        VLB=-VUB;
        D=[0 0];
        [K,SS]=dlqr(A,B,Q,R);

    case 4 %Helikopter lab from Optimalisation and control 6 states
        initF;
        h=0.01;
        xk=[-.51 .1 .52 -.51 .41 .21]';
        %x =[lambda r p p_dot e e_dot]'
%         A1=[1 h 0 0 0 0; 0 1 -h*K_2 0 0 0;0 0 1 h 0 0 ;
%            0 0 -h*K_1*K_pp 1-h*K_1*K_pd 0 0;0 0 0 0 1 h;
%            0 0 0 0 -K_3*K_ed*h 1-K_3*K_ep*h]; % marginal stable system
        A=[0 1 0 0 0 0; 0 0 -K_2 0 0 0;0 0 0 1 0 0 ;
            0 0 -K_1*K_pp -K_1*K_pd 0 0;0 0 0 0 0 1;
            0 0 0 0 -K_3*K_ed -K_3*K_ep]; % marginal stable system
        A=eye(6,6)+A*h;
        B=[0 0;0 0; 0 0;K_1*K_pp 0; 0 0;0 K_3*K_ep];% mx x mu
        B=B*h;
        C=[1 0 0 0 0 0]; % only one output y1
        Q=diag([1,1,1,1,1,1])
        R=[1 0;0 1]% mu x mu
        N=2
        VUB=[20*pi/180;20*pi/180];  % u= [ P_c  e_c]'
        VLB=-VUB;
        D=[0 0];
        [K,SS]=dlqr(A,B,Q,R);

    case 5 % Helicopter lab with 4 states, 2 controllers.
        initF;
        h=0.1;
        xk=[0.9 0.5 0.7 0.3 ]';
        % X=[ p ; p_dot  ; e ; e_dot ]
        A=[0 1 0 0; -K_1*K_pp -K_1*K_pd 0 0;0 0 0 1;0 0 -K_3*K_ed -K_3*K_ep]; % marginal stable system
        A=eye(4,4)+h*A;
        B=[0 0;K_1*K_pp 0; 0 0; 0 K_3*K_ep]; % mx x mu
        B=B*h
        C=[1 0 0 0]; % only one output y1
        Q=diag([1,1,1,1]);
        R=[0.01 0;0 .01];% mu x mu
        N=2;
        VUB=[2;2];
        VLB=-VUB;
        D=[0 0];
        [K,SS]=dlqr(A,B,Q,R);
    case 6 % ship model Roger Sjetne
        h=0.01;
        A=10/17.5*[0 1 0;
                   0 -0.7072 -0.286;
                   0 -4.1078 -2.6619];
        A=eye(3,3)+h*A;
        B=10/17.5*[0 0;
                    -0.2081 -0.1;
                   -1.52 0.5] ;
        B=B*h
        C=[1 0 0];
        xk=[-0.13 .1 0.14]';
        Q=diag([1,1,1]);
        R=[1 0;0 1]; % mu x mu
        N=2;
        VUB=[1;1];
        VLB=-VUB;
        D=[0 0];
        [K,SS]=dlqr(A,B,Q,R);
    case 7 % roger Sjetnes båt unstable
        h=0.01;
        leng=31.35;
        leng=17.5;
        speed=70;

        A=speed/leng*[0 1 0;
                   0 -0.7072 -0.286;
                   0 -4.1078 -2.6619];
        A=eye(3,3)+h*A;
        B=speed/leng*[0 0;
                    -1.1081 1.1081;
                   -.152 -.152] ;
        B=B*h
       %B=B*h
        C=[1 0 0;
            0 1 0;
            0 0 1];
        xk=[2.2 -1.3 0.5]';
        Q=diag([1,1,1]);
        R=[.1 0;0 .1];% mu x mu
        N=2;
        VUB=[1;1];
        VLB=-VUB;
        D=[zeros(3,2)];
        [K,SS]=dlqr(A,B,Q,R);
    case 8 % Heli labb 3 states state x3 is unstable. sys is marginal stable
             % with saturation of u for stat point to far from 0
        initF;
        h=0.01;
        xk=[-2 1. -1]';
        % x=[ r_dot ; P ; P_dot ]
        A=[0 -K_2 0;0 0 1; 0 -K_1*K_pp -K_1*K_pd ];
        A=eye(3,3)+h*A;
        % B= [ P_c ; P_e ]
        B=[0; 0 ;K_1*K_pp ]
        B=B*h;
        C=[1 0 0]; % only one output y1
        Q=diag([1,10,10]);
        R=[1];% mu x mu
        N=2;
        VUB=[5];
        VLB=-VUB;
        D=[0 0];
        [K,SS]=dlqr(A,B,Q,R);

     case 9 %Helikopter lab from Optimalisation and control 5 states
        initF;
        h=0.1;
        xk=[1 2 1 1 0.1]';
        %x =[ r p p_dot e e_dot]'
        A=[ 1 -h*K_2 0 0 0; 0 1 h 0 0 ;
             0 -h*K_1*K_pp 1-h*K_1*K_pd 0 0; 0 0 0 1 h;
             0 0 0 -K_3*K_ed*h 1-K_3*K_ep*h]; % marginal stable system
        B=[0 0; 0 0;K_1*K_pp*h 0; 0 0;0 K_3*K_ep*h] % mx x mu
        C=[1 0 0 0 0]; % only one output y1
        Q=diag([1,1,1,1,1]);
        R=[.1 0;0 .1]% mu x mu
        N=2;
        VUB=[pi/2;pi/2];
        VLB=-VUB;
        D=[0 0];
        [K,SS]=dlqr(A,B,Q,R);
     case 10 % helikopter labb, nadi 6 states pådrag spenning
         % continuos system
        h=0.1;
        initF;
        % x=[ lambda  r  p  p_dot  e  e_dot  ]'
        xk=[2 1 .41 .1 0.1 0.5]';
        k1=K_f*l_h/J_p;
        k2=K_p*l_a/J_t;
        k3=K_f*l_a/J_e;
```

