Fast Real-Time MPC for Fighter Aircraft

Amanda Andersson & Elin Näsholm
Master of Science Thesis in Electrical Engineering

**Fast Real-Time MPC for Fighter Aircraft**

_Amanda Andersson & Elin Näsholm_

LiTH-ISY-EX--18/5143--SE

**Supervisors:**
- Erik Hedberg  
  ISY, Linköping University
- Daniel Simon, PhD.  
  Saab Aeronautics
- Peter Rosander, Lic.  
  Saab Aeronautics

**Examiner:**
- Johan Löfberg, Assoc. Prof.  
  ISY, Linköping University

*Division of Automatic Control*  
*Department of Electrical Engineering*  
*Linköping University*  
*SE-581 83 Linköping, Sweden*

Copyright © 2018 *Amanda Andersson & Elin Näsholm*
Abstract

The main topic of this thesis is model predictive control (MPC) of an unstable fighter aircraft. When flying it is important to be able to reach, but not exceed the aircraft limitations and to consider the physical boundaries on the control signals. MPC is a method for controlling a system while considering constraints on states and control signals by formulating it as an optimization problem. The drawback with MPC is the computational time needed and because of that, it is primarily developed for systems with a slowly varying dynamics.

Two different methods are chosen to speed up the process by making simplifications, approximations and exploiting the structure of the problem. The first method is an explicit method, performing most of the calculations offline. By solving the optimization problem for a number of data sets and thereafter training a neural network, it can be treated as a simpler function solved online. The second method is called fast MPC, in this case the entire optimization is done online. It uses Cholesky decomposition, backward-forward substitution and warm start to decrease the complexity and calculation time of the program.

Both methods perform reference tracking by solving an underdetermined system by minimizing the weighted norm of the control signals. Integral control is also implemented by using a Kalman filter to observe constant disturbances. An implementation was made in MATLAB for a discrete time linear model and in ARES, a simulation tool used at Saab Aeronautics, with a more accurate nonlinear model.

The result is a neural network function computed in tenth of a millisecond, a time independent of the size of the prediction horizon. The size of the fast MPC problem is however directly affected by the horizon and the computational time will never be as small, but it can be reduced to a couple of milliseconds at the cost of optimality.
Acknowledgments

First of all we want to thank the people at the department at Saab Aeronautics for giving us a chance to write our thesis there and for welcoming us into the world of aircraft.

We are especially grateful for having such helpful and knowledgeable supervisors. Daniel Simon for introducing us to the MPC theory and guiding us in the right direction and Peter Rosander for teaching us about aircraft and advising us whenever we get stuck. Our morning meetings have always brought up interesting discussions.

Our gratitude goes also to Johan Löfberg at the Department of Electrical Engineering for giving us helpful tips and tricks and to Erik Hedberg for the help with the Kalman filter and for reading our thesis over and over again.

Lastly we want to send a thank to our families and friends for the support even though not always understanding us when talking about MPC.

Linköping, May 2018
Amanda Andersson & Elin Näsholm
## Contents

List of Figures ix

List of Tables xi

Notation xiii

1 Introduction 1
   1.1 Problem Formulation .............................. 1
       1.1.1 Problem Statements ............................ 2
   1.2 Delimitation ........................................ 2
   1.3 Methodology ........................................ 3
       1.3.1 Software ..................................... 3
   1.4 Outline ............................................ 4

2 Flight Mechanics 5
   2.1 Dynamic Model .................................... 5
       2.1.1 Load Factor .................................. 8

3 Optimization 11
   3.1 Karush–Kuhn–Tucker Conditions .................... 11
   3.2 Weighted Norm Minimization ........................ 12
   3.3 Interior-Point Method ................................ 13
   3.4 Infeasible Start Newton’s Method .................. 14
       3.4.1 Backtracking Line Search ..................... 15

4 Model Predictive Control 17
   4.1 Linear Model ..................................... 17
   4.2 Limitations ....................................... 19
   4.3 Slack Variables ................................... 19
   4.4 Reference Tracking ................................ 20
   4.5 Integral Control ................................... 21

5 Model Predictive Control Techniques 23
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Explicit MPC</td>
<td>23</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Simplifications</td>
<td>26</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Neural Network</td>
<td>27</td>
</tr>
<tr>
<td>5.2</td>
<td>Fast MPC</td>
<td>30</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Interior-Point Method with Approximations</td>
<td>30</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Alternative Methods</td>
<td>32</td>
</tr>
<tr>
<td>6</td>
<td>Implementation</td>
<td>35</td>
</tr>
<tr>
<td>6.1</td>
<td>Linear MPC</td>
<td>35</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Models &amp; Penalty Matrices</td>
<td>36</td>
</tr>
<tr>
<td>6.1.2</td>
<td>MPC Model</td>
<td>37</td>
</tr>
<tr>
<td>6.1.3</td>
<td>Observer</td>
<td>39</td>
</tr>
<tr>
<td>6.2</td>
<td>Explicit MPC</td>
<td>40</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Neural Network</td>
<td>41</td>
</tr>
<tr>
<td>6.3</td>
<td>Fast MPC</td>
<td>43</td>
</tr>
<tr>
<td>6.4</td>
<td>ARES</td>
<td>44</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Simulink Layout</td>
<td>44</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Neural Network</td>
<td>46</td>
</tr>
<tr>
<td>6.4.3</td>
<td>Fast MPC</td>
<td>46</td>
</tr>
<tr>
<td>6.4.4</td>
<td>Kalman Filter</td>
<td>48</td>
</tr>
<tr>
<td>7</td>
<td>Results</td>
<td>49</td>
</tr>
<tr>
<td>7.1</td>
<td>MATLAB Model</td>
<td>49</td>
</tr>
<tr>
<td>7.1.1</td>
<td>Neural Network</td>
<td>51</td>
</tr>
<tr>
<td>7.1.2</td>
<td>Fast MPC</td>
<td>54</td>
</tr>
<tr>
<td>7.1.3</td>
<td>Comparison</td>
<td>57</td>
</tr>
<tr>
<td>7.2</td>
<td>ARES Model</td>
<td>58</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Neural Network</td>
<td>58</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Fast MPC</td>
<td>59</td>
</tr>
<tr>
<td>7.A</td>
<td>Step Responses Ares</td>
<td>63</td>
</tr>
<tr>
<td>8</td>
<td>Discussion &amp; Conclusion</td>
<td>67</td>
</tr>
<tr>
<td>8.1</td>
<td>Choice of Method</td>
<td>67</td>
</tr>
<tr>
<td>8.2</td>
<td>Interpretation of Results</td>
<td>69</td>
</tr>
<tr>
<td>8.3</td>
<td>Conclusion</td>
<td>69</td>
</tr>
<tr>
<td>8.4</td>
<td>Further Work</td>
<td>70</td>
</tr>
<tr>
<td>9</td>
<td>Bibliography</td>
<td>71</td>
</tr>
</tbody>
</table>
List of Figures

2.1 The aircraft seen from the side. The angle of attack, \( \alpha \), is defined as the angle between the velocity vector along the \( x \)-axis of the aircraft and \( V \) projected onto the \( xz \)-plane. .................. 6

2.2 The aircraft seen from above. The angle of sideslip, \( \beta \), is defined as the angle between the velocity vector along the \( x \)-axis of the aircraft and \( V \). .................. 6

2.3 The aircraft seen from the front taking a turn. The roll, or bank angle \( \phi \) is defined as the angle between the earth fixed \( z \)-direction and the extension of the lift vector, when flying wings level. .... 8

3.1 The barrier function. Decreasing \( \kappa \) gives steeper edges, here the lowest value is shown in blue with limits for \( x \) at 0 and 10. .... 14

5.1 The explicit solution of a problem with two states and one input. It has two saturated areas connected by a linear segment. Different colors show different regions with a certain optimal affine function \( z^* \). .................. 25

5.2 The structure of the neural network with input layer in green, hidden layers in yellow and output layer in red. .................. 27

5.3 Weights and bias for one node. .................. 28

5.4 The shape of the activation functions. .................. 29

5.5 The mean square error for a system with its training completed. It has stopped after 79 epochs finding its minimum after 59 iterations. .................. 30

6.1 A step response in \( \alpha \) for the linearized aircraft model with an MPC controller for \( N = 240, 60, 20 \) for the envelope point with Mach 0.6 and altitude 1000 m. The canard is the control signal that starts at zero. .................. 39

6.2 The regions of the feasible set. To generate a graph, the reference and disturbance are fixed to 0\(^\circ\). .................. 40

6.3 Data set of 150 points. .................. 41
6.4 The regression for two different activation functions when stopped early. The target values are compared with the outputs of the neural network. The hyperbolic tangent is slower to train and is not as fitted at this epoch. ................................................................. 42

6.5 The structure of the neural network function. ................................. 43

6.6 Simulink implementation of the MPC controller. Alfa_Cmd is the reference signal commanded from the pilot. ......................................................... 45

6.7 Simulink implementation of the neural network controller. Port 1 and 2, represent the states. ............................................................ 46

6.8 Simulink implementation of the fast MPC controller with warm start. ......................................................................................... 47

6.9 A step response with a too aggressively tuned Kalman filter. The penalty matrices from Equation 6.11 in Section 6.1.3. .................. 48

7.1 The step responses to 3° for different velocities on an altitude of 6000 m with the same controller. The canard wing is the control signal that start at a positive value. ............................................ 50

7.2 A desired step response for α with disturbances on both states. The observer is displayed in magenta and the real values in black. . . . 50

7.3 The minor difference of using three hidden layers compared with two. ......................................................................................... 52

7.4 The regression when not using enough nodes. The negative saturation is especially hard to fit appropriately. ............................. 53

7.5 Step response for the neural network without disturbances. . . . 53

7.6 A lateral step response, where the reference for p is set to 5°/s and β to 0°. The elevon is the control signal starting with a positive value. The simulation is started at the quasi stationary state of β = 5, p=0, r=0. ................................................................ 54

7.7 Step in the angle of attack with a reference of 3° for a perfect model for different κ. The control signal of the canard wing is the input with a steady state value of 0°. .......................................................... 55

7.8 The difference of limiting K_{max} to 3 compared to letting the algorithm find the relaxed optimum. Even with a restricted number of iterations the solution resembles the optimal solution. ........ 56

7.9 A step response in α for the fast MPC without disturbances. . . . 57

7.10 The resulting NN controller. .......................................................... 59

7.11 The effect of varying the prediction horizon when taking steps to 4° and 18° (the maximum limit). The reference is green and the aircraft response is blue. ............................................................. 61

7.12 The controller hitting the load factor limit, when α = 18. .......... 62

7.13 Fast MPC, N = 60. With an initial Mach of 0.8 and altitude of 6000 m. ......................................................................................... 64

7.14 The controllers following a reference and then returns to trim state and q = 0. With Mach 0.8 and altitude 6000 m. .............. 65
List of Tables

6.1 Parameters for using the MPC controller in ARES. All with the default value of 0. .................................................. 44

7.1 A comparison of the NN values at steady state for different sizes of data sets with the RELU and the hyperbolic tangent as activation functions. ................................................................. 51

7.2 A comparison between different number of neurons and hidden layers. ................................................................. 52

7.3 The number of iterations and time taken for cold/warm start with and without the load factor constraint. $K$ is limited to $K_{\text{max}} = 20$. The reference of $\alpha$ is set to $1.0^\circ$ and $9.0^\circ$ where it exceeds the $n_z$ limitation). ................................................................. 56

7.4 A comparison of the methods at steady state. $m$ stands for Mach and $h$ for altitude. .................................................. 58

7.5 A comparison of the methods at steady state with the disturbances $w = \dot{w} = [0.05, 0.2]^T$. .................................................. 58

7.6 The improvement with the alternative methods. Time measured in microseconds ($\mu$s). .................................................. 60

7.7 The time in microseconds ($\mu$s) taken for each iteration. $K_{\text{max}} = 3$, $\kappa = 1$. .................................................. 60
### Notation

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_0(x)$</td>
<td>Objective function</td>
</tr>
<tr>
<td>$f_i(x)$</td>
<td>Inequality constraint</td>
</tr>
<tr>
<td>$h_i(x)$</td>
<td>Equality constraint</td>
</tr>
<tr>
<td>$K$</td>
<td>Kalman gain</td>
</tr>
<tr>
<td>$K_{\text{max}}$</td>
<td>Maximum number of optimization cycles</td>
</tr>
<tr>
<td>$L$</td>
<td>Lower quadratic matrix</td>
</tr>
<tr>
<td>$l$</td>
<td>Number of constraints for one time step</td>
</tr>
<tr>
<td>$m$</td>
<td>Length of the control vector</td>
</tr>
<tr>
<td>$N$</td>
<td>Prediction horizon</td>
</tr>
<tr>
<td>$n$</td>
<td>Load factor</td>
</tr>
<tr>
<td>$n$</td>
<td>Length of the state vector</td>
</tr>
<tr>
<td>$P$</td>
<td>Penalty matrix for the weighted norm minimization</td>
</tr>
<tr>
<td>$P_N$</td>
<td>Terminal state penalty for $x_N$, solution to the Riccati equation</td>
</tr>
<tr>
<td>$p$</td>
<td>Rotational velocity w.r.t. the $x$-axis</td>
</tr>
<tr>
<td>$Q$</td>
<td>Penalty matrix for $x$</td>
</tr>
<tr>
<td>$Q_K$</td>
<td>Covariance matrix for the state</td>
</tr>
<tr>
<td>$q$</td>
<td>Rotational velocity w.r.t. the $y$-axis</td>
</tr>
<tr>
<td>$q$</td>
<td>Column vector for the weighted norm minimization</td>
</tr>
<tr>
<td>$R$</td>
<td>Penalty matrix for $u$</td>
</tr>
<tr>
<td>$R_K$</td>
<td>Covariance matrix for the observer output</td>
</tr>
<tr>
<td>$\mathbb{R}$</td>
<td>The set of real numbers</td>
</tr>
<tr>
<td>$r$</td>
<td>Rotational velocity w.r.t. the $z$-axis</td>
</tr>
<tr>
<td>$r$</td>
<td>Residual</td>
</tr>
<tr>
<td>$r$</td>
<td>Reference vector</td>
</tr>
<tr>
<td>$r_d$</td>
<td>Dual residual</td>
</tr>
<tr>
<td>$r_p$</td>
<td>Primal residual</td>
</tr>
<tr>
<td>Notation</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>$s$</td>
<td>Step length</td>
</tr>
<tr>
<td>$T_s$</td>
<td>Sample time</td>
</tr>
<tr>
<td>$\mathbb{U}$</td>
<td>Matrix of the $N$ following inputs</td>
</tr>
<tr>
<td>$u$</td>
<td>Control vector</td>
</tr>
<tr>
<td>$\bar{u}$</td>
<td>Trimmed input</td>
</tr>
<tr>
<td>$u_r$</td>
<td>Steady state input for reference</td>
</tr>
<tr>
<td>$\bar{V}$</td>
<td>Airspeed vector</td>
</tr>
<tr>
<td>$V(x)$</td>
<td>Objective function, MPC problem</td>
</tr>
<tr>
<td>$x$</td>
<td>State vector</td>
</tr>
<tr>
<td>$x_r$</td>
<td>Steady state for reference</td>
</tr>
<tr>
<td>$\bar{x}$</td>
<td>Trimmed state</td>
</tr>
<tr>
<td>$x^*$</td>
<td>Optimal value</td>
</tr>
<tr>
<td>$x^+$</td>
<td>Consecutive step</td>
</tr>
<tr>
<td>$z$</td>
<td>Vector with all states and inputs in the prediction horizon</td>
</tr>
<tr>
<td>$z_k$</td>
<td>Input value formed by the affine function</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>Angle of attack</td>
</tr>
<tr>
<td>$\beta$</td>
<td>Angle of sideslip</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>Penalty scalar for slack variable</td>
</tr>
<tr>
<td>$\delta_a$</td>
<td>Aileron control surface deflection</td>
</tr>
<tr>
<td>$\delta_c$</td>
<td>Canard control surface deflection</td>
</tr>
<tr>
<td>$\delta_e$</td>
<td>Elevator control surface deflection</td>
</tr>
<tr>
<td>$\delta_{ea}$</td>
<td>Elevon control surface deflection</td>
</tr>
<tr>
<td>$\delta_r$</td>
<td>Rudder control surface deflection</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>Slack variable</td>
</tr>
<tr>
<td>$\theta$</td>
<td>Pitch angle</td>
</tr>
<tr>
<td>$\kappa$</td>
<td>Barrier coefficient for the barrier function</td>
</tr>
<tr>
<td>$\mathcal{L}$</td>
<td>The Lagrangian</td>
</tr>
<tr>
<td>$\lambda_i$</td>
<td>Lagrange multiple, inequality constraints</td>
</tr>
<tr>
<td>$\nu_i$</td>
<td>Lagrange multiple, equality constraints</td>
</tr>
<tr>
<td>$\Phi$</td>
<td>Euler angles</td>
</tr>
<tr>
<td>$\phi$</td>
<td>Roll angle</td>
</tr>
<tr>
<td>$\phi(x)$</td>
<td>Barrier function</td>
</tr>
<tr>
<td>$\chi$</td>
<td>Feasible region</td>
</tr>
<tr>
<td>$\psi$</td>
<td>Yaw angle</td>
</tr>
<tr>
<td>$\omega$</td>
<td>Angular velocity</td>
</tr>
<tr>
<td>$\hat{\omega}$</td>
<td>Observed disturbance</td>
</tr>
</tbody>
</table>
### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASM</td>
<td>Active-Set Method</td>
</tr>
<tr>
<td>CR</td>
<td>Critical Region</td>
</tr>
<tr>
<td>IPM</td>
<td>Interior-Point Method</td>
</tr>
<tr>
<td>KKT</td>
<td>Karush-Kuhn-Tucker</td>
</tr>
<tr>
<td>LP</td>
<td>Linear Programming</td>
</tr>
<tr>
<td>LQ</td>
<td>Linear–Quadratic</td>
</tr>
<tr>
<td>MPC</td>
<td>Model Predictive Control</td>
</tr>
<tr>
<td>MP-QP</td>
<td>Multi-Parametric Quadratic Program</td>
</tr>
<tr>
<td>MSE</td>
<td>Mean Square Error</td>
</tr>
<tr>
<td>NN</td>
<td>Neural Network</td>
</tr>
<tr>
<td>NED</td>
<td>North, East, Down</td>
</tr>
<tr>
<td>QP</td>
<td>Quadratic Program</td>
</tr>
<tr>
<td>RELU</td>
<td>Rectified Linear Unit</td>
</tr>
</tbody>
</table>
Model predictive control (MPC) is a control strategy that relies on the dynamical model of the system and takes the limitations of the system into account when calculating the optimal control sequence for each sample time. The method predicts the response of the system a number of time steps into the future, to find the optimal solution. Today it is primarily used in the process industry and chemical plants for multiple, input multiple output systems with long time horizons. MPC can handle both soft and hard constraints of the inputs, states and outputs, with the possibility to achieve high performance with a small effort while fulfilling demands of e.g. safety.

