NTNU

Norwegian University of
Science and Technology

Model predictive control of an LNG liquefaction process using Jmodelica.org

Oliver Sale Haugberg

Chemical Engineering and Biotechnology
Submission date: June 2018
Supervisor: Johannes Jäschke, IKP
Co-supervisor: Adriaen Verheyleweghen, IKP

Norwegian University of Science and Technology
Department of Chemical Engineering

Preface

This thesis is written as the final part of my Master of Science Degree in Chemical Engineering at the Norwegian University of Science and Technology during the Spring of 2018.

I would like to thank my supervisors Johannes Jäschke and Adriaen Verheyleweghen for the opportunity to work on this exciting project and guiding me along the way. Their continuous support and feedback along the way has been very important for all the work I have done here.

Declaration of Compliance

I hereby declare that this thesis is an independent work in agreement with the exam rules and regulations of the Norwegian University of Science and Technology (NTNU).

Trondheim, June 15, 2018
Oliver Sale Haugberg

Abstract

In this thesis Model Predictive Control (MPC) theory has been used to develop a controller for a Liquefied Natural Gas (LNG) liquefaction process using the Jmodelica.org framework. Jmodelica.org is an open source platform that combines Modelica models with the possibility of running dynamic optimization problems directly on these models. The model used is a cascade liquefaction process model written in Modelica, with the corresponding Optimal Control Problem (OCP) used by the MPC written in Optimica, where Optimica is an extension to the base Modelica language. The possibilities of using Jmodelica.org for the implementation of a Robust MPC are also briefly looked into.

During initial simulations the MPC is found to control the process to an optimal operating point under a given set of constraints on the inputs and system variables. The controllers objective is mainly to minimize the energy used by the three compressors for a given natural gas load. The MPC controls the process to this point, which is previously known optimal point, both with and without measurement noise on the dynamic states of the model. The MPC's performance is tested and compared to a Proportional Integral (PI) controller for different changes to environmental variables. For these tests the MPC controller performs better with regards to minimizing the total energy used, especially for changes that require new set-points to be used by the PI controller. However due to the PI controller not being fully optimized, and as there are clear improvements that can be made to the PI controller, no final decision can be made on the actual improvements introduced by the MPC. The robust MPC variant is also tested and compared to the standard MPC for a small unnoticed change. The robust controllers ability to withstand expected disturbances built in to its implementation, is shown at the cost of operating in a more conservative manner, with regards to total energy used by the three compressors in the process.

The nature of the Jmodelica.org platform lends itself in a practical manner to the development and implementation of MPC control structures. The benefits introduced by doing the process modeling work in Modelica, are combined with the wide range of options for solving dynamic optimization problems in Jmodelica. These problems can then be solved using efficient numerical solvers and algorithms already embedded within the platform. Some specific areas where more work can be done to improving the model and improve the MPC's performance are also discussed. These areas are most notably by either modifying the model, or adding more accurate process model units to better describe the dynamics present in each refrigeration cycles. As for the MPC controller itself, some changes that can be made to the transcription process are also discussed, which can potentially help further reduce the computational time of the controller.

Sammendrag

I denne masteroppgaven har Model Prediktiv Kontroll (MPC) teori blitt brukt for å utvikle en kontroller for en naturgass kondenserings prosess. Dette ble gjort i Jmodelica.org, som en åpen kildekode plattform hvor programmeringsspråket Modelica blir kombinert med innebygde muligheter for å løse dynamiske optimaliseringsproblemer. Kondensering prosessen som er modellert er en kaskade prosess og er skrevet direkte i Modelica. En viktig del av kontrolleren er beskrivelsen av det dynamiske problemet som da har blitt skrevet i Optimica, en utvidelse av Modelica språket som blir brukt av Jmodelica.org. Muligheten for å bruke rammeverket for implementering av robust MPC har også blitt utforsket.

I de første simuleringene ble det vist at MPCen kontrollerer prosessen til et optimalt punkt for diverse begrensninger på manipulerte og kontrollerte variabler. Kontrolleren gjør dette når målet dens er hovedsakelig å minimalisere energiforbruket til de tre kompressorene i prosessen. Kontrolleren når dette punktet både med og uten støy på målingene som blir gjort mellom steg. MPC kontrolleren har også blitt testet og sammenlignet med en Proporsjonel Integrasjon (PI) kontroller for diverse prosess forandringer. For testene som har blitt gjort kontrollerer MPCen bedre med tanke på å minimalisere det totale energi forbruket. Forskjellene er størst når det er nødvendig med nye settpunkter for PI kontrolleren. Ettersom det ikke er blitt gjort mye arbeid på å optimalisere PI kontrolleren, og det er noen tydelige forbedringer som kan gjøres, kan ikke en endelig beslutning tas på hvor mye bedre MPCen er. Den robuste MPC kontrolleren har også blitt sammenlignet med den vanlige implementasjonen for en ikke observert forandring av prosessens variabler. Den robuste kontrollerens evne til å håndtere forventede forandringer har blitt vist, på bekostning av at den opererer på en mer konservativ måte som fører til et høyere energiforbruk av kompressorene.

Jmodelica.org rammeverket fungerer meget godt for implementering av MPC kontrolleren. Fordelene med å gjøre modellerings arbeidet i Modelica er kombinert med Jmodelica.org rammeverkets muligheter for å spesifisere og løse diverse dynamiske optimaliseringsproblemer. For å gjøre dette blir effektive numeriske løsere, som er innebygget i rammeverket brukt. Noen forbedringer områder har blitt diskutert, både forbedringer av modellen og forbedringer av MPC algoritmen. Modellen kan forbedres spesielt ved å også beskrive dynamikken som er tilstede i de forskjellige varmeveksler enhetene. Forbedringer som er mulige for MPC algoritmene handler hovedsakelig om å effektivisere den for å forbedre kjøretid.

Table of Contents

List of Tables

List of Figures

Nomenclature

Symbols

α_i	Compressibility constants
η	Polytropic efficiency
γ	Average adiabatic heat capacity ratio
$\mathbf{p}$	Approximation polynomial
τ	Normalized time unit
C_p	Heat capacity
c_{choke}	Valve constant
d	Disturbance
e	Error of control variable
f	Differential functions
g	Algebraic functions
g	Gravitational constant
H	Enthalpy
h_i	Discretization element length
$h_{poly,scaled}$	Scaled polytropic head
h_{poly}	Polytropic head
J	Objective function
k	PI tuning parameter

l	Lagrange basis polynomial
m	Molar flow rate
M_m	Molar mass
m_{cond}	Cooling water molar flow
m_{rec}	Receiver holdup
N_c	Collocation points
N_e	Discretization elements
N_{comp}	Rotational speed of compressor
P	Pressure
$q_{suction}$	Volumetric flow rate
Q_{unit}	Heat transfer in process unit
R	Gas constant
T	Temperature
t	Time
u	Input functions
u_{valve}	Valve opening position
UA_{units}	Specific overall heat transfer coefficient
V	Volume
W_{comp}	Compressor work
x	Differential states
y	Output variable
Z	Slack variable
z	Algebraic states
Z_i	Compressibility factor

Subscripts

i	Discretization element
k	Collocation point
L	Lower limit

U Upper limit

E Ethane

L LNG

M Methane

P Propane

Abbreviations

CV Control Variable

DAE Differential Algebraic Equations

JT Joule-Thompson

LNG Liquefied Natural Gas

MPC Model Predictive Control

MV Manipulated Variable

NLP Nonlinear Problem

OCP Optimal Control Problem

PI Proportional Integral

Chapter 1

Introduction

1.1 Introduction

Production of natural gas and its use as an energy source has continued to grow in recent years. As natural gas is mainly made up of methane gas, it is a relatively cleaner and more environmentally friendly alternative to its main competitors, oil and coal. There are a wide range of different uses of natural gas, including power production, domestic heating, transportation, and as a raw material for many industrial and petrochemical processes. As a result of this the demand for natural gas is expected to grow in the following decades.

Whereas coal and oil can easily be loaded on to freighters and transported over very large distances, transportation of natural gas is more complicated. This is due to the very low density of natural gas, when compared to its competitors other competitors. For medium distance transportation pipelines are used for natural gas, but these become unfeasible over long distances. As a result, the gas markets are often disjointed with large geographical differences in price.

A solution to the problem of transportation is to instead transport it as liquefied natural gas (LNG). Because the density of LNG is significantly higher that of the natural gas itself, when cooled down to a liquid state it can be transported in a cost effective and safe manner over very large distances. Compared to simply pressurizing the gas, when liquefied and kept at a low temperature the liquid can be stored at relatively low pressures. LNG is produced by adding a liquefaction step in the treatment process of natural gas. The liquid product is then loaded onto purpose built transportation vessels which can ship the LNG globally. After arriving to a terminal the liquid it can be regasified and further transported within the local gas infrastructure.

LNG liquefaction processes are however quite energy intensive and add to the cost of production. As the demand for natural gas has steadily increased there certainly are many commercially competitive LNG plants. But due to small margins and many competitors/

alternative sources of energy, minimizing the costs of operations is very important. Not only is it important that the design of the plant is optimal, but also that operations are. Because of the large amount of energy required by the liquefaction processes, there are various process designs that have been studied, optimized, and developed over the last few decades. Optimal operation can also significantly affect the feasibility of different plant due to their scale, meaning small increases in efficiency can have large impact on economic performance.

One source of information related to the optimal operation of refrigeration cycles, which are at the core of all LNG liquefaction processes, is found in Jensen [2008], where the problem is studied in depth. The main topics are finding the relevant steady state degrees of freedom in the process that can be used for optimal operation. These degrees of freedom are then used to control a set of variables at optimal set-points. An alternative to this constant set-point control approach is Model Predictive Control (MPC). When using a model predictive controller, computers are used to calculate the optimal input for a process. These calculations are then redone at each step, as new information is sampled from the process.

For any MPC framework to give acceptable control performance, detailed dynamic process models are required. The process models need to capture the complex nature and dynamics which are in the process to be controlled. Properly expressing and laying out these dynamics models can be time consuming and quite challenging. It is therefore important that the choice of modeling environment is chosen to fulfill all requirements needed, but also not be too overtly complex. Low level programming languages are often used, as they offer speed and capability to be extremely efficient for specific problems. However, a major challenge with low level languages, is often the re usability and opportunity of separating code into different units. The architecture of the language might not always be a natural fit to the architecture of the model being developed. As a result of this, several high level and purpose built languages have been developed to fill these more specific needs. One of these languages is Modelica, a language purpose built for technical modeling of different systems. By using a high level programming environment instead, where the architecture is more component oriented, the modeling work can be greatly eased. There are of course downsides such as computation efficiency and the ability to tailor fit models to specific needs.

As opposed to other programming languages, assignment statements are not used in Modelica. Instead, algebraic equations and differential equations are directly described without any inherent causation. Equations do not need to be written for specific variables, they are instead written down to signal how variables relate to each other. Modelicas object oriented approach to process units and blocks greatly eases the possibility of reusing model components. The vast array of open source libraries can also reduce the time spent modeling all sorts of different physical systems or control layers. As Modelica is an open source project, the language and several different modeling libraries are continuously developed and open for all [Modelica]. However, the main focus of Modelica is not dynamic optimization, a key requirement for model predictive control. Its main focus is rather

the development and simulation of models. Different simulation environments, both open source and commercial, are available that use Modelica. There is for example the commercial software Dymola, or the open source tool OpenModelica which will be used in this thesis. Although there are some optimization tools available in environments such as Dymola, these capabilities are mainly focused on design optimization, or analyzing dynamic systems. Like most other programs their focus is simulation and development of models.

For dynamic optimization, detailed numerical algorithms are required to solve problems reliably and within reasonable time frames. As a result of this, these numerical algorithms are often written in low level languages with specific requirements for how problems are to be formulated. Interfacing dynamic models with these numerical solvers can then be very challenging, depending on what computer language the models are written in. For the solvers to work, models could then be required to be written in the same low level language in very specific ordinary differential equation syntax. As previously explained, this can cause great challenges to the modeling phase. To summarize, a major problem is then often interfacing the desired high level environment for process modeling, with the necessary low level environments used by numerical solver algorithms.

As a result of the points explained above the Jmodelica.org framework will be used for the implementation of model predictive control in this thesis. Jmodelica.org is an open source project focusing on dynamic optimization and simulation of models described by the Modelica language. The platform works by combining several open source tools and standards together with its built in compilers and algorithms, to solve various different problems. For dynamic optimization the platform combines the Modelica language with Optimica, an extension to the base Modelica language explained in Åkesson [2008]. The Optimica extension allows for high level formulations of dynamic optimization problems, in syntax very similar to Modelica. This allows for the required problem formulation parameters such as objective functions, variable constraints, and the optimization horizons to easily be expressed using already built in Modelica units. Together with Modelica/Optimica models, the platform compilers then transform the models and optimization problems expressions into flattened C and XML code, to increase efficiency. These lower level model representations can then be interfaced with the numerical solver algorithms which are used for solving the specified problems in an efficient manner. In Åkesson et al. [2010] a thorough explanation of the rational behind the development of the Jmodelica.org framework, and its intended uses, are laid out in detail.

Since Jmodelica.org's initial release the platform has seen steady development with new capabilities and features added. It is currently maintained by Modelon AB together with academia. There are various examples where the platform has been successfully implemented for various problems. Some examples related to the implementation of MPC include, Larsson et al. [2013] where Jmodelica.org was successfully used to set up a framework and implementation of an MPC for the start up phase of a power plant model. In Cavey et al. [2014] an MPC with moving horizon estimation is implemented to control the heating of a building. Modelica is also widely used for modeling of non thermo fluid processes, which can also be interfaces with Jmodelica.org. One example of this is shown in

Berntorp and Magnusson [2015], where the framework is used for lane control of a vehicle.

In this thesis an MPC will be implemented using Modelica and Jmodelica.org to control a LNG liquefaction process. To do this a liquefaction model which was originally laid out in Verheyleweghen and Jäschke [2018] will be used. The model will first be rewritten in Modelica, as it is originally written in Matlab, and not used for MPC. After this the necessary Optimica code will be written to specify the dynamic optimization problems used by the controller, together with various Python scripts handling the interfacing. The possibilities of using the Jmodelica.org platform for implementation of a robust MPC will also be investigated. Robust MPC being an variant of the standard MPC where disturbances on variables are build into the controller itself, with the aim of having the control action being optimal with disturbances in mind. A standard proportional integral (PI) feedback controller will also be developed as an alternative to the model based controllers. It will mainly be used for comparison purpose to showcase a different control method.

The layout of the thesis described the necessary background knowledge and theory first before moving on to the actual implementation and findings. In the first chapter, which covers control theory, the theory of how the MPC works, and is implemented, is given together with a rational and explanation of robust MPC. Subsequently the process theory chapter lays out the process model, together with a brief overview of refrigeration cycles in general. In Chapter 4 the specific implementation of the control for the process is explained and the controllers behavior showcased. The penultimate chapter covers a few tests that are done on the controllers together with some general discussion. Finally, a conclusion is presented on the thesis.

Chapter 2

Model predictive control theory

In this chapter the general theory behind Model predictive control will be given. Firstly, a brief overview of the MPC algorithm itself, and the logic of how it usually is implemented is given. Following this, a mathematical explanation of how the optimal control problems (OCP), which are the problem formulation within the MPC, are transcribed into nonlinear problems (NLP) is given. This transcription of the problem prepares it for a structure that can be used and solved by the numerical solvers.

Further, the basics of how the robust MPC works is given, together with an explanation of why this variant can be very beneficial for systems containing model variable uncertainties. The architecture of the Jmodelica.org framework and how it specifically handles this process is also briefly explained. Lastly, the alternative basic feedback PI controller is explained with the equations and theory behind it.

2.1 MPC overview

Model predictive control is an advanced control tool generally used to control multivariable control problems. It's objective to control a process or system to a desired state where it minimizes a certain objective function, whilst also satisfying inequality constraints on input or output variables Seborg et al. [2010]. To do this different tools are required, most notably a model of the process itself, a problem formulation including constraints and an objective function, and a general controller algorithm. The process model is used to predict the dynamics and the future behaviour of the system. Using this process model, an informed choice can be made as to what should be done to steer the system in the desired direction.

For real life applications, models can be obtained in a few different ways. Sometimes by inducing a step response to the process and analyzing the response. Alternatives are models built using transfer functions or state-space representations. In this thesis Modelica is used to express the non linear model, so the model is therefore defined by a set of

differential algebraic equations (DAE's).

Together with the process model a dynamic optimization problem, or in this case more specifically an optimal control problem, has to be formulated. This formulation includes the objective function, inequality/equality constraints on variables, and an optimization horizon. As the optimization problems naturally include constraints in their definition, MPC has the advantage of naturally including these within the controller. Many of the benefits of using MPC can be seen in the formulation of the optimal control problem. Depending on the problem to be solved, the objective functions formulation can vary greatly. Some problems might focus on minimizing deviations of certain variables from predefined set-points or trajectories. In other applications the goal could be maximizing or minimizing certain variables. This could be either maximizing the profit or production rate, or minimization of energy consumption. The objective functions can also include penalties on rapid changes to input variables. This can be done to prevent situations where the optimal solutions require very erratic changes to valves positions. Too rapid changes can lead to damage, or simply be unfeasible to actually perform. In general the formulation is very flexible allowing for many different possibilities, including the multiple interactions between different inputs and outputs in complex systems.

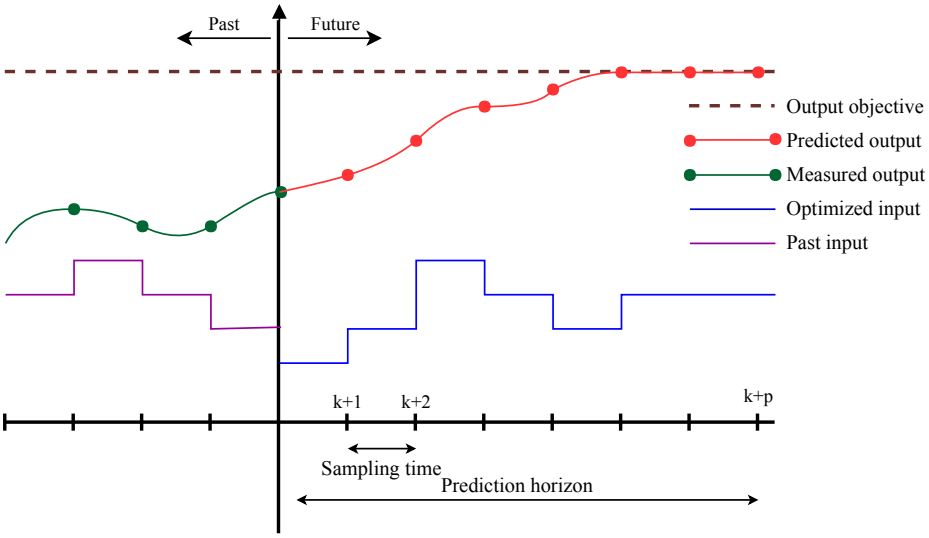


Figure 2.1: A general illustration of how an MPC works at a given time step. Using the past measurements and a prediction model of the process a set of optimized inputs are computed with a goal of the output reaching a certain objective.

The general idea for the MPC controller is shown in figure 2.1. At time t, given the current known state of the system, an optimal control problem is formulated and then solved. This is done for a predefined finite time horizon, known as the prediction horizon. This prediction should ideally last until the system has reached a new steady state, thereby

including all dynamics present in the process. The solution to the optimization problem is then the optimal input trajectory $u(t)$ for the input variables, given the stated objective function. In the figure, this optimized input is shown by the blue line. This is in other words the best set of inputs for the system, based on how the system model expects the system to behave over the prediction horizon. The first set of the optimal input is then applied to the system for a small time frame from k to $k+1$, usually the process sampling rate. The input is often held constant during the sampling duration, as the figure shows. At the next sampling instance, point $k+1$, new measurements are taken of the process state and the whole algorithms starts over again. The changes for the new instance being that a new set of initial conditions are used based on the sampling of the system states, or output. The predictions horizon is also shifted forward in time to account for the change in time. As a result of this, if certain changes can be predicted to happen, the controller can act ahead of time to minimize their future impact. This feed forward capability is a major benefit when compared to other pure feedback controllers. Depending on the model accuracy, MPC can also help control larger time delays or higher order dynamics, which can be harder for more basic controllers.