```
%
       A=[ 0 0 1 0 0 0 % e_dot
           0 0 0 1 0 0 % p_dot
           0 0 0 0 0 0 % e_dotdot
           0 0 0 0 0 0 % P_dotdot
           0 -k2 0 0 0 0 % r_dot=  -k2 p
           0 0 0 0 1 0]% lambda_dot=r
       A=eye(6,6)+h*A;

       B=[0 0 % e
          0 0 % p
          k3 k3 % e_dot
          k1 -k1% p_dot
          0 0% r
          0 0] % lambda

       B=h*B;

       C=[ 1 0 0 0 0 0];
       Q=diag([1 1 1 1 1 1]);
       R=diag([.01 .01]);
       N=1
       VUB=[5;5]; % Vf ; Vb
        VLB=-VUB;
        D=[0 0];
        K=dlqr(A,B,Q,R)
    case 11 % paper machine, page 46 MPC book
%         h=2;
%         A=[ -1.93 0 0 0
%             0.394 -0.426 0 0
%             0 0 -0.63 0
%             0.82 -0.784 0.413 -0.426];
%         A=eye(4,4)+h*A;
%         B=[1.274 1.274
%             0 0
%             1.34 -0.65
%             0 0];
%         B=B*h
        %discrete model
        A=[0.0211 0 0 0
            0.1062 0.4266 0 0
            0 0 0.2837 0
            0.1012 -0.6688 0.2893 0.4266]
        B=[0.6462 0.6462
            0.28 0.28
            1.5237 -0.7391
            0.9929 0.1507]

        C=[1 0 0 0];
        xk=[0 0 0 0]';
        Q=diag([1,0,1,1]);
        R=[.1 0;0 .1]; % mu x mu
        N=2;
        VUB=[10;10];
        VLB=-VUB;
        D=[0 0];
        [K,SS]=dlqr(A,B,Q,R);
    case 12 % B-26 aircraft model, optimal control book page 96
        h=0.01;
        A=[ 0 1 0 0
            0 -1 -73.14 3.18
            0.086 0 -0.11 -1
            0.0086 0.086 8.95 -0.49];
        A=eye(4,4)+h*A;
        B=[0 1
            0 -3.91
            0.035 0
            -2.53 31];
        B=B*h
        C=[1 0 0 0];
        xk=[30*pi/180 0.6 .20 0.4]';
        Q=diag([1,1,1,1]);
        R=[.1 0;0 .1]; % mu x mu
        N=2;
        VUB=[1;1];% rudder deflection, Aileron deflection
        VLB=-VUB;
        D=[0 0];
        [K,SS]=dlqr(A,B,Q,R);
    case 13 % aircraft, same like case 12 but with 3 states
        h=0.1;
        A=[ -1 -73.14 3.18
             0 -0.11 -1
             0.086 8.95 -0.49];
        A=eye(3,3)+h*A;
        B=[ 0 -3.91
            0.035 0
            -2.53 31];
        B=B*h
        C=[1 0 0
            0 1 0
            0 0 1];
        xk=[.1 .1 .1]';
        Q=diag([1,1,1]);
        R=[.1 0;0 .1]; % mu x mu
        N=2;
        VUB=[60*pi/180;60*pi/180];% rudder deflection, Aileron deflection
        VLB=-VUB;
        D=[0 0];
        [K,SS]=dlqr(A,B,Q,R);
    case 14 % F404 Aircraft Engine
        %
        % u = [ W_f (pph)fuel injection   A_8 (sq in) areal nozzle ]
        % x = [ N_2 (rpm)  N_25 (rpm)  T_45 (deg Fah) ]
        % y = [ N_2 (rpm)              T_45 (deg Fah) ]
        % discrete model
        A=[-0.9305 0 0.1107
            0.0077 0.9802 -0.0173
            0.0142 0 0.8593]
        B=[0.0217 0.2510
            0.0192 -0.0051
            0.0247 -0.0030]
        C=[1 0 0
            0 1 0];
        xk=[.10 .10 .10]';
        Q=diag([10 10 10])
        R=diag([.050 .050])
        N=5;
        VUB=[1;1];
        VLB=-VUB;
        D=[0 0];
        [K,SS]=dlqr(A,B,Q,R);
    case 15 % Citation aircraft model, MPC book
%        h=0.2;
        Ts=0.01;
%         A=[-1.2822 0 0.98 0
%             0 0 1 0
%            -5.4293 0 -1.8366 0
%            -128.2 128.2 0 0];
        A=[-1.2822 0 0.98 0
           0 0 1 0
           -5.4293 0 -1.8366 0
           -128.2 128.2 0 0];