Applying this theory to the aeronautical field is a challenging task, with fast dynamics along advanced trajectories, rapid maneuvers and high frequency disturbances, where you want to use the aircraft at its maximum possible performance. At each of the sampling instants an optimization problem has to be solved, but the complexity and computational time grows rapidly with the number of states and the prediction horizon. This poses a problem which is addressed herein.

1.1 Problem Formulation

The purpose of this master thesis is to identify and analyze algorithms for real time implementation of MPC controllers.

MPC is demanding in the sense that an optimization problem has to be solved in each sample interval which, for the aircraft application studied here, is limited to 1/120 of a second. With the capacity and methods used today, this is a real challenge. In this thesis different methods will be examined and compared in the respect of speed, functionality- and real time performance.
The thesis will begin with a literature study of suitable methods, for example fast online optimization solvers or approximate explicit MPC-algorithms, where the performance and feasibility are discussed. The result will be an implementation of the appropriate methods and simulations to show the performance of the controllers.

The work will be focused on aircraft, where the system dynamics of both longitudinal and of lateral movement are studied. The goal is to demonstrate the applicability of one of the algorithms by using Saab Aeronautics’ simulation environment ARES and to implement a functioning method for the unstable JAS 39 Gripen.

### 1.1.1 Problem Statements

The questions to be answered are:

- How can an MPC controller be integrated in the system of an aircraft?
- Which (approximation) methods can be applied to simplify and speed up the solving process?
- Which performance can be obtained with explicit versus faster implicit methods, considering time and optimality?

### 1.2 Delimitation

The dynamics of an aircraft can be described more accurately with a nonlinear model \( f(\frac{dx}{dt}, x, u, t) = 0 \), but in this thesis only a linearized model will be used when designing the controller. This, together with the assumption that the constraints for the optimization task will be linear, gives a convex problem to solve. When a local minimum is found, it is always equivalent to the global optimum. Certain aircraft dynamics will not be taken into account, for example the phugoid mode in the longitudinal motion as well as the spiral mode in the lateral motion, and faster dynamics such as servo and valve dynamics.

Only discrete problems are examined with the methods presented. This means that the control signals are seen as constant in between the sampling points. In the linearized model, \( x_{i+1} = Ax_i + Bu_i \), the matrices \( A \) and \( B \) are assumed to be time invariant and known for the simulations. All states in the state space models have outputs that can be measured. No disturbances such as extra loads (fighters often have multiple combinations of military equipment) or fuel distribution will be considered. The input will be affected by disturbances and/or noise and should in fact be denoted \( \hat{u} \), as an estimate of the real signal, but this notation will be ignored.

The optimization problem of minimizing the cost function will only have quadratic penalties for \( x \) and \( u \). It will not contain any terminal constraints since proving stability of the MPC controllers is problematic anyway. Since it is not the focus of
1.3 Methodology

The first part of the project was based on literature studies about optimization, flight dynamics, design structures of MPC, explicit MPC and fast MPC. Different approximation methods for these were researched to see if it is possible to decrease the iteration time without losing the control performance.

A basic controller with an unsimplified structure was designed for the aircraft dynamics and solved by using YALMIP [Löfberg, 2004] along with the multiparametric toolbox (MPT3) [Herceg et al., 2013]. First the regulator problem was considered, driving all the states to zero and thereafter the model was extended to solve the servo problem. The result of this was used as the benchmark for the explicit and fast MPC algorithms.

One method from each one of the branches, distinguished as approaches with good results and the potential to be implemented for the fast system, were chosen and code was written in MATLAB. The performance of these were evaluated and then implemented for the nonlinear model in ARES, by transforming the algorithm into C-code. In MATLAB it is possible to autogenerate code directly with some limitations of not using certain commands.

The computational speed was then measured for different reference signals. In this process, adjustments were made to improve the algorithms further and to adapt them slightly for nonlinear behaviors.

1.3.1 Software

Down below is a short description of some software used during the process.

**MATLAB**

MATLAB is a desktop computing environment with its own programming language developed by MathWorks. It was the primary tool used during the project for development, testing and evaluation. A wide variety of toolboxes have done the process easier, for example the neural network toolbox where the process of training a network can be overviewed, the optimization toolbox solving optimization problems and the statistics toolbox generating random data.

**YALMIP**

YALMIP is a toolbox for modeling and optimization in MATLAB with an alternative setup for solving optimization problems.

**MPT3**

MPT3 is an open source toolbox for MATLAB used for MPC problems to solve the optimization. It was utilized in combination with YALMIP to obtain the implicit
and explicit piecewise affine solution. The implicit one is used throughout the thesis to give the correct behavior and for comparing the different methods.

**ARES**

ARES (Aircraft Rigid-Body Engineering System) is an in-house simulation environment used by Saab Aeronautics. It is based on a Vegas model in the Simulink based simulation tool ADMIRE, developed by Saab Aeronautics and the Swedish Defence Research Agency to resemble JAS 39 Gripen. It connects the flight control system with other subsystems of the aircraft; aerodynamics, hydraulics, structural loads etc. The nonlinear simulation environment can be used both for desktop simulation and software and hardware simulator testing. [Simon, 2017, p. 40]

### 1.4 Outline

The thesis is organized as follows. Chapter 2-5 present relevant theory. First, theory of flight mechanics is described followed by some optimization methods. Both chapters give specific knowledge needed for further understanding of the thesis. Then the basics of MPC are discussed where the MPC formulation is extended with slack, reference tracking and integral control. Thereafter both explicit and fast MPC is exploited together with the two chosen methods most applicable to the problem. By this follows Chapter 6 with the implementation of the methods and the process of filtering away inadequate solutions. Then the result of the implementation in the different software are shown in Chapter 7. Chapter 8 contains a discussion about the result and further development and the conclusions from this thesis. It all ends with appendices containing tables, Simulink diagram and graphs.
The MPC techniques are applied on a system resembling a JAS 39 Gripen. This chapter starts with a brief description of the flight mechanics necessary to understand the implementation in this thesis. The longitudinal and lateral dynamics models are presented as separate systems, describing the modes, input and state vectors and how these are obtained for all configurations of altitude and speed. After that the load factor is shortly presented, it can be interpreted as a measurement of the acceleration perceived on-board the aircraft which limits some maneuvers. This chapter is based on [Nelson, 1998].

2.1 Dynamic Model

To describe the dynamics of an aircraft, several right-handed coordinate systems are needed. Making the assumption that the earth is flat, the first is the earth fixed coordinate system which is a north, east, down (NED) coordinate system with its origin on the surface of the earth. The second one is the body-fixed coordinate system of the aircraft. This coordinate system, with its origin at the centre of gravity, has its x-axis pointing through the nose and the z-axis pointing downwards, see Figure 2.1.

To describe the orientation between the NED and the body-fixed coordinate systems, Euler angles are used and defined as

$$\Phi = [ \phi, \theta, \psi ],$$

where $\phi$ is the roll angle, $\theta$ the pitch angle and $\psi$ the yaw angle. Another coordinate system used is the wind axis coordinate system, which has its $x$-axis aligned with $V = [u, v, w]^T$ or the incoming airflow. The relation between the
wind axis coordinate system and the body-fixed coordinate system is described by \( \alpha = \arctan \frac{w}{u} \), called the angle of attack, and \( \beta = \arcsin \frac{v}{|V|} \), called the angle of sideslip, see Figure 2.1 and 2.2.

**Figure 2.1:** The aircraft seen from the side. The angle of attack, \( \alpha \), is defined as the angle between the velocity vector along the \( x \)-axis of the aircraft and \( V \) projected onto the \( xz \)-plane.

**Figure 2.2:** The aircraft seen from above. The angle of sideslip, \( \beta \), is defined as the angle between the velocity vector along the \( x \)-axis of the aircraft and \( V \).

The flight dynamics are described with Newton’s second law and aerodynamic forces and moments. The Newton forces acting on the aircraft are expressed by \( F = m\ddot{V} + \omega \times mV \) and the moments \( T = I\dot{\omega} + \omega \times I\omega \). \( F \) and \( T \) are the total forces and moments, \( m \) the aircraft mass, \( I \) the inertia and \( \omega = [p, q, r]^T \). From the airflow around the aircraft when flying through the atmosphere, aerodynamic forces and moments arises.

The flight dynamics of an aircraft is usually seen as two separate models, longitudinal and lateral. The longitudinal model describes the pitch dynamics which
includes the phugoid mode and the short-period mode while the lateral model
addresses the yaw and roll dynamics with the roll mode, dutch roll mode and
the spiral mode. They are all described in the list below. The phugoid and spiral
mode are not considered in the linearization.

- The **phugoid mode** is a slowly oscillating mode where the angle of attack
remains approximately constant. This motion can easily be counteracted
by the pilot.

- The **short-period mode** is a faster oscillating mode (sometimes even unsta-
able) where the changes in $\alpha$ and $\theta$ are almost the same. The aircraft is pitch-
ing around the velocity vector, a phenomenon which has to be damped.

- The **roll mode** is dominated by the ability to make a rotation round the
$x$-axis i.e. change the angular velocity $p$.

- The **dutch roll mode** is an oscillating mode with a relatively short period
damped by the fin. It is a coupled motion where the aircraft’s tail is wagging
from side to side.

- The **spiral mode** gives a rolling/yawing-motion. By making a turn (or hav-
ing a roll disturbance) the inner wing is dropped, developing a sideslip in
the same direction as the rotation causing a force on the fin setting the air-
craft o
ff to a spiral. It is relatively normal that the pole is unstable but it is
easy for a pilot to avoid the divergence due to a long time constant.

A standard linearization of the aircraft dynamics, on the form $\dot{x}(t) = Ax(t) + Bu(t)$,
is given by the longitudinal state space model

\[
\begin{bmatrix}
\Delta \dot{\alpha} \\
\Delta \dot{q}
\end{bmatrix} =
\begin{bmatrix}
\frac{Z_a}{u_0} & 1 \\
M_a + M_\dot{\alpha} & M_q + M_\dot{\alpha}
\end{bmatrix}
\begin{bmatrix}
\Delta \alpha \\
\Delta q
\end{bmatrix} +
\begin{bmatrix}
Z_{\delta_c} \\
M_{\delta_c} + M_{\dot{\alpha}}
\end{bmatrix}
\begin{bmatrix}
\Delta \delta_c \\
\Delta \delta_e
\end{bmatrix},
\]

(2.1)

and the lateral model

\[
\begin{bmatrix}
\Delta \dot{\beta} \\
\Delta \dot{\rho} \\
\Delta \dot{r}
\end{bmatrix} =
\begin{bmatrix}
\frac{Y_\beta}{u_0} & \frac{Y_\rho}{u_0} & \frac{Y_r}{u_0} - 1 \\
L_\beta & L_\rho & L_r \\
N_\beta & N_\rho & N_r
\end{bmatrix}
\begin{bmatrix}
\Delta \beta \\
\Delta p \\
\Delta r
\end{bmatrix} +
\begin{bmatrix}
0 \\
L_{\delta_\alpha} & L_{\delta_r} \\
N_{\delta_\alpha} & N_{\delta_r}
\end{bmatrix}
\begin{bmatrix}
\Delta \delta_\alpha \\
\Delta \delta_r
\end{bmatrix},
\]

(2.2)

where $\delta_c$, $\delta_e$, $\delta_\alpha$ and $\delta_r$ are the canard control surface deflection, the elevator con-
trol surface deflection, the aileron control surface deflection and the rudder con-
trol surface deflection respectively [Nelson, 1998, p. 149, 195]. The states $p$, $q$
and $r$ are the angular velocities with respect to the $x$-, $y$- and $z$-axes of the body-
fixed coordinate system and $u_0$ is the velocity in the body-fixed $x$ direction when
linearizing the system. All other variables are aerodynamic and stability deriva-
tives, which describe how forces and moments on the aircraft changes due to
small variations of the states and control signals.

For an aircraft, the parameters in the $A$ and $B$ matrices depend on both dynamic
pressure, Mach number (i.e. the velocity as a fraction of the speed of sound),
mass and inertial moment. The state space model is thus evaluated in a number
of envelope points, since in the linearized model $A$ and $B$ vary depending on the operating conditions.

Due to some alterations in wing configuration, the state space model will differ slightly from the model used to describe JAS 39 Gripen which has a canard delta configuration. For example the elevator and aileron are combined to an elevon, $\delta_{ea}$, which control both roll and pitch motions. It is thus possible to combine the two models into one large due to the cross coupling.