2.2 Optimal control problem

Although the model equations are described in Modelica, the model that is actually handled within Jmodelica.org is a transcribed and flattened version. In the resulting MPC algorithm, the model is represented by a system of differential algebraic equations (DAE-s). These equations differ from normal systems of ordinary differential equations in that not all variable derivatives are explicitly stated. Instead, some of the variables are expressed using algebraic equations. A general expression of a DAE system can be given by

$$\begin{aligned} \dot{x} &= f(x(t), z(t), u(t), p) \\ 0 &= g(x(t), z(t), u(t), p) \end{aligned} \tag{2.1}$$

For this system, $x(t)$ and $z(t)$ are the differential and algebraic states respectively. They are given by the differential functions $f(x(t), z(t), u(t), p)$, and the algebraic functions $g(x(t), z(t), u(t), p)$. In the system to be used, the input variables are described by $u(t)$, and system parameters which are not a function of time are denoted by p. The initial conditions on the differential states of the system are given by x_0. For DAE-s it is critical that these initial conditions are consistent with the system at whole. In other words, if the initial conditions are consistent, they can be used to find a solution for the system at the initial state.

$$x(t_0) = x_0 \tag{2.2}$$

The constraints imposed on the system will vary depending on how the model is formulated and what the optimal control problem formulation is trying to do. There are different types of constraints such as variable bounds, where there can be upper or lower limits for certain variables. This could be due to physical limitations, such as for a valve opening variable where only values between 0 and 1 make physical sense. Variable bounds can

also be imposed from a safety point of view by imposing limits on pressures, temperature, or liquid levels. Another type of constraint is a point constraints that is only imposed at a certain time point. Most notably are terminal or initial constraints that which are only applied at the initial time or the end time of the optimization horizon.

$$\begin{aligned} x_L &\leq x(t) \leq x_U \\ z_L &\leq z(t) \leq z_U \\ u_L &\leq u(t) \leq u_U \end{aligned} \tag{2.3}$$

Here L denotes the lower bound and U the upper bound of the different variable types. How the objective function, or cost function, is expressed will depend on the goal of the optimization problem. The function is a minimization function, meaning the solver will look for a solution resulting in a minimal value of the function. If the goal is tracking some desired output trajectory, quadratic penalties can be expressed on output variables deviation from these desired values. Other problem formulations can simply be minimization of certain values such as energy usage or maximization of product throughput. In 2.4 a general optimal control problem formulation is given for a system of DAE-s. The cost function to be minimized over the optimization horizon is given by J being a function of the differential, algebraic and input variables. The optimization is done from the initial time point t_0 to the final time point t_f.

$$\begin{aligned} \min_{u(t)} \quad & \int_{t_0}^{t_f} J(x(t), z(t), u(t), p)dt \\ s.t \quad & \dot{x} = f(x(t), z(t), u(t), p) \\ & x(t_0) = x_0 \\ & 0 = g(x(t), z(t), u(t), p) \\ & x_L \leq x(t) \leq x_U \\ & z_L \leq z(t) \leq z_U \\ & u_L \leq u(t) \leq u_U \\ & \forall t \in [t_0, t_f] \end{aligned} \tag{2.4}$$

In the context of the MPC formulation, this is the optimization problem that is solved at every step of the controller. The optimization time $t_f - t_0$ is then the prediction horizon. The optimal input trajectory, the solution to the problem, $u(t)$ is then applied to the system for the sampling duration. When a new sampling can be taken, the new initial states x_0 are applied, the horizon shifted and the problem is solved again.

2.3 Transcription to Nonlinear optimization problem

As the optimal control problem is solved using a direct method, it must first be discretized to some degree before it can be solved. By discretizing of the problem over time the problem is instead solved at a specific number of time points, not continuously. The problem

is instead reduced to a finite dimensional problem, from an infinite dimensional problem which it originally is. The optimal control problem, with its constraints and objectives as described by 2.4, must then be rewritten for these discrete time points instead. The objective function, the model equations themselves and the constraints are then solved at each discretion point. New equations and constraints are also imposed to keep the problem consistent between the different points, and still reflect a solution true to the original continuous problem. An explanation of how this process is done is therefore given below based on Magnusson and Åkesson [2015], where a more detailed explanation of how dynamic optimization in the Jmodelica.org framework is done.

There are different ways of solving optimization problems, with two main methods for higher order systems known as direct and indirect methods. One way of looking at the difference is the order of how things are done. For indirect methods, conditions for the optimal solution are first found. This can be thought of as the solution for the minimum of the objective function or its root. This area is where ΔJ is close enough or equal to zero for the point to be considered an optimal solution. The problem is then transcribed together with these optimally conditions and then solved. Direct methods instead first transcribe the problem, then by iteration look for the optimal solution. Making sure that after each iteration the solution of the objective functions continuous to be smaller and smaller. This is reiterated until an optimal solution is found.

Both indirect and direct methods use transcription methods to discretization the problem. There are also for this step different methods that can be used, with the two main methods being the shooting methods, and the simultaneous methods. For shooting methods, the dynamics of the system are applied by simulation using the original equations. In the simplest method, known as the single shooting method, only one simulation is done with the only condition being that the final state is close enough to the desired solution. The more complex variant is the multiple shooting method, where the optimization horizon is split into different segments. Within each segment the trajectory is simulated until the next segment with an initial guess. In multiple shooting methods, new variables are introduced as the initial state at the beginning of each segment, new constraints are also applied to the problem requiring that the error between the different segments it not to large. These constraints are known as the defect constraints. The dynamics are thereby explicitly stated along the trajectory, but due to the defect constraint systems states are not perfect continuous between different elements.

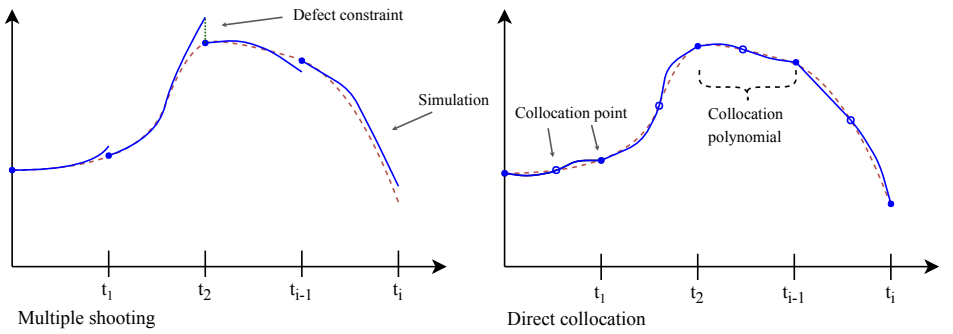


Figure 2.2: Multiple shooting method and simultaneous or collocation method for approximation a function. The shooting method simulates each segment from an initial point, whilst the collocation method builds a polynomial based on collocation points in each element.

The other transcription method is the simultaneous method. Which is also known as the collocation method, where instead of simulation, the system trajectory is approximated between different points. Instead of the system dynamics and equation being applied by simulating from the beginning of one element to the next, they are instead explicitly applied at certain time points. These points are known as the collocation points and can be at any point in each element. The trajectory is then approximated between the points using approximation polynomials, also known as splines. In figure 2.2 the two methods are shown approximating a function displayed as the red dashed line. For the collocation polynomial in this figure there are three points, one at the beginning, one at the end, and one in the middle. The polynomial is equal to the true function in these points and otherwise approximated. In Jmodelica, direct collocation is the main method used to transcribe the problem into the final NLP formulation. Therefore a more detailed overview of how this is done is given in the section bellow.

2.3.1 Direct collocation

In this section a more in depth explanation of direct collocation is given. The goal of this transcription is to transform the optimal control problem, which is infinitely dimensional, into a finite dimensional nonlinear problem which then can be solved by the numerical solvers in Jmodelica. A more thorough explanation of the theory behind the algorithms and the transcription process in Jmodelica can be found in Magnusson [2016], or in Lennernäs [2013] which is more focused on the actual implementation of the collocation algorithms in Jmodelica.org themselves.

Firstly, the time frame for the optimization problem $[t_0, t_f]$ is separated into N_e elements. These elements are then treated separately with the system dynamics being approximated using Lagrange basis polynomials. These polynomials and the different elements are in figure 2.2 separated by t_1, t_2 etc. These approximation polynomials are constructed by interpolation between different interpolation points within each element. By using the value of the system states, equations, and parameters at these points the polynomials are

constructed. Each collocation element N_i which has a length of h_i is then also given N_c collocation points. For each element a normalized time unit τ_i is also used. The relationship between this new normalized time unit, and the old non normalized prediction horizon time unit is then given by.

$$t_i(\tau) := t_{i-1} + h_i(t_f - t_0)\tau \quad \forall \tau \in [0;1] \forall i \in [1, N_e] \tag{2.5}$$

Where t_i are the time points of the beginning and end of each collocation element N_i. The length units h_i are also normalized so that their sum is equal to 1. The value of τ_i will therefore go from 0 to 1 from the start to the end of each discretization element.

The differential, algebraic and input functions from the system of DAE-s are replaced by their corresponding series of Lagrange polynomials. There is then one set of polynomials for each discretization element instead of a set of equations.

$$\begin{aligned} \dot{x}(t) &\rightarrow [\dot{x}_0, \dot{x}_1, \dot{x}_2, ..., \dot{x}_n] \\ x(t) &\rightarrow [x_0, x_1, x_2, ..., x_n] \\ z(t) &\rightarrow [z_0, z_1, z_2, ..., z_n] \\ u(t) &\rightarrow [u_0, u_1, u_2, ..., u_n] \end{aligned} \tag{2.6}$$

Where n is the number of collocation elements N_c, and each number denotes one set of polynomials. These time dependent variable polynomials can then be grouped together and denoted with $\boldsymbol{p}_i$.

$$\boldsymbol{p}_i = [\dot{x}_i, x_i, z_i, u_i] \tag{2.7}$$

If there then are k collocation points in each element in $[1..N_c]$, then the notation τ_k is the local normalized time at an elements collocation point k. Further, the values of the polynomials at this time point τ_k , can be denoted as $\boldsymbol{p}_i(\tau_k)$. The values at these points then become a part of the decision variables in the NLP that will have to be determined by the solver.

$$\boldsymbol{p}_i(\tau_k) = \boldsymbol{p}_{i,k} = [\dot{x}_{i,k}, x_{i,k}, z_{i,k}, u_{i,k}] \tag{2.8}$$

The collocation polynomials can then themselves be defined. They are given by the following equations and are defined as functions of the basis polynomials and the system values at the different collocation points.

$$x_i(\tau) = \sum_{k=0}^{N_c} x_{i,k} \cdot \tilde{l}_k(\tau) \tag{2.9}$$

$$z_i(\tau) = \sum_{k=1}^{N_c} z_{i,k} \cdot l_k(\tau) \tag{2.10}$$

$$u_i(\tau) = \sum_{k=1}^{N_c} u_{i,k} \cdot l_k(\tau) \tag{2.11}$$

As the differential states polynomials x_i have to be continuous between the discretization elements, they are slightly different. For this to be implemented successfully an extra collocation point is added at the beginning of each element. It can thereby be specified to be equal to the end value of the previous element. The Lagrange basis polynomials denoted by $\tilde{l}_k$ and l_k are equal for all elements as a result of the normalized time used in each element. The basis polynomials themselves are defined by

$$\tilde{l}_k(\tau) := \prod_{l=0, l\neq k}^{N_c} \frac{\tau - \tau_l}{\tau_k - \tau_l} \tag{2.12}$$

$$l_k(\tau) := \prod_{l=1, l\neq k}^{N_c} \frac{\tau - \tau_l}{\tau_k - \tau_l} \tag{2.13}$$

These basis polynomials have the property that they are equal to 1 when $j = k$ and zero for $j \neq k$. Because each basis polynomial k is multiplied by the corresponding system variable value at point k, the polynomial will have the correct value at this point. Each basis polynomial makes sure the value is correct for it's point, without interfering with the other points.

$$l_k(t_j) = \begin{cases} 1 & \text{if} \quad j = k \\ 0 & \text{if} \quad j \neq k \end{cases} \tag{2.14}$$

For the polynomials replacing the differential variable derivatives $\dot{x}_i$, the state variable polynomials are differentiated with respect to time. This is done by combining equation 2.9 and 2.5 and then using the chain rule. The corresponding approximation polynomial is then expressed by

$$\dot{x}_i(\tau) = \frac{dx_i}{dt_i}(\tau) = \frac{d\tau}{dt_i}\frac{dx_i}{d\tau}(\tau) = \frac{1}{h_i(t_f - t_0)} \sum_{k=0}^{n_c} x_{i,k} \cdot \frac{d\tilde{l}_k}{d\tau}(\tau) \tag{2.15}$$

The properties and accuracy of these constructed polynomials can be changed, by changing where in the discretization elements the collocation points are placed. Different placements can cause instability, longer convergence rates, or worse approximations. Algorithms in Jmodelica.org have the possibility to use a few different methods, however for this thesis only Radau quadrature rules will be used. There are other resources that give a more detailed overview of polynomial interpolation and numerical integration in general. A more detailed explanation behind the rational for the Lagrange basis polynomials and the point placements can be found in Süli and Mayers [2003].

The final nonlinear problem formulation is then possible to construct. It is essentially the same optimal control problem as stated in 2.4, but expressed for discreet time points over the prediction horizon as opposed to for continuous time. The dynamics are no longer expressed by the original state equations, but by the collocation polynomials. The inequality constraints are replaced with new constraints of the decision variables at the collocation time points. The objective function is also changed from continuous integral function to a

function minimizing the sum of the target variables, at the different collocation points. A general expression for a NLP is then given by

$$\begin{aligned} \min \quad & J(\boldsymbol{p}_{i,k}) \\ s.t \quad & g_e(\boldsymbol{p}_{i,k}) = 0 \\ & f_i(\boldsymbol{p}_{i,k}) \leq 0 \\ & \forall i \in [1..N_e] \\ & \forall k \in [0..N_k] \end{aligned} \tag{2.16}$$

Where $\boldsymbol{p}_{i,k}$ are then the decision variables of the NLP problem. The model equations and constraints are now either equality constraints expressed by g_e, or inequality constraints expressed by f_i. The model equations and initial conditions are now instead expressed as equality constraints at the collocation points. As the differential states have to be continuous between elements, new sets of constraints also have to be expressed. These are expressed as $x_{i,N_c} = x_{i+1,0}$, and are expressed within the g_e constraints in 2.16.

2.4 Robust Model predictive control

Advanced control methods, such as an MPC, introduce many benefits to the controller such as the ability to deal with coupled multiple inputs and outputs, different constraints and an explicit controller objective. The downside however, is that due to the heavy use of models, the accuracy or quality of the models become very important. If the model is not accurate enough due to plant model mismatch, or if there are significant parameter value uncertainties, measurement errors, or noise, the controller could have difficulties working correctly. Optimal operation, in regards to the objective function, is most often at the limit of certain inequality constraints. Disturbances to the process or unreliable parameter values can therefore easily cause the constraint to be violated, if the controller cannot react quickly enough.

The presence of model-plant mismatch or uncertainties will cause the predicted future behavior to not properly reflect the plant itself. The then sub optimal input can cause either constraint violations, or non optimal performance. Due to this, robust control implementations therefore include variable uncertainties explicitly in the design of the controller. The robustness of the controller is said to be its ability to keep the process stable or properly controlled with accepted performance for a given set of uncertainties. An in depth look into robust model predictive control can be found in Lucia [2014].

There are different approaches of how to implement robust MPC, with one of the earlier formulations being min-max MPC, as described by Campo and Morari [1987]. In this variant the optimization problem is solved for the worst case scenario of the expected disturbance, whilst also including new constraints that all other scenarios values of the uncertainty be satisfied. This causes the solution to be very conservative considering it does not take into account that any new information could occur in the future. Multi-stage MPC, which is the robust MPC variant that will be explored here, is another type of robust

MPC, where new information is taken into account. Here it is assumed that the value of the uncertain parameter can change at each sampling instance which is then taken into account.

In multi-stage MPC certain variables are assumed to have disturbances or uncertainties that can change between sampling instances. To take this into account a scenario tree is used for the given variable possibilities. Different value scenarios for a variable are then simulated in parallel. This can be represented with a scenario tree branching out together with the corresponding optimal input. Then at the next instance each different scenarios again branches out with new possibilities.

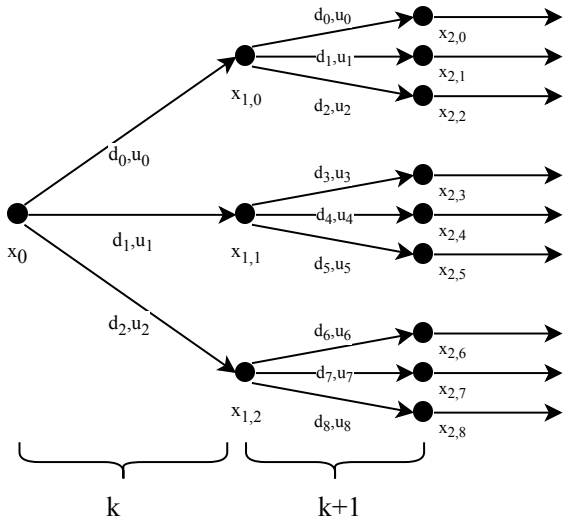


Figure 2.3: Scenario tree for different permutations of the disturbance with corresponding input, where k indicates a discretization step

In figure 2.3 a scenario tree is shown for a case where three permutations of one disturbance d are taken into account and simulated. From the initial state x_0, three permutations of the disturbances d_0, d_1, and d_2 and their corresponding optimal inputs u_0, u_1, and u_2 are shown leading to three new states. From these three new states, each scenario will result in three new scenarios, each again with their own new optimal input. This is what causes the multi-stage variant to be a less conservative approach than the min-max variant. Disturbances are not assumed to be either at the worst or the best case scenario. As only one set of input values can actually be used from the optimization, a new set of constraints are introduced. These constraints requiring that all input solutions from one node have to be equal, $u_0 = u_1 = u_2$. The solution must be the optimal first input when all three possibilities are possible, but where they can also be different at the following steps. The total problem formulation then includes the constraints for all the three disturbances. The different scenarios can then be though of as an expected value, an upper, and a lower bound. As long as the effect of the disturbance is convex , the optimized input should not violate any constraint within the range of the disturbance.

A problem with this successive branching is that if it is done rigorously for each possibility, the problem very quickly becomes way to large to be solvable. If branching of the problem is done over the whole prediction horizon at each sampling step, or if branching is done for not only one variable but for multiple, the problem will scale exponentially in size. The continuous branching allows for optimizing that assumes the disturbance can could change between every sample. This is however not necessarily the case, and by instead assuming that it is constant after a given point, known as the the robust horizon, the problem size is significantly reduced. The problem size can also be significantly reduced if only one uncertain variable is taken into account. If branching is then only done at the first step, the scheme becomes what is known as a two stage robust MPC. Also as only the first step of the solution is implemented by the controller anyway, this approach will often give results close to more rigorous variants as shown in Lucia and Engell [2012].

As the implementation of robust MPC is not the focus of this thesis, the implementations done are of the two-stage variant looking at three different cases for a single disturbance at a time. As the model is already relatively large, and because of the number of discretization points done, anything of a larger scale would become to big for the tools used to be solvable. This is another general disadvantage of the implementation of robust MPC algorithms. As the problems become exponentially larger, the required computational time can very quickly become too big, or even the problem too large for NLP solvers. The more thorough one wished the controller to be, the quicker the problem size increases.