%        A=eye(4,4)+A*h;
%        B=[-.3 0; 0 0; -25 -12; 0 0];
        B=[-.3; 0; -17;0];
%        B=B*h;
        C=[0 1 0 0
            0 0 0 1
            -128.2 128.2 0 0 ];
        D=[0];
        sys=ss(A,B,C,D);
        [sysb,T] = ssbal(sys);
%         [A,B,C,D] = c2dm (sysb.a,sysb.b,sysb.c,sysb.d,0.01,'zoh')
       [A,B,C,D] = c2dm (sys.a,sys.b,sys.c,sys.d,0.01,'zoh')
        xk=[0 0.1 0.1 0]';
        Q=diag([1 1 1 1])
        R=diag([20])
        N=2;
        VUB=[60*pi/180];
        VLB=-VUB;
        [K,SS]=dlqr(A,B,Q,R)
    case 16
        DSRVnadi;
        h=0.001;
        C=eye(4,4); D=zeros(4,2)
        sys=ss(Ac,Bc,C,D);
        [A,B,C,D] = c2dm(sys.a,sys.b,sys.c,sys.d,0.01,'zoh')
        xk=[0 .10 .1 0]';
        Q=diag([100 1 1 1])
        R=diag([1 1])
        N=2;
        VUB=[30*pi/180;30*pi/180];
        VLB=-VUB;
        D=[0 0];
        [K,SS]=dlqr(A,B,Q,R);
    case 17 % DC motor
        % DC motor maxon motors, A 2520-853
        % x=[ w_m i_a ]
        Ts=0.001;
        J=0.01;
        b=0.1;
        K=0.01;
        R=1;
        L=0.5;
        Ac=[-b/J   K/J
           -K/L   -R/L];
        Bc=[0
           1/L];
        C=[1   0];
        D=0;
        sys=ss(Ac,Bc,C,D);
        [A,B,C,D] = c2dm(sys.a,sys.b,sys.c,sys.d,Ts,'zoh')
        xk=[0.1 0.1]';
        N=2;
        VUB=15;
        VLB=-VUB;
        Q=diag([10 10]);
        R=.01;
        [K,SS]=dlqr(A,B,Q,R)
    case 18 % Pitch controller of plane from www
        Ac = [-0.313 56.7 0;
             -0.0139 -0.426 0;
              0 56.7 0];
        Bc = [ 0.232;
            0.0203;
            0];
        Cc=[0 0 1];
        Dc=[0];
        Ts=1/100;
        [A,B,C,D] = c2dm (Ac,Bc,Cc,Dc,Ts,'zoh')
        xk=[0.1 0.1 0.2]';
        N=45
        VUB=pi/3;
        VLB=-VUB;
        Q=diag([1 100 1]);
        R=1;
        [K,SS]=dlqr(A,B,Q,R)
    case 19 % plane slotine 224
        Ac=[0 1 0 0;-4 -4 0 0; 0 0 0 1;6 0 0 0]
        Bc=[0 3 0 -1]'
        Ts=1/100;
        Cc=[0 0 1 0];
        Dc=[0];
        Ts=1/100;
        [A,B,C,D] = c2dm (Ac,Bc,Cc,Dc,Ts,'zoh')
        xk=[0.1 0.1 0 0.1]';
        N=2;
        VUB=pi/3;
        VLB=-VUB;
        Q=diag([100 100 10 100]);
        R=.1;
        [K,SS]=dlqr(A,B,Q,R)
   end
mx=length(A); dimu=size(B,2);
%Finding optimal K

%[K,SS]=dlqry(A,B,C,D,1,1);
phi=A-B*K

%Testing Controllability and observability
co=ctrb(A,B); if length(A)-rank(co)>0

  disp('WARNING: NOT CONTROLLABLE')
  break
end
```

C.1.7 init3

```
function  [A,A1,A2,B,C,xk,Q,R,N,VLB,VUB,phi,K,SS,mx,dimu]=init3
%KUN FOR ROBUSTHET!!

switch 4
    case 1 % Nadi simulering SSDMPC (from QP anyway).
        xk=[ .1 .1 ]' %start value
        %xk=[-0.1  0.1 ]';%for N=0
        A=[1 0.1;0 1]; % marginal stable system
        B=[0 0.0787 ]';
        C=[1 0];
        %Q=C*C';
        Q=diag([1 1]);
        R=.1
        N=20
        VUB=1;
        VLB=-VUB;
        D=[0];
        [K,SS]=dlqr(A,B,Q,R)
        %[K,SS]=dlqry(A,B,C,D,Q,R);
        %Q=C'*C;
    case 2  % ;Multi variables
         xk=[3 2]'; %start value
        A=[1 0.1;0 1]; % marginal stable system
        B=[0.2 0.1;1.1 0.1] % mx x mu
        C=[1 0]; % only one output y1
        Q=diag([1 1]);
        R=[.1 0;0 .1];% mu x mu
        N=2;
        VUB=[1;1];
        VLB=-VUB;
        D=[0 0];
        [K,SS]=dlqr(A,B,Q,R);
        Q=C'*C;

    case 4 % Helilab Roobust. K3 has now sin(p)
        initF;
        h=0.01;
        delta=0.07; % Uncertainty
        xk=[-.51 .1 .52 -.51 .41 .21]';
        %x =[lambda r p p_dot e e_dot]'
        %A=[1 h 0 0 0 0; 0 1 -h*K_2 0 0 0;0 0 1 h 0 0 ;
        %    0 0 -h*K_1*K_pp 1-h*K_1*K_pd 0 0;0 0 0 0 1 h;
        %    0 0 0 0 -K_3*K_ed*h 1-K_3*K_ep*h]; % marginal stable system
        A=[0 1 0 0 0 0; 0 0 K_2 0 0 0;0 0 0 1 0 0 ;
            0 0 K_1*K_pp K_1*K_pd 0 0;0 0 0 0 0 1;
            0 0 0 0 K_3*K_ed K_3*K_ep]; % marginal stable system
        A1=[0 1 0 0 0 0; 0 0 K_2+delta 0 0 0;0 0 0 1 0 0 ;
            0 0 K_1*K_pp K_1*K_pd 0 0;0 0 0 0 0 1;
            0 0 0 0 K_3*K_ed K_3*K_ep]; % marginal stable system
        A2=[0 1 0 0 0 0; 0 0 K_2-delta 0 0 0;0 0 0 1 0 0 ;
            0 0 K_1*K_pp K_1*K_pd 0 0;0 0 0 0 0 1;
            0 0 0 0 K_3*K_ed K_3*K_ep]; % marginal stable system
        A=eye(6,6)-A*h;
        A1=eye(6,6)-A1*h;
        A2=eye(6,6)-A2*h;
        B=[0 0;0 0; 0 0;K_1*K_pp 0; 0 0;0 K_3*K_ep];% mx x mu
        B=B*h;
        C=[1 0 0 0 0 0]; % only one output y1
        Q=diag([1,1,1000,10,1,10])
        R=[80 0;0 80]% mu x mu
        N=2;
        VUB=[20*pi/180;20*pi/180];  % u= [ P_c  e_c]'
        VLB=-VUB;
        D=[0 0];
        [K,SS]=dlqr(A,B,Q,R);