### 2.1.1 Load Factor

The load factor $n$ is defined as the ratio of the aircraft lift force to its weight, at the centre of gravity it is calculated as $n = L/mg$. For level flight the load factor is 1, when flying upside-down it has the same magnitude but changes the sign. In Figure 2.3, assuming that the aircraft is doing a balanced turn, the load factor is $n = 1/cos(\phi)$. Although the ratio is dimensionless it is often denoted with $g$.

![Figure 2.3: The aircraft seen from the front taking a turn. The roll, or bank angle $\phi$ is defined as the angle between the earth fixed z-direction and the extension of the lift vector, when flying wings level.](image)

When perceived by the pilot an additional term needs to be added as in Equation 2.3 where $\Delta l$ is the distance from the centre of gravity to the cockpit.

$$n_z = n + \frac{\dot{q}\Delta l}{g} \quad (2.3)$$

An example of the longitudinal dynamics, for an arbitrary envelope point, can be seen below based on the state space formulation.

$$\begin{bmatrix} \Delta \alpha \\ \Delta q \\ \Delta n_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.349 & 0.008 & 0.009 \end{bmatrix} \begin{bmatrix} \Delta \alpha \\ \Delta q \\ \Delta \delta_c \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0.009 & 0.123 & 0 \end{bmatrix} \begin{bmatrix} \Delta \delta_c \\ \Delta \delta_e \end{bmatrix} \quad (2.4)$$

As shown by the matrices the load factor is highly connected to the angle of attack. At high velocities it is primarily $n_z$ which is the limiting factor to the fea-
sible region while at lower velocities it is $\alpha$. By making this limitation too large accelerations on the pilot and major stresses on the structure are avoided. The magnitude of the minimum value of the load factor is often much lower than the maximum one.
In this chapter some optimization methods are presented which are later used in the implicit and explicit implementation. First the Karush–Kuhn–Tucker (KKT) conditions are formulated which are used to investigate if a point is optimal and feasible in a region. When solving the MPC-problem explicitly the conditions are utilized to obtain regions over which the optimal input is piecewise affine in the state, see Section 5.1. Then the equations for the weighted norm minimization are derived. These are used to determine states and inputs for a certain reference at steady state. For the fast MPC in Section 5.2 the interior-point method (IPM) is combined with the infeasible start Newton’s method and backtracking line search. The IPM restructures the problem by eliminating the inequality constraints and replaces it with a barrier function. To find the optimum when stepping through the feasible region the infeasible start Newton’s method chooses where to start and in which direction it should move. The backtracking line search determines how long these steps should be.

### 3.1 Karush–Kuhn–Tucker Conditions

The KKT conditions are necessary conditions for a point $x^*$ to be optimal considering a general optimization problem as in Equation 3.1, where $f_0(x)$ is the objective function and $f_i(x)$ and $h_i(x)$ the constraint functions. The set of $x$ which fulfills the inequality and equality equations is called the feasible region.

\[
\begin{align*}
\text{minimize} & \quad f_0(x) \\
\text{subject to} & \quad f_i(x) \leq 0, \ i = 1, \ldots, m \\
& \quad h_i(x) = 0, \ i = 1, \ldots, p 
\end{align*}
\tag{3.1}
\]
The KKT conditions are stated as:

\[ f_i(x) \leq 0, \ i = 1, \ldots, m \]  \hspace{1cm} (3.2a)
\[ h_i(x) = 0, \ i = 1, \ldots, p \]  \hspace{1cm} (3.2b)
\[ \lambda_i \geq 0, \ i = 1, \ldots, m \]  \hspace{1cm} (3.2c)
\[ \lambda_i f_i(x) = 0, \ i = 1, \ldots, m \]  \hspace{1cm} (3.2d)

\[ \nabla f_0(x) + \sum_{i=1}^{m} \lambda_i \nabla f_i(x) + \sum_{i=1}^{p} v_i \nabla h_i(x) = 0 \]  \hspace{1cm} (3.2e)

where \( \lambda_i \) and \( v_i \) are the Lagrange multipliers. Equation 3.2a and 3.2b are needed to check primal feasibility, 3.2c dual feasibility and 3.2d complementary slackness. Fulfilling these constraints, a point is optimal if there exist \( \lambda_i \) and \( v_i \) solving 3.2e. It means that the Lagrangian \( \mathcal{L} \) (Equation 3.3) is minimized. The function \( \mathcal{L} \), also called dual function, is used to find stationary points by looking at the dual problem.

\[ \mathcal{L}(x, \lambda, v) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} v_i h_i(x) \]  \hspace{1cm} (3.3)

For a convex problem the solution \((x^*, \lambda^*, v^*)\) satisfying Equation 3.2 is global and primal-dual optimal.

### 3.2 Weighted Norm Minimization

The norm minimization presented here is an optimization problem where the objective function is the norm constrained by equality constraints.

\[ \text{minimize} \quad \frac{1}{2} \|Px - q\| \]  \hspace{1cm} (3.4)
\[ \text{subject to} \quad Ax = b \]

Here the 2-norm is considered which leads to a quadratic problem, where \( P \) is the weight matrix. By using Equation 3.3, the Lagrangian becomes:

\[ \mathcal{L}(x, v) = \frac{1}{2} (Px - q)^T (Px - q) + v^T (Ax - b) \]
\[ = \frac{1}{2} x^T P^T P x - x^T P^T q + \frac{1}{2} q^T q + x^T A^T v - v^T b \]  \hspace{1cm} (3.5)

The optimality conditions are then fulfilled when the gradients are equal to zero.

\[ \nabla_x \mathcal{L} = P^T P x - P^T q + A^T v = 0 \]
\[ \nabla_v \mathcal{L} = A x - b = 0 \]  \hspace{1cm} (3.6)

This can be written into matrix form

\[
\begin{bmatrix}
P^T P & A^T \\
A & 0
\end{bmatrix}
\begin{bmatrix}
x \\
v
\end{bmatrix} =
\begin{bmatrix}
P^T q \\
b
\end{bmatrix}.
\]  \hspace{1cm} (3.7)
when solved for, the optimal point is obtained.

### 3.3 Interior-Point Method

The interior-point method is an optimization strategy for both linear and non-linear programming that solves convex problems. To find the best solution it searches through the interior of the feasible set. For an optimization problem with linear equality constraints rewritten to a minimization (or maximization) problem over a convex set

\[
\begin{align*}
\text{minimize} & \quad f_0(x) \\
\text{subject to} & \quad f_i(x) \leq 0, \; i = 1, \ldots, l \\
& \quad Ax = b,
\end{align*}
\]  

(3.8)

where \( f_0 \) to \( f_l \) are twice continuously differentiable and convex. (Works for QP:s, LP:s etc.) The feasible solution is found by reducing Equation 3.8 to a linear equality constrained problem formulated as:

\[
\begin{align*}
\text{minimize} & \quad f_0(x) + \kappa \phi(x) \\
\text{subject to} & \quad Ax = b.
\end{align*}
\]  

(3.9)

By using the barrier method the inequality constraints is replaced with a penalty function \( \phi(x) \). In this case the logarithmic barrier is used.

\[
\phi(x) = -\sum_{i=1}^{l} \log(-f_i(x))
\]  

(3.10)

The value of \( \kappa \), called the barrier coefficient, is strictly larger than zero and determines how much being adjacent to a boundary should be penalized. See Figure 3.1 where \( \kappa \) is altered. A high value is used when it is important not to go outside the feasible set, even though the objective is minimized at the edges the solution will move toward the middle of the set.

For solving Equation 3.9 various methods are possible. Focus in this thesis has been laid on the infeasible start Newton’s method, see Section 3.4. The iterations are made with a decreasing value for \( \kappa \), with the earlier optimal value as the starting point. As \( \kappa \) decreases, the solution to Equation 3.9 approaches the QP solution.

For larger MPC problems the performance of the IPM is better than for the active set method (ASM), where combinations of constraints are used to find an optimal point (see [Boyd and Vandenberghe, 2004] for further description). One benefit is that the complexity is not affected by degeneracy. For the ASM the number of iterations grows linearly with the number of inequality constraints while it approximately stays the same when using the IPM [Lau et al., 2009].
3.4 Infeasible Start Newton’s Method

The infeasible start Newton’s method is an extension of the original Newton’s method. The difference is, as the name implies, that the starting point must not be feasible, only the inequality constraints have to be met. Further, instead of only minimizing the objective function the residual of the equality constraint is also minimized. The linear set of equations below determines which search direction that is needed, and how to update the dual variable $\nu$. [Boyd and Vandenberghe, 2004, p. 531-534].

$$\begin{bmatrix} \nabla^2 f(x) & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \nu \end{bmatrix} = -\begin{bmatrix} r_d \\ r_p \end{bmatrix} \quad (3.11)$$

The function $f(x)$ is the objective function added to the penalty function, $f(x) = f_0(x) + \kappa \phi(x)$. The vector on the right hand side is the residual of the problem, where $r_p$ and $r_d$ are the primal and dual residual respectively.

$$\begin{bmatrix} r_d \\ r_p \end{bmatrix} = \begin{bmatrix} \nabla f(x) + A^T \nu \\ Ax - b \end{bmatrix} \quad (3.12)$$

In every new iteration the search point is updated by taking the primal (Newton) and dual search step according to:

$$\begin{align*}
x^+ &= x + s\Delta x \\
\nu^+ &= \nu + s\Delta \nu \quad (3.13)
\end{align*}$$

The first value of $\nu$ can be chosen arbitrarily.
3.4 Infeasible Start Newton’s Method

3.4.1 Backtracking Line Search

After finding a feasible direction, the step length $s$ must be determined, this can be done with an inexact line search. With the goal to minimize the residuals in the direction calculated above, backtracking is used until the value is reduced below the threshold $\varepsilon > 0$. As long as Equation 3.14 holds, $s$ is updated to $\beta s$ with the constants $\alpha \in (0, 0.5)$ and $\beta \in (0, 1)$.

$$
||r(x + s\Delta x, \nu + s\Delta \nu)||_2 > (1 - \alpha s)||r(x, \nu)||_2
$$

(3.14)

A pseudo code of the full method is described in Algorithm 1. [Boyd and Vandenberghe, 2004, p. 534]

Algorithm 1 Algorithm for the infeasible start Newton’s method

1: function SUBOPTIMAL(x, \nu, \alpha, \beta, \varepsilon)
2:     while not( ||r(x, \nu)||_2 < \varepsilon and Ax = b + |\delta| ) do
3:         Compute $\Delta x$ and $\Delta \nu$ (Equation 3.11)
4:         s=1
5:         while $||r(x + s\Delta x, \nu + s\Delta \nu)||_2 > (1 - \alpha s)||r(x, \nu)||_2$ do
6:                 s=s$\beta$
7:         end while
8:         Update $x = x + s\Delta x$
9:         Update $\nu = \nu + s\Delta \nu$
10:     end while
11:     return x
12: end function

$\delta$ is a small value to handle numerical errors.
Model Predictive Control is a method to control a system with constraints. In this chapter most of the general theory is presented. First the linear MPC is described showing how the objective function is designed and the constraints are stated. This is followed by extensions, such as slack, reference tracking and integral control, with an explanation of why they have to be implemented and a presentation of different alternative methods. The goal is to bypass the main drawback of the method, which is the computational time that restricts the system to having few states or being slowly varying.

4.1 Linear Model

In discrete time, a linear system can be described by Equation 4.1.

\[ x_{i+1} = Ax_i + Bu_i \]  \hspace{1cm} (4.1a)
\[ y_i = Cx_i + Du_i \]  \hspace{1cm} (4.1b)

\( x_i \in \mathbb{R}^n \) is the state vector and \( u_i \in \mathbb{R}^m \) the control vector. In the case considered, there is no direct connection between the output and the input, i.e. \( D \) is a zero matrix. Equation 4.1a will be part of the MPC formulation as an equality constraint that has to be fulfilled for each step in the prediction horizon.
The basic formulation of the quadratic optimization problem, which is solved at every time sample is formulated as

$$\text{minimize } \sum_{i=0}^{\infty} (x_{k+i}^T Q x_{k+i} + u_{k+i}^T R u_{k+i})$$

subject to

$$x_{k+i+1} = Ax_{k+i} + Bu_{k+i}$$  \((4.2)\)
$$F_x x_{k+i} \leq b_x$$
$$F_u u_{k+i} \leq b_u$$
$$x_k = x(t).$$

It contains an objective function subject to the system dynamics equation and a number of inequality and equality constraints that shall be fulfilled for each time instant \(k > 0\). The current time point is denoted \(k\), and \(i\) corresponds to the prediction step. \(Q\) and \(R\) are called penalty matrices for the state and input. This formulation will minimize an infinite amount of terms, an implementation which is impossible. The sum is therefore divided into two terms making it more applicable.

$$\text{minimize } \sum_{i=0}^{N-1} (x_{k+i}^T Q x_{k+i} + u_{k+i}^T R u_{k+i}) + \sum_{i=N}^{\infty} (x_{k+i}^T Q x_{k+i} + u_{k+i}^T R u_{k+i})$$  \((4.3)\)

\(N\) is the prediction horizon, meaning that \(N\) time steps ahead are predicted. The second part of the expression is upper bounded by the function \(\Psi(x_{k+N})\). A heuristic approach is to write that as a penalty on the last time step \(x_{k+N}\), assuming that an LQ controller can be run at that point.

$$\sum_{i=N}^{\infty} (x_{k+i}^T Q x_{k+i} + u_{k+i}^T R u_{k+i}) \leq \Psi(x_{k+N}) = x_{k+N}^T P_N x_{k+N}. \quad (4.4)$$

The objective function is thus

$$V_k = \sum_{i=0}^{N-1} (x_{k+i}^T Q x_{k+i} + u_{k+i}^T R u_{k+i}) + x_{k+N}^T P_N x_{k+N}. \quad (4.5)$$

This function can be adjusted by adding cost variables or by varying the values in the matrices. It is an iterative work to obtain desired performance. The penalty matrices \(P_N\), \(Q\) and \(R\) should be real, symmetric matrices. \(P_N\) and \(Q\) are positive semi-definite while \(R\) is positive definite. The terminal state penalty, \(P_N\), is commonly chosen as the solution of the Ricatti equation commonly used for discrete LQ-problems.

$$A^T P_N A - P_N - A^T P_N B (B^T P_N B + R)^{-1} B^T P_N A + Q = 0 \quad (4.6)$$
The structure of the finite MPC problem is thereby

\[
\begin{align*}
\minimize_{u_{k+i}, x_{k+i}} & \sum_{i=0}^{N-1} (x_{k+i}^T Q x_{k+i} + u_{k+i}^T R u_{k+i}) + x_{k+N}^T P_N x_{k+N} \\
\text{subject to} & \\
& x_{k+i+1} = A x_{k+i} + B u_{k+i} \\
& F_x x_{k+i} \leq b_x \\
& F_u u_{k+i} \leq b_u \\
& x_k = x(t).
\end{align*}
\]

The result of the minimization is a series of optimal input values, \( u_{k+i} \), for all of the \( N \) following sample points. By repeating the process with a new \( x_0 \) and pushing the horizon forward in time at every sample, you get what is called a receding horizon controller. By defining \( U \) as the vector of \( N \) inputs \( \{u_i\}_{i=0}^{N-1} \) the optimal input sequence is denoted \( U^* \). When only applying the first value of \( U^* \) at each time instant \( k \) to the real system, a feedback controller is obtained.

From a computational perspective it is an advantage to have a short prediction horizon during the optimization since the computational time of the problem increases cubically with \( N \) [Wang and Boyd, 2008]. However, a longer prediction horizon can anticipate the future more accurately. A general guideline is to select the prediction horizon long enough such that a step response for a closed loop system can be made within the prediction time [Simon, 2017, p. 133].