2.5 Optimal control in Jmodelica.org

The process model itself is written in OpenModelica, an open source simulation environment for Modelica model development. The resulting model together with the Optimica code specifying the Optimal control problem is then handled by the Jmodelica.org built in compilers. These compilers then flatten the model, removing unnecessary information and transforms the model and problem into C and XML code. The C code expressing the model equations themselves, and XML the meta data for the model, such as parameter values names etc. After a flattened version of the model has been made CasADi, Andersson et al. [2018], is used to create a symbolic representation of the model.

CasADi is used as it can quickly perform algorithmic differentiation obtaining the derivatives of the different variables. The original integration of Jmodelica.org and CasADi was largely based on mapping of the XML code as described in Andersson et al. [2011]. With the current implementation, which further combines CasADi within the collocation algorithms is explained in more detail in Lennernäs [2013]. The reason the CasADi is useful is because when actually solving the NLP, the iterative methods used by the solver require the first and/or second order derivatives of the cost and constraint functions with respect to other NLP decision variables. As CasADi uses algorithmic differentiation on the symbolic representation, these required derivatives can be calculated very efficiently and quickly. For a more detailed overview of how different software is used, Magnusson [2016] Chapter 3 has a detailed explanation. After the collocation algorithms, working

with the CasADi representation, has finished transcribing the problem into the final NLP representation, the problem can be solved. In Jmodelica.org the default solvers is IPOPT, Wächter and Biegler [2006], which is used for all computation in this thesis.

All interfacing with Jmodelica.org is done using python scripts written by the user. This is where the different tools and packages are brought together. Specified parameters for the optimization problems, initial guesses, and initial trajectories are also given here. After an optimal control problem has been for an MPC step, and an optimal set of inputs has been computed, the optimal set of inputs can be used to run simulations. Simulation of Modelica models is handled through python, using the Functional Mock-up Interface standard. This standard interface allows for model exchange, simulation etc. After then transferring the model into a Functional Mock-up Unit object, simulations can be done on this object.

2.6 PI control

An alternative control structure to compare with the MPC implementation is a PI controller which is also implemented and used to control the process. Being a non model based controller, proportional integral (PI) control uses feedback to control certain control variables (CV's) by adjusting the input value of other manipulated variables (MV's). This is done by measuring the error of the control variables $e(t)$ from their predetermined set-points. The corresponding input is then adjusted in a way to try minimizing the deviation from the set-point. The added control action input for a manipulated variable can be represented by the equation

$$u(t) = k_p e(t) + k_i \int_0^t e(\tau) d\tau' \tag{2.17}$$

Where the error is given as a deviation from a predetermined set-point $y_s(t)$

$$e(t) = y_s(t) - y(t) \tag{2.18}$$

A block diagram for the feedback controller is shown in figure 2.4. For the PI controller used a feedback block is added to introduce some time delay on the measurements.

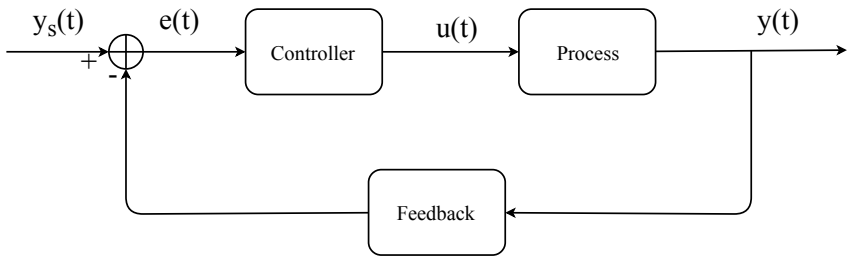


Figure 2.4: Feedback block diagram showing idea behind feedback control. For the PI controller used the feedback block acts as a delay, with no measurement errors on the outputs.

The first term is the proportional action tuned with k_p, where the control action is simply a function of the current value of the error. Large deviations from the set-point thereby cause greater control action. The second term is the integral control action and is tuned with k_i. The integral control action looks at the history of the error from the start, until the current time. This term removes the steady state error which is often present for pure proportional controllers. As long as there is an error present from the set-point, the integral term of the equation will continue to grow until the new control action has removed the error. A third term can also be included, known as the derivative term, which would cause it to be a PID controller. The derivative term looks at the current slope, or rate of change of the error. If the current value of the CV is approaching its set-point too quickly, it is very possible that it could overshoot the set-point. The derivative term will try to minimize too rapid changes and cause the value to settle quicker to its set-point. Derivative control action is however not used in this thesis, as it is generally not required.

Chapter 3

Process description

In this section the type of LNG liquefaction process used, and the corresponding process model will be described. Starting with a brief introduction to refrigeration cycles in general, which are at the core of any liquefaction process, before moving on to the specific cascade liquefaction process used. The design principles behind this model are explained, together with the implementation of the model in Modelica. The equations that make up the different process units are explained together with the thermodynamic equations and state equations used by the model.

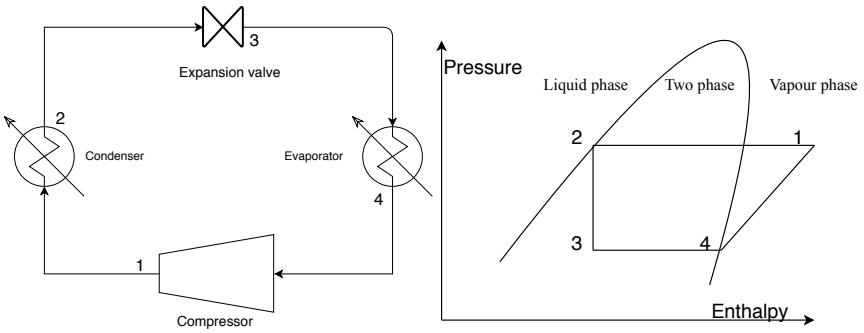
3.1 Refrigeration cycles

Refrigeration cycles are used in a wide range of applications, from areas such as household air conditioning to large scale heat transfer in industrial process plants. The underlying principles of using pressure, evaporation and condensation to move energy are however always the same. In this section a brief introduction to refrigeration cycles and how different design choices can affect how they operate is given. A thorough investigation into different designs and optimal operation of refrigeration cycles can be found in Jensen [2008].

In a refrigerant cycle a refrigerant is continuously cycled in a closed loop from a condenser section to an evaporator section. This is done so that the working fluid in the cycle can move energy from a heat source to a heat sink. Whether the goal of the process is cooling the heat source, or heating the heat sink will depend on the actual process. In the case of LNG liquefaction, the goal is cooling and condensation of the natural gas stream to a pure liquid state. The final heat sink of the process is the cooling water present in the condensers, and the heat source the natural gas stream. In figure 3.1 a basic refrigeration cycle and a possible corresponding pressure enthalpy diagram is shown. The curve in the pressure enthalpy diagram is the saturation line showing the boundary between the different phase regions. Inside of the curve, the system is in the two phase area where it contains both liquid and vapour phase. The left of the curve is the pure liquid phase, and

the right side is the pure vapour phase. The numbers on the figure reference the outlets of the different cycle units.

In the refrigeration cycle, the refrigerant first enters the compressor in a pure gas phase. The gas leaving the compressor, point 1 in figure 3.1 will then be a high temperature, high pressure gas. This gas is then cooled down to its condensation temperature, where it will cross the saturation curve. It will then be further condensed from vapour until it is all in a liquid state. In figure 3.1 point 2, the exit point of the condenser, is exactly on the saturation line. This indicates that the refrigerant is fully condensed, but not cooled any further. If the point where to be shifted to the left, there would be sub-cooling present which is when the temperature of the liquid is cooled from the saturation temperature. This cooled but still high pressure liquid fluid is then sent through a Joule-Thompson valve. In this valve an isenthalpic process causes the pressure, and temperature with it, to significantly fall to point 3. Isenthalpic meaning that there is no heat loss or work done by the fluid through the valve, which causes the enthalpy to be equal on both sides. The then low pressure, low temperature two phase fluid is then sent to the evaporator section where it is fully evaporated to point 4. It is also here possible for the exit point to be shifted to the right, which would indicate super heating in the cycle. Super heating being when the exiting gas phase is heated from its saturation temperature. After being fully evaporated, the then pure vapour phase is sent back to the compressor.



(a) Refrigeration cycle **(b)** Pressure enthalpy diagram

Figure 3.1: Example of a simple refrigeration cycle and the corresponding pressure enthalpy diagram

By further cooling the liquid out of the condenser, or further heating the vapour out of the evaporator, sub-cooling or super-heating can be introduced. In real application of refrigeration cycles, some degree of super heating is always required. This is because if liquid droplets enter the compressor, it can be very damaging to the equipment. It is however generally agreed that no super heating in the cycle is optimal with regards to energy usage. The issue of sub-cooling however is slightly different, as in Jensen [2008] it is show that some degree of sub-cooling can be optimal for certain operations.

By using different designs for the condenser or evaporator, sub-cooling and super-

heating can be included or removed from the process by design if desired. For example by using flooding tanks for the condenser, sub cooling can be eliminated for the cycle by letting the liquefied fluid drain out of the bottom. In this way the liquid phase is not in contact with the heat exchanging element. In the same manner super-heating can be removed by letting the vapour phase evaporate out the top of a tank. The heat exchanging area is fully submerged in the fluid thereby not affecting the gas phase. If instead plug flow type heat exchanger designs are used, the refrigerant can experience both sub-cooled and super-heated when exiting the heat exchanger. But again for these designs both phenomena can be removed at whole by first sending the outgoing fluids through receiver tanks. These receiver tanks then act as buffers where their outlets only contain either vapour or liquid phase by design.

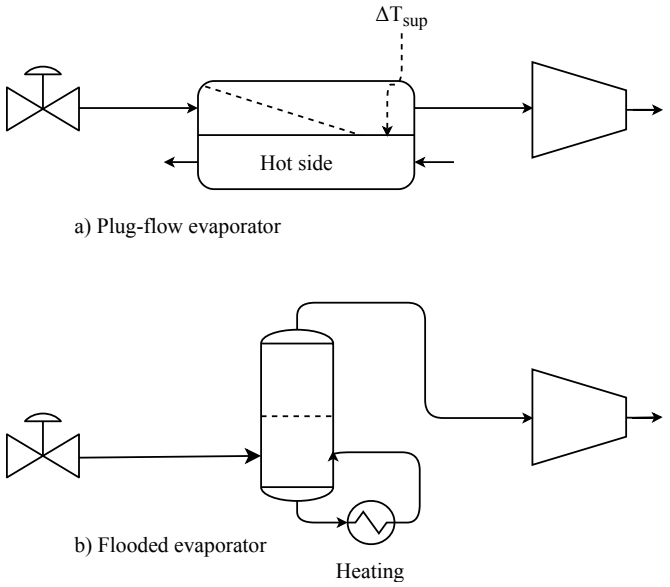


Figure 3.2: Two different designs for the evaporator section. In design a) the plug flow evaporator allows for super heating, whilst in design b) the vapour phase leaves at the saturation point. For design a) receivers can be introduces between the compressor and the heat exchanger to eliminate super heating.

For simple refrigeration cycles, as the one that has been described so far, there are generally five degrees of freedom Jensen [2008]. This is the case if one allows for both sub-cooling and super-heating in the cycle. These five degrees come from the following variables

1. Heat transferred in the condenser
2. Heat transferred in the evaporator
3. Power to the compressor
4. Valve opening position
5. Active charge in cycle

These five variables can be adjusted to change how the cycle operates. They relate to the five design degrees of freedom that can be adjusted when designing the process. Those being the load, the two pressure leves P_h and P_l and the the amount of sub-cooling and super-heating ΔT_{sub} and ΔT_{sup}.

3.2 LNG liquefaction process

There are a few different LNG liquefaction process designs used today. The majority of these either being based on AP-C3MR or cascade technology N. Usama et al. [2011]. The first of these two AP-C3MR, uses two refrigeration cycles where pure propane is used in a precooling cycle and the main cycle containing a mixed refrigerant doing the main cooling. The other cascade based technology uses three cycles each of which containing a pure refrigerant in a cascade setup between the cycles. There are also other designs used in various applications where the differences generally come down to the refrigerant makeup, or the setup of the refrigeration cycles.

The process model to be used in this thesis is an adaption of a cascade based liquefaction process originally modeled and laid out in [Verheyleweghen and Jäschke, 2018]. In this paper the process model was written in MATLAB with the main goal of implementing a self optimizing control structure for the process. The focus of the paper was to look at how a self optimizing control structures would compare to other more direct temperature control structures, when trying to minimize the average steady state loss during disturbances. In this thesis however, as the Jmodelica.org framework is used for development of the MPC framework, the model has been rewritten in Modelica. Due to the different structure of these languages the model setup is quite different and some changes have been made. But as the model units are made up of the same equations with the same thermodynamics the process behaves in the same way.

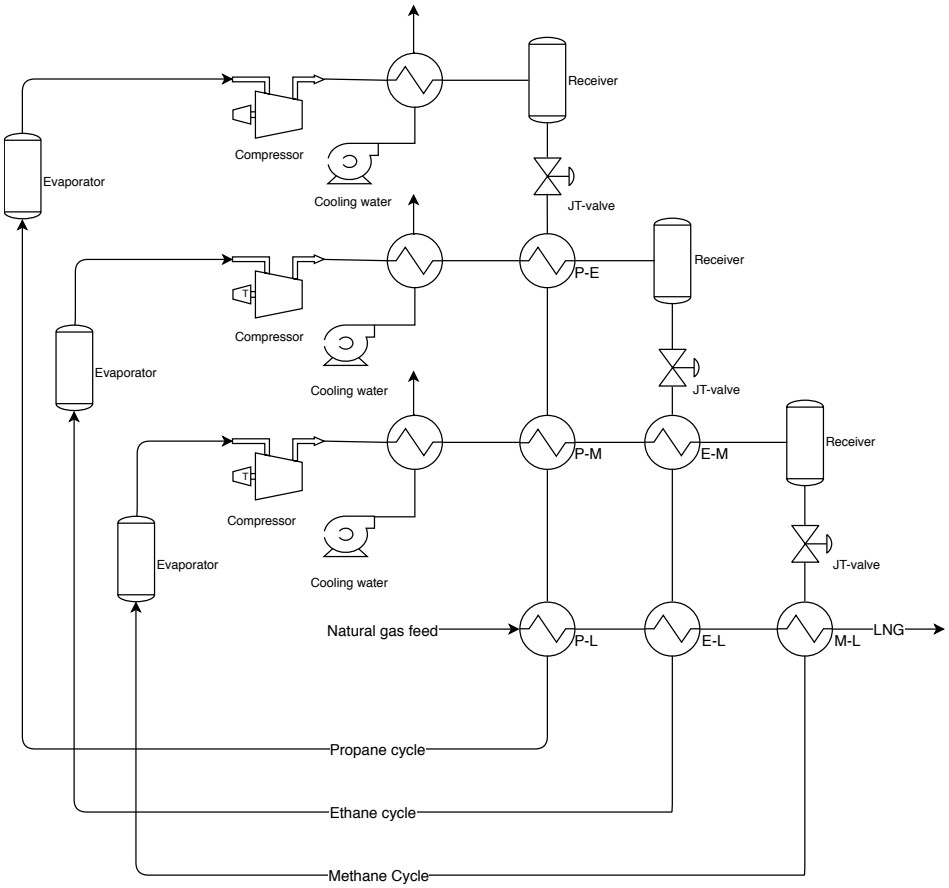


Figure 3.3: Flow diagram for LNG liquefaction process used

A flow diagram for the cascade process modeled is shown in figure 3.3. In a cascade process of this kind, the natural gas feed is sent through successively colder heat exchangers that cool and condense the gas until it is in a liquid state. The cold side of the three different heat exchangers each use three different refrigerants from the three cycles. The three refrigerants used are propane, ethane and methane, each in their own cycle. Meaning that the refrigerant cycles only contain one refrigerant each, as opposed to a mixed refrigerant system. Due to the different phase change characteristics of the three refrigerants, the LNG can be fully cooled to its liquid state. The cascade characteristics of the process come from the fact that the cycles exchange heat with each other in order to increase the thermodynamic efficiency of the process.

In the model the evaporator and condenser sections are models as a series of counter current heat exchangers. In real life implementation of cascade liquefaction processes, such as the ”Optimized Cascade Process” by [ConocoPhilips], the evaporator and condenser sections are large multi-stream heat exchangers. Because these complex heat ex-

changer designs are significantly more complicated to model due to the added geometry, a series of counter current heat exchanger models are instead used. As figure 3.3 shows the three refrigeration cycles interact not only with the LNG steam, but also with each other. For example in the condenser section of the methane cycle, the three heat exchangers each interacts with a different stream on their cold side. The first one with the cooling water, followed by one with the propane cycle, and then with the ethane cycle. This can also be seen in the evaporator section of the propane cycle, where the propane is first evaporated by the methane stream, then ethane, followed by the natural gas itself. This is done to minimize the mean temperature difference, which increases the efficiency of the liquefaction process at a whole.

3.3 Implementation in Modelica

In Modelica models are expressed algebraic equations, differential equations, and functions that when put together make up a process unit or class. These classes which can also be thought of as smaller models, which are then connected together depending on their relation to make up the process model at a whole. The sub-models are connected together using a different Modelica component known as a connector, which is an important feature of the language. Connectors are added to the different sub units, so they can be connected together using special connect function, which then creates a new set of balance and equality equations over the connection. For example by connecting the outlet of one tank to the inlet of another, new equations are made for the flow and potential variables of those inlets and outlets. For flow variables, such as mass flow or energy flow, balance equations are made. For potential variables, such as temperature or pressure, equality equations are made.

$$\begin{aligned} f_1 + f_2 &= 0 \\ p_1 &= p_2 \end{aligned} \tag{3.1}$$

The new equations are shown in 3.1 where f is a flow variable and p a potential variable. This feature allows for different units of a process model to be constructed independently, and later connected when expressing the process at a whole. This of course assumes that all units are made with the same standards in mind and use the same connectors. The inlet and outlet connectors must be of the same class, both having the same variables. In the following sections the equations used in the different process units are described, followed by the thermodynamic equations and state equations used in the model.

3.3.1 Process units

Each of the process units that can be seen in the process flow diagram, figure 3.3 have their own set of equations. They are built up as independent units with equations describing their function in the process, with Modelica connectors binding them together as described. The equations laid out are originally formulated in Verheyleweghen and Jäschke [2018]. Not all units have specific equations for all variables as they might not have an effect on these variables. One example of this is the model for the heat exchanger where there is no effect the pressure of the fluid. The units describing the heat exchangers therefore have equations

that specify that the inlet pressure has to be equal to the outlet pressure. These equations are however not written out in this section.

Compressors

For the compressor unit a polytropic compressor model is used. These processes have in common that they all follow the relation.

$$P_{inn}V_{inn}^{n} = P_{out}V_{out}^{n} \tag{3.2}$$

Where P and V are the pressure and volume for the inlet and outlet flows. The exponent n indicates the the polytropic exponent, which will vary depending on the type of polytropic process.

In the model used the following relation is valid for the temperature and pressure of both the inlet and the outlet.

$$\frac{T_{out}}{T_{inn}} = \frac{P_{out}}{P_{inn}}^{\frac{1}{k}} \tag{3.3}$$

Where T is temperature for the inlet and outlet of the compressor. The exponent k is the polytropic coefficient, and defined by the following relation

$$k = \frac{n}{n-1} = \eta\frac{\gamma}{\gamma-1} \tag{3.4}$$

Here η is the polytropic efficiency of the compressor, γ the average adiabatic heat capacity ratios from the inlet and outlet of the compressor. It is given by the following equation.

$$\gamma = \frac{1}{2}\Big(\frac{C_{p,inn}}{C_{p,inn}-R} + \frac{C_{p,out}}{C_{p,out}-R}\Big) \tag{3.5}$$

Where R is the universal gas constant, and C_p the heat capacity at the inlet and outlet.