    case 18 % Pitch controller of plane from www
        Ac = [-0.313 56.7 0;
             -0.0139 -0.426 0;
              0 56.7 0];
        Bc = [ 0.232;
            0.0203;
            0];
        Ac1 = [-0.313 56.7 1;
             -0.0139 -0.426 0;
              0 56.7 0];
        Bc1 = [ 0.232;
            0.0203+0.1;
            0];
        Ac2 = [-0.313 56.7 -1;
             -0.0139 -0.426 0;
              0 56.7 0];
        Bc2 = [ 0.232;
            0.0203-0.1;
            0];
        Cc=[0 0 1];
        Dc=[0];
        Ts=1/100;
        [A,B,C,D] = c2dm (Ac,Bc,Cc,Dc,Ts,'zoh')
        [A11,B11,C1,D1] = c2dm (Ac1,Bc1,Cc,Dc,Ts,'zoh')
        [A12,B12,C1,D1] = c2dm (Ac2,Bc1,Cc,Dc,Ts,'zoh')
        [A21,B21,C1,D1] = c2dm (Ac1,Bc2,Cc,Dc,Ts,'zoh')
        [A22,B22,C1,D1] = c2dm (Ac2,Bc2,Cc,Dc,Ts,'zoh')
        xk=[0.1 0.1 0.2]';
        N=2;
        VUB=pi/3;
        VLB=-VUB;
        Q=diag([1 100 1]);
        R=1;
        [K,SS]=dlqr(A,B,Q,R)

   end
mx=length(A); dimu=size(B,2);
%Finding optimal K

%[K,SS]=dlqry(A,B,C,D,1,1);
phi=A-B*K;

%Testing Controllability and observability
co=ctrb(A,B); if length(A)-rank(co)>0

  disp('WARNING: NOT CONTROLLABLE')
  break
end
```

C.1.8 DSRVnadi.m

```
% [xdot, U] = DSRV(x,ui,U0) returns returns the speed U in m/s (optionally) and the
% time derivative of the state vector x = [ w q x z theta ]' for a
% deep submergence rescue vehicle (DSRV) L = 5.0 m, where
%
% w     = heave velocity                (m/s)
% q     = pitch velocity                (rad/s)
% x     = x-position                    (m)
% z     = z-position, positive downwards (m)
% theta = pitch angle                   (rad)
%
% The inputs are:
% ui     = delta (rad), where delta is the stern plane
% U0     = nominal speed (optionally). Default value is U0 = 4.11 m/s = 13.5 knots.
%
% Reference : A.J. Healey (1992). Marine Vehicle Dynamics Lecture Notes and
%             Problem Sets, Naval Postgraduate School (NPS), Monterey, CA.
```

```
delta_max = 30;             % max stern plane angle (deg)
U0 = 2.11;                   % U0 = 4.11 m/s = 13.5 ft/s
W0 = 0;

%U = sqrt( U0^2 + (W0+x(1))^2 ); Ulineart
U = sqrt(U0^2+0.5^2) ; % w= 0.5 m/sec downward
% Normalization variables
L = 5.0; Iy  =  0.001925; m   =  0.036391;

Mqdot  = -0.001573; Zqdot  = -0.000130; Mwdot  = -0.000146; Zwdot
= -0.031545; Mq     = -0.01131;  Zq     = -0.017455; Mw     =
0.011175; Zw     = -0.043938; Mtheta = -0.156276/U^2; Mdelta =
-0.012797; Zdelta = 0.027695;

% Masses and moments of inertia
m11 = m-Zwdot; m12 = -Zqdot; m22 = Iy-Mqdot; m21 = -Mwdot;

detM = (m11*m22-m12*m21);

a11=m22*Zq/detM-m12*Mq/detM; a12=m22*Zw/detM-m12*Mw/detM;
a14=-m12*Mtheta/detM; a21=-m21*Zq/detM + m11*Mq/detM;
a22=-m21*Zw/detM + m11*Mw/detM; a24=m11*Mtheta/detM;

b11=m22*Zdelta/detM-m12*Mdelta/detM; b21=-m21*Zdelta/detM +
m11*Mdelta/detM;

Ac=[a11 a12 0 a14
    a21 a22 0 a24
    0    0  0  -U0
    0    1  0   0];

% Ac=[a11 a12  a14
%     a21 a22  a24
%     0    1     0];
Bc=[b11 0.01
    b21 -1
    0  0
    0  0 ];
% Bc=[b11 b21 0 0]';
```

C.1.9 mod_cholesky.m

```
function [Hessian_m,condition,E]=mod_cholesky(a)
% Modified Cholesky algorithm,
% alg. 6.5
% Nadi Bar
% Feb 2003

%a=diag([-2 12 4]);
%a=[4 2 1; 2 6 3 ;1 3 -0.004] % eksempel 6.2
sam=0;
% to design the parameters
gamma=0;epsilon=0; for stj=1:length(a)
    for sti=1:length(a)
        if sti==stj
            gamma=max(gamma,abs(a(sti,stj)));
        else
            epsilon=max(epsilon,abs(a(sti,stj)));
        end
    end
end maskin=10^-3;%
delta = maskin*max(gamma+epsilon,1); %
```