Observe that normally another constraint of \( x_N \in T \) is a part of the optimization problem as a terminal constraint. It is necessary for guaranteeing stability of the closed loop system and motivates the LQ-based \( \Psi \). The region \( T \) has a complex formulation, especially with reference tracking, and is excluded in the implementation in this thesis.

### 4.2 Limitations

When setting up the structure of the MPC problem, the constraints \( b_x, b_u \) are primarily maxima and minima. The states \( p, q, r, \alpha, \beta \) along with the input signals \( \delta_{ea}, \delta_c, \delta_r \) are all restricted. The load factor \( n_z \) is another limitation, not present as a state but highly connected to \( \alpha \), see Section 2.1.1. In the methods used, the origin must be within the feasible region of all states and input signals.

### 4.3 Slack Variables

Model errors and disturbances of the system can, when being close to the limit, make the problem infeasible for next time step and hence no new control input can be calculated. One way to handle the infeasibility problem is to add a slack variable, \( \epsilon \), to the problem to soften the constraints. The slack variable is a non-
negative vector, \( \mathbb{R}^n \), added to the right hand side of the state constraints

\[
F x_i \leq b_x + \varepsilon_i \\
0 \leq \varepsilon_i ,
\]

(4.8)

where the size of the slack variables depends on the size of the associated violation of the constraints. Hence with a slack variable greater than zero, the constraints are softened and the problem becomes feasible.

The slack variables are also added, with a scalar weight, to the objective function to ensure that the slack is not used excessively.

\[
\minimize_{u_i,x_i,\varepsilon_i} \sum_{i=0}^{N-1} (x_i^T Q x_i + u_i^T R u_i + \gamma \varepsilon_i^T \varepsilon_i) + x_N^T P_N x_N
\]

(4.9)

The penalty of the slack variable is based on the 2-norm \( \|\varepsilon\|_2 \), but it is also possible to use \( \|\varepsilon\|_1 \) or \( \|\varepsilon\|_\infty \). If using a linear penalty function an exact relaxation is obtained, meaning that the constraints are relaxed only when necessary. In [Kerrigan and Maciejowski, 2000], it is discussed how to obtain this by letting the penalty weight \( \gamma \) being less than the Lagrange multiplier vector.

Slack is not added on the input constraints, considering that the input often originates from an actuator which has hard limits constraining the force, torque etc. Or as in this case, where the deflection surfaces can only rotate to a specific angle.

### 4.4 Reference Tracking

There are a few different approaches to obtain reference tracking. The objective function can be altered and/or new constraints added. [Rossiter, 2006] uses a pseudo reference while [Camacho and Alba, 2007] penalize the difference in input signal \( \Delta u \) between two consequent time steps. By using Equation 4.10 as objective function, instead of just minimizing the signals, the deviation from the reference values are forced to be as small as possible.

\[
\minimize_{u_i,x_i,\varepsilon_i} \sum_{i=0}^{N-1} (x_i - x_r)^T Q (x_i - x_r) + (u_i - u_r)^T R (u_i - u_r) \\
+ \gamma \varepsilon_i^T \varepsilon_i + (x_N - x_r)^T P_N (x_N - x_r)
\]

(4.10)

To obtain these values another equality constraint is introduced.

\[
\begin{bmatrix}
A - I & B \\
C & 0
\end{bmatrix}
\begin{bmatrix}
x_r \\
u_r
\end{bmatrix} =
\begin{bmatrix}
0 \\
r
\end{bmatrix}
\]

(4.11)

It is based on the fact that at steady state \( x_{k+1} = x_k = x_r \). The corresponding input that will maintain this state is called \( u_r \). The linear system has a unique solution when the number of inputs and outputs are the same. Else, if the system is underdetermined it can be solved as an optimization problem where the input signal should be minimized [Meadows and Badgwell, 1998].
4.5 Integral Control

Due to model errors, the output will have a nonzero offset from the reference with the model above. To avoid this problem, a constant disturbance term $\hat{w} \in \mathbb{R}^n$ is added to the model. An observer will be used in an outer loop to estimate the disturbances. With this the dynamical model will be

$$
\begin{align*}
    x_{i+1} &= Ax_i + Bu_i + \hat{w}_i \\
    y_i &= Cx_i \\
    \dot{\hat{w}}_{i+1} &= \hat{w}_i = \hat{w}.
\end{align*}
$$

(4.12)

The equality constraints in Equation 4.11 are also extended with the disturbance to:

$$
\begin{bmatrix}
    A - I \\
    C
\end{bmatrix} 
\begin{bmatrix}
    x_r \\
    u_r
\end{bmatrix} = 
\begin{bmatrix}
    -\hat{w} \\
    r
\end{bmatrix}
$$

(4.13)
This thesis is primarily focused on two different branches of MPC, explicit and fast MPC. The purpose is to reduce the time of the online calculations but still preserve the MPC characteristics. The time available to do all the control calculations for the aircraft is of the magnitude of 1/100 of a second, i.e. the computational time is critical for the problem. In explicit MPC, the solving of the optimization problem is moved offline, which results in a simpler search algorithm to be executed online. One explicit approximation is the neural network (NN) that can reproduce the behavior of a complex function. In fast MPC the optimization is instead solved online. To reduce the calculation time, smarter algorithms and approximations are implemented producing a similar result. Theories of the different branches are discussed in this section.

5.1 Explicit MPC

In explicit MPC, the entire control law is calculated offline which preserves the characteristics of the MPC solution. The result is later used as a look-up table, where all states have a corresponding optimal input signal on the form $z^*_k = F_i x_k + g_i$. The online computation at each time step is replaced with the process of finding the corresponding region and execute a simple linear function, instead of solving an expensive quadratic program. This gives a significant reduction of the computational time. A normal way to obtain the solution is to use the KKT conditions (see Section 3.1) where, for the feasible set of states, the optimal solution is obtained as piecewise affine functions divided into subsets depending on the active constraints. The following derivation in this section is based on [Bemporad et al., 2002].
The offline control law is solved with multi-parametric quadratic programming (MP-QP). By exploiting that \( x_{k+i+1} = Ax_{k+i} + Bu_{k+i} \) at each time step \( i \) and that \( x(t) \) is known, the influence of the future states can be eliminated with

\[
x_{k+i} = A^i x(t) + \sum_{j=0}^{i-1} A^j B u_{k+i-1-j},
\]

which only leaves a dependency on the input values \( u_{k+i} \) collected in the vector \( U \), and the current state \( x(t) = x_k \). The optimization problem in Equation 4.7 can then be expressed on a MP-QP form as

\[
V(x_k) = \frac{1}{2} x_k^T Y x_k + \min_U \frac{1}{2} U^T H U + x_k^T F U \\
\text{subject to } G U \leq W + E x_k,
\]

where the term \( \frac{1}{2} x_k^T Y x_k \) is seen as constant. Since \( x_k = x(t) \) is known at each time step \( k \) it will not be further considered, since it does not affect the minimization problem.

By defining \( z = U + H^{-1} F^T x_k \) and \( S = E + G H^{-1} F^T \), where \( H = B^T Q B + R \) and \( F^T = B^T Q A \), the problem can be restructured to:

\[
V_z(x_k) = \min_z z^T H z \\
\text{subject to } G z \leq W + S x_k,
\]

where \( V_z(x_k) = V(x) - \frac{1}{2} x_k^T (Y - FH^{-1} F) x_k \). It is here clear that the solution is only dependent on the current state and that the solution can be determined independent of time. The solution to the MP-QP is approached by employing the KKT conditions to the problem. Making the assumption that the corresponding rows of the matrix \( G \) are linearly independent, in the feasible domain of \( x_k \in \chi \), for all admissible combinations of active constraints. If \( H \) is positive definite and \( G \) fulfills the assumption above, let the corresponding set of vectors \( x_k \) with the same combination of active constraints be called a critical region \( \text{CR} \). Then the optimal \( z^* \) and the corresponding Lagrange multiplier \( \lambda \) is uniquely defined by an affine function of \( x_k \) over \( \text{CR} \), i.e. the solution \( z^* \) remains optimal in a neighborhood of \( x_k \) were the same set of constraints is active.

The KKT conditions for the MP-QP are stated as in Equation 3.2.

\[
Hz + G^T \lambda = 0, \quad \lambda \in \mathbb{R}^q
\]
\[
\lambda_i (G^T z - W^i - S^i x_k) = 0, \quad i = 1, \ldots, q
\]
\[
0 \leq \lambda
\]
\begin{align}
\tilde{G}H^{-1} \tilde{G}^T \tilde{\lambda} + \tilde{W} + \tilde{S}x_k &= 0 \quad \text{must hold, which gives the dual variable} \\
\tilde{\lambda} &= -(\tilde{G}H^{-1} \tilde{G}^T)^{-1}(\tilde{S}x_k + \tilde{W}),
\end{align}
\tag{5.5}

that is an affine function of \(x_k\). Substitution of \(\tilde{\lambda}\) from Equation 5.5 in the solution for 5.4a gives \(z^*_k\) as an affine function of \(x_k\) on the form \(z^*_k = F_i x_k + g_i\).

\begin{align}
z^*_k &= H^{-1} \tilde{G}^T (\tilde{G}H^{-1} \tilde{G}^T)^{-1}(\tilde{S}x_k + \tilde{W})
\end{align}
\tag{5.6}

The set of active constraints will result in different values of \(F_i\) and \(g_i\), hence \(z^*_k\) is dependent on the different critical region \(CR_i\).

The next step is to calculate the \(CR_i\) where the optimal solution is valid, since the KKT conditions only give a solution locally in the surroundings of \(x_k\). Using the knowledge that \(z^*_k\) has to fulfill Equation 5.4c and the primal feasibility constraint,

\begin{align}
\tilde{G}H^{-1} \tilde{G}^T (\tilde{G}H^{-1} \tilde{G}^T)^{-1}(\tilde{S}x_k + \tilde{W}) &\leq Sx_k + W
\end{align}
\tag{5.7}

is constructed and redundant inequalities deleted where the \(CR_i\) is a polytope in the feasible region. After defining one critical region, the remaining feasible region is partitioned into non-overlapping polytopic subsets. The partition can be made by switching the sign of one of the active constraints. In the newly defined polytopic subset a new optimal solution is calculated and a new \(CR_i\) determined.

To get the full explicit MPC controller is hence an iterative procedure. Algorithm 2, [Simon, 2017, p. 64], divides the entire feasible set into convex polytopic regions, see Figure 5.1 for an example of the final result.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure5_1.png}
\caption{The explicit solution of a problem with two states and one input. It has two saturated areas connected by a linear segment. Different colors show different regions with a certain optimal affine function \(z^*\).}
\end{figure}
Algorithm 2: The offline calculation creating an explicit MPC controller.

1: Select a state $x_k \in \chi$ inside the initial feasible constraint region.
2: Solve the KKT conditions.
3: Calculate $F_i$ and $g_i$.
4: Determine the active constraints.
5: Define the set $\chi_i$.
6: Divide the remainder of the feasible region $\chi$ in non-overlapping polytopic subsets, $\chi_i \cap \chi = \emptyset$.
7: Repeat from 1 with the divided regions as the initial set of constraints.

When the explicit solution has been created, the online computation consists of searching through all critical regions $\text{CR}_i$ and calculating the input signal given $F_i$ and $g_i$. Hence, the complexity of the online computation is highly dependent on the number of critical regions. This complexity is dependent on the number of constraints and the prediction horizon.

If several $\text{CR}_i$ have the same optimal solution $z_k^* = F_i x_k + g_i$, and if the union of the regions constitute a convex set, they can be merged together to reduce the complexity of the controller.

With reference tracking the model in Equation 5.2 is extended so that it depends on the deviation from the reference, $x_k - x_r$ and $U - U_r$ instead of the state $x_k$ and the input sequence $U$. The critical regions are then dependent on both current state, reference signal and the estimated disturbance, $\text{CR}_i(x_k, r, \hat{w})$, instead of only the current state $\text{CR}_i(x_k)$. The disturbance is needed for integral action and is added to the structure, which extends the complexity and dimension of the explicit controller.

### 5.1.1 Simplifications

To find the correct region based on the states, reference and disturbance is called the point location problem. [Bayat et al., 2011] and [Johansen and Grancharova, 2003] present search methods as hash tables and binary search trees by structuring the tables of control law in different ways and splitting the feasible region into larger areas. The binary search tree is often combined with a method of dividing the space into hyperboxes (or sometimes simplices) where the feasible region is split into equally large squares and the control law is evaluated at the edges for each partition. If they differ too much, the process repeats itself obtaining more and more regions. Structurally it has similarities with the branches of a tree and it avoids the problem of checking all constraints, making a trade-off between accuracy and speed. The method is more adapted for problems with constant maximum and minimum values for both states and inputs. Another way to simplify the partition is by merging adjacent regions with the same control law, that would though result in an NP-hard optimization problem [Kvasnica and Fikar, 2012] and it is still difficult to decrease the regions significantly.
5.1 Explicit MPC

5.1.2 Neural Network

A neural network (NN) is a nonlinear regression approximation model inspired by the biological network in the brain. It can be used to approximate complex functions and reproduce the behavior of a system. [Rojas, 1998] The idea is to make a replica of the MPC problem and to generate a function which can be used online.

The multilayer perceptron model (feed forward neural network with backpropagation) contains at least three layers of neurons connected with weighted lines, see Figure 5.2. From left to right are the input layer, hidden layer(s) and output layer.

![Figure 5.2: The structure of the neural network with input layer in green, hidden layers in yellow and output layer in red.](image)

The number of nodes in the input and output layer are equal to the number of inputs and outputs of the function or system. It is more difficult to choose the depth of the hidden layer, in most of the cases it is enough with two. No mathematically founded rule has been presented, but there exists a wide variety of rules of thumb empirically found, adapted for certain applications. Often it only depends on the input and output layers and not on the sample size, which is questioned by [Thomas et al., 2015]. [Huang, 2003] recommends a more narrow second layer for better performance. It is a trade-off, if there are few neurons in the network it gets underfitted and can not model the system adequately, neither the training set data nor an arbitrary point. Too many nodes will do the opposite and fit the points precisely, including noise and disturbances (called overfitting), and give a complex structure which is harder to process.

Figure 5.3 shows one single neuron with the total input $i_j$. It is the sum of the bias $\theta_i$ along with the sub-inputs $a_j$ weighted with $\omega_{ij}$, where $j$ runs from 1 to the number of nodes in the previous layer.
The input is passed through an activation function $F(\cdot)$ according to Equation 5.8 to get the output $o_i$. The function can be seen as a switch which determines if the node is activated or not.

$$o_i = F(i_i) = F\left(\sum_j w_{ij}a_j + \theta_i\right)$$  \hspace{1cm} (5.8)

The bias is needed to move the threshold when the activation function is triggered. It moves the function horizontally without changing its shape [Mohammed et al., 2017]. The activation function can be chosen freely, often a nonlinear function is used, primarily the sigmoid function

$$F(i_i) = \frac{1}{1 + e^{-i_i}},$$  \hspace{1cm} (5.9)

or the hyperbolic tangent function

$$F(i_i) = \frac{e^{i_i} - e^{-i_i}}{e^{i_i} + e^{-i_i}}.$$  \hspace{1cm} (5.10)

Considering that the explicit solution is piecewise affine the rectified linear unit (ReLU) is another reasonable alternative.

$$F(i_i) = \begin{cases} 
0, & \text{for } i_i < 0 \\
i_i, & \text{for } i_i \geq 0.
\end{cases}$$  \hspace{1cm} (5.11)

In fact any function can be approximated with the RELU, [Pascanu et al., 2013] suggest that a network with L hidden layers with n nodes and $n_0$ nodes in the input layer gives at least $O(L^{n_0}n^{n_0})$ linear regions. It is also fast to train due to the gradient which is either 0 or 1. The activation functions can be seen in Figure 5.4

When training the network, the weights are repeatedly changed to minimize the squared errors, see Equation 5.12, for a set of points called the training set, by using a backpropagation algorithm [Rojas, 1998].

$$E = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$  \hspace{1cm} (5.12)
5.1 Explicit MPC

\[ N \] is the size of the training set, \( \hat{y}_i \) the output generated by the network and \( y_i \) the correct output. The algorithm uses a gradient descent method, evaluating the partial derivative of the error with respect to the weights. Thus, it is harder to use activation functions which are not continuously differentiable. The initial weights are generally randomly selected from a normal distribution. The iteration of calculating the weighted sum \( i_i \), evaluating the error \( E \) and adjusting the weights \( \omega_i \) for a training set is called an epoch.