The polytropic head for the process is given by the equation

$$h_{poly} = \frac{kZ_{inn}R}{gM_m}(T_{out} - T_{inn}) \tag{3.6}$$

Where Z_{inn} is the compressibility of the refrigerant at the inlet of the compressor, M_m its molar mass and g the universal gravitational constant.

Because the model uses compressor maps, the efficiency η and the polytropic head h_{poly} are are found from curves defined by empirical data. They are therefore defined by the following equations

$$\eta = e_1 h_{poly,scaled} + e_2 - 2 * 10^{(e_3 h_{poly,scaled} - e_4)} \tag{3.7}$$

$$\frac{q_{suction}}{N_{comp_4}^{C}} = \frac{C_1 h_{poly,scaled} - C_2}{C_3} \tag{3.8}$$

The constants e_i and C_i are constants specific to the different compressors and can be found in appendix [A]. The speed of the compressor, which is the input variable used to operate the compressor is given by N_{comp}. The volumetric flow rate at the inlet is denoted with $q_{suction}$. The volumetric flow rate is also then be expressed by the following equation-.

$$q_{suction} = \frac{n_{inn} R T_{inn}}{P_{inn}} \tag{3.9}$$

Where n_i is the molar flow rate at on the compressor inlet side. The scaled compressor head $h_{poly,scaled}$ is given by the equation

$$h_{poly,scaled} = \frac{h_{poly}}{{u_{comp}}^{C_5}} \tag{3.10}$$

where C_5 is also a constant specific to the compressor and found in appendix [A]. All three cycles use the same compressors equations, but as the compressor maps are different for the three refrigerants the constants are different.

Condensers and evaporators

For the condensers and evaporator sections, counter current heat exchanger models are used. In each heat exchanger units, the logarithmic mean temperature difference is used to calculate the heat transfer rate.

$$Q_{unit} = U A_{unit} \Delta T_{lm} \tag{3.11a}$$

$$\Delta T_{lm} = \frac{\Delta T_1 - \Delta T_2}{\log(\Delta T_1) - \log(\Delta T_2)} \tag{3.11b}$$

$$\Delta T_1 = T_{cold,out} - T_{hot,inn} \tag{3.11c}$$

$$\Delta T_2 = T_{cold,in} - T_{hot,out} \tag{3.11d}$$

Where Q_{unit} is the heat transferred in each different condenser or heat exchange unit. ΔT_{lm}, the logarithmic mean temperature difference is calculated from the inlet and outlet temperatures of the cold and hot side. It is the average logarithmic difference of these temperature values. The specific overall heat transfer coefficient UA is constant and specific to each heat exchanger unit. The values used for these coefficients can be found in appendix [A].

For the condenser section of each refrigeration cycle, the first heat exchanges cold side is cooling water. For these units the energy transfer is also a function of the temperature change for the cooling water. It is therby given by the function

$$Q_{condenser} = m_{cond} C_{p,water} (T_{amb} - T_{cond}) \tag{3.12}$$

For this equation m_{cond}, the molar flow rate of the cooling water also acts as an input that can be adjusted for the process. T_{amb} and T_{cond} are respectively the temperatures of the cooling water at the inlet, the ambient temperature, and at the outlet from the condenser.

The constant heat capacity of the water is denoted by $C_{p,water}$.

For the hot side of these, and for the heat exchangers not using cooling water, the change in enthalpy for the streams is set to be equal to the heat transferred Q_{unit}. The relation between the enthalpy of the streams and their temperature or pressure is found in the thermodynamic section.

$$Q_{unit} = H_{hot,out} - H_{hot,inn} \tag{3.13a}$$
$$Q_{unit} = H_{cold,out} - H_{cold,inn} \tag{3.13b}$$

One issue with these equations are that they do not allow for any dynamics to in the condenser and evaporator sections. The temperatures out of the units are instantaneously set by the inlet temperatures and their flow rates. The logarithmic mean temperature difference concept is based on the heat exchange being at steady state. The equations also assume that the overall heat transfer coefficient is constant, which is is not the case as the refrigerants undergo phase change through the units. These simplification are however done as proper dynamic heat exchanger models for systems that undergo phase change are surprisingly difficult to actually implement. In the case of these condensers and evaporators, it is due to the fact that the gas entering the condensers is heated beyond its saturation temperature. And it is due to the fluids leaving the sections can experience both sub-cooling or super-heating. The points in the heat exchangers where the streams go from one phase area to another are very very difficult to model with dynamics.

By allowing for sub-cooling and super-heating in a dynamic model new challenges are faced in the formulation of the model. Ideally the model for the heat exchangers should be robust enough to handle all refrigerant phase possibilities, at any point in the exchanger. Those three possibilities being pure liquid or vapour phase, or a mixture of both as shown in the pressure enthalpy diagram figure 3.1. The model equations would need to describe the dynamics of all three possible phase regions, and be able to handle the points where the transitions between these regions can occur. The main difference for the regions being that while in the single phase region, heat transfer causes temperature change for the refrigerant, whilst in the two phase region it causes either evaporation or condensation.

Because it is not possible to determine when and where the transitions between phase regions can occur, the model must be able to handle these points online during simulation and act accordingly on the fly. For example, the vapour that enters the condenser section will first experience a drop in temperature. It will cool down until it reaches the saturation temperature, where it will start to condense into a liquid instead. Rapid changes to the system parameters can cause sudden drops in the heat transfer rate resulting in vapour leaving the condenser. If these possibilities are allowed for in the process model, the resulting heat exchanger model would be required to handle all such possibilities.

These issues related to dynamic heat exchangers models are therefore ignored by instead using the steady state model described. In the overall model is the level of sub-cooling and super-heating is also controlled to zero. The enthalpies out of the condenser and evaporator sections are set to be equal equal to the corresponding saturation tem-

peratures. By doing this two degrees of freedom are however lost for each cycle as the enthalpies have to be controlled to these respective values. For example the valve opening levels can not be used as a free input as it must be able to fulfill one of these requirements. The system model cant be over specified.

Receivers

In the condensers section the working fluid refrigerants enter the receivers which introduce dynamics into the cycles. The volume of the different receiver act as a buffers in each cycles. This will then cause changes in the condenser section to not have an immediate effect on the evaporator section. The outlet streams are set to be equal to the conditions in the tanks, not the inlet to the tank. The energy balance equation for the methane and ethane receivers are as follows

$$\frac{dH_{rec,M}}{dt} = H_{out,M} - H_{in,M} \tag{3.14a}$$

$$\frac{dH_{rec,E}}{dt} = H_{out,E} - H_{in,E} \tag{3.14b}$$

Where H_{rec} is the enthalpy in the receiver and H_M, H_E the enthalpy flow in and out of the different units. It is also assumed that for the receivers there is perfect level control causing no flow dynamics, only energy dynamics. This causes the inlet and outlet molar flows to be equal. The dynamics in the different cycles then instead comes from the enthalpies of the flows. The dynamics can be rewritten for specific enthalpy in the following way

$$\frac{dh_{rec,M}}{dt} = \frac{h_{out,M} - h_{in,M}}{m_{rec}} \tag{3.15a}$$

$$\frac{dh_{rec,E}}{dt} = \frac{h_{out,E} - h_{in,E}}{m_{rec}} \tag{3.15b}$$

$$\tag{3.15c}$$

For the propane receiver, the dynamic equation is for the temperature instead. It is also here assumed the inlet and outlet molar flows are equal, results in the following differential equation

$$\frac{dT_{rec,P}}{dt} = \frac{m_{in}}{m_{rec}}(T_{out} - T_{in}) \tag{3.16}$$

In this equations, and for the two others m_{rec} denotes the molar holdup in each receiver. Higher values causing larger receiver thereby causing slower dynamics.

Valves

After the receiver, the refrigerant which is then a saturated liquid passes through the isenthalpic Joule-Thompson valve. Here the refrigerant is expanded down into the two phase

region, as the pressure is significantly lowered. The molar flow rate of the cycle is also here related to the pressure drop by the following valve equation

$$m = u_{valve} c_{choke} \sqrt{P_{inn} - P_{out}} \tag{3.17}$$

Where u_{valve}, is the valve opening, m_i the molar flow rate through the valve, and c_{choke} the valve constant which can be found in appendix [A].

3.3.2 Other equations

When later optimizing the operation of the process model, one of they key characteristics of its behaviour is how the task of cooling the LNG stream is spread over the different cycles. In other words how much energy is actually transferred in the three heat exchangers. The cycles also interact with each other so how the evaporation of propane is spread through its three evaporator heat exchangers is also of interest. Theses different modes of operation are a key factor in determining what setup will cause minimum energy usage by the compressors, which is the optimal mode of operation. There are therefore no specific constraints or equations specifying the exact duty in each heat exchanger. What must be specified however is the total energy transferred in and out of each refrigeration cycle. Because the refrigerants are in a closed loop, they must return to their initial state for the loop to be consistent. The energy added to the refrigerant in the evaporator and the compressor, must equal the energy leaving it in its condenser section. The equations specifying this are as follows

$$Q_{comp,M} + Q_{hex,M,L} = Q_{cond,M} + Q_{hex,E,M} + Q_{hex,P,M} \tag{3.18a}$$
$$Q_{comp,E} + Q_{hex,E,L} + Q_{hex,E,M} = Q_{cond,E} + Q_{hex,P,E} \tag{3.18b}$$
$$Q_{comp,P} + Q_{hex,P,L} + Q_{hex,P,M} + Q_{hex,P,E} = Q_{cond,P} \tag{3.18c}$$

Where Q indicates the change in energy for the refrigerant over the different process units.

The receivers after the evaporator sections are implemented without any dynamic equations. This was done as any attempts to implement them would cause the system to not initiate properly. In the current implementation they simply state that the inlet has to be equal to the outlet. In the actual model all variables are also scaled down to be of the same magnitudes. This is done to minimize the risk of solvers having issues with extreme differences in variable values. For example all temperatures are multiplied with 0.01, and all pressures are scaled down by on magnitude. Theses change are however always taken into account in the equations and scaled back when presented in plots and tables.

3.3.3 Compressibility

Because the compressibility factors of the gases are needed for the equations for the compressors, the Dranchuk and Abou-Kassems equation of state is used Dranchuk and Kassem

[1975]. The equation uses a generalized starling equation and is fitted to data. The compressibility factor is then given by the equation

$$\begin{aligned} Z_i &= 1 + \big(a_1 + \frac{a_2}{T_{r,i}} + \frac{a_3}{T_{r,i}^3} + \frac{a_4}{T_{r,i}^4} + \frac{a_5}{T_{r,i}^5}\big)B \\ &+ \big(a_6 + \frac{a_7}{T_{r,i}} + \frac{a_9}{T_{r,i}^2}\big)B^2 \\ &- \big(\frac{a_7}{T_{r,i}} + \frac{a_8}{T_{r,i}^2}\big)a_9 B^5 \\ &+ a_{10}(1 + a_{11}B^2)\Big(\frac{B^2}{T_{r,i}^3}\Big)\text{exp}(-a_{11}B^2) \\ B &= \frac{0.27 P_{r,i}}{Z_i T_{r,i}} \end{aligned} \tag{3.19}$$

Here the a_x parameters are constants and can be found in appendix[ref to appendix]. $T_{r,i}$ and $P_{r,i}$ are reduced temperature and pressure for the different refrigerants respectively, and are given by

$$T_{r,i} = \frac{T_i}{T_{c,i}} \tag{3.20a}$$

$$P_{r,i} = \frac{P_i}{P_{c,i}} \tag{3.20b}$$

Where $T_{c,i}$ and $P_{c,i}$ are the critical temperature and pressure values for the refrigerants. These can be found in appendix[A].

3.3.4 Thermodynamics

As the energy usage of the compressors, or the heat transferred in the different heat exchangers units is calculated as a change in enthalpy, equations relating the enthalpy to the refrigerants states are required. Equations for the saturation temperatures are also needed and are calculated as a function of the pressure. These values are calculated from models made from data calculated in AllProps, Lemmon et al. [1994]. AllProps is a software that uses the Helmholtz equation to calculated the required thermodynamic data. For the saturation temperature the equations are then

$$T_{sat} = \sum_{i=0}^{2} c_{sat,i} \log(P)^i \tag{3.21}$$

And the equations for the saturation enthalpies for the liquid and vapour phases are

$$H_{liq} = \sum_{i=0}^{6} c_{liq,i} P^i \tag{3.22a}$$

$$H_{vap} = \sum_{i=0}^{6} c_{vap,i} P^i \tag{3.22b}$$

The coefficients c_i used in these polynomials are specific to the different refrigerants and can be found in appendix [A]. All three functions are polynomials and also functions of the refrigerant pressure. Heat capacities for the gas phase are calculated as a function of the temperature. These coefficients are also found in appendix [A].

$$C_P = \left(c_{C_p,1} + c_{C_p,2} * T + c_{C_p,3} T^2\right) R \tag{3.23}$$

Chapter 4

Setup and implementation

In this chapter, an overview of how the MPC is implemented for the process model outlined in chapter 3 is explained. First the setup and implementation of what is required for the optimal control problem is shown. After this the solution to a specific OCP is shown and explained. The the same is then done for the actual algorithm for the standard and robust implementation of the MPC together with their solutions. Towards the end of the chapter the implementation of the PI controller is also briefly discussed.

4.1 Optimal control problem

In chapter 2 the formulation of an optimal control problem and its requirements such as the objective function, a set of equations describing the model, a prediction horizon and constraints was explained. As the equations making up the process model have already been laid out in chapter 3, this section will focus on explaining the rest of the problem formulation in relation to the specific model. Some other parameters for the discretization and collocation steps of the problem are also explained, such as blocking factors on the inputs or the prediction horizon itself.

Constraint softening

When systems are operated at their optimum point when constraints are present, they often end up at the boundary of one or more of these constraints. Meaning that the optimum value of one or more variables is at its maximum or minimum allowed value. If this is the case, the constraint is said to be an active constraint. One problem that can then arise is that the solver could be faced with an unfeasible problem due to it being to close to an unfeasible area. The problem is running so close to an edge, that any disturbance or unexpected behavior can cause the system to have no feasible solution that won't violate the constraint. This could occur from both environmental variables changing unexpectedly or measurements errors on the states causing no possible change in input to uphold feasibility. This problem can be solved by changing the constraint from a hard constraint

to a soft constraint.

When a constraint is softened the variable can cross the boundary if necessary, but it will be heavily penalized in the objective function if if does so. For the liquefaction process model the most important constraint that is always active is the final outlet temperature of the LNG stream. This is because its temperature is directly linked to what is considered optimal, it will always be as close as possible to its limit which is where is fully liquefied. This constraint can also be softened, as opposed to some other hard constraint. The temperature output constraint will usually have some pullback from the actual saturation temperature. This is different from some absolute physical constraints such as a valve opening. It is acceptable that the temperature is slightly above its limit for a limited period, but a valve opening cant be more than fully opened or have a negative value.

The constraint is softened by introducing a new slack variable $Z(t)$. As it is only the constraint on the LNG temperature that is softened, only one slack variable is introduced for this problem. The slack variable is included with the other constraint so the new upper limit is the set value plus the slack variables value. A constraint restricting only positive values on the slack variable is also included as negative values don't make sense. The slack variable value is also heavily penalized in the objective function to keep constraint violations at a minimum.

Objective function

The objective function decides what optimal operation is for the model. Depending on how the model structure and how it is laid out, what is considered optimal operation can vary. For the LNG liquefaction process, one could say that maximizing throughput of liquefied LNG is one way of looking at optimally. For this problem however the flow of the LNG is constant and is said to be set at a value. The goal of the problem is then to successfully cool all of the gas down to the required temperature, whilst trying to minimizing the total energy used by the compressors.

Minimizing the energy used by the compressors $W_{comp,i}$, is the main focus of the objective function. The first set of heat exchanges in the condensers sections use cooling water, where the cooling water flow together with the compressor rotation speeds, is a control input. Depending on the location of the LNG plant, the cost of cooling water can either be trivial, or something that must be taken into account. In the objective function there is therefore a very small penalty added to the flow $m_{cond,i}$ of the cooling water. This is mainly done as otherwise all these flows would constantly be at their maximum allowed values, even if the extra effect of doing this is extremely low. The slack variable for the outlet temperature is also included in the objective function, so constraint violations are kept as small and as few as possible. The objective function is then

$$\begin{aligned} J = & W_{comp,M} + W_{comp,E} + W_{comp,P} \\ & + 0.001\big(m_{cond,M} + m_{cond,E} + m_{cond,P}\big) \\ & + 25(Z) \end{aligned} \tag{4.1}$$

Constraints

Although Modelica allows for some constraints such as min/max values on variables, to be built into the definition of variable types themselves, all constraints are put in the Optimica formulation of the dynamic problem for consistency. There are both constraints on the input variables, and on certain state variables. The slack variable is also included in the constraints. The constraints used with a brief explanation are given in table 4.1.

Table 4.1: Constraints for input and state variables in the optimal control problem

Mathematical constraint	Justification
$T_{LNG} \leq -150[^{\circ}C] + Z(t)$	Maximum temperature allowed at LNG outlet
$0.6 \leq u_{comp} \leq 1.1$	Compressor operating range
$0 \leq m_{cond} \leq 1 * 10^4$ [kgmol/h]	Cooling water flow operating range
$0 \leq x_{valve} \leq 1$	JT-Valve opening range
$P_i \geq 0.4$ [bar]	Minimum pressure level allowed in cycles
$Z(t) \geq 0$	Slack variable has to be positive

Other more basic constraints such as temperatures not being negative, or flow variables not suddenly changing signs are not explicitly stated. They are not included as there are several parts of the model that would break down way before they could occur. As long as the model is properly initiated the model and its variables will stay in the same solution space, meaning variables wont suddenly jump to some completely different solution even if it somehow where to be mathematically possible.

Disturbances variable

The disturbances variables are the variables that are not controlled or adjustable, and they cant be predicted. They are implemented as a set of exogenous inputs where the value can change independently of the other process variables. To properly implement this the variables are then actually implemented as dynamic states variable. This is done because in the Jmodelica.org transcription process of the Modelica model into the dynamic problem, parameters and dynamic variables are handles quite differently. Parameters cant be changed after transcription, meaning they would have to be constant for the entire control period. Dynamic states however can of course be changed by defining the derivative of the variable. Disturbance variables can therefore be adjusted by defining their derivative which is implemented as

$$\frac{d\boldsymbol{x}}{dt} = \alpha(\boldsymbol{x}_{set} - \boldsymbol{x}) \tag{4.2}$$

The variables can be initiated at a steady state by having its initial x_0 value equal to the x_{set} value. If can also then be changed at a given time point by changing x_{set} to the new desired value. The parameter will then change as a first order response where the time constant of its change is adjusted with the constant α. This is done to look at how the controllers deal with different changes to environment variables.

Table 4.2: Environmental variables that can change independently of other variables

Variable	Description
T_{amb}	Temperature of cooling water
$T_{LNG,in}$	Temperature of natural gas at the inlet
UA_M	Heat transfer coefficient for methane condenser
UA_E	Heat transfer coefficient for ethane condenser
UA_P	Heat transfer coefficient for propane condenser

In table 4.2 the variables that are implemented in this fashion, where they can be subjected to unnoticed changes are listed.

Blocking factors

Within each discretization element the solver will try to find the optimal input values at each collocation point. This means that the input will be able to change during a single sampling instance. This possibility of change within each element can be removed by using blocking factors on the inputs, which will causes them to be piece-wise constant. Firstly there is a computational advantage to this as there then only is one input value that must be found per discretization element. Blocking factors can also be used to set for how long inputs must be constant. The control horizon, which is the final value of the input can then also be set. It can for example be half of the prediction horizon, meaning that for the last half of the prediction horizon all inputs are held at a constant value.