```
beta1 = max(gamma,epsilon/sqrt(length(a)^2-1));
%minimizing the norm of the E  ||E,inf||
beta=sqrt(max(beta1,maskin)); % Gill Murray, Wright[104]
n=length(a); c=zeros(size(a)); l=zeros(size(a)); d=zeros(3,1);

for k = 1:n %Initialization
   c(k,k)= a(k,k);
   d(k)= a(k,k);
end

for j = 1:n
   for s=1:(j-1)
      l(j,s)=c(j,s)/d(s);
   end
   for i=j+1:n
       for s=1:(j-1)
           sam=sam+l(j,s)*c(i,s);
       end
       c(i,j)=a(i,j)-sam;
       sam=0;
   end
   phi(j)=0;
   if j<n
       phi(j)=max(abs(c(j+1:n,j)));
   end
   d(j)=max(abs(c(j,j)),(phi(j)/beta)^2);
   d(j)=max(d(j),delta);
   if j<n
       for i=j+1:n
           c(i,i)=c(i,i)-c(i,j)^2/d(j);
       end
   end
end

% the E matrix that should correct the Hessian:
for i=1:length(a)
    E(i,i)=d(i)-c(i,i);
end %for

Hessian_m=E+a; % the answer H_m
condition=cond(Hessian_m);
```

C.1.10 svec.m

```
function v = svec(X)
% maps the set of symmetric matrices S^n into R^n(n+1)/2

[r,c] = size(X);

if (norm(X-X')>1e-3),
   error('input to svec must be symmetric');
end

if not(r==c),
   error('input to svec must be square');
end

v=[]; for i=1:r,
   v = [v, X(i,i:r)];
end v=v';
```

C.1.11 funvalue2.m

```
function [F, LogDetInv, lagrF, penaltyF ] =
funvalue2(x,Lambda,cmult,Tx,kappa);
% Evaluates objective function, used in linesearch.
% Nadi Bar

Qvec=x(1:length(x)/2); Pvec=x(length(x)/2+1:end);

Qm=invsvec(Qvec); Pm=invsvec(Pvec);

LogDetInv = kappa*log(det(inv(Tx*Qm*Tx'))) ; % del I av Lagrangian
lagrF = trace(Lambda*(Qm*Pm-eye(size(Qm))));% del II av lagragian
penaltyF = (cmult/2)*trace((Qm*Pm-eye(size(Qm)))'*(Qm*Pm-eye(size(Qm)))); % del III av Lagragian
F = LogDetInv + lagrF + penaltyF; % hele 4.17 L funksjon
```

C.1.12 sgrad.m

```
%Returns the "gradient" of log det (A^-1),
%found by the operator d/dsvec(A)

function [sgrad]=sgrad(A)
%grad=(-inv(A));
sgrad=svec(-inv(A));
```

C.1.13 sHessian.m

```
function [sHessianK]=sHessian(A)
%Returns the hessian of log det (A^-1),
%when A as input

if length(A)==2 x=A(1,1); y=A(2,1); z=A(2,2); sgrad(A^-1);
te=svec(A^-1)*svec(A^-1)'; SHessianM=te+[0 0 -1; 0 1 0;-1 0
0]/(x*z-y*y); end

sHessianK=T(A)*(skron(inv(A),inv(A),length(A)))*T(A);
```

C.1.14 T.m

```
function t = T(X)
% returns a diagonal matrix with length vec(X), and with 1's corresponding to the diagonal entries
% of X, and sqrt(2) to the entries strictly above the diagonal

[r,c] = size(X);

if (norm(X-X')>sqrt(10^-10)),
   error('input to t must be symmetric');
end

if not(r==c),
   error('input to t must be square');
end

t=[];
for i=1:r, % Exersize: do this without loops!
   t = [t, 1, ones(size(X(i,i+1:r)))*sqrt(2)];
end
```

```
t=diag(t);
```

C.1.15 invsvec.m

```
function X = invsvec(v);

r = max(size(v));

if (min(size(v))~=1),
  error('input must be vector');
end

%size of output matrix, n by n,
%Formula found from n^2+n-2*r=0
n = (-1+sqrt(1+8*r))/2;

if (not(abs(n-round(n))<1e-8)),
  error('input format not correct');
end

X=zeros(n);

%if input vector is column vector, flip to row vector
[r,c]=size(v); if r >= c v = v'; end

for i=1:n,
  tst = v(1:n-i+1);
  X(i,i:n) = tst;
  X(i:n,i) = tst';
  v = v(n-i+2:end);
end
```

C.1.16 initF.m

```
%Data file needed to initialize the helicopter model parameters.
% to be used with cases 4 and 5.

m_w = 1.87 ;% motvekt
m_h = 1.15; % helikopter
l_a = 0.66; %lengde fra pivotpunkt til helikopterkropp
l_h = 0.177; % lengde fra midten på helikopterkroppen til motor
J_e = 2 * m_h * l_a *l_a; % treghetsmoment om elevasjonsaksen
J_p = 2 * ( m_h/2 * l_h * l_h); % treghetsmoment om pitchaksen
J_t = 2 * m_h * l_a *l_a; %treghetsmoment om vandringsaksen

K_f = 0.5; % Kraftkonstant (N/V)
K_p =  0.686; % Kraften som trengs for å holde helikopteret i likevekt.