One large challenge for the neural network is to avoid overfitting. The risk of overfitting is large when using the same set for both training and validation, that in combination with an excess of nodes make the network approximate the training samples precisely but can not reproduce a general behavior. To avoid overfitting, the set is divided into three parts.

**Training set** - the data which the net uses to compute the gradient and update the weights and biases.

**Validation set** - determines when the training stops, based on when the squared error of this data ceases to decrease.

**Test set** - classifies the performance of a fully trained network.

This is called cross validation. The network stops training when the validation error has reached its minimum or if a specified number of epochs has been completed (early stopping). Another way to prevent the complication is to use more samples. See Figure 5.5 for training of an arbitrary system.
30 5 Model Predictive Control Techniques

Figure 5.5: The mean square error for a system with its training completed. It has stopped after 79 epochs finding its minimum after 59 iterations.

5.2 Fast MPC

Fast MPC is an online computation of MPC using one or several approximation methods to decrease the complexity and computational time. There are several approaches to achieve fast MPC, for example it is possible to exploit the structure of the problem to decrease computational effort, use thresholds allowing the controller to find a suboptimal solution or to use move blocking to limit the number of states. The latter is a way to simplify and limit the calculations by clustering consecutive input signals. The following section is based mainly on [Wang and Boyd, 2008].

5.2.1 Interior-Point Method with Approximations

The method is based on the problem formulation in Equation 4.2 with the modification of writing the inequality constraints on the form

\[ F_1 x_i + F_2 u_i \leq f_i, \quad i = 1, \ldots, N - 1. \]  (5.13)

Where \( F_1 \in \mathbb{R}^{l \times n}, \) \( F_2 \in \mathbb{R}^{l \times m}, \) \( f \in \mathbb{R}^l \) and the elements in vector \( f \) are strictly positive. If the constraints are purely maximum and minimum limitations, \( F_1 \) and \( F_2 \) only contain the values \([-1, 0, 1]\). The future states and reference signals are joined in a vector \( z \in \mathbb{R}^{mN+n(N-1)} \)

\[ z = [u_k, x_{k+1}, u_{k+1}, \ldots, x_{k+N-1}, u_{k+N-1}]^T \]  (5.14)

and the problem is rewritten to an alternative matrix form

\[
\begin{align*}
\text{minimize} & \quad z^T H z + g^T z \\
\text{subject to} & \quad P z \leq h \\
& \quad C z = b.
\end{align*}
\]  (5.15)
Where $H = \text{diag}(R, Q, R, \ldots, Q, R)$, i.e. a block diagonal quadratic matrix. Since there are no linear dependencies in the objective function in our case, $g$ is a column vector of zeros with the same length as $z$. It will however be kept in the formulation for later, when it is needed for the reference tracking extension. The rest of the matrices in 5.15 are as follows:

$$P = \begin{bmatrix} F_2 & 0 & 0 & \cdots & 0 & 0 \\ 0 & F_1 & F_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & F_1 & F_2 \end{bmatrix}$$ (5.16)

$$h = \begin{bmatrix} f - F_1 x_k \\ f \\ \vdots \\ f \end{bmatrix}$$ (5.17)

$$C = \begin{bmatrix} -B & I & 0 & 0 & \cdots & 0 & 0 \\ 0 & -A & -B & I & \cdots & 0 & 0 \\ 0 & 0 & 0 & -A & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & I & 0 \\ 0 & 0 & 0 & 0 & \cdots & -A & -B \end{bmatrix}$$ (5.18)

$$b = \begin{bmatrix} Ax_k + \hat{w} \\ \hat{w} \\ \vdots \\ \hat{w} \end{bmatrix}$$ (5.19)

The primal barrier method is used for solving the problem, see Section 3.3. The formulation is achieved by replacing the inequality constraint in Equation 5.15 with a penalty function:

$$\phi(x) = -\sum_{i=1}^{IN} \log(h_i - p_i^T z),$$ (5.20)

where $p_i^T$ and $h_i$ are the $i$:th row in matrix $P$ and vector $h$, respectively. The primal and dual residuals, $r_p$ and $r_d$, are constructed based on Equation 3.12.

$$r_d = 2Hz + g + \kappa P^T d + C^T v = 0$$

$$r_p = Cz - b = 0$$ (5.21)

Where $2Hz + g + \kappa P^T d$ is the derivative of the objective function, with $d = \frac{1}{h_i - p_i^T z}$.

By also calculating the second derivative, $\Phi = 2H + \kappa P^T \text{diag}(d)^2 P$, Equation 3.11
Model Predictive Control Techniques

\[
\begin{bmatrix}
\Phi & C^T \\
C & 0
\end{bmatrix}
\begin{bmatrix}
\Delta z \\
\Delta \nu
\end{bmatrix} = -
\begin{bmatrix}
r_d \\
r_p
\end{bmatrix}
\tag{5.22}
\]

Solving the equation above gives the primal and dual search step, \(\Delta z\) and \(\Delta \nu\). The task to determine the step size, \(s\), is done by Algorithm 1 in section 3.4.1 with \(r = [r_d^T, r_p^T]^T\).

### 5.2.2 Alternative Methods

Solving Equation 5.22 directly without exploiting the structure has a cost of \((1/3)N^3(2n + m)^3\) flops [Wang and Boyd, 2008]. The methods presented below are used to improve the speed for solving the problem. To make the Newton step calculations faster the search steps are updated by solving Equation 5.23:

\[
Y = C\Phi^{-1}C^T \\
\beta = -r_p + C\Phi^{-1}r_d \\
Y\Delta \nu = -\beta \\
\Phi\Delta z = -r_d - C^T\Delta \nu
\tag{5.23}
\]

The first two lines are based on that \(Y\) is the Schur complement [Horn and Zhang, 2005] to the matrix in Equation 5.22. The resulting \(Y\) is block tridiagonal, see Equation 5.24, where \(Y_{ij}\) are matrices \(n\times n\) large. These are symmetric, meaning \(Y_{ij} = Y_{ji}^T\), and it is thereby possible to obtain a simpler notation.

\[
Y = 
\begin{bmatrix}
Y_{11} & Y_{12} & 0 & \cdots & 0 & 0 \\
Y_{21} & Y_{22} & Y_{23} & \cdots & 0 & 0 \\
0 & Y_{32} & Y_{33} & \cdots & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & \cdots & Y_{N-1,N-1} & Y_{N-1,N} \\
0 & 0 & 0 & \cdots & Y_{N,N-1} & Y_{N,N}
\end{bmatrix}
\tag{5.24}
\]

The elements in the \(Y\) matrix can be computed more effectively than by multiplying \(Y = C\Phi^{-1}C^T\) the normal way by exploiting the structure. The inverse of the \(\Phi\) matrix (Equation 5.25) is diagonal with the same structure as \(H\), thus a lot of flops can be avoided by calculating the inverse block-wise. Algorithm 3 describes how this is done, only requiring an order of \(O(N(n + m)^3)\) flops.
\[ \Phi^{-1} = \begin{bmatrix} \tilde{R}_0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & \tilde{Q}_1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \tilde{R}_1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \tilde{R}_{N-1} & 0 \\ 0 & 0 & 0 & \cdots & 0 & \tilde{P}_N \end{bmatrix} \] (5.25)

**Algorithm 3** Algorithm for calculating the block matrices in Y.

1: function elements_of_Y(A, B, \( \Phi^{-1} \))
2: \( D_1 = B \tilde{R}_0 B^T + \tilde{Q}_1 \)
3: for \( i = 2, \ldots, N \) do
4: \( E_{i-1} = -A \tilde{Q}_{i-1}^T \)
5: \( D_i = A \tilde{Q}_{i-1} A^T + B \tilde{R}_{i-1} B^T + \tilde{Q}_1 \)
6: end for
7: end function

Instead of inverting \( Y \) when solving for \( \Delta \nu \), it is possible to Cholesky decompose it to a lower triangular matrix and its transpose, \( Y = LL^T \). The advantage is that backward and forward substitution can be used to obtain the solution of \( \Delta \nu \). For a general 3x3 matrix \( K \), the elements \( l_{ij} \) in \( L \) can be explicitly calculated with Equation 5.26.

\[
K = LL^T = \begin{bmatrix} l_{11}^2 & \text{symmetric} & \varepsilon_1 \\ l_{21} l_{11} & l_{22}^2 + l_{22}^2 & \varepsilon_2 \\ l_{31} l_{11} & l_{32} l_{21} + l_{32} l_{22} & l_{33}^2 + l_{33}^2 + l_{33}^2 \end{bmatrix} \] (5.26)

The Cholesky decomposition can be time consuming when having a dense matrix, but by taking advantage of the known sparse structure this can be prevented. When \( Y \) is tridiagonal the algorithm can be split into smaller problems, gaining a sparse lower matrix as in Equation 5.27 where all blocks have the same \( n \times n \) size, which requires \( O(Nn^3) \) flops.

\[
L = \begin{bmatrix} L_1 & 0 & 0 & \cdots & 0 & 0 \\ M_1 & L_2 & 0 & \cdots & 0 & 0 \\ 0 & M_2 & L_3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & L_{N-1} & 0 \\ 0 & 0 & 0 & \cdots & M_{N-1} & L_N \end{bmatrix} \] (5.27)

A pseudo code of the full decomposition is described in Algorithm 4. The function \( \text{chol}(\cdot) \) solves Equation 5.26 of necessary size, for a larger problem the structure looks the same.
Algorithm 4 Algorithm for the Cholesky decomposition of a tridiagonal block matrix.

1: function CHOLESKY_DECOMPOSITION(Y)
2:     \[L_1 = \text{chol}(D_1)\]
3:     for \(i = 1, N - 1\) do
4:         \[M_i = E_i L_i^{-T}\]
5:         \[L_{i+1} = \text{chol}(D_{i+1} - M_i M_i^T)\]
6:     end for
7:     return \(L\)
8: end function

Warm start is another method available where the choice of starting point is determined by known information about the problem. It is possible to use feasible points from earlier iterations like in Equation 5.28, where \(z^*_k\) is the computed optimal value for the previous optimization. The whole vector is shifted one time step to the left and filled with zeros at the empty spaces.

\[
z^*_k = \left[ \bar{u}_k^T, \bar{x}_{k-1}^T, \bar{x}_{k+1}^T, \ldots, \bar{x}_{k+N-2}^T, \bar{u}_{k+N-2}^T \right]^T
\]

\[
z_{\text{init},k} = \left[ \bar{u}_k^T, \bar{x}_{k+1}^T, \bar{x}_{k+N-1}^T, \bar{u}_{k+N-1}^T, 0, 0 \right]^T
\]

Normally, warm starting the IPM does not lead to any significant reduction of iterations due to \(\kappa\) which repeatedly changes. The solution usually lies close to the boundary, and in the consecutive step where the barriers have become less steep the objective value changes significantly. Changes in \(A\) and/or \(B\) between the time steps can even make the point unfeasible.

A way to solve this is to fix \(\kappa\) such that the iteration become a Newton process. Some downsides of fixed \(\kappa\) are that the exact optimal solution might not be found and that the number of steps can become large. In this application the first problem is not an issue, the solution only needs to approach the optimal value to obtain a well functioning controller. The second problem can be solved by limiting the number of iterations to \(K_{\text{max}}\). [Wang and Boyd, 2008] recommends a small value between 3-10, except at the first time step (when there is no previous trajectory to follow) where the value of \(K_{\text{max}}\) has to be increased. Also, by using warm start less Newton steps are required.
6 Implementation

The theories in Chapter 4 and 5 are in this chapter applied for a system resembling JAS 39 Gripen. The implementation of the linear MPC controller is presented first, which both the explicit and the fast MPC are based on, and will be compared to. Thereafter the implementation of the selected explicit and fast solvers are discussed. The design choices made for the methods in MATLAB are described with the intention to create a controller appropriate for the aircraft model. Necessary parameters will be tuned to get a smooth behavior similar to the original optimal setup. At last the methods are implemented in the ARES environment by building the system in Simulink and generating it into C-code.

6.1 Linear MPC

The aircraft model is a nonlinear system. To apply the linear theories from earlier chapters, the model is linearized in a number of envelope points, since the flight models (as in Equation 2.1 and 2.2) vary across the flight envelope. For the implementation, six envelope points with a velocity of 0.4, 0.6 and 0.8 Mach for an altitude of 1000 and 6000 m are used. Both the longitudinal and lateral model have to be trimmed for each envelope point. In the sections below, primarily the state space models at Mach 0.6 and altitude 6000 m are utilized if nothing else is stated.
6.1.1 Models & Penalty Matrices

Here the implemented dynamic models, penalty matrices and limitations are presented. $Q$ and $R$ are tuned individually for the longitudinal and lateral problems to achieve good performance for an implicit MPC controller tested on the linearized model. The slack penalty $γ$ was chosen in the magnitude of $γ = 10^4$, this high value ensures that the slack is not used excessively. The terminal state penalty $P_N$ was determined as the solution to the discrete Riccati equation 4.6.

Since MPC is solved in a receding horizon fashion, the MPC problem 6.5 is solved in 120 Hz that is the main frequency of the flight control system, i.e. $T_s = 1/120$ seconds.

**Longitudinal**

In MATLAB a discretized linear model was used for the longitudinal system according to

$$
\begin{bmatrix}
\alpha_{k+1} \\
q_{k+1}
\end{bmatrix} = 
\begin{bmatrix}
0.9916 & 0.0081 \\
0.0345 & 0.9908
\end{bmatrix}
\begin{bmatrix}
\alpha_k \\
q_k
\end{bmatrix} + 
\begin{bmatrix}
0 & -0.0037 \\
0.0519 & -0.1558
\end{bmatrix}
\begin{bmatrix}
\delta_{c,k} \\
\delta_{ea,k}
\end{bmatrix}.
$$

(6.1)

With the weight matrices $Q$ and $R$ chosen as

$$
Q = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0.1
\end{bmatrix} \quad R = \begin{bmatrix}
1 & 0 \ \\
0 & 1
\end{bmatrix}.
$$

(6.2)

This tuning gave a relatively quick response, while driving the attack angle to its reference and trying to keep the input signals low. The limits were $α \in [-8, 18]$, $q \in [-100, 100]$, $δ_c \in [-50, 25]$, $δ_{ea} \in [-25, 25]$ and $n_z \in [-3, 9]$.