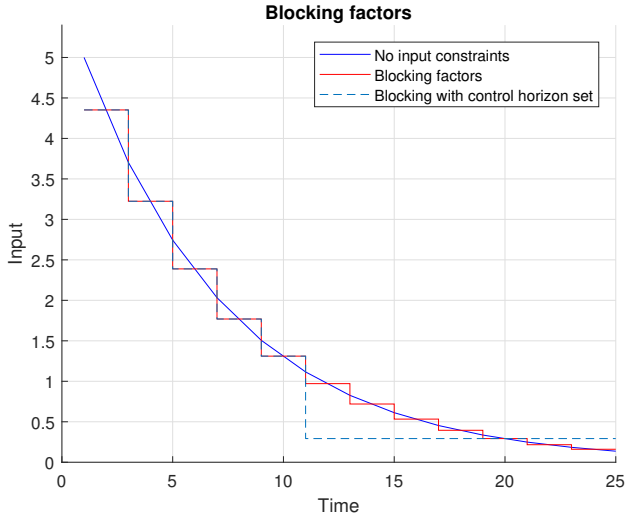


Figure 4.1: Example of how blocking factors can be used to keep inputs piece-wise constant for different lengths of time

In figure 4.1, how blocking factors can be used to control the inputs values is illustrated. For the blue line there are no blocking factors meaning the input can change freely within each element. For the red line however each input value must be held constant over the different elements, in this case of length 2 seconds. The dashed line is an example of when the blocking factors are used to adjust the the control horizon, as after 11 seconds the input value must be held constant.

Another reason control inputs are often help constant between samples, is that within each element exactly how the changes affect the model are not always known. Changing the input between samples might not be the most optimal way of controlling the process, and it ads more complexity which is not desired. As the process states are only measured in intervals, not continuously, it makes sense to hold the inputs constant until new measurements can be taken. However as the control input is then discreet and not continuous, some non continuous behaviour is introduced to the model. Because the valve opening positions also then need to change between elements, discreet behaviour is introduced on the outlet temperature. This is due to the model itself and not a feature of refrigeration cycles in general. This will be explained further bellow.

Initial guess and prediction horizon

The model is first initialized by using a certain set of inputs values, and using special initial equations that require the derivatives of the state variables to be zero. Internal Modelica functions then find a consistent set of initial conditions for all variables in the system model that are then given as start values. By then changing the inputs values and simulating for a certain time horizon the system will react to the new inputs and after a certain time

stabilize at a new steady state solution. The initialization input values and the simulation input values are shown in table 4.3 with the behaviour of the state variables of the receiver tanks shown in figure 4.2.

Table 4.3: Inputs for acquiring initial trajectory of the system variable. Compressor rotation values are dimensionless, and cooling water flows are 10^4[kgmol/h].

Input		Initialization value	Simulation value
$comp_M$	u_1	0.94	0.93
$comp_E$	u_2	0.89	0.87
$comp_P$	u_3	0.86	0.82
$mcond_M$	u_4	0.1	0.05
$mcond_E$	u_5	0.1	0.03
$mcond_P$	u_6	0.3	0.25

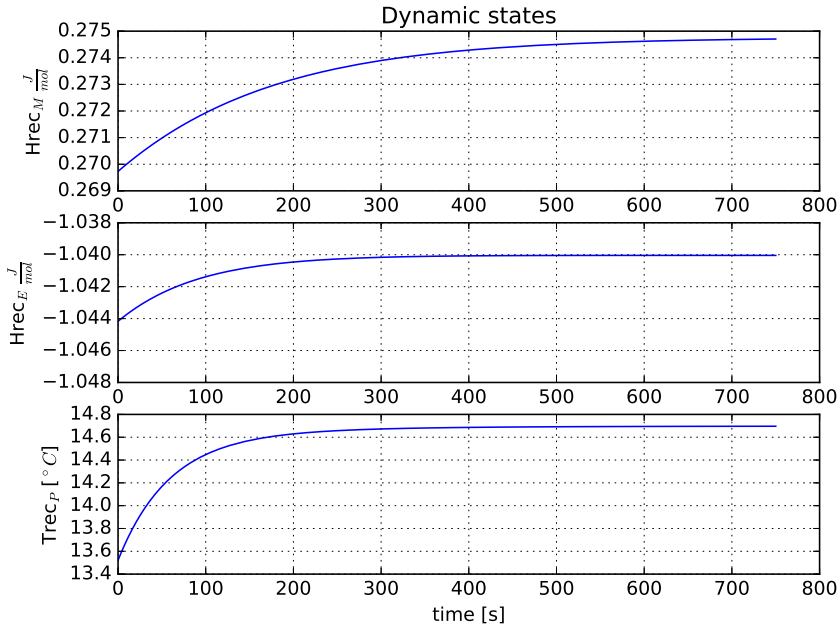


Figure 4.2: Changes to dynamic variables in the receivers after input changes shown in figure 4.3

Because the simulation values are lower than the initialization values, the three tanks can be see to all go to higher values in figure 4.2. For the methane and ethane receiver the specific enthalpies are shown and for the propane tank the temperature. Higher values in this case indicating less potential to cool the LNG stream. The main use of this initial simulation is to use it as an initial trajectory guess for the dynamic optimization problem, as it will be so solved significantly quicker when there are valid values for all decision

variables in the NLP. What it also shows is that the new steady state is reached after approximately 500 seconds with sight differences between the receivers due to them having different holdups. The predictions horizon of the OCP should be long enough so that it captures all of the dynamics in the system and is therefore set to 750 seconds. This is slightly longer than what might be necessary, but due to undesired solutions towards the end of the optimization is is implemented this way. To get a high enough resolution for quicker dynamics in the system, the number of discretization elements is set to be 50 elements with 3 collocation points each. Each element will then have a length of 15 second causing this to be being the sampling time, or duration where the inputs are held constant in the MPC implementation.

Optimal control problem

With the settings explained then used, the optimal control problem can be solved. It is initialized with the same initial conditions as those used for the previous simulation. The solution for the three compressor inputs N_M, N_E, and N_P which are the main inputs are then

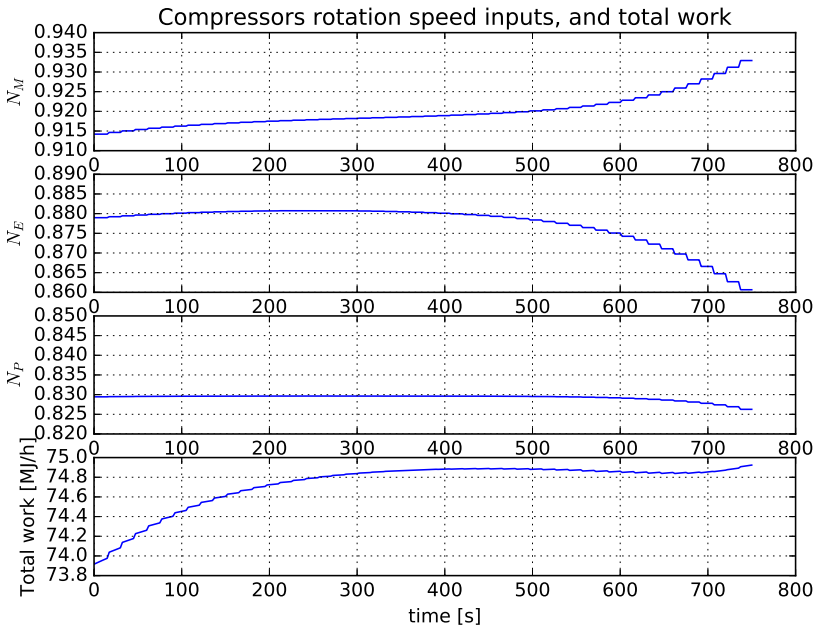


Figure 4.3: Optimized compressor speeds for each cycle as computed by optimal control problem, together with total compressor work

From figure 4.3 it is clear the system is initialized in a state where to much work is done as the total work required is low in the beginning, when the cooling potential in the reservoirs is used. Compared to the initialized values shown in table 4.3, each compressors

inputs have a lower input value which corresponds to a lower duty required. As the times moves towards 500 second mark, the inputs are quite stable as the total works starts to plateau at a steady rate in this area. After 500 seconds however the inputs all start to quickly divert from their optimal values, which is why the extra 250 seconds were added to the prediction horizon. In a proper and ideal solution for the system, the inputs would drive the system to an optimal state and then stabilize. Then for as long as there are no disturbances or change in conditions it would try to keep it there. The reason the calculated inputs don't do this and start to change towards the end, is due to the objective function not including anything beyond the prediction horizon. It is only trying to optimize within the 750 seconds, so it starts to utilize built up cooling potential in the receivers. In other words, the system can for a short term use less total energy whilst still not violating any constraints. It can do this in the short term as long as it does not have to get back to the long term optimal state. This is easier to visualize by looking at the behaviour of some of the other variables. For example the same can seen in the ethane condenser where the cooling water flow is significantly increased.

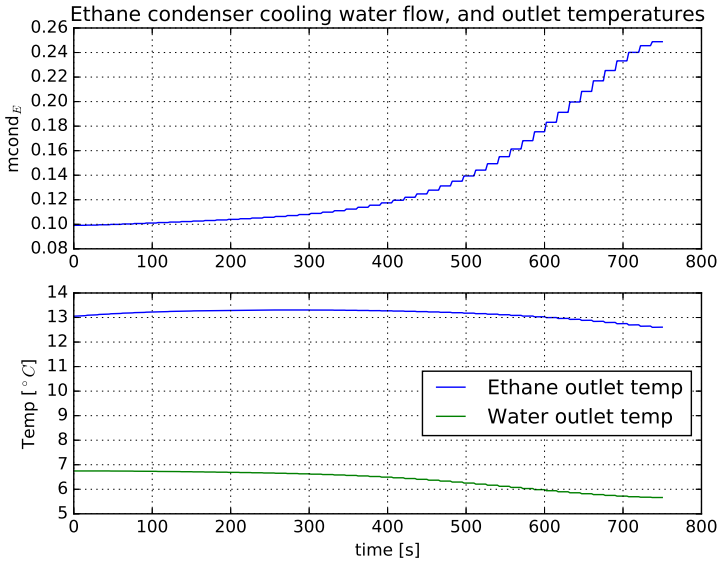


Figure 4.4: Ethane condenser cooling water flow and outlet temperatures

The increase in flow rate is much larger as the flow rate of the cooling water is significantly less penalized in the objective function. The positive effects of this increase on the cycle is however quite small as it only decreases the outlet temperature of the water, and thereby decreases the outlet temperature of the ethane slightly. The same behaviour is also clear from the flow rates in all the cycles. It can be see to increased towards the end as shown in figure 4.5 due to the valves opening up. The potential of the cold refrigerant stored in each receiver is used by increasing the flow rate from them. The system having been initiated at a to conservative state is also clear from the flow rates, which can be seen

to be quite low initially.

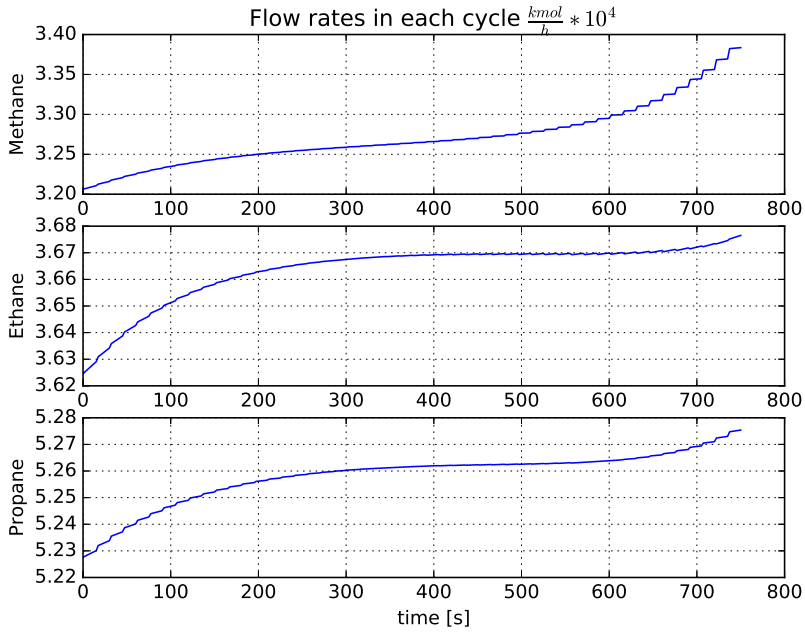


Figure 4.5: Refrigerant flow rate in each cycle as estimated in the optimal control problem

The main point of this is that the extra 250 seconds are added to the prediction horizon because what the solver find to be optimal towards the end, is not what actually is optimal in the long term. The real optimization horizon of the MPC is not only the prediction horizon, but for however long it is intended to run. Expect for adding some extra computational time as the problem is larger, this added time should however not be a problem. This is due to the fact that the solution can be seen to first go towards the desired steady state solution, meaning the initial desired response is not linked to the undesired response towards the end. And as only the first input is actually used from the solution , the later responses should have minimum effect on controller inputs used. As the prediction horizon is shifted at every step, the undesired inputs are also always shifted away.

Another thing of note for the process model is the behaviour from having no receivers after the evaporator section. Because the input variables in the compressor and condenser are discreet due to the blocking factors, the valve position also need to move in a discreet behavior between elements to fulfill other process conditions. Such as controlling the level of super heating or not violating the LNG temperature constraint. In figure 4.6 this is sort of seen for the first few valve samples. Through each samples the positions drifts to fulfill process conditions, but when the input variables change, is must jump to the new value before drifting again. This will also be visible later when plotting the LNG outlet

temperature during larger changes to the inputs. In figure 4.6, the temperature of the LNG outlet temperature is also plotted and it is clear the optimal solution is for it to be as close to its limit as possible as it is an active constraint.

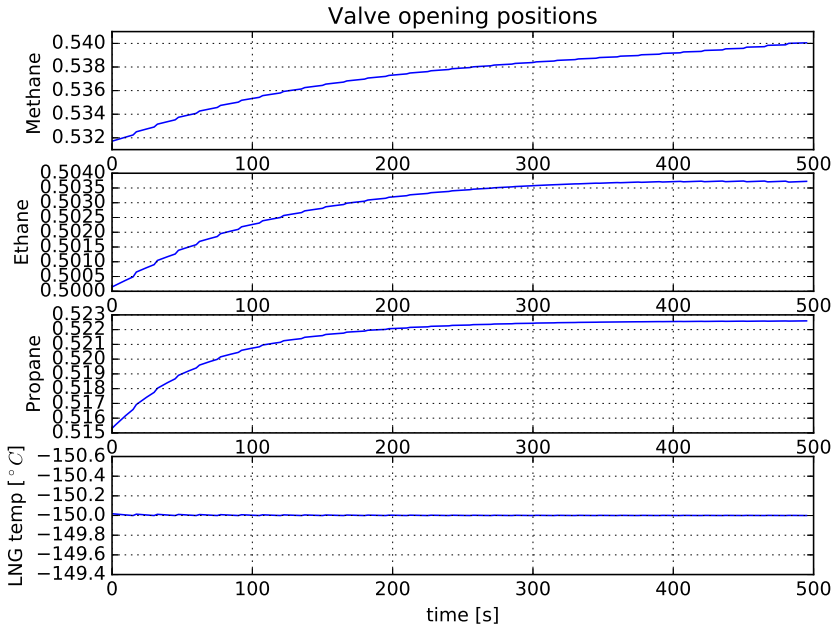


Figure 4.6: Opening position of the three JT-valves and outlet temperature of the LNG

4.2 MPC

The initialization and solution of the OCP can then be included in an MPC algorithm to actually control the process online. The main sett-up is similar to that for only the optimal control problem with some added extra steps. After a solution is found the first set of inputs for the six input variables are extracted and logged. A separate simulation model with the same initial conditions and initialization is then used to simulate for the duration of the sampling time using the logged set of optimal inputs. The final values of the state variables after the simulation is done can then be taken out and used as new initial conditions for the next optimal control problem to be solved. The optimization horizon is also shifted forward by one sampling duration, in this case 15 seconds.

The computation time used for the transcribing and initialization of the problem into an NLP is a significant part of the total time used by each step. But as each successive problem is identical in structure to the previous one, meaning it is made up of the exact same variables and equations, the discretization can be reused. The same initialized discretization of the problem is then used in each optimization step by changing the initial

conditions and parameter, and by shifting the prediction horizon. By doing this the only significant part of the computation time left is the actual solving of the NLP by the solver IPOPT. New initial guesses between steps are then also given by using the warm start capabilities of IPOPT. When a warm start is used, the solver object uses a previous solution with different parameters values as initial guesses for the decision variables. Alternatives to this would be to explicitly give the previous optimization new trajectories as a given initial guess, but as the warm start option is found to be the quicker option it is used instead.

For the first simulation there are no changes to any environment variables or any measurement noise on the states. This causes the estimated model behaviour to be very similar to how the controlled model actually behaves, due to them being the same model. The only difference between them stemming from the discretization occurring for the NLP. The controller will then simply controls the model until it reaches a state where it is in a state considered optimal. This can be seen from figure 4.7 where the value of the dynamic variables can be seen to stabilizing at their respective optimal values.

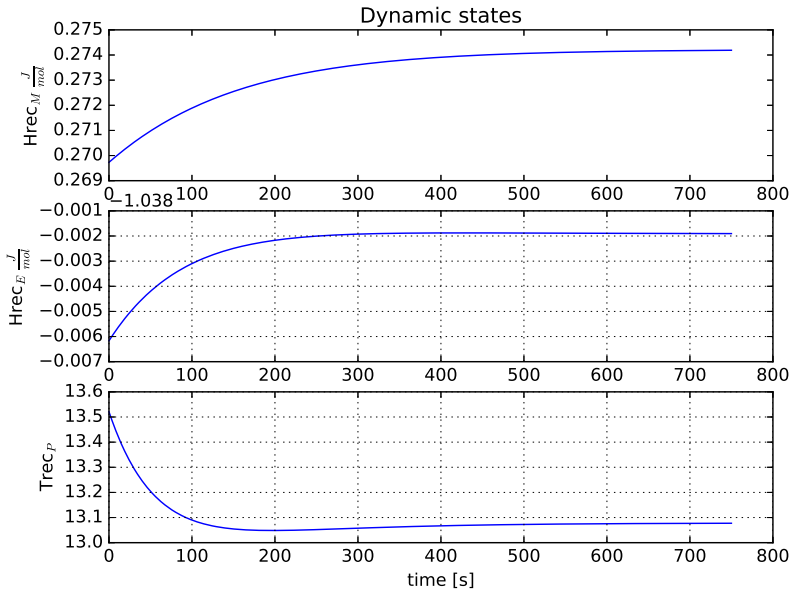


Figure 4.7: Values of dynamic states in receivers for MPC simulation with no disturbances or changes

The same can be seen for the input values for the compressors used which are shown in figure 4.8. Here each step shown is the first optimal input computed in each successive optimal control problem, which are the inputs actually used

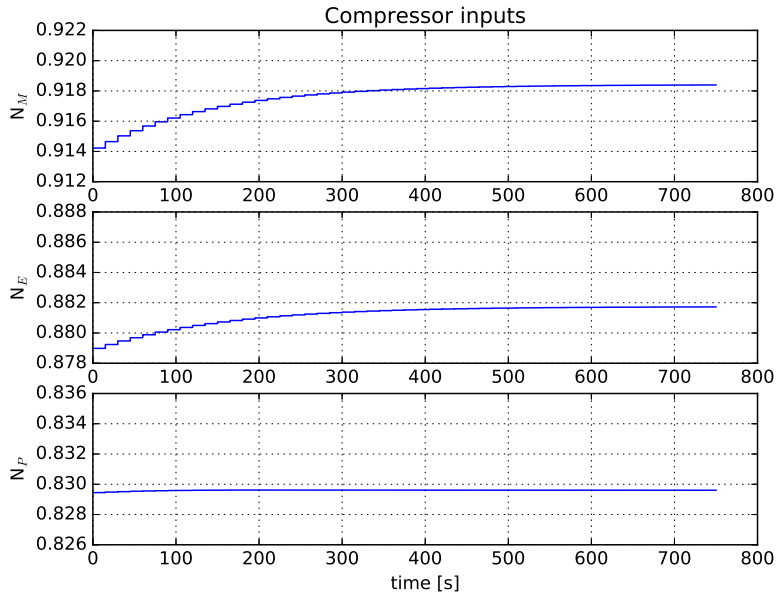


Figure 4.8: Optimal inputs used for each compressor during simulation of model controlled by MPC

By simulating the system with the MPC controlling it for a long time, the system will reach its the optimal operating point. At least what is considered the optimal solution according to the objective function. The values of different variables at this point give an indication of how the system is then operating. In table 4.4 values for the different variables of the refrigerant streams are shown. These are the low and high pressure levels, the different temperatures, and the flow rate in each cycle. The different temperature values, are for the condenser section where the refrigerant will have a different temperature after each successive heat exchanger unit as shown in figure 3.3.