K_ep = 15.1243; K_ed = 9.3779; K_ei = 1.4216;

K_pp = 8.0467; K_pd = 2.9616; K_pi = 0;


K_1 = l_h*K_f/J_p; K_2 = K_p*l_a/J_t; K_3 = K_f*l_a/J_e;
```

C.1.17 grad.m

```
%Returns the "gradient" of log det (A^-1),
%found by the operator d/dsvec(A)

function [grad]=grad(A) grad=(-inv(A))
```

C.2 Online algorithms

C.2.1 Online simulation

```
% Must have results from SSDMPC.m (the whole workspace)
% Px0 is from nrf (ERPC)
% Pmx = inv(Tx*Qm*Tx') is the ellipse from SSDMPC. It is only for plotellipse purpose.
% to send to Newrap.m: inv(Qm)=Pm and not Pmx (because from eq 3.28, newrap
%   is the z' inv(Qz) z <1 ellipse).

%Qe=Qm;
%Pe=Pm
clear Pmx x r1 r2 c1 c Pe=Pz21; N=2; U=[]; disp('start time system
simulation')

k=0; clear c1 c x
%Ts=0.01;
Iter=1000;
% x0=[-0.01 -0.1 0.3 -0.1]'; % for case 12
x(:,1)=x0;k=0; U=zeros(dimu,50); K=KK(:,1:mx); while k<Iter
    k=k+1;
    tic;
    [c,mu,count,F,Ic,Pxx,Pcx,Pcc,info]=newrap(Tx,Pe,x(:,k),N,mx,dimu);
    c1(:,k)=c;
    CtimeOnline(k)=toc;
    U(:,k)=K*x(:,k)+c(1:dimu) + 12;
    %U(:,k)=K*x(:,k);%+c(1:dimu);
    if abs(U(1,k))>umax(1)
        U(1,k)=sign(U(1,k))*umax(1);
    end

    if dimu==2
        if abs(U(2,k))>umax(1)
            U(2,k)=sign(U(2,k))*umax(2);
        end
    end

    x(:,k+1)=A*x(:,k)+B*U(:,k);
end
%c1(1:20)
timee=[0:Ts:Iter*Ts]; timeeU=[0:Ts:Iter*Ts-Ts]; figure(51)
subplot(2,2,1) hold on plot(timee,x) subplot(2,2,3) hold on
plot(timeeU,U,'b') subplot(2,2,2) plot(timeeU,c1,'b') hold on
subplot(2,2,4);hold on plot(timeeU,CtimeOnline(1:Iter),'b')
```

C.2.2 Online_initial

```
% To find whether the initial xk is inside the ellipsoide.
% Pz0 is from nrf (ERPC)
% Pmx = inv(Tx*Qm*Tx') is the ellipse from SSDMPC. It is only for plotellipse purpose.
% to send to Newrap.m: inv(Qm)=Pm and not Pmx (because from eq 3.28, newrap
%   is the z' inv(Qz) z <1 ellipse).

Pe=Pz2; N=2 r1=1;r2=2;

U=[]; disp('initializing Online problem') x=[]; k=0;

xk=[0 0.4 1.2 -1 0 0.1]'
%xk=x0;

Tx=zeros(mx,mx+dimu*N); Tx([1:mx],[1:mx])=eye(mx); % x=T*z
Tc=zeros(N*dimu,N*dimu+mx);Tc=[zeros(N*dimu,mx) eye(N*dimu)];% f=Tc*z
%Partitioning Pz  fra eq 3.29
S3=Tx*Pe*Tx';  % P11 , Pxx
S2=Tc*Pe*Tx'; %P12 = P21' ,  Pcx
S1=Tc*Pe*Tc';   %P22, Pcc

setlmis([]); [f_online,n,f_online_dec]=lmivar(2,[N*dimu,1]);

lmi_on=newlmi;

lmiterm([-lmi_on 1 1 0],1-xk'*S3*xk);
lmiterm([-lmi_on 1 1 -f_online],-2,S2*xk); %-f'*S2*x
lmiterm([-lmi_on 1 2 -f_online],1,1); lmiterm([-lmi_on 2 2
0],S1^-1);

onlinelmi = getlmis;

[lmi_min,x_ini]=feasp(onlinelmi); if lmi_min>0
    disp('The initial xk is not in the ellipsoide zT Pz z <1 ')
end
```

C.2.3 Real-Time Experiment

Newton Raphson S-function

```
function [sys,x0,str,ts] =
newrapS(t,x,u,flag,N,dimu,mx,Tx,Ic,Pcc,Pcx,Pxx)
% Dispatch the flag. The switch function controls the calls to
% S-function routines at each simulation stage of the S-function.
%
switch flag,
  %%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%
  % Initialize the states, sample times, and state ordering strings.
  case 0
    [sys,x0,str,ts]=mdlInitializeSizes;

  %%%%%%%%%%%
  % Outputs %
  %%%%%%%%%%%
  % Return the outputs of the S-function block.
  case 3
    sys=mdlOutputs(t,x,u,N,dimu,mx,Tx,Ic,Pcc,Pcx,Pxx);

  %%%%%%%%%%%%%%%%%%%
  % Unhandled flags %
  %%%%%%%%%%%%%%%%%%%
  % There are no termination tasks (flag=9) to be handled.
  % Also, there are no continuous or discrete states,
  % so flags 1,2, and 4 are not used, so return an emptyu
  % matrix
  case { 1, 2, 4, 9 }
    sys=[];

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % Unexpected flags (error handling)%
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  % Return an error message for unhandled flag values.
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end
%
%=============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
%
function [sys,x0,str,ts] = mdlInitializeSizes()

sizes = simsizes; sizes.NumContStates  = 0; sizes.NumDiscStates  =
0;
sizes.NumOutputs     = -1;  % dynamically sized
sizes.NumInputs      = -1;  % dynamically sized
sizes.DirFeedthrough = 1;   % has direct feedthrough
sizes.NumSampleTimes = 1;

sys = simsizes(sizes); str = []; x0  = [];
ts  = [-1 0];   % inherited sample time