**Lateral**

The linear model for the discretized lateral system used was

$$
\begin{bmatrix}
\beta_{k+1} \\
p_{k+1} \\
r_{k+1}
\end{bmatrix} = 
\begin{bmatrix}
0.9980 & 0.0005 & -0.0082 \\
-0.11819 & 0.99833 & 0.0044 \\
0.0308 & -0.0010 & 0.9965
\end{bmatrix}
\begin{bmatrix}
\beta_k \\
p_k \\
r_k
\end{bmatrix} + 
\begin{bmatrix}
-0.0005 & 0.0007 \\
0.4107 & 0.0405 \\
0.0392 & -0.0418
\end{bmatrix}
\begin{bmatrix}
\delta_{ea,k} \\
\delta_{r,k}
\end{bmatrix}.
$$

(6.3)

With the weight matrices $Q$ and $R$ chosen as

$$
Q = \begin{bmatrix}
100 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0.1
\end{bmatrix} \quad R = \begin{bmatrix}
5 & 0 \\
0 & 5
\end{bmatrix}.
$$

(6.4)

The limits were set as $β \in [-15, 15]$, $p \in [-170, 170]$, $r \in [-100, 100]$, $δ_{ea} \in [-25, 25]$, $δ_r \in [-25, 25]$ and $n_y \in [-1, 1]$. In fact it is the sum of the longitudinal and lateral control signal for the elevon which should lie within the range of ±25. For the implementation this will not be considered, since with the penalty matrices chosen there is a margin to the limit for all cases tested. See Section 8.4 for more discussion.
6.1 Linear MPC

6.1.2 MPC Model

The chosen MPC structure, Equation 6.5, is a basic linear MPC with the addition of reference tracking, integral action with disturbances and slack.

\[
\begin{align*}
\text{minimize} & \quad \sum_{i=0}^{N-1} (x_{k+i} - x_r)^T Q (x_{k+i} - x_r) + (u_{k+i} - u_r)^T R (u_{k+i} - u_r) \\
& + \gamma \varepsilon_{k+i}^T \varepsilon_{k+i} + (x_{k+N} - x_r)^T P_N (x_{k+N} - x_r)
\end{align*}
\]

subject to

\[
\begin{align*}
& x_{k+i+1} = Ax_{k+i} + Bu_{k+i} + \hat{w}_k + (I - A)\bar{x} - B\bar{u} \quad (6.5b) \\
& F_x x_{k+i} \leq b_x + \varepsilon_{k+i} \quad (6.5c) \\
& F_u u_{k+i} \leq b_u \quad (6.5d) \\
& 0 \leq \varepsilon_{k+i} \quad (6.5e) \\
& x_k = x(t) \quad (6.5f)
\end{align*}
\]

Where \( x_r \) and \( u_r \) are determined by

\[
\begin{bmatrix}
A - I & B \\
C & 0
\end{bmatrix}
\begin{bmatrix}
x_r \\
u_r
\end{bmatrix} =
\begin{bmatrix}
-\hat{w}_k - (I - A)\bar{x} + B\bar{u} \\
r
\end{bmatrix}.
\]

(6.6)

The prediction model (Equation 6.5b) is extended with the trim states \( \bar{x} \) and \( \bar{u} \) since the linearized models do not contain the actual state but the deviation from the linearization point.

\[
(x_{k+1} - \bar{x}) = A(x_k - \bar{x}) + B(u_k - \bar{u})
\]

\[
x_{k+1} = Ax_k + Bu_k + (I - A)\bar{x} - B\bar{u} \quad (6.7)
\]

Upper and lower limits of the load factor \( n_z \), that are depended on both \( x_{k+i} \) and \( u_{k+i} \) as described in Section 2.1.1, are also incorporated to the structure. To include the load factor constraints into the formulation, Equation 6.5c and 6.5d are combined into one as

\[
F_1 x_{k+i} + F_2 u_{k+i} \leq f. \quad (6.8)
\]

It will result in a problem with \( 3n + m \) inputs \((x_k, x_r, d_k, u_r)\) and \( m \) outputs \((u_k)\). It has \( N(n + m) - n + N(n + 1) = N(2n + m + 1) - n \) free variables, the first term contains all states and inputs except for \( x_k \) and the second term all the slack variables, one for each state and one for the load factor. In this case when the inequality constraints only concern the extreme values and the load factor, the total amount of constraints is \( 2N(n + m + 1) + Nn + N(n + 1) = N(4n + 2m + 3) \) corresponding to maximum and minimum values for \( x, u, n_z \) along with Equation 6.5b and 6.5e for all of the time steps.
Reference Tracking

Equation 6.6 is underdetermined if $B$ has more columns than $C$ has rows, i.e. the system has more input than output signals. Due to the non unique solution this can lead to strange behaviors of the input signals if the wrong combination is used. Since the reference, $r$, is constant during one iteration, Equation 6.6 can be solved separately outside of the MPC optimization problem. $x_r$ and $u_r$ can instead be estimated with the weighted norm minimization method (see Section 3.2) and substitute $r$ as input to the optimization problem. With the intention of minimizing the deviation from the trimmed input signals, the objective function in Equation 3.4 becomes

$$
\begin{align*}
\text{minimize} & \quad \frac{1}{2} \left\| \begin{bmatrix} 0 & 0 & x_r \\ 0 & I & u_r \end{bmatrix} - \begin{bmatrix} 0 & 0 & \bar{x} \\ 0 & I & \bar{u} \end{bmatrix} \right\|_p \\
\text{subject to} & \quad \begin{bmatrix} A - I & B \\ C & 0 \end{bmatrix} \begin{bmatrix} x_r \\ u_r \end{bmatrix} = \begin{bmatrix} -\hat{w}_k - (I - A)\bar{x} + B\bar{u} \\ r \end{bmatrix} \\
& \quad \begin{bmatrix} \bar{x} \\ \bar{u} \end{bmatrix} \end{align*}
$$

which is solved with Equation 3.7.

For the longitudinal model disturbances are added on both the angle of attack $\alpha$ and pitch rotation $q$. The reference is used for $\alpha$ and is divided into two parts, $r = \bar{\alpha} + \Delta\alpha_{cmd}$, where $\bar{\alpha}$ is the trimmed state and $\Delta\alpha_{cmd}$ the commanded reference from the pilot. When no command is received, i.e. the pilot has released the control stick, the aircraft shall level out and have no pitch rotation ($\alpha = \bar{\alpha}$ & $q = 0$). When solving Equation 6.9 for $r = \bar{\alpha}$, with the assumption of no or small disturbances, the input reference $x_r$ becomes

$$
x_r = \begin{bmatrix} \bar{\alpha} \\ \bar{q} \end{bmatrix},
$$

where $\bar{q} \approx 0$.

For the lateral model, disturbances are also added to all states. The varying reference is on $p$ but you also want to keep the sideslip angle $\beta$ close to zero.

Prediction Horizon

A normal step response for the pitch dynamics takes about two seconds, therefore a prediction horizon of $N = 240$ would be suitable. From simulations with the MPC controller for the linearized aircraft model, it shows that there was no noticeable loss in performance between a controller that uses a prediction horizon of $N = 240$ compared to $N = 60$, see Figure 6.1. When shortening the prediction horizon to $N = 20$ it is harder for the controller to handle constraints, see Figure 6.1 where the controller reaches the load factor constraint. The longer prediction horizons stops and holds $\alpha$ constant, but for $N = 20$ $\alpha$ continues to rise while the magnitude of the control signals increases in an unwanted way.
6.1 Linear MPC

To reduce the complexity of the problem and the calculation time, a prediction horizon of $N = 60$ is used in the following explicit and fast MPC implementations. The longitudinal problem thus has 8 inputs, 2 outputs, 900 constraints and 418 free variables and the lateral problem has 11 inputs, 2 outputs, 1140 constraints and 537 free variables.

### 6.1.3 Observer

To estimate the disturbance $\hat{w}$, a steady state Kalman filter is used.

$$
\begin{bmatrix}
\hat{x}_{k+i+1} \\
\hat{w}_{k+i+1}
\end{bmatrix}
= \begin{bmatrix} A & I \\ 0 & I \end{bmatrix}
\begin{bmatrix}
\hat{x}_{k+i} \\
\hat{w}_{k+i}
\end{bmatrix}
+ \begin{bmatrix} B \\ 0 \end{bmatrix} u_{k+i}
+ \begin{bmatrix}
(I-A) & -B \\ 0 & 0
\end{bmatrix}
\begin{bmatrix}
\bar{x} \\
\bar{u}
\end{bmatrix}
+ K (y - C \begin{bmatrix}
\hat{x}_{k+i} \\
\hat{w}_{k+i}
\end{bmatrix})
$$

(6.10)

Where $C$ has the form $[I 0]$ since all states but not the disturbance are measurable. The gain $K$ is based on the assumption that there is process noise on the estimated states $\hat{x}$ and $\hat{w}$ and observation noise on the output $y$. They are both white noises with zero mean and a covariance of $Q_K$ and $R_K$ respectively. To determine the matrices, $Q_K$ was varied while $R_K$ was kept as an identity matrix until a good behavior without oscillations was obtained. See [Glad and Ljung, 2003, p. 146-150] for the theory regarding Kalman filter.

Figure 6.1: A step response in $\alpha$ for the linearized aircraft model with an MPC controller for $N = 240, 60, 20$ for the envelope point with Mach 0.6 and altitude 1000 m. The canard is the control signal that starts at zero.
The longitudinal covariance matrices were chosen as

\[
Q_K = \begin{bmatrix}
0.001 & 0 & 0 & 0 \\
0 & 0.001 & 0 & 0 \\
0 & 0 & 0.1 & 0 \\
0 & 0 & 0 & 0.1 \\
\end{bmatrix}
\]

\[
R_K = \begin{bmatrix}
1 & 0 \\
0 & 1 \\
\end{bmatrix}.
\]  

(6.11)

6.2 Explicit MPC

To get an understanding of the problem size, an explicit solution was generated with YALMIP and MPT3 for a simplification of the longitudinal case. The problem had four input parameters consisting of two states \((\alpha, q)\), reference tracking and disturbance on \(\alpha\) (i.e. the critical regions are dependent on \(\text{CR}_i(\alpha, q, r_\alpha, \hat{w}_\alpha)\)) generating one output. For such a dynamical system with slack for the entire prediction horizon of \(N = 10\), the feasible set was divided into 37897 regions, with a total number of 127 constraints and 57 optimization variables.

For the same system as above but with \(N = 6\), the total number of constraints was reduced to 79 and the optimization variables to 33, the feasible set was divided into 6485 regions. As can be seen, the number of regions is highly correlated with the prediction horizon \(N\). Figure 6.2a and 6.2b illustrate the various regions, with the reference and disturbance parameters fixed to zero after the partition.

Creating an online solver directly for these regions would be problematic, especially when the polytopes are many, small and irregular. To find the correct region would take too much time when checking a number of inequality constraints for each partition in the list. Especially with 8 input parameters for the lateral system it seemed to be too complex to use binary search trees or hash tables. Further, it would only consider a limited prediction horizon. When trying \(N = 20\) the partition took more than a day to generate and the result was difficult

\[\text{Figure 6.2: The regions of the feasible set. To generate a graph, the reference and disturbance are fixed to } 0^\circ.\]
to process, so to get the desired prediction horizon of $N = 60$ is highly impractical. By merging adjacent regions with the same control law only a fraction of the regions would be eliminated. Also the methods with hyperboxes or simplices was discarded, the approaches are easier for problems with only box constraints which is not the case with the load factor.

The focus of interest was thereby to choose a method that could approximate the explicit solution to one or several functions. The most prominent was to use a neural network for reproducing the behavior, with the advantage of not needing to generate an explicit solution, being able to increase the prediction horizon and decrease the required data storage.

### 6.2.1 Neural Network

During the implementation of the NN, MATLAB’s neural network toolbox was used for the training. It gives, for instance, easy access to the linear regression for all the subsets and the progression of the mean squared error (Equation 5.12). The regression is a value of how well the output correspond to the target.

**Generating Data**

To generate the data set, points were selected as combinations of the states, reference and disturbances distributed in a hyper-box. They were limited by the extreme values of $x$, $r$ and approximated values which the disturbance would lie within. The points were generated from a quasi random sequence, which has low discrepancy i.e. the result was random points evenly distributed in space. See Figure 6.3 for the difference to random points in two dimensions. This decision was made when working with a smaller problem at the beginning, where the data points were evenly distributed in a grid. When then choosing a reference among the grid samples the steady state value was accurate, but for other references the NN had harder to adapt. Especially at the middle of an interval the steady state er-

![Figure 6.3: Data set of 150 points.](a) Random points. Some of the points are overlapping at the same time as there are large empty surfaces. (b) Quasi random points more evenly distributed.
ror grew larger. To receive an adequate result in that manner a more detailed grid had to be made, increasing the number of points drastically. By instead using the quasi random points, it was possible to keep the set smaller.

The optimization problem was solved point by point, and the first input \( u_k \) in the input vector saved. The data set was divided into sub-parts, 80 % of it was used for training and 20 % for validation. \( N \) was selected to 60, considering that one calculation should be made within a reasonable amount of time. If having the time though, it could easily be increased to hundreds of time steps. Generally, sets which had many points that demanded saturated control signals gave a worse result. The extreme values are easy to detect, instead you want to have more information about how the input affects the unsaturated areas.

**Network Design**

For the input layer, different parameter combinations were tried. At the beginning the parameters \( (x, r, \hat{w}) \) were chosen as the ones fed to the NN. That worked well if Equation 6.6 was of full rank, but since it does not have a unique solution the parameter \( r \) had to be replaced with \( x_r \) and \( u_r \). The output layer always consisted of \( m \) nodes.

As activation function, both linear and nonlinear functions were tested for the hidden layers (see Section 5.1.2 for equations). The difference between the sigmoid and hyperbolic tangent was minor, the number of epochs was similar and the error had the same magnitude. A major distinction appeared when the layers contained the rectifier, only a fraction of the epochs were used. See Figure 6.4 for the regression when the training was stopped after 20 epochs. Considering that the explicit solution is piecewise affine, it is reasonable to use the piecewise

![Figure 6.4](image)

*(a) RELU.*

*(b) Hyperbolic tangent.*

**Figure 6.4:** The regression for two different activation functions when stopped early. The target values are compared with the outputs of the neural network. The hyperbolic tangent is slower to train and is not as fitted at this epoch.
affine RELU function. With a hyperbolic function it is harder to adjust to the linear problem and the saturated regions are as in Figure 6.4 not as fitted and can be seen as vertical lines of data points.

Consequently to determine the size of the layer, the process started with only a few nodes. More and more were then added until no major improvement was seen. The lateral problem is bigger with more inputs and therefore a bigger data set and more nodes were used to obtain satisfactory performance.

Training was made for all envelope points which resulted in a function where the weights were changed depending on which altitude and velocity the aircraft is flying at. Figure 6.5 shows a schematic image of the NN function. The neural network function then contains information about the dynamical matrices and extreme values at each envelope point.

### 6.3 Fast MPC

To follow a reference trajectory, some changes had to be made with the matrices derived in Section 5.2. The vector \( z \) is elongated with the extra time step \( x_{k+N} \) and a new vector \( z_r = [u_r, x_r, \ldots, u_r, x_r]^T \) is introduced, where \( x_r \) and \( u_r \) are the solution to Equation 6.6. \( P_N \) is put as the last block element of the diagonal matrix \( H \) to penalize the final state. The minimization will then be of the expression

\[
(z - z_r)^T H (z - z_r) = z^T H z - 2 z_r^T H z + z_r^T H z_r
\]  

(6.12)

assuming \( H \) is symmetric. The last term is constant and can thereby be removed since it does not affect the minimization. Equation 6.12 can then be compared with Equation 5.15, obtaining \( g^T = -2 z_r^T H \). In \( P \), another column with zeros is added to the right, the same is done in \( C \) but with an \( n \times n \) identity matrix at the bottom. The vectors \( h \) and \( b \) remain unchanged. Due to the changes in \( H \), \( \Phi^{-1} \) will lose some of its structure at the last elements (if \( P_N \) has cross couplings), where an \( n \times n \) block is added that has to be solved separately.