Table 4.4: Value of different variables at optimal steady state during operation

Methane variables			
P_{low}	1.76 bar	P_{high}	33.45 bar
T_{comp}	83.4 $°C$	$T_{cond,1}$	13.2 $°C$
$T_{cond,2}$	-39.8 $°C$	$T_{cond,3}$	-92.7 $°C$
T_{valve}	-154.7 $°C$	m_{flow}	3.263 $\cdot$ 10^4 [kgmol/h]
Ethane variables			
P_{low}	0.42 bar	P_{high}	7.95 bar
T_{comp}	78.6 $°C$	$T_{cond,1}$	13.4 $°C$
$T_{cond,2}$	-39.1 $°C$	T_{valve}	-97.8 $°C$
		m_{flow}	3.672*10 $\cdot$ 10^4 [kgmol/h]
Propane variables			
P_{low}	0.89 bar	P_{high}	6.90 bar
T_{comp}	41.2 $°C$	$T_{cond,1}$	13.1 $°C$
T_{valve}	-43.3 $°C$	m_{flow}	5.265 $\cdot$ 10^4 [kgmol/h]

The same can also be done for the different process units in each cycle. The power consumption of the compressors and the heat transferred in the heat exchanger units are also of interest at the optimal point.

Table 4.5: Value of different heat transfer rates for the process units and the compressor duties at optimal operation. All units are in [MJ/h]

Compressor duties [MJ/h]					
$Comp_M$	19.933	$Comp_E$	28.893	$Comp_P$	26.153
LNG heat exchangers [MJ/h]					
$hex_{L,P}$	12.048	$hex_{L,E}$	20.387	$hex_{L,M}$	7.746
Cooling water heat exchangers [MJ/h]					
hex_M	8.330	hex_E	13.303	hex_P	93.528
Cross cycle heat exchangers [MJ/h]					
$hex_{E,M}$	18.032	$hex_{P,M}$	5.620	$hex_{P,E}$	54.009

What these numbers show is quite interesting for a few reasons. For example as the propane heat exchanger is modeled as having a significantly larger heat transfer coefficient, its total heat transfer is naturally larger than for the other two. This is also due to the fact that for the propane cycle, it is only the cooling water that contributes to condensing it. The other two cycle are also condensed by the other streams, for example the ethane stream which can be seen to mostly be condensed by the propane stream in the $hex_{P,E}$ heat exchanger. Otherwise most numbers are mostly spread out evenly, such as the duties of the different compressors, or the heat transferred from each refrigerant cycle to the LNG steam, which are all of the same magnitude.

Measurement noise

Because there is no actual plant being controlled, the model used for the simulation is the same as the one used for the optimal control problem. Therefore what the controller estimates to to be the plants response, is almost exactly the same as the actual response during the simulation. There is therefore no plant-model mismatch for the controller. To simulate some mismatch randomly generated noise is added to the measurement of the three receiver states $H_{rec,M}$, $H_{rec,E}$ and $T_{rec,P}$. Randomly drawn numbers from normal distribution with a standard deviation of 0.5% their nominal values are added to each state during the sampling. The state which is then used to compute the next set of inputs wont be exactly the same causing the inputs to not be perfectly optimal for the system.

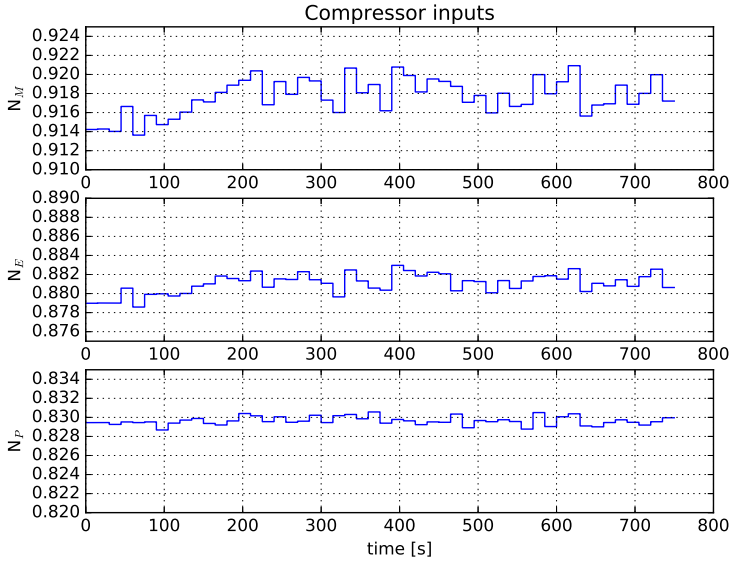


Figure 4.9: Optimal inputs used by compressors in simulation with MPC controller with measurement noise on states

From the computed compressor inputs it is clear that although the inputs are erratic,

due to the the disturbances on the measured states, their averaged are however the same as for the ideal scenario. The effect is significantly less visible on the states themselves as they are not actually changing, only the measurement of them is wrong. The small changes that are visible do however come from the inputs always being changed as they are trying to keep them as close to the optimal as possible.

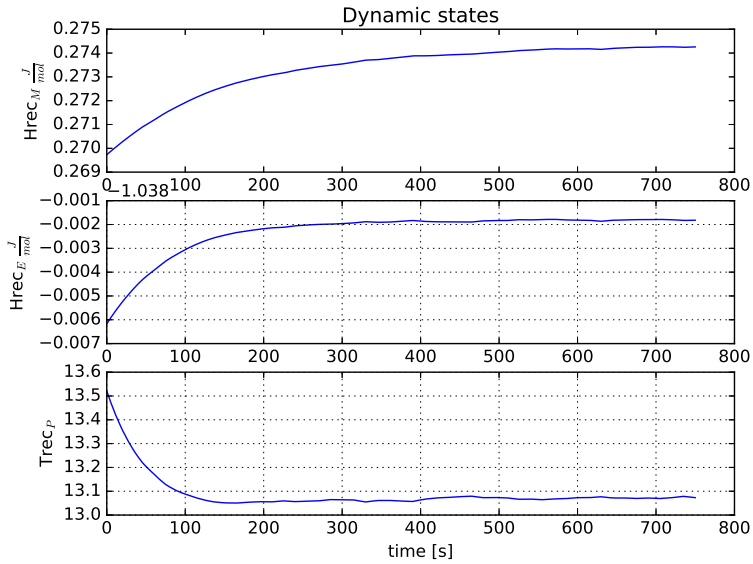


Figure 4.10: Dynamic state values for MPC simulation with noise

The same effect is clear on the opening position of the valves. As their position is not set as a control input determined but determined by the model, they must constantly changed to make up for the ”mistakes” made by the compressors and the condensers in order to keep other process conditions intact.

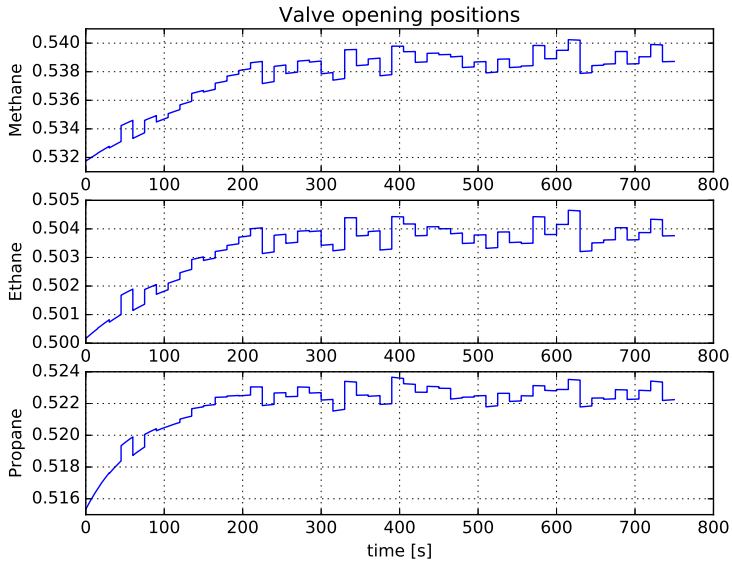


Figure 4.11: Valve opening positions for MPC simulation with noise

4.3 Robust MPC

For the robust MPC there are a few changes from the standard MPC formulation. In the following example is it assumed there are uncertainties on the cooling water temperature T_{amb}. For the formulation of the model itself there are then three object made of the process class. Where they each have a different value for the parameter that specifies the temperature of the cooling water. One of these values being the expected value which is the same as for the previous MPC, and the other two an upper and a lower limit of expected values. The controller should then be able to handle any value within this range. The three temperature instances are then

$$\begin{aligned} T_{amb} &= 5°C \\ T_{amb,l} &= 4°C \\ T_{amb,u} &= 6°C \end{aligned} \tag{4.3}$$

Each object instance then has its own full set of equations and variables in the OCP formulation. As for the objective functions and the constraints described in equation 4.1 and table 4.1, they are then also introduced to the OCP for each new instance. New constraints are also added for requiring that the first set of input values have to be equal for all inputs

across the three instances. These new constraints are then

$$\begin{aligned} u_1(0) &= u_{1,l}(0) = u_{1,u}(0) \\ u_2(0) &= u_{2,l}(0) = u_{2,u}(0) \\ u_3(0) &= u_{3,l}(0) = u_{3,u}(0) \\ u_4(0) &= u_{4,l}(0) = u_{4,u}(0) \\ u_5(0) &= u_{5,l}(0) = u_{5,u}(0) \\ u_6(0) &= u_{6,l}(0) = u_{6,u}(0) \end{aligned} \tag{4.4}$$

These point constraints are applied at the initial time of the prediction horizon, and as they are all limited by the blocking factors are then valid for the entire first sampling period.

Because the problem become significantly larger due to the three instances being computed and optimized in parallel, some changes are made to the prediction horizon and the number of discretization elements. The new prediction horizon is set at 700 seconds, with 35 elements N_e. The number number of collocation points is however unchanged. Although this reduction helps with the problem size, the downside is that the new sampling period is 20 seconds. The prediction of rapid changes is therefore less accurate and if there are changes to environmental variables, these could go unnoticed for an extra 5 seconds as a result of the longer sampling period. Prediction of slower dynamics and general process behaviour should however not be affected by this decrease in resolution.

Except for changing the controller to the robust MPC variant, all other parameters are the same for this simulation. In figure 4.12 the three different optimal trajectory paths for the compressor inputs are shown. As required by the new constraints, they all have the same input for the first sampling period. One thing of interest for the three different compressors are the values they stabilize towards. For the methane and ethane compressor they both drift towards the same value, especially the methane compressor. This is not seen for the propane one and is probably explained by the fact that the propane cycle is significantly more influenced by the temperature of the water as previously discussed due to its higher total heat transferred in its condenser. As for the general trajectory paths for the three cases they do make sense. For the lower case when the cooling water is colder, less energy is needed by the compressor and the opposite for the upper case where more energy is needed. If the refrigerants can be cooled lower, they wont need to be as highly pressurized by the compressors. In figure 4.12 the trajectories plotted are the results from the first OCP after initialization.

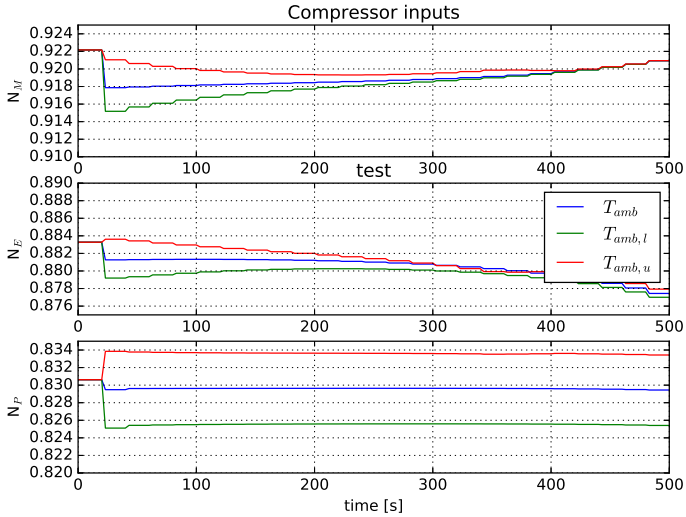


Figure 4.12: Optimal input trajectories for the compressors for the three different cooling water temperature cases estimated in robust MPC

It is also clear that the inputs used are slightly more conservative then what the estimated middle case requires. This is quite clear for the methane and ethane inputs, as the requirements for the higher temperature case drags the first set of inputs up. In the other two temperature cases, the estimated case and the lower case, the ideal input is lower as the figure shows. The effects of this when simulating is a more conservative controller that is over cooling the natural gas beyond what is required by the constraints. This can be seen in the temperature of the LNG at the outlet which is plotted in figure 4.13. The constraint on the outlet temperature is still the same as before, that it must be bellow -150 $\circ C$. But now, due to the controller taking the other cases into consideration, it is operating at a point where the temperature is at steady state is slightly bellow the constraint.

4.4 PI controller

When implementing the feedback PI controller it is important that the controller is used to control the correct variables for it to work well. In the same way as for the MPC there are here also constraints that must be controlled. In Jensen [2008] the topic of optimal operation of refrigeration cycles is discussed in detailed for a PRICO process, and for this model in particular self optimizing control is implemented in Verheyleweghen and Jäschke [2018]. Where Self optimizing control is a strategy for selecting control variables that minimize loss for of objective function when there are disturbances. Where the aim is to find a set of variables such that when controlling them to a constant set-point only leads to an acceptable loss during any disturbances.

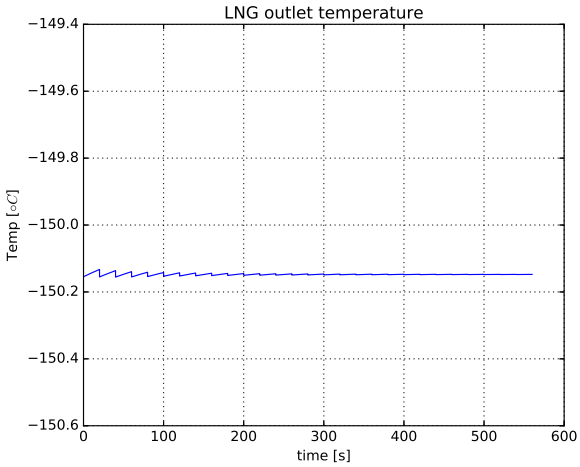


Figure 4.13: LNG outlet temperature during simulation

The LNG outlet temperature is of course also an active constraint for the PI controller, meaning it must directly be controlled. The methane compressor is used to control the outlet temperature as is is the methane cycle which decides the LNG streams final temperature. For the sake of simplicity the cooling water flows are kept constant at their optimal values when the PI controller is used. This leaves the two other compressor inputs as MV's that can be used to control other variables. These controllers can be used in both a MIMO fashion meaning multiple input multiple output, or a SISO fashion for single input single output. For a MIMO controller a combination of measurements are used to determine the MV's value, whilst for SISO controller only one variable is used.

As the PI controller is not the main focus of the thesis, and it's purpose is primarily for comparison with the MPC controller, only a simple temperature feedback strategy is used. For this the other two PI controller for the compressors are used to keep the temperature of the LNG stream, after the other two heat exchangers, at their nominal values. The set-points used for this are taken from the optimal state found in 4.4, which causes the three set-points for the temperatures to be : -37.9 $^\circ C$, -92.2 $^\circ C$, and -150 $^\circ C$ respectively after each heat exchanger unit.

Chapter 5

Controller comparison and discussion

In this chapter more specific test will be done showing how the controllers behave for disturbances. How the controllers decide to act when changes to environmental variables are induced will be shown. The MPC controller will be compared to the PI controller, with the aim of better displaying the differences between the two control strategies. A comparison between the standard MPC and the robust MPC will also be shown for an unnoticed change. Towards the end of the chapter, a more general discussion of the controllers, and how Jmodelica.org as platform worsks for MPC development will be given.

5.1 Comparison between MPC and PI controller

Change to cooling water temperature

For the first controller comparison a change in the cooling water temperature used by the condensers will be looked at. For all comparisons the system is initialized at the optimal optimal point previously found in chapter 4. Then at the 75 second mark, a 5 degree increase in the waters temperature will be induced. The time constant for this change is set by the value of the α parameter in equation 4.2, which will for this test be set to 0.05. This will in other words cause the time constant to be 20 seconds, meaning the change will be 95% complete complete after 60 seconds. It will then be a relatively fast change for the system in comparison to the slower dynamics of the receiver states. For the tests the main points of interest is minimizing the total power consumption of the three compressor. Any constraint violations on the LNG outlet temperature during are also of interest. If there are to many violations of the temperature constraint there is a need for a back for the constraint limit. If the constraint is backed of, in other words lowered, it will cause more conservative control and thereby increase the energy used by the controller. For comparison purposes the measurement noise which was previously present for the state measurements is turned

off in these tests.

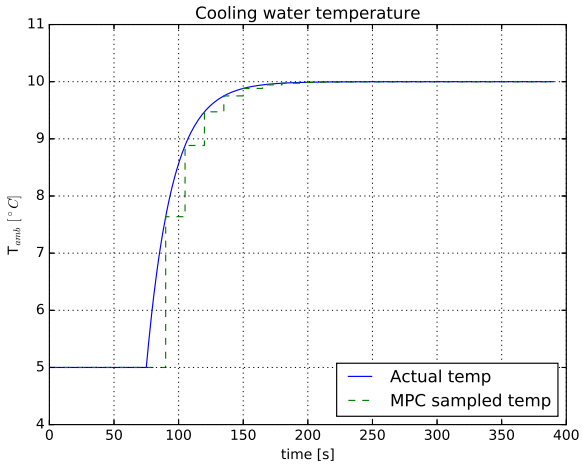


Figure 5.1: Temperature change of cooling water temperature with temperature in the simulation model and values sampled by the MPC shown

In figure 5.1 the temperature of the cooling water used in the simulation model is plotted together with the temperatures which are sampled by the MPC. Due to the MPC only sampling at 15 second intervals, it will not immediately be aware of the changes when they are made. It can only update the information it uses when it samples the process. Because the MPC is controlling the LNG outlet temperature as close to its constraint limit as possible, these initial unnoticed changes will cause it to violate the constraint in small segments. The MPC controller cant take any action before it is aware that there has been a change to the cooling water temperature. Due to the fact that the information is correct at the beginning of each sampling element, it will always be start of correct at the beginning of each sample before maybe drifting away. Because there is implemented time delay for the PI controller, there will also be a period where it will not taking any control action.