% end mdlInitializeSizes

%
%=============================================================================
% mdlOutputs
% Return the output vector for the S-function
%=============================================================================
%
function sys = mdlOutputs(t,x,u,N,dimu,mx,Tx,Ic,Pcc,Pcx,Pxx) tic
xk=u; Tc=zeros(N*dimu,N*dimu+mx);Tc=[zeros(N*dimu,mx)
eye(N*dimu)];
%check for c=0 case...
if (xk'*Pxx*xk <1)
   %disp('c=0 is feasible-----------------------------');
   J=0;
   %k=zeros(aug-v,v);
   mu=0;
   F=0;
   count=0;
   sys=[zeros(mx,1)];
   return
end
%init loop
tol=1.e-9 ;count=0;
oldmu=1;mu=0; % mu is set because of while loop.
while (abs(oldmu-mu) >= tol) & (count < 500)
  oldmu=mu;
  g =-oldmu*inv(Ic+oldmu*Pcc)*Pcx';
  F =xk'*(Pxx + 2*Pcx*g +g'*Pcc*g)*xk-1;     %eq 3.32
  dgdmu=inv(Ic+oldmu*Pcc)*[oldmu*Pcc*inv(Ic+oldmu*Pcc)-Ic]*Pcx';
  dFdmu=xk'*[2*Pcx*dgdmu+dgdmu'*Pcc*g+g'*Pcc*dgdmu]*xk;
  if(abs(dFdmu)<tol)
    disp('WARNING, division by near to zero')
    sys=[zeros(mx,1)];
    t
    return
  end
  mu=oldmu-F/dFdmu;
  count=count+1;
end
%----------------------NR finished------
g=-mu*inv(Ic+mu*Pcc)*Pcx'; c1=g*xk; if (isnan(c1(1)) |
isinf(c1(1)))
    c1=zeros(dimu,1);
else if dimu==2
            if (isnan(c1(2)) | isinf(c1(2)))
               c1=zeros(dimu,1);
           end
     end
end tid=toc;
% only the first dimu elements are used from c1.
% sys is c. The output must be size mx. Therefore additional mx-dimu zeros
sys=[c1(1:dimu);tid;zeros(mx-dimu-1,1)];

% end mdlOutputs
```

Appendix D

Visualization algorithms

D.1 Ellipsoid information

D.1.1 Volume_size.m

```
% Information about the dimension and orientation / condition of the ROA
% Nadi Bar

Qe1=Qz0; Txp=zeros(mx,length(Qe1));
% Txp(1,1)=1;Txp(2,2)=1;
% if mx>2
%     Txp(3,3)=1;
% end
Txp=[eye(mx,mx) zeros(mx,length(Qe1)-mx)]; Qex=Txp*Qe1*Txp';
Pe1=inv(Qex)

vol=det(Pe1)^-0.5 % = det (Qex^-1 )^0.5
cond(Pe1) temp2=eig(Pe1)
```

D.1.2 plotellipse.m

```
function[p] = plotellipse(P, fignum,line)
%
% function [p] = plotellipse(P,fignum);
% plot the 2-D ellispe, x'Px=1 on figure fignum
% this does not clear the plot, so use clf
% before calling this if required.
% p is the handle to the plot

if (nargin ~= 3)
   disp('ERROR: usage plotellipse(P,fignum)');
   return
end

if ((size(P,1) ~= 2) | (size(P,2) ~= 2))
   disp('ERROR: P must be 2x2 to plot ellipse')
   P
   return
end
```

```
[V,Lam] = eig(P); a = 1/sqrt(Lam(1,1)); b = 1/sqrt(Lam(2,2));

figure(fignum); hold on; grid on;

th = [0:0.1:(2*pi+0.1)]; coord = V * [a*cos(th);b*sin(th)];
p=plot(coord(1,:),coord(2,:),line);
```

D.1.3 SYS_test.m

```
% Test for new state space matrices systems
% behavior of the system in open and close loops
% with constraints and without.
% Nadi Bar

[A,B,C,xk,Q,R,N,umin,umax,phi,K,SS,mx,dimu]=init2; Iter=1000;
k=0;x=[];U=zeros(dimu,Iter); Ts=0.001;

% med tilbakekobling
 x0=[1 1 1 1]'; % for case 12
 x=[];k=0;U=zeros(dimu,Iter);
 x(:,1)=x0;
 while k<Iter
    k=k+1;
     if k>0
        U(:,k)=-K*x(:,k);%+[100;40];
    end
    if abs(U(1,k))>umax(1)
        U(1,k)=sign(U(1,k))*umax(1);
    end
    if length(umax)>1
        if abs(U(2,k))>umax(2)
            U(2,k)=sign(U(2,k))*umax(2);
        end
    end
    x(:,k+1)=A*x(:,k)+B*U(:,k);
end
% U=zeros(1,N);x3=[];
% x3(:,1)=[x0;0.4]
% k=0;
% while k<N
%     k=k+1;
%     [x3(:,k+1),U00]=DSRV(x3(:,k),U(:,k),4.11);
% end

timee=[0:Ts:Iter*Ts]; timeeU=[0:Ts:Iter*Ts-Ts]; figure(51)
subplot(2,2,1) hold on plot(timee,x,':') legend('x1','x2','x3')

subplot(2,2,3) hold on plot(timeeU,U,':')
```

D.2 3D visualization technique

D.2.1 visual3D.m

```
% ******************************
% 3D ellipsoid Visaulization
% Nadi Bar, Juni 2003
%*****************************
```

```
f=4;