When deciding on a fixed value of \( \kappa \), the algorithm was run repeatedly, starting with a large value and thereafter decreasing with a tenth for each step until it was nearly identical to the original controller.
Another design parameter to determine was $K_{\text{max}}$. If using a too low value, the equality constraints will not be correct which results in an infeasible solution and the optimized input signal can push the states beyond the limitations. Often there are no physical limits on the states, thus this will not give any major problem in the system but extra precautions have to be implemented in the software for mixed constraints when using warm start. Note that, unlike for the explicit MPC, the structure of the fast MPC algorithm is not designed with slack.

The implementation of warm start has to be altered slightly. For initial feasibility in time step $k + N - 1$, the last state vector has to be replaced with zeros, as in Equation 6.13, when having combined constraints of $x$ and $u$.

$$z_{k-1}^* = [\bar{u}_k^T, \bar{x}_{k-1}^T, \ldots, \bar{x}_{k+N-2}^T, \bar{u}_{k+N-2}^T, \bar{x}_{k+N-1}^T]^T$$

$$z_{\text{init},k} = [\bar{u}_k^T, \bar{x}_{k+1}^T, \ldots, \bar{x}_{k+N-2}^T, \bar{u}_{k+N-2}^T, 0, 0, 0]^T$$  \hspace{1cm} (6.13)

### 6.4 ARES

The ARES model describes the nonlinear behavior of an aircraft including e.g. pilot models, sensors, servo dynamics and affecting surrounding factors. The longitudinal and lateral controller are designed separately, and are as default LQ controllers. The MPC solvers have only been implemented for longitudinal movement. To run the MPC controller in the ARES model, several parameters have to be set. They can all be seen in Table 6.1 together with their description. For all images of ARES simulations, the MPC function is activated when taking a step at 2 seconds. The implementation in ARES is generated from a Simulink environment.

**Table 6.1:** Parameters for using the MPC controller in ARES. All with the default value of 0.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sw_MPC</td>
<td>Switch to activate the MPC, ON = 1 and OFF = 0.</td>
</tr>
<tr>
<td>sw_NN</td>
<td>Switch to choose between NN = 1 and fast MPC = 0.</td>
</tr>
<tr>
<td>sw_kal</td>
<td>Switch to choose if the Kalman filter shall estimate one = 0 or two = 1 disturbances.</td>
</tr>
<tr>
<td>sw_warm</td>
<td>Switch to choose between warm start = 1 or cold start = 0.</td>
</tr>
<tr>
<td>Kmax</td>
<td>Sets the value of $K_{\text{max}}$.</td>
</tr>
<tr>
<td>kappa</td>
<td>Sets the value of $\kappa$.</td>
</tr>
</tbody>
</table>

### 6.4.1 Simulink Layout

The controllers and the observer are created as subsystems that are composed of Simulink blocks or MATLAB functions and then generated to C-code for the ARES simulations. Figure 6.6 shows that the MPC controller consist of three main blocks; neural network, fast MPC and Kalman filter. The line going into the blocks, coming from the rest of the model, is a vector with the measured...
input containing the states and pilot command. The lines going out from the controllers are the output signals continuing to the servos. This implementation allows the user to switch between the fast controller and the neural network. The Kalman filter is used to observe the disturbances that will be used as inputs to the controllers.

Figure 6.6: Simulink implementation of the MPC controller. Alfa_Cmd is the reference signal commanded from the pilot.
Since the linearized matrices used in both the fast and the explicit (NN) solution were generated only in the envelope points, linear interpolation of the elements in the matrices was done to get a good approximation. The interpolation was made between the four envelope points nearest the current Mach number and altitude. Outside the mapped region the closest value was chosen (no extrapolation).

### 6.4.2 Neural Network

Figure 6.7 shows the implementation of the neural network. First $x_r$ and $u_r$ are calculated to obtain the right input signals for the network. Like for the state space matrices, linear interpolation of the neural network output was made by using the outputs from the four nearest envelope points and interpolate first in altitude and thereafter in velocity. $col$ and $row$ are used to activate the correct NNs in the function block which contains the weights and biases for all envelope points generated in MATLAB.

*Figure 6.7: Simulink implementation of the neural network controller. Port 1 and 2, represent the states.*

### 6.4.3 Fast MPC

Figure 6.8 shows the Simulink implementation of the fast MPC controller. The loop is used to save the latest $z$-vector for the warm start. Originally the function took seconds, thus the implementation started with measuring the computational time for the fast MPC without simplifications, to identify which parts that slowed it down. The main problem was the matrix multiplications $P^T \text{diag}(d)^2 p$ and $C\Phi^{-1}C^T$. Due to the sparse/diagonal structure of the matrices, only smaller
blocks had to be multiplied decreasing the time by a factor 1000. Thereafter it was possible to distinguish the improvements in time using the alternative methods discussed in Section 5.2.2.

Figure 6.8: Simulink implementation of the fast MPC controller with warm start.
6.4.4 Kalman Filter

In the ARES model, the penalty matrices for the Kalman filter were set to

\[ Q_K = \begin{bmatrix} 10^{-3} & 0 & 0 & 0 \\ 0 & 10^{-6} & 0 & 0 \\ 0 & 0 & 10^{-1} & 0 \\ 0 & 0 & 0 & 10^{-6} \end{bmatrix}, \quad R_K = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}. \quad (6.14) \]

The Kalman penalty matrices were changed since the controller became too aggressive with the matrices as in Section 6.1.3, see Figure 6.9.

![Figure 6.9: A step response with a too aggressively tuned Kalman filter. The penalty matrices from Equation 6.11 in Section 6.1.3.](image-url)
In this chapter the primary results are presented. The controllers are both tested in MATLAB and in the nonlinear ARES model. First the original MPC is evaluated with the observer and then tables and figures describing the choice of tuning variables and parameters are shown. The computational time and performance for the explicit and fast implicit controllers are measured and the final step responses are displayed to later on be compared and discussed. The chapter ends with an appendix containing graphs with a variety of variables when making a step for the angle of attack, it shows for example how the speed and altitude changes with time.

### 7.1 MATLAB Model

Figure 7.1 shows that the system behaves differently dependent on envelope point with the same weight matrices. A lower Mach number gave a 1.5 times longer closed loop rise time and also a slight overshoot.

The performance of the observer is shown in Figure 7.2 run with the original MPC problem. Random disturbances were added to the state space model with a mean of $\mu_w = [0.05, 0.2]^T$ and a variance of $\sigma_w^2 = [10^{-5}, 10^{-4}]^T$. With Kalman filter $\alpha$ reaches the right steady state value although a bit noisy. Without a filter approximating the disturbances, the corresponding step response had a steady state value of $4.1^\circ$. 
**Figure 7.1:** The step responses to $3^\circ$ for different velocities on an altitude of 6000 m with the same controller. The canard wing is the control signal that start at a positive value.

**Figure 7.2:** A desired step response for $\alpha$ with disturbances on both states. The observer is displayed in magenta and the real values in black.
7.1.1 Neural Network

Table 7.1 shows that the number of iterations and time training are clearly higher when using the hyperbolic tangent function. Increasing the data set significantly does not give any improvement, rather the opposite. The reason was that the number of nodes was kept constant when generating the table. If the data set grows larger, the number of hidden nodes should also be increased. From the table it is seen that a set with 20 000–100 000 data points gives the best result for the longitudinal case. Considering that a lot of configurations were going to be tested, a set of 20 000 data points was chosen, giving an adequate result while still being fast to train.

**Table 7.1:** A comparison of the NN values at steady state for different sizes of data sets with the RELU and the hyperbolic tangent as activation functions.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Activation Function</th>
<th>Steady state $\alpha$ [deg]</th>
<th>Time [h:min:sec]</th>
<th>Epochs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>-1.4 0 5.2 8.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 000</td>
<td>RELU</td>
<td>-1.42 0.00 5.12 8.27</td>
<td>1:40</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>Hyp. Tang.</td>
<td>-1.44 -0.07 5.14 8.32</td>
<td>4:17</td>
<td>62</td>
</tr>
<tr>
<td>20 000</td>
<td>RELU</td>
<td>-1.40 0.01 5.19 8.29</td>
<td>7:14</td>
<td>108</td>
</tr>
<tr>
<td></td>
<td>Hyp. Tang.</td>
<td>-1.37 0.01 5.17 8.30</td>
<td>11:54</td>
<td>160</td>
</tr>
<tr>
<td>50 000</td>
<td>RELU</td>
<td>-1.40 0.01 5.20 8.30</td>
<td>16:49</td>
<td>117</td>
</tr>
<tr>
<td></td>
<td>Hyp. Tang.</td>
<td>-1.38 -0.01 5.18 8.30</td>
<td>27:17</td>
<td>221</td>
</tr>
<tr>
<td>100 000</td>
<td>RELU</td>
<td>-1.39 0.01 5.20 8.30</td>
<td>38:41</td>
<td>177</td>
</tr>
<tr>
<td></td>
<td>Hyp. Tang.</td>
<td>-1.41 -0.01 5.21 8.32</td>
<td>3:17:04$^1$</td>
<td>800$^1$</td>
</tr>
<tr>
<td>200 000</td>
<td>RELU</td>
<td>-1.41 -0.05 5.20 8.29</td>
<td>28:00</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>Hyp. Tang.</td>
<td>-1.39 0.00 5.22 8.31</td>
<td>3:39:20</td>
<td>578</td>
</tr>
</tbody>
</table>

As can be seen in Table 7.2, using only one hidden layer is not sufficient with respect to regression and error. The usage of two hidden layers clearly give a better result, probably due to the large data set of 20 000 samples. Three layers gave the best result in regression but it takes time to train and when making step responses the difference is minor compared to just using two layers, see Figure 7.3.

$^1$The training made an early stop, limited at 800 epochs.
Table 7.2: A comparison between different number of neurons and hidden layers.

<table>
<thead>
<tr>
<th>Layers</th>
<th>Nodes</th>
<th>Time [min:sec]</th>
<th>Iterations</th>
<th>Regression Training</th>
<th>Regression Validation</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[10]</td>
<td>0:23</td>
<td>71</td>
<td>0.9939</td>
<td>0.9937</td>
<td>1.722</td>
</tr>
<tr>
<td>1</td>
<td>[30]</td>
<td>7:24</td>
<td>126</td>
<td>0.9979</td>
<td>0.9979</td>
<td>0.565</td>
</tr>
<tr>
<td>1</td>
<td>[60]</td>
<td>10:30</td>
<td>140</td>
<td>0.9990</td>
<td>0.9987</td>
<td>0.360</td>
</tr>
<tr>
<td>2</td>
<td>[20 10]</td>
<td>7:58</td>
<td>81</td>
<td>0.9995</td>
<td>0.9989</td>
<td>0.304</td>
</tr>
<tr>
<td>2</td>
<td>[25 20]</td>
<td>5:46</td>
<td>85</td>
<td>0.9997</td>
<td>0.9995</td>
<td>0.135</td>
</tr>
<tr>
<td>2</td>
<td>[30 25]</td>
<td>6:10</td>
<td>51</td>
<td>0.9997</td>
<td>0.9996</td>
<td>0.123</td>
</tr>
<tr>
<td>2</td>
<td>[50 30]</td>
<td>35:34</td>
<td>45</td>
<td>0.9997</td>
<td>0.9997</td>
<td>0.094</td>
</tr>
<tr>
<td>3</td>
<td>[25 20 15]</td>
<td>29:47</td>
<td>126</td>
<td>0.9999</td>
<td>0.9998</td>
<td>0.041</td>
</tr>
</tbody>
</table>

To some extent, more neurons generally give a better fit, especially when using relatively large sets. Figure 7.4 shows the regression when insufficient nodes in the hidden layers were used, where the fully trained NN has issues fitting the saturated areas. At the same time, using many neurons could make the network too complex and the training would take a lot longer.

Figure 7.3: The minor difference of using three hidden layers compared with two.
Figure 7.4: The regression when not using enough nodes. The negative saturation is especially hard to fit appropriately.

Longitudinal

For the longitudinal matrices in Section 6.1.1 with $N = 60$, 20 000 data points, RELU used as activation function, 25 nodes in the first hidden layer and 20 in the second, the step response is presented in Figure 7.5. For a reference of $3^\circ$ without any disturbances there is no steady state error.

Figure 7.5: Step response for the neural network without disturbances.
As a mean it took approximately 22 minutes to generate one set of data and 4 minutes to train the NN. The training though, could vary a lot, it took between 2-20 minutes mainly depending on the envelope point trained but even the same set of data could differ.

**Lateral**

For the lateral matrices in Section 6.1.1 with $N = 60$, a data set of 50 000 points, RELU used as activation function, 50 nodes in the first hidden layer and 30 in the second, the step response is presented in Figure 7.6. When the reference on the roll rotation $p$ is $5^\circ/s$ the rotational speed becomes only $0.02^\circ/s$ wrong while the sideslip angle $\beta$ reaches exactly $0^\circ$. Thus the neural network training also works for larger problems that are nearly impossible to generate a explicit piecewise affine function of.

![Figure 7.6: A lateral step response, where the reference for $p$ is set to $5^\circ/s$ and $\beta$ to $0^\circ$. The elevon is the control signal starting with a positive value. The simulation is started at the quasi stationary state of $\beta = 5$, $p=0$, $r=0$.](image)

**7.1.2 Fast MPC**

Simulations of a step with alternating values of $\kappa$ shows differences from the exact solution when starting with $\kappa = 10$ (Figure 7.7a). Reduced to $\kappa = 0.1$ the step response resembled the exact solution well (Figure 7.7b). If choosing a relatively high value more iterations are required to find a solution than for a smaller one. It can however give some improvement in speed when being close to an edge of the feasible set.
Figure 7.7: Step in the angle of attack with a reference of 3° for a perfect model for different $\kappa$. The control signal of the canard wing is the input with a steady state value of 0°.
Figure 7.8: The difference of limiting \( K_{\text{max}} \) to 3 compared to letting the algorithm find the relaxed optimum. Even with a restricted number of iterations the solution resembles the optimal solution.

The value of \( K_{\text{max}} \) can be set quite low as there is no major distinction between using \( K_{\text{max}} = 3 \) and not restricting the number of iterations for a linear system, see Figure 7.8. Table 7.3 shows that there is a slight reduction in needed iterations for warm start compared to cold start when the load factor limitation is disregarded. Only at high references a difference starts to show. With the load factor, the required number of iterations increases, especially at the boundaries (a reference of 9.0° here hits the load factor limitation). At the boundaries with \( n_z \) warm and cold start does not matter significantly from a time perspective. The cold start is a bit faster due to the multiplications with the \( z \) vector which MATLAB computes faster when it is filled with zeros. A larger contrast between cold and warm start would have been noticeable if \( K_{\text{max}} \) was larger than 20 iterations.

Table 7.3: The number of iterations and time taken for cold/warm start with and without the load factor constraint. \( K \) is limited to \( K_{\text{max}} = 20 \). The reference of \( \alpha \) is set to 1.0° and 9.0° where it exceeds the \( n_z \) limitation).

<table>
<thead>
<tr>
<th>Reference</th>
<th>( K_{\text{mean}} ) [it]</th>
<th>Time/it [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.0</td>
<td>9.0</td>
</tr>
<tr>
<td>Without ( n_z ) constraint</td>
<td>Cold start</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Warm start</td>
<td>1</td>
</tr>
<tr>
<td>With ( n_z ) constraint</td>
<td>Cold start</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Warm start</td>
<td>2</td>
</tr>
</tbody>
</table>
For the fast MPC with $N = 60$, $K_{\text{max}} = 3$, $\kappa = 0.1$, $\beta = 0.5$, $\alpha = 0.01$ and warm start, the step response can be seen in Figure 7.9 where it shows that there is no visible difference from the original MPC controller.