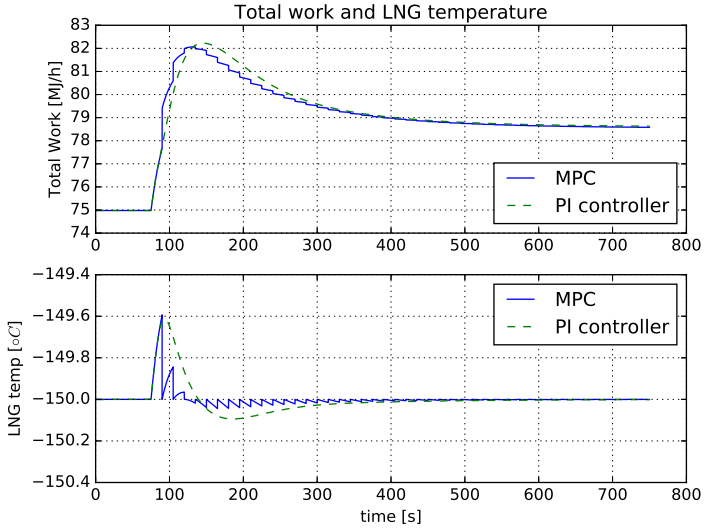


Figure 5.2: Total compressor work and the LNG outlet temperature for cooling water temperature change

As for the actual responses they are surprisingly similar between the MPC and the PI controller. Especially the total work used by the compressors as is shown in figure 5.2. During the transient period between the initial steady state and the new steady state, the total work done by the model controlled with the MPC is generally lower than when controller by the PI controller. This is especially true for the time after the peak. This makes sense considering the MPC is trying to find the optimal way of getting to the new steady state. The PI controller does not know anything about the total work done, and is just trying to minimize error for it's set points. Minimizing the work done during the transition is also in the objective function for the MPC. This different approach is also reflected in the temperature of the LNG outlet during this transition. The MPC is controlling in more "strict" fashion, trying to keep the temperature as close to the limit as possible while minimizing work. The PI controller however slightly overshoots its set point when it is taking control action. It is therefore operating to conservatively for a brief period. As the controllers both reach the new steady state, the total work done by both is relatively similar compared to the absolute change. The MPC does however find a slightly lower point than the PI controller at the new steady state as figure 5.3 shows. This indicates that for this disturbance in particular, choosing to control the temperature of the LNG after each condenser, is not to bad a choice of CV's. The set point for the PI controller are quite close to what the MPC would consider optimal. As for the MPC's outputs not being continuous in the plots as the PI controllers are, is due to the blocking factors. The blocking factors cause the control inputs for the MPC to be discreet between elements. The PI controller has no restrictions on the inputs for the compressors, and they are therefore continuous throughout the simulation.

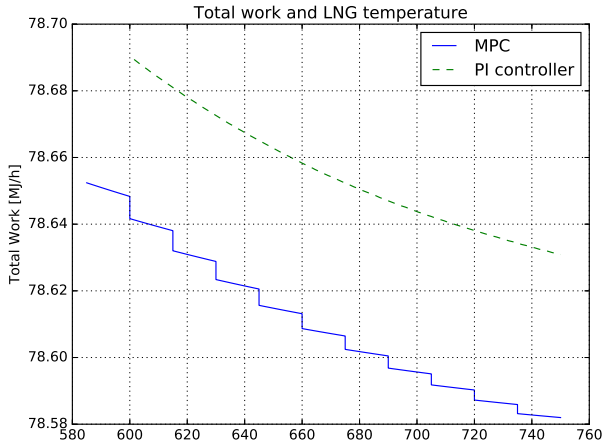


Figure 5.3: Total compressor work for the MPC controller and the PI controller as they approach the new steady state

Natural gas inlet temperature

For the next comparison the temperature of the natural gas is increased by five degrees at it's inlet. This change is done slightly slower than the previous one, now with a time constant of 33 seconds. Otherwise the comparison is done with all conditions being the same as for the previous test. In figure 5.4 the total compressor work and the LNG outlet temperature are again shown for both controllers. Although there are practically no constraint violations, or changes to the LNG temperature at the outlet, there are clear differences that can be seen for the total work done by the compressors. For the same reasons previously noted for the other test, during the transition period the MPC calculate a better and more efficient transition in regards to the power consumption. Its path from one steady state to the next one is more efficient.

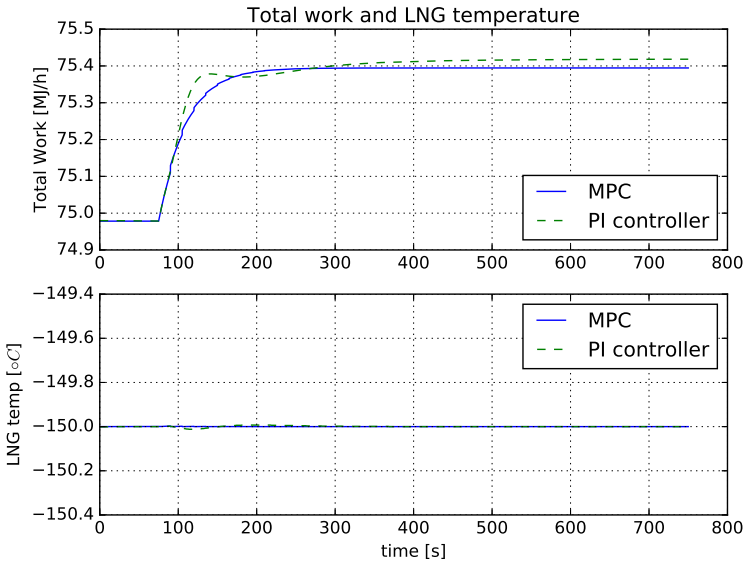


Figure 5.4: Total compressor work and LNG outlet temperature for natural gas inlet temperature change

What is interesting in this example however, is that the controllers clearly settle at different new steady states. Although the axis for the total work is smaller in scale, the MPC is clearly bellow the PI controller in regards to the total work. What is actually occurring is more clearly shown by instead looking at the actual compressor inputs, which are shown in figure 5.5. Although the methane cycle compressors input barely changes for either controller, the other two compressor inputs have very different control action. The input for these two controllers both change but they divert in completely different directions. The PI controller chooses to increases the propane compressor speed and decreases the ethane speed, whilst the MPC does the opposite. The PI controller increasing the work done by the propane compressor makes perfect sense considering what change is induced

to the process by the temperature increase. As the set point for the temperature out of the propane/natural gas heat exchange is the same as before the change, the extra duty required to cool it is all put on this first heat exchanger. Which is the heat exchanger mainly affected by the propane compressor. Due to the cascade structure of the process, the ethane compressor now requires slightly less work as it is now cooled even more by the propane cycle. The effects are however mostly evened out for the methane refrigeration cycle, as its input barley changes.

The MPC however takes a different control approach. In it's case, it instead increases the duty of the ethane compressor and slightly decreases the other two compressors while still properly controlling the process. It has found that increasing the power to the ethane cycles is a more efficient way to cool the natural gas, given the change in circumstances. This is a good example of how the MPC's objective function can find a new optimal operating point for changing circumstance. It is not bound by constant set points that are calculated offline. This is exactly the case for the PI controller, as they are taken from the optimal point based on the plant under other circumstances. They are therefore not necessarily the optimal set point for changes to the environmental variables, such as is the case for the change to the temperature.

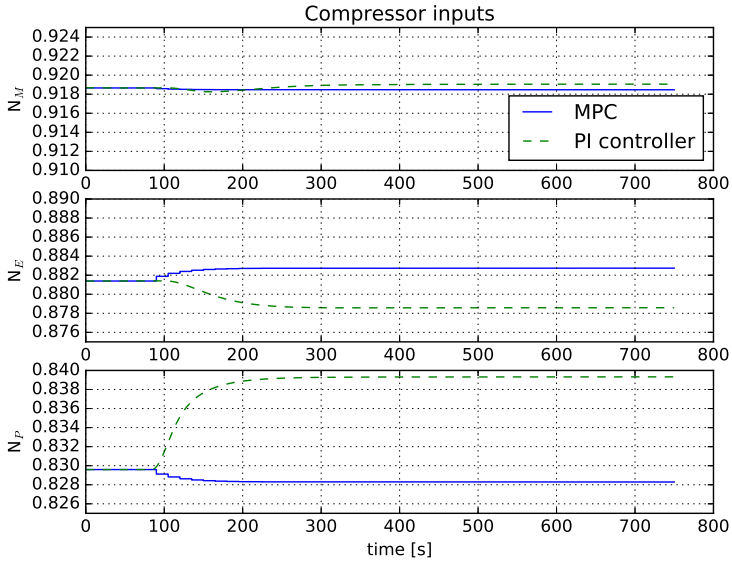


Figure 5.5: Compressor inputs for change to LNG inlet temperature

5.2 Robust MPC compared with Standard MPC

From the first comparison where the cooling water temperature is increase by 5 degree, it is clear the MPC can easily adapt and control it as it finds a new optimal operating point.

The reason that there are constraint violation by the outlet temperature is due to the sampling rate not immediately picking up the changes. This causes the MPC to operate for brief periods of time using wrong information. However, this assumes that when the MPC samples the process, the samples of the cooling water temperature are perfect and the MPC then knows exactly what the temperature is. If there instead where to be a change to the temperature the MPC did not pick up and was unaware of, there very well could be serious violations of the constraint over time.

This is where the benefits of the robust MPC implementation come into play. When the robust MPC is controlling the process, disturbances to the variable that was used for the different cases should not result in any violations of constraints. This of course assumes the disturbance is withing the range which was used in the implementation of the controller. To test this the standard MPC and the robust MPC are both compared when the model is induced to a small change in cooling water temperature. An increase of 0.75 $^{\circ}C$ is done to the cooling water temperature, but now this information is not sampled by either controller. They will operate not knowing that any change has been made. Except for this, the settings used by both the standard MPC and the robust MP are the same as the previous tests.

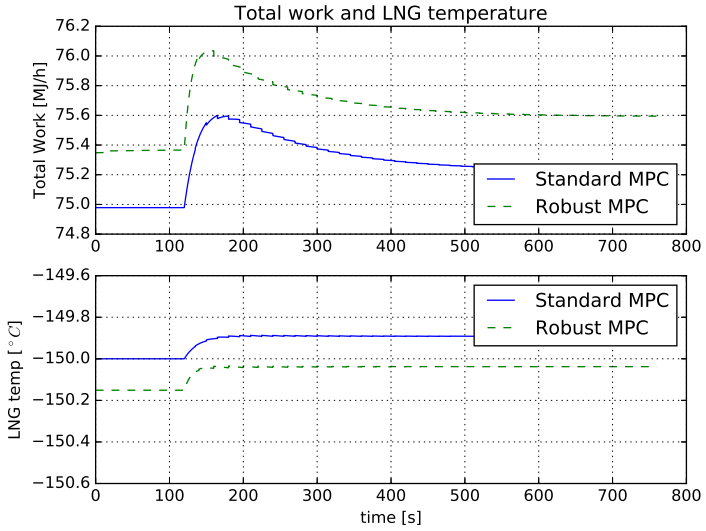


Figure 5.6: Total work and LNG outlet temperature for unnoticed change in cooling temperature

Because the robust implementation is optimizing the input under the assumption that the cooling water could be one degree higher, it is operating more conservatively. This can be seen from figure 5.6, where the total work is higher than the standard, and the outlet temperature is settles at a lower point than for the standard MPC. What it considered the better performing controller obviously depends on the nature of the constraint. If it is a constraint that can occasionally be slightly violated, such as if there where a buffer tank

after the outlet. Then the standard implementation could be operating acceptably assuming that it eventually picks up on the change and controls correctly. On the other hand, if the constraint can not be violated at any time point then the robust implementation is preferred. Another alternative could be the standard MPC but with a change of the constraint limit to be lowered further. The drawback of course being the extra work due to the controller being more conservative all the time. The robust MPC is also giving the optimal input when all three cases have to be taken into account. The lowered sampling rate used by the robust implementation, and its increased computational time are also things that must be evaluated when consideration which controller is preferred.

One thing to note is that the controller inputs do change for both controller for this comparison, even though they are not aware of the temperature change. The unnoticed change does however affect the other states of the process model, most notably the receiver. The receivers for instance will find new steady state values for the dynamic variables, which will cause the controllers to adjust themselves accordingly. The inputs used for both controllers are shown in figure 5.7.

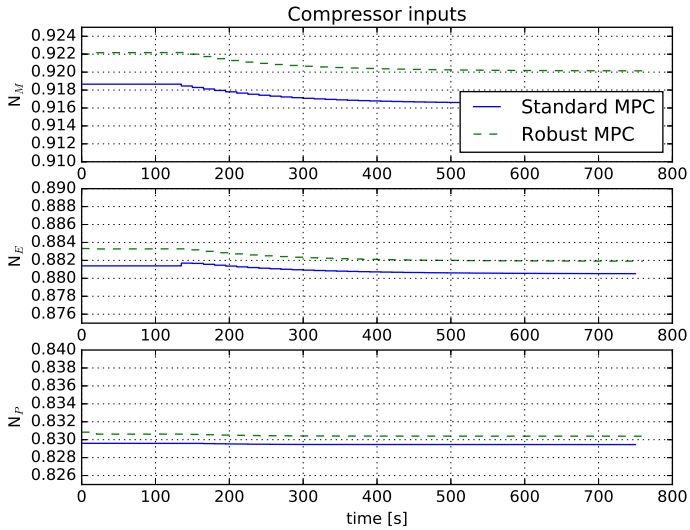


Figure 5.7: Compressor inputs for unnoticed change to cooling water temperature temperature

5.3 General discussion

The different comparison tests that have been shown, have demonstrated some of the different control structures made in this thesis. The point of this to validate the controllers and look at how their control behaviour differs. Not to explicitly decide which performs better overall. If a more concrete decision where to be made, more work would have been needed to be done optimizing the controllers, especially the PI feedback controller. By

better tuning the PI controller, and using a better choice of feedback variables to control, its performance would certainly have been improved. As it stands however, the PI controller does quite well highlight the differences in behaviour between MPC controller and traditional temperature feedback controller which was the main point.

There are also a few different areas where improvement can be made to the MPC controller. Firstly, by reformulating the objective function, or using different tuning of the control objectives, the controllers behavior can be altered. For example if it where wished that the cooling water molar flow where to be more heavily penalized in the objective function. With the OCP formulation used by the MPC, there is in general a lot of freedom in how one wants to control the model. Blocking factors can for example be used to limit controller input but this can also be done by altering the objective function. By penalizing large changes to input, or setting an explicit constraint on the inputs rate of change it can also be altered by the OCP. One reason one might want to limit rapid changes to control input could be to minimize the wear and tear of it continuously changing. The MPC's formulation naturally allows for this, something that cant be done in the same way with a PI controller.

For the MPC controller there is however the possibility that computational time for the solution starts to become a real problem. Although this has not been a problem for the test done in this thesis, as the solution times for each step have generally been 10-15 seconds for the standard MPC, and roughly 20 seconds for the robust MPC. If changed where made to include more discretization elements, or more branches and stages are added to the robust MPC, computational time could become a problem. Many of the modeled units that have been used, such as the heat exchangers, are also quite simplified from how they actually behave. Updating the process model with more accurate units, and including more dynamic states for the process could significantly increase the computational time for each step, which would be negative for the model based MPC controller.

There are however also things that can be done do improve the computational time, which has not been looked into in this thesis. Not only are there are many settings that can be changed within the solvers, but different solver could also be used. IPOPT, the solver which was used, has embedded within itself the ability to use many different linear solvers, each having different advantages and disadvantages. However, the only linear solver that has been used had been MUMPS. How the initial guess used by the warm start option can also be tuned, which could maybe have decreased the solution time for each step. Higher up in the control framework, changes can also be made to the transcription process itself. The computational time is generally quite closely related to the size of the problem in this case. Due to the long prediction horizon used, and the requirement for a high resolution of the trajectory, the resulting problem size has been quite large. One way the size of the problem could be reduced, is by adjusting the size of each discretization element to more closely match its required size. Except for it being easier and more simple, there is no reason the discreet elements all have to be of the same length as they have been. Improvements can probably be made by increasing the length of the elements towards the end of the prediction horizon. As previously talked about, the behaviour of the system towards

the end is not really of any interest anyway. Jmodelica.org does include the ability of changing the length of each discretization element, in the same way the blocking factors can be used to control inputs lengths. By using this feature, a high resolution could be used in the beginning, with lower resolution towards the end.

One of the crucial downsides noted for the PI controller, is it's lack of ability to change the set points it is trying to control the process towards. If implemented in a real world application this would probably not be a problem, as there would usually be another control layer on top of the PI controllers. An online optimization layer, such as an MPC controller, would usually sit on top and compute what the set points for the PI controllers should be. Thereby not controlling the process directly, but the set point the slave controllers control towards. The actual actuators would instead be controlled by simpler controllers such as the PI controller, thereby utilizing the benefits from both controller strategies. If there where to be something wrong with the behaviour of the control system, it would be easier to identify these problems if PI controller are used on the ground. It can often be a lot harder to debug a controller such as an MPC, due to its actions not necessarily being easy to understand or directly logical in the short term. There are no benefits gained by implementing a complicated MPC control system, if operators are not comfortable using it. Traditional feedback controllers are well understood and easier to operate and debug. By using the MPC controller to instead set the set-points, the overall control structure is still making sure the process is controlled at its optimal point for the give circumstances.

Performance of model based controller methods such as MPC are as opposed to non model based controllers, very dependent on the quality and accuracy of the model itself. The model used is was originally developed as a steady state model, so there are improvements that can be made for the dynamic parts. As implemented now, the first sections that should be looked for improvements the model are adding the extra receivers in the cycles, and updating the heat exchanger models. The benefits of MPC controllers are more clear if there are higher level dynamics, or larger time delay's between control action and a process response. Simple feedback controllers such as a PI controller generally struggle with large time delays.

Changes or updates to different process units in the future is an example of why using Modelica for the modeling work is a great advantage. Future changes to the model can very quickly be implemented without the need for much extra interfacing work due to the object oriented approach of Modelica. As long as the new model units are implemented with the current input/output connectors already used, they simply need to replace the old units in the model structure.

A major benefit of using the Jmodelica.org framework is the separation of the modeling work to the Modelica code section, the definition of the OCP to the Optimica code, and the framework/ organizational of the MPC to Python scripts. By having the three areas separate from each other, making changes can often be easy as the interfacing capabilities of Jmodelica automatically take care of changes. For example, if new constraints are added or changes are made to the objective functions, these changes are automatically taken care of during the transcription process without the need to change the model or framework.

By also writing different Optimica code sections for the dynamic problem formulations, the same model can be reused for different problems. The freedom offered by handling of the platform in Python also allows for the use of the platform for a wide range of areas, such as the MPC.

The biggest advantage of using the Jmodelica.org platform is however the ability to combine the benefits of doing modeling work in Modelica, and the ability to solve complex dynamic problems with the numerical algorithms and tools embedded in the platform. Once a model has been properly implemented in Modelica and made sure to work properly, it can quite easily and quickly be used with Jmodelica.org to solve dynamic optimization problems. The reliable and flexible interfacing between these tools is of great value when used properly. There is off course also a slight downside to using the Jmodelica transcription algorithms, as it might be hard to solve problems outside the scope of the platform. Although the entire platform is open source, making changes to the already implemented algorithms is probably more challenging and time consuming than adapting a smaller tailor made program. This is not to say the Jmodelica platform is not flexible, as it clearly is shown for example by the implementation of the robust MPC.

Chapter 6

Conclusion

In this thesis a dynamic model of an LNG liquefaction process was developed in the Modelica programming language, with the intent to use it for control purposes. Then in the Jmodelica.org framework an MPC controller and a robust MPC controller where developed and implemented for the process model. For these controllers the intent was to successfully control the processes whilst mainly minimizing the energy used by the three compressors. A simple PI temperature feedback controller was also implemented as an alternative control structure that could be used to compare during various tests.