% ProjEllip(Qe,f,r1,r2,r3,'b-')
%subplot(1,2,1)
r1=2;r2=3;r3=4;
if mx==6
    str1=labels(r1);str2=labels(r2);str3=labels(r3)
end Qe=Qz21; Txp=zeros(3,length(Qe));
Txp(1,r1)=1;Txp(2,r2)=1;Txp(3,r3)=1; Pe=inv(Txp*Qe*Txp');
col1=0.5;col2=.5;col3=.5;trans=0.2;
visual(Pe,f,col1,col2,col3,trans) ProjEllip(Qe,f,r1,r2,r3,'b-')

Txp=zeros(3,length(Qz00)); Txp(1,r1)=1;Txp(2,r2)=1;Txp(3,r3)=1;
Pe=Txp*Px00*Txp'; col1=1;col2=0;col3=.1;trans=0.20;
visual(Pe,f,col1,col2,col3,trans,str1,str2,str3)
% Qe=Qz0;
% Txp=zeros(3,length(Qe));
% Txp(1,r1)=1;Txp(2,r2)=1;Txp(3,r3)=1;
% Pe=inv(Txp*Qe*Txp');
% col1=1;col2=0;col3=.1;trans=0.16;
% visual(Pe,f,col1,col2,col3,trans)
%ProjEllip(Qe,f,r1,r2,r3,'m-')
```

D.2.2 visual.m

```
function visual(C,fig,col1,col2,col3,trans,str1,str2,str3)
% Nadi Bar
% called by visual_try.m
%************************************
if nargin==6
    str1='x_1'; str2='x_2'; str3='x_3';
end
%Qe=Q;
figure(fig);
% Txp=zeros(3,length(Qe));
% Txp(1,1)=1;Txp(2,2)=1;Txp(3,3)=1;
% C = inv(Txp*Qe*Txp');

% Then the region of interest would be an ellipsoid centered at M, whose
% size will be govered by C.

% U: Eigenvectors - dictate orientation of ellipsoid
% L: Eigenvalues - decide size of ellipsoid
[U,L] = eig(C);

% For 2 standard deviation spread of data, the radii of the eliipsoid will
% be given by 2*SQRT(eigenvalues).

radii = sqrt(diag(inv(L)));

[xc,yc,zc] = ellipsoid(0,0,0,radii(1),radii(2),radii(3));
% second, rotate it with orientation matrix U
a = kron(U(:,1),xc); b = kron(U(:,2),yc); c = kron(U(:,3),zc);
data = a+b+c;  n = size(data,2);
xd = data(1:n,:); yd = data(n+1:2*n,:); zd = data(2*n+1:end,:); % Nadi

mymap1(:,:,1) = repmat(col1,length(zd),length(zd)); mymap1(:,:,2)
= repmat(col2,length(zd),length(zd)); mymap1(:,:,3) =
repmat(col3,length(zd),length(zd));
```

```
%scPm = surf(xd,yd,zd,mymap1);
scPz10 = surf(xd,yd,zd,mymap1,'edgecolor','n'); hold on
%view(3);

shading interp

camlight headlight % add light for 3-D effect
lighting phong
cameratoolbar('show') % camera toolbar for animation

alpha(scPz10,trans), camlight xlabel(str1), ylabel(str2),
zlabel(str3) grid on
```

D.2.3 ProjEllip.m

```
% ******************************
% 3D ellipsoid Visaulization Projection in 2D plane
% Nadi Bar, Juni 2003
%******************************
function ProjEllip(Q,fig,r1,r2,r3,lin)
%lin is the definition of the line properties: ex lin= '*b--'
% lin must be a string. defualt is 'b--'
% r1 r2 3r are the bases to draw on
if nargin==5
    lin='b--'
end

Txp=zeros(2,length(Q));Txp(1,r1)=1;Txp(2,r2)=1; Pmx =
inv(Txp*Q*Txp'); [V,Lam] = eig(Pmx); a = 1/sqrt(Lam(1,1)); b =
1/sqrt(Lam(2,2)); th = [0:0.1:(2*pi+0.5)];
z=ones(1,length(th))*-5; coord = V * [a*cos(th);b*sin(th)];
figure(fig);hold on; p=plot3(coord(1,:),coord(2,:),z,lin);
% xz plane
Txp=zeros(2,length(Q));Txp(1,r1)=1;Txp(2,r3)=1; Pmx =
inv(Txp*Q*Txp'); [V,Lam] = eig(Pmx); a = 1/sqrt(Lam(1,1)); b =
1/sqrt(Lam(2,2)); th = [0:0.1:(2*pi+0.5)];
z=ones(1,length(th))*0.2; coord = V * [a*cos(th);b*sin(th)];
figure(fig); p=plot3(coord(1,:),z,coord(2,:),lin);
% yz plane
Txp=zeros(2,length(Q));Txp(1,r2)=1;Txp(2,r3)=1; Pmx =
inv(Txp*Q*Txp'); [V,Lam] = eig(Pmx); a = 1/sqrt(Lam(1,1)); b =
1/sqrt(Lam(2,2)); th = [0:0.1:(2*pi+0.1)]; z=ones(1,length(th))*5;
coord = V * [a*cos(th);b*sin(th)]; figure(fig);
p=plot3(z,coord(1,:),coord(2,:),lin);
```

D.2.4 Animate.m

```
   % Visualization techniques
   % Animations for movie creation
   % Nadi Bar

   clear MOV h
   h=plot(plot_time(1),plot_data_u1(1))
   hold on
   plot([1:200],-20*ones(1,length(1:200)),'k:')
   axis([0 200 -22 20])
   set(h,'EraseMode','none','MarkerSize',10)
```

```
n=0;tel=0;
for i=0:0.005:50
       n=n+1;
       drawnow
       set(h,'XData',plot_time(n),'YData',plot_data_u1(n))
       if mod(n,100)==0 & (n>=100)
           tel=tel+1;
           MOV(tel) = getframe;
       end %if
end
```