![Figure 7.9: A step response in $\alpha$ for the fast MPC without disturbances.](image)

### 7.1.3 Comparison

In Table 7.4 the steady state values are tabulated for different references at two of the envelope points. The system for the altitude 6000 m has, unlike the one at the altitude of 1000 m, unstable poles. When solving the original problem without any approximations, the error tends toward zero in all cases except for reference $8.3^\circ$ at Mach 0.8. At this envelope point the load factor limit is more easily reached, already at $\alpha = 7.72^\circ$ the load factor reaches $n_z = 9.01$ (slack is used). For that reference the fast MPC is further away from the right value caused by the absence of slack in the design. The NN performs slightly better in all the cases, especially at Mach 0.8 where the steady state error is up to 2 %. A similar table but with disturbances included can be seen in Table 7.5. It can be concluded that the disturbances do not affect the performance noticeably, except for the neural network at $8.3^\circ$ where the network is not trained sufficiently for disturbances in the saturated region. The errors are all sufficiently small, some sensors and actuators do not even have such high sensitivity.
Table 7.4: A comparison of the methods at steady state. \( m \) stands for Mach and \( h \) for altitude.

<table>
<thead>
<tr>
<th>Steady state ( \alpha ) [°]</th>
<th>(-1.4)</th>
<th>0</th>
<th>5.2</th>
<th>8.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural Network</td>
<td>m0.6 h6000</td>
<td>-1.40</td>
<td>0.00</td>
<td>5.20</td>
</tr>
<tr>
<td></td>
<td>m0.8 h1000</td>
<td>-1.41</td>
<td>-0.01</td>
<td>5.20</td>
</tr>
<tr>
<td>Fast</td>
<td>m0.6 h6000</td>
<td>-1.40</td>
<td>0.00</td>
<td>5.20</td>
</tr>
<tr>
<td></td>
<td>m0.8 h1000</td>
<td>-1.37</td>
<td>0.00</td>
<td>5.19</td>
</tr>
</tbody>
</table>

Table 7.5: A comparison of the methods at steady state with the disturbances \( w = \hat{w} = [0.05, \ 0.2]^T \).

<table>
<thead>
<tr>
<th>Steady state ( \alpha ) [°]</th>
<th>(-1.4)</th>
<th>0</th>
<th>5.2</th>
<th>8.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural Network</td>
<td>m0.6 h6000</td>
<td>-1.40</td>
<td>0.00</td>
<td>5.20</td>
</tr>
<tr>
<td></td>
<td>m0.8 h1000</td>
<td>-1.41</td>
<td>-0.01</td>
<td>5.20</td>
</tr>
<tr>
<td>Fast</td>
<td>m0.6 h6000</td>
<td>-1.40</td>
<td>0.00</td>
<td>5.20</td>
</tr>
<tr>
<td></td>
<td>m0.8 h1000</td>
<td>-1.38</td>
<td>0.00</td>
<td>5.19</td>
</tr>
</tbody>
</table>

7.2 ARES Model

Figure 7.13 in Appendix 7.A shows a variety of variables measured when taking a smaller and a larger step. These are generated with the fast MPC controller at Mach 0.8 and altitude 6000 m with \( N = 60 \). This shows how the aircraft changes envelope points and still follows the reference accurately (similar behavior can be seen with the neural network controller).

7.2.1 Neural Network

In Figure 7.10 the NN controller is taking steps of different amplitudes. Figure 7.14a shows a step and then what happens when releasing the stick going back to trim state. The total calculation time per iteration for the neural network was \( t_{NN} \in [17 \cdot 10^{-6}, \ 63 \cdot 10^{-6}] \) with a mean of 21.8 \( \mu s \).
7.2.2 Fast MPC

In Table 7.6 the computational time in ARES for the different methods are listed. To the left are the measurements for the built in MATLAB commands generated into C-code and to the right are the alternative methods from Section 5.2.2. The time for the matrix multiplication to form $Y$ is included in the decomposition. The alternative methods have decreased the time with 99.9 % for the Cholesky decomposition and 82.7 % for the inverse when $N = 60$. When decreasing the prediction horizon with a factor 6, the time for the built in commands decreases with $O(N^3)$ for both the $\Phi$ inverse and the Cholesky decomposition. The alternative inverse has a quadratic growth, not due to inverting the diagonal but due to forming the matrix with a nested for-loop. The Cholesky decomposition grows linearly with $N$, as concluded in [Wang and Boyd, 2008].
Table 7.6: The improvement with the alternative methods. Time measured in microseconds (µs).

<table>
<thead>
<tr>
<th>N</th>
<th>Built-In Command</th>
<th>Alternative Method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min</td>
<td>Max</td>
</tr>
<tr>
<td>Φ Inverse</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>1372</td>
</tr>
<tr>
<td>Cholesky</td>
<td>10</td>
<td>271</td>
</tr>
<tr>
<td>Decomposition</td>
<td>60</td>
<td>67844</td>
</tr>
</tbody>
</table>

The total time in ARES with all methods included is shown in Table 7.7. The original version uses the built-in MATLAB command for the inverse and Cholesky decomposition. Parameters are set to $K_{\text{max}} = 3$ and $\kappa = 1$, and the reference to $18^\circ$ which is the maximum value of the angle of attack. This was chosen to see the difference in computational time when $\alpha$ gets close to the limit compared to when the controller is working in a region far from the borders at the beginning of its step.

The total time of the original version of the fast MPC has a cubic growth, $O(N^3)$, while the adjusted implementation reduces to a quadratic dependence of the prediction horizon, $O(N^2)$. There is a noticeable difference in time using warm and cold start when in the middle of region, i.e. no state or load factor constraints are active. At the limit, the calculation time for using warm start is only slightly better. In total at least 89% of the mean time is eliminated with the adjusted warm start when $N = 60$.

Table 7.7: The time in microseconds (µs) taken for each iteration. $K_{\text{max}} = 3$, $\kappa = 1$.

<table>
<thead>
<tr>
<th>N</th>
<th>Middle of Region</th>
<th>At Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min</td>
<td>Max</td>
</tr>
<tr>
<td>Original</td>
<td>10</td>
<td>1152</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>157676</td>
</tr>
<tr>
<td>Adjusted</td>
<td>10</td>
<td>550</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>18224</td>
</tr>
<tr>
<td>Adjusted w. Warm Start</td>
<td>10</td>
<td>549</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>18238</td>
</tr>
</tbody>
</table>
This concluded that a shorter prediction horizon was needed to reach the sample time of a couple of milliseconds. With $N = 10$ the maximum time was beneath the limit but this resulted in a violation of the constraints, see Figure 7.11 with the same tendencies as in Figure 6.1. The increasing reference around $4^\circ$ depends on the speed, which is decreasing and the quite strange behavior of the larger reference depends on the load factor which reaches its maximum value, see Figure 7.12.
Figure 7.12: The controller hitting the load factor limit, when $\alpha = 18$.

For the fast MPC implementation the total calculation time per iteration was for $N = 60$, $t_{fast} \in [18238 \cdot 10^{-6}, \ 28424 \cdot 10^{-6}]$ where the average time per iteration changes if the constraints are active or not. A reduction of the prediction horizon to $N = 10$ gave $t_{fast} \in [549 \cdot 10^{-6}, \ 2032 \cdot 10^{-6}]$. The calculation time then never exceed 2 ms, but had a loss in performance and accuracy. Since it is important to have short online calculation time $N = 10$ was chosen as the prediction horizon for the fast MPC implementation in ARES and the resulting step response can be seen in Figure 7.14b.
Appendix

7.A Step Responses Ares
Figure 7.13: Fast MPC, $N = 60$. With an initial Mach of 0.8 and altitude of 6000 m.
Figure 7.14: The controllers following a reference and then returns to trim state and \( q = 0 \). With Mach 0.8 and altitude 6000 m.
This chapter discusses the methods and results from this project and contain a conclusion as an answer to the problem statements. It ends with some recommendations of how to continue the work and which pitfalls to avoid in the future.

8.1 Choice of Method

In this thesis, a linear interpolation between the envelope points was used for the matrices in the aircraft model. The relationship when altering the velocity or altitude is nonlinear, but a linear interpolation is assumed to be good enough. One drawback is that a greater model error for some envelope point will affect the flight behavior everywhere around that point. No extrapolation was made assuming that a linear relationship can not continue to be valid. When reaching a point outside the region, the matrices will retain the values of the matrices from the nearest envelope point which will lead to an increase of the disturbances. Too large disturbances, even though estimated correctly, could be hard for the NN controller to handle accurately if not trained for it.

To get a more correct view of how the aircraft actually behaves, more envelope points have to be used. This will reduce the problems that could arise with interpolation, since a denser grid gives smaller gaps and with small enough gaps a linear relationship is valid. Adding more envelope points will not entail any further changes in the algorithms than that the matrices have to be tabularized. Weights and biases are trained separately for all envelope points in the NN and added to the code.

The weight matrices $Q$ and $R$ were kept constant at all envelope points. In most flight conditions this gave a good response, but at certain velocities it resulted in
an overshoot as in Figure 7.1. It is clear that the weight matrices have to change for the different envelope points, though the aircraft should give consistent responses, independent of flight condition, when the pilot gives a certain command. Also the Kalman filter could have been tuned better, the reason for using a lower gain in ARES was to avoid oscillations as the ones seen in Figure 6.9. A model error affecting the disturbance of $q$ made the observer change too fast with the former weights.

A lot of time has been spent on determining the steady state values $x_r$ and $u_r$. The method chosen has performed nicely in the tested cases but with other weights and combination of states, the constraints can easily be violated. Further, for the current formulation it is assumed that there are no disturbances on the measured signals. If present, it affects $x_r$ which would differ from the reference. The weight matrix $P$ could be chosen more carefully. By using the identity matrix as the weight for $u_r$ an interpretation would be that the efficiency is the same, which is not true. These values should advantageously be scaled accordingly. It is also possible to penalize the state(s) without reference if it grows too large. We did not have this problem though, if minimizing the control signals the rotation will also be kept low.

When trying to maintain $\alpha_{trim}$ and having large disturbances it would be advantageous to swap the controller to one with reference $q = 0$, by changing to $C = [0, 1]$. Currently a large error will cause a rotation, which will cause the speed and altitude to change.

**Neural Network**

The selection of data sets for the NN could have been made even better, the goal is to use a limited amount of data that contains significant information. One alternative would be to select the set from a normal distribution. It would give a better performance at the middle of the range where it is important to have good accuracy for reaching the reference, while the other cases which more rarely happens could have a bigger error. Another way would be to collect data when flying to see which combinations occurs and then use those. The limits which the disturbances lie within have been especially hard to approximate.

More tests could be made on the design of the NN, altering the size of the data set, the nodes in the layers more thoroughly and so on. A lot of the executed tests confirm that all inputs for the training might not be necessary. Removing the disturbance is an example as $x_r$, $u_r$ indirectly contain that information. An interesting approach could be to not only use $u_k$ in the output layer, but also some of the sequent time steps.

**Fast MPC**

One way to decrease the computational time for the fast MPC further would be to choose $P_N$ diagonal which leads to a $\Phi$ only containing diagonal elements, making its inverse even easier and faster to calculate. It is vital to pre-compute as many matrices as possible considering the amount of envelope points. Every ex-
tra flop makes it trickier to compute the optimization within the necessary time span.

It is possible that better results could have been received if more time was put on tuning all variables such as $\alpha$, $\beta$ and $\epsilon$ for the line search algorithm. The values were chosen to get a working behavior for one test system and were thereafter never changed. A more considered choice of those values would perhaps have given a better solution of the inexact algorithm, which also could have reduced the time slightly.

### 8.2 Interpretation of Results

The most promising method is the NN, mainly due to its low computational demand. It would be recommended to use a bigger data set of 50 000 points, with more nodes in the hidden layers than the 20 000 points network, with $[25 \ 20]$ neurons. The smaller one works well but data with a wider range of disturbances should preferably be added. We had some problems generating the correct state space models in ARES and because of that, the controller performs worse at some envelope points. System errors are a part of the disturbance and for now the NNs are trained for rather small values.

The fast MPC algorithm with $N = 10$ does not have all the properties desired when run in ARES. For example it is difficult for it to stop at the extreme values. This probably depends on the state space matrices which are not exact and varies too fast, since it performs much better for the linear case. The number of flops have to be reduced even more to increase the prediction horizon, unfortunately some matrix multiplications are heavy and the autogenerated code has some time-consuming parts. The best thing would be to make changes directly in the C-code, currently it builds some constant matrices many times and make a large number of inefficient matrix multiplications consisting of big nested for-loops. For the same reason it is doubtful that the lateral dynamics could be computed within the time as well.

In Chapter 7.2 an overshoot can be seen for the angle of attack when the reference is high. It is caused by the high requested rotation, which has to be damped. The behavior does not show in the linear system due to the servo dynamic which is missing. A solution is to low-pass filter the step, or just split it into two smaller pieces.

### 8.3 Conclusion

An MPC controller can be implemented as a part of the aircraft control system as an inner loop which requests the wanted control surface deflections based on the current states and reference. Both of the methods presented in this thesis, NN and fast MPC, are suitable implementations in respect to accuracy when combining them with a Kalman filter and by choosing the reference with the minimization
norm method.

From a time perspective the NN controller is the best solution. It has a computational time of microseconds and the designer has the possibility to choose a long prediction horizon due to that the majority of the optimization is made offline.

The fast MPC controller does not meet the time limit when using the desired prediction horizon. Despite making Cholesky decomposition, using warm start, inverting blocks, relaxing optimality etc. $N$ has to be reduced significantly, and by doing this the performance suffers. Especially with more constraints, when the feasible region gets more irregular it is hard to tune a good behavior w.r.t. following reference and not violating the constraints.

### 8.4 Further Work

It is appropriate to continue exploring the idea of two separate models for longitudinal and lateral movement. The complexity would probably be too large otherwise as the NN needs more data due to the increase in dimensions and the matrices for the fast MPC grows bigger, demanding a shorter prediction horizon for the same sampling rate. The only drawback is the limitation of the elevon wing, it is the sum of the commands from both elevator and aileron that should lie within the range. An idea would be to have the limit as a parameter in the NN training for one of the systems. Also when the problem is split, one can solve the problems in parallel on different cores. The lateral model has to be further investigated to choose more appropriate weight matrices, amount of hidden layers and sizes of data sets.

A way to simplify the NN code and to look at the problem differently would be to consider it all as a big neural network containing all flight conditions. By using the data from all the envelope points and having speed and velocity as inputs to the function, no switches would be needed to activate certain weights and biases. Further, this could be tried with a nonlinear system model of the aircraft. The drawback would mainly be when fixing small errors or changing the model. In that case, the entire network has to be re-trained, unlike our case where only the weights for a certain envelope point have to be updated.

To verify that the controller is safe extensive testing is necessary, it is hard to confirm stability with these methods. Even to guarantee it for the original setup is difficult without having terminal constraints.
F. Bayat, T. A. Johansen, and A. A. Jalali. Using hash tables to manage the time-
storage complexity in a point location problem: Application to explicit model

A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos. The explicit linear
Cited on page 23.

S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press,

E. F. Camacho and C. B. Alba. *Model Predictive Control*, volume 2. Springer-

page 39.

In *Proc. of the European Control Conference*, pages 502–510, Zürich, Switzer-

F. (eds) *The Schur Complement and Its Applications. Numerical Methods and

G. B. Huang. Learning capability and storage capacity of two-hidden-layer feed-
forward networks. *IEEE Transactions on Neural Networks*, 14(2):274–281,

T. A. Johansen and A. Grancharova. Approximate explicit constrained linear
model predictive control via orthogonal search tree. *IEEE Transactions on Au-


E. C. Kerrigan and J. M. Maciejowski. Soft constraints and exact penalty func-
tions in model predictive control. *United Kingdom Automatic Control Coun-