When run without any disturbances or changes to any environmental variables, the MPC controlled the model to an operating point which is previously known to be an optimal point. This point was controlled to both with and without measurement errors on the dynamic states of the receivers. Two tests where done comparing the MPC's performance to the PI controller for changes to environmental variables. For the first test, the temperature of the cooling water was increased by five degrees. In this test the most notable improvements on total work used, was during the transition from the initialized steady state to the next steady state. In the other comparison test the LNG inlet temperature was increase. In this test, although the change to total work done was relatively smaller overall, the improvements during operation at the new steady state were clearer. This was due to the PI controllers requiring new set points for the change to the parameter. Due to the PI controller not being optimized, no clear decision can be made as to the actual overall benefits introduced by using an MPC controller instead.

The robust MPC was also compared to the standard MPC for a small unnoticed change to the cooling water temperature. Because the change to the variable value was within the range built in to the robust MPC itself, it successfully controlled the process without any constraint violations. The standard MPC did however violate the constraint, as it is trying to control the process as close to the allowed limit as possible. The downside for the robust controller is however the more conservative behaviour which increases energy use, and the increase in computational time required for each step solution.

Overall the Jmodelica.org framework performs very well for MPC development. The benefits of using Modelica are combined with Jmodelica's ability to setup and solve dynamic optimization problems. Some areas have been noted where improvements can be made, such as developing better process model units for the heat exchangers, or adding the extra receiver for each cycle. Computational time can also be improved by either making changes to the transcription process, or better tuning the solvers used for this problem in particular.

Bibliography

Jørgen Bauck Jensen. *Optimal Operation of Refrigeration Cycles*. PhD thesis, NTNU, 05 2008.

Modelica. The modelica association home page. `https://www.modelica.org/`. 2018-04-20.

Johan Åkesson. Optimica—an extension of Modelica supporting dynamic optimization. In *In 6th International Modelica Conference 2008*, Bielefeld, Germany, March 2008. Modelica Association.

J. Åkesson, K.-E. rzn, M. Gfvert, T. Bergdahl, and H. Tummescheit. Modeling and optimization with optimica and jmodelica.orglanguages and tools for solving large-scale dynamic optimization problems. *Computers Chemical Engineering*, 34(11):1737 – 1749, 2010. ISSN 0098-1354.

P. O. Larsson, F. Casella, F. Magnusson, J. Andersson, M. Diehl, and J. kesson. A framework for nonlinear model-predictive control using object-oriented modeling with a case study in power plant start-up. In *2013 IEEE Conference on Computer Aided Control System Design (CACSD)*, pages 346–351, Aug 2013.

Mats Cavey, Roel De Coninck, and L Helsen. Setting up a framework for model predictive control with moving horizon state estimation using jmodelica. pages 1295–1303, 03 2014.

K. Berntorp and F. Magnusson. Hierarchical predictive control for ground-vehicle maneuvering. In *2015 American Control Conference (ACC)*, pages 2771–2776, July 2015.

Adriaen Verheyleweghen and Johannes Jäschke. Self-optimizing control of an lng liquefaction plant. 2018.

D.E. Seborg, D.A. Mellichamp, T.F. Edgar, and F.J. Doyle. *Process Dynamics and Control*. John Wiley & Sons, 2010.

Fredrik Magnusson and Johan Åkesson. Dynamic optimization in jmodelica.org. 3(2): 471–496, 2015. ISSN 2227-9717.

Fredrik Magnusson. *Numerical and Symbolic Methods for Dynamic Optimization*. PhD thesis, Lund University, 11 2016.

Björn Lennernäs. A casadi based toolchain for jmodelica.org, 2013. Student Paper.

E. Süli and D.F. Mayers. *An Introduction to Numerical Analysis*. Cambridge University Press, 2003.

Sergio Lucia. *Robust Multi-stage Nonlinear Model Predictive Control*. PhD thesis, TU Dortmund, 2014.

P. J. Campo and M. Morari. Robust model predictive control. In *1987 American Control Conference*, pages 1021–1026, June 1987.

S. Lucia and S. Engell. Multi-stage and two-stage robust nonlinear model predictive control. *IFAC Proceedings Volumes*, 45(17):181 – 186, 2012. 4th IFAC Conference on Nonlinear Model Predictive Control.

Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 2018.

Joel Andersson, kesson Johan, Casella Francesco, and Diehl Moritz. Integration of casadi and jmodelica.org. pages 218–231, 06 2011.

Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, Mar 2006.

M N. Usama, Sherine Awad, and M Shuhaimi. Technology review of natural gas liquefaction processes. 11:3541–3546, 12 2011.

ConocoPhilips. Conocophilips liquefaction homepage. `http://lnglicensing.conocophillips.com/what-we-do/lng-technology/optimized-cascade-process/`. 2018-04-24.

P.M. Dranchuk and H Kassem. Calculation of z factors for natural gases using equations of state. 14, 07 1975.

E. W. Lemmon, R. T. Jacobsen, S. G. Penoncello, and S. W. Beyerlein. *Computer Programs for the Calculation of Thermodynamic Properties of Cryogens and other Fluids*, pages 1891–1897. Springer US, Boston, MA, 1994.

Appendix

A - Constants

If not otherwise noted in the main text, all constant are taken from Verheyleweghen and Jäschke [2018].

Table 6.1: Compressor efficiency constants

Cycle	e_1	e_2	e_3	e_4
Methane	0.0251571	-0.074	0.138053	6.5963
Ethane	0.03448682	-0.074	0.18925109	6.5963
Propane	0.061733	-0.074	0.338767	6.5963

Table 6.2: Poly-tropic head calculation constants

Cycle	C_1	C_2	C_3	C_4	C_5
Methane	0.27146	11.765	45.040	1.79	2.11
Ethane	1.9928	63.00	32.855	1.79	2.11
Propane	1.9928	35.1962	18.3546	1.79	2.11

Table 6.3: Heat transfer coefficients for heat exchanger units

Heat exchanger	UA_{unit}
hex_M	0.27001
hex_E	0.45
hex_P	5.1
$hex_{E,M}$	0.8298083
$hex_{P,M}$	0.2947781
$hex_{P,E}$	2.6744397
$hex_{L,P}$	0.3299167
$hex_{L,E}$	0.8925934
$hex_{L,M}$	0.5384013

Table 6.4: Valve constants

	$c_{choke,M}$	$c_{choke,E}$	$c_{choke,P}$
Values	3.4	8.4	13

Table 6.5: Receiver sizes

	$m_{rec,M}$	$m_{rec,E}$	$m_{rec,P}$
Size	500	350	300

Table 6.6: Constants used for heat capacity calculations

	$c_{Cp,1}$	$c_{Cp,2}$	$c_{Cp,3}$
Methane	1.702	9.0819e-3	-2.164e-6
Ethane	1.131	19.225e-3	-5.561e-6
Propane	1.213	28.785e-3	-8.824e-6

Table 6.7: Constants used for saturation temperatures

	$c_{sat,0}$	$c_{sat,1}$	$c_{sat,2}$
Methane	1.4966	0.51530	0.13110
Ethane	2.4141	0.77135	0.21080
Propane	1.213	0.93582	0.24817

Table 6.8: Constants used for liquid saturation enthalpy

	Methane	Ethane	Propane
$c_{liq,0}$	-15187	-9447.4	-9992.7
$c_{liq,1}$	5974.2	12605	15991
$c_{liq,2}$	-5488.7	-12132	-11110
$c_{liq,3}$	3138.6	7061.9	4838.4
$c_{liq,4}$	-943.33	-2146.5	-970.54
$c_{liq,5}$	138.42	319.1	50.621
$c_{liq,6}$	-7.6457	-18.124	4.8879

Table 6.9: Constants used for vapour saturation enthalpy

	Methane	Ethane	Propane
$c_{vap,0}$	-6622.6	5906	9036.9
$c_{vap,1}$	2117.7	6200.8	11005
$c_{vap,2}$	-1650.9	-7884.9	-12980
$c_{vap,3}$	518.47	5412.9	9271.9
$c_{vap,4}$	-59.04	-1973	-3644.6
$c_{vap,5}$	0	356.06	721.52
$c_{vap,6}$	0	-25.103	-56.405

Table 6.10: Critical temperature and critical pressure for the refrigerants

	$T_{c,i}$ [K]	$P_{c,i}$ [bar]
Methane	190	46.055
Ethane	305	48.813
Propane	370	42.503

Table 6.11: Constants used for compressibility calculations

Parameter	Value
a_1	0.3265
a_2	-10.700
a_3	-0.5339
a_4	0.01569
a_5	-0.05165
a_6	0.5475
a_7	-0.7361
a_8	0.1844
a_9	0.1056
a_{10}	0.6134
a_{11}	0.7210

B-Code

The code used in this thesis can be found in its entirety at at : http://folk.ntnu.no/jaschke/. MPC simulations are run using Jmodelia.org 2.1
The first code snippet is the main Optimica code for the definition of the optimal control problem, and the python is the main code file handling the standard MPC.

Optimica

This is the definition of the Optimal control problem which is used by the standard MPC. This snippet of Optimica code is at the end of the main Modelica model file. The ”opt” object is the core of the Optimica extension and includes everything required for solving each step. The ”process” object is the model itself.

```
optimization opt(objectiveIntegrand=
        (((25*Z)+
        process.compM.Wcomp+process.compE.Wcomp+process.compP.Wcomp+
        0.001*(process.condM.mcond+process.condE.mcond+process.condP.mcond))),
        startTime=0,
        finalTime=750)

     //Model itself
      Process process();

     // Set inputs
     input Real u1(free = true) = process.compM.N_M;
     input Real u2(free = true) = process.compE.N_M;
     input Real u3(free = true) = process.compP.N_M;
     input Real u4(free = true) = process.condM.mcond;
     input Real u5(free = true) = process.condE.mcond;
     input Real u6(free = true) = process.condP.mcond;

     input Real Z(free = true,nominal = 0.0001);

     // Dynamic states
     parameter Real HrecM;
     parameter Real HrecE;
     parameter Real TrecP;

     parameter Real alphaM = 1;
     parameter Real alphaE = 1;
     parameter Real alphaP = 1;
     parameter Real Tamb = 2.78;
     parameter Real LNGtemp = 2.9305;
     parameter Real LNGflow = 3.074;

     parameter Real alphaM_set = 1;
     parameter Real alphaE_set = 1;
     parameter Real alphaP_set = 1;
     parameter Real Tamb_set = 2.78;
     parameter Real LNGtemp_set = 2.9305;
     parameter Real LNGflow_set = 3.074;
```

```
  initial equation
     process.receiverM.Hrec = HrecM;
     process.receiverE.Hrec = HrecE;
     process.receiverP.Trec = TrecP;

     process.Tamb = Tamb;
     process.LNGtemp = LNGtemp;
     process.LNGflow = LNGflow ;

     process.alphaM = alphaM;
     process.alphaE = alphaE;
     process.alphaP = alphaP;


  equation
     // Equations used for changing environmental variables
     der(process.alphaM) = 0.5*(alphaM_set-process.alphaM) ;
     der(process.alphaE) = 0.5*(alphaE_set-process.alphaE);
     der(process.alphaP) = alphaP_set-process.alphaP ;
     der(process.Tamb) = 0.05*(Tamb_set-process.Tamb);
     der(process.LNGtemp) = 0.03*(LNGtemp_set-process.LNGtemp) ;
     der(process.LNGflow) = LNGflow_set-process.LNGflow ;

  constraint

     process.LNGsink.inlet.T <= 1.23 + Z;
     Z >= 0;

     process.compM.N_M >= 0.8;
     process.compE.N_M >= 0.8;
     process.compP.N_M >= 0.8;

     process.compM.N_M <= 1.1;
     process.compE.N_M <= 1.1;
     process.compP.N_M <= 1.1;

     process.valveM.x >= 0;
     process.valveE.x >= 0;
     process.valveP.x >= 0;

     process.valveM.x <= 1;
     process.valveE.x <= 1;
     process.valveP.x <= 1;

     process.condM.mcond >= 0;
     process.condE.mcond >= 0;
     process.condP.mcond >= 0;

     process.condM.mcond <= 1;
     process.condE.mcond <= 1;
     process.condP.mcond <= 1;

     process.valveM.outlet.P >= 0.04;
     process.valveE.outlet.P >= 0.04;
     process.valveP.outlet.P >= 0.04;

end opt;
```

Python

This is the main python script that handles the standard MPC. The Modelica model and the Optimica code used are in their entirety found within the ”LNGtest.mop” file. The various ”LNG.XXX” files opened in the script are sections of the main file used for interfacing with different parts. For example ”LNG.easysim” is for interfacing with the simulation model, while ”LNG.opt” it the optimal control problem shown in the Optimica code section-.

```
import os.path
import csv
import numpy as N
import matplotlib.pyplot as plt

# Import the needed JModelica.org Python methods
from pymodelica import compile_fmu
from pyfmi import load_fmu
from pyjmi import transfer_optimization_problem, get_files_path
from pyjmi.optimization.casadi_collocation import BlockingFactors
from pyjmi.symbolic_elimination import BLTOptimizationProblem, EliminationOptions

def run_demo(with_plots=True):
    file_path = os.path.join(get_files_path(), "LNGtest.mop")
    init_fmu = compile_fmu("LNG.easy", file_path)

    #Simulation model
    init_model = load_fmu(init_fmu)

    #Inputs used for initial simulations
    #compM_A = 0.94
    #compE_A = 0.89
    #compP_A = 0.86
    #mcondM_A = 0.1
    #mcondE_A = 0.1
    #mcondP_A = 0.3

    #Input for stationary optimal point A
    compM_A = 0.9186759
    compE_A = 0.8813871
    compP_A = 0.8295928
    mcondM_A = 0.1219701
    mcondE_A = 0.1082355
    mcondP_A = 0.9999975
    init_model.set('u1',compM_A)
    init_model.set('u2',compE_A)
    init_model.set('u3',compP_A)
    init_model.set('u4',mcondM_A)
    init_model.set('u5',mcondE_A)
    init_model.set('u6',mcondP_A)


    # Solve the initialization problem A using FMI
    init_model.initialize()
    #Save dyanmic states at point A
    [HrecM_A,HrecE_A,TrecP_A] = init_model.get(['process.receiverM.Hrec',
```

```
                        'process.receiverE.Hrec','process.receiverP.Trec'])

    #simulation options
    opts = init_model.simulate_options()
    opts['initialize'] = False
    opts['CVode_options']['atol'] = 1.0e-9
    opts['CVode_options']['rtol'] = 1.0e-7

    #Other random input values
    compM_B = 0.93
    compE_B = 0.87
    compP_B = 0.82
    mcondM_B = 0.05
    mcondE_B = 0.03
    mcondP_B = 0.25
    #Set input B
    init_model.set('u1',compM_B)
    init_model.set('u2',compE_B)
    init_model.set('u3',compP_B)
    init_model.set('u4',mcondM_B)
    init_model.set('u5',mcondE_B)
    init_model.set('u6',mcondP_B)

    #Simulate with input B for initial trajectory
    init_res = init_model.simulate(start_time=0., final_time=750.,options = opts)

    #Optimization problem
    op = transfer_optimization_problem('LNG.opt', file_path)

    #Simulation model (model controlled)
    sim_fmu = compile_fmu("LNG.easysim", file_path)
    sim_model = load_fmu(sim_fmu)

    #Initial simulation conditions
    sim_model.set('HrecM',HrecM_A)
    sim_model.set('HrecE',HrecE_A)
    sim_model.set('TrecP',TrecP_A)
    #Simulation model options
    opts_S = init_model.simulate_options()
    opts_S['initialize'] = False
    opts_S['CVode_options']['atol'] = 1.0e-9
    opts_S['CVode_options']['rtol'] = 1.0e-7

    #Discretization options
    sample_time = 15
    horizon = 750
    n_e = horizon/sample_time
    final_t = 150
    num_input = final_t/sample_time

    #set Initial receiver states in OCP
    op.set('HrecM',float(HrecM_A))
    op.set('HrecE',float(HrecE_A))
    op.set('TrecP',float(TrecP_A))
    #options
    opts = op.optimize_options()
    opts['n_e'] = n_e # Number of elements
```

```
opts['init_traj'] = init_res # Initial trajectory
opts['nominal_traj'] = init_res
opts['IPOPT_options']['tol'] = 1e-7

#Blockingfactors
# Set blocking factors
factors = {'u1': opts['n_e'] / 1 * [1],
           'u2': opts['n_e'] / 1 * [1],
           'u3': opts['n_e'] / 1 * [1],
           'u4': opts['n_e'] / 1 * [1],
           'u5': opts['n_e'] / 1 * [1],
           'u6': opts['n_e'] / 1 * [1]}
#Quadric penalties can be used (not used here)
du_quad_pen = {'u1':10,
               'u2':10,
               'u3':10,
               'u4':10,
               'u5':10,
               'u6':10}

bf = BlockingFactors(factors)
opts['blocking_factors'] = bf #Use blocking factors

#Lists used for logging simulation model states
HM = []
HE = []
TP = []
compM = []
compE = []
compP = []
compW = []
waterM = []
waterE = []
waterP = []
valveM = []
valveE = []
valveP = []
tempL = []
Tamb = []
Te = []
Press = []
mm =[]
hexEL = []
hexPL = []
time = []
time_o = []
Tamb_o = []

#Control loop, each iteration is a control step
for n in range(num_input):
    if n == 0:
        #Initialize sim model and solver
        sim_model.initialize()
        solver = op.prepare_optimization(options=opts)
    #Environment variables can be changed
    if n == 5:
        #sim_model.set('LNGtemp_set',2.9805)
```

```
    sim_model.set('Tamb_set',2.83)
#if n == 8:
   # sim_model.set('Tamb_set',2.7875)
    #sim_model.set('Tamb_set',2.83)


#Solve optimization problem
#Set new time for solver
solver.collocator.t0 = n*sample_time
solver.collocator.tf = n*sample_time+horizon
#Solve problem
res = solver.optimize()

#Logging time and ambient temperature used by solver
Tamb_o.extend(res['process.Tamb'][:4]*100-273)
time_o.extend(res['time'][:4]+n*sample_time)


#Get opt input from solution in res
opt_input = res.get_opt_input()

#Simulate of model using optimal inputs just calculated
sim_res = sim_model.simulate(start_time= n*sample_time, final_time=(n+1)*sample_ti
                    input=(opt_input), options=opts_S)

#Log changes to dynamic states inputs for plotting
HM.extend(sim_res['process.receiverM.Hrec']*10)
HE.extend(sim_res['process.receiverE.Hrec']*10)
TP.extend(sim_res['process.receiverP.Trec']*100-273)

compM.extend(sim_res['process.compM.N_M'])
compE.extend(sim_res['process.compE.N_M'])
compP.extend(sim_res['process.compP.N_M'])
compW.extend(sim_res['process.compM.Wcomp']*100+sim_res['process.compE.Wcomp']*100

#Logging random variables
waterM.extend(sim_res['process.condM.mcond'])
waterE.extend(sim_res['process.condE.mcond'])
waterP.extend(sim_res['process.condP.mcond'])
valveM.extend(sim_res['process.valveM.x'])
valveE.extend(sim_res['process.valveE.x'])
valveP.extend(sim_res['process.valveP.x'])
hexEL.extend(sim_res['process.hexEL.outlet_LNG.T'])
hexPL.extend(sim_res['process.hexPL.outlet_LNG.T'])
tempL.extend(sim_res['process.LNGsink.inlet.T']*100-273)
Tamb.extend(sim_res['process.Tamb']*100-273)
Te.extend(sim_res['process.condE.outlet.T'])
Press.extend(sim_res['process.compM.outlet.P'])
mm.extend(sim_res['process.valveM.outlet.m'])
time.extend(sim_res['time'])

#Generate  random noise
num1 = N.random.normal(1,0.005)
num2 = N.random.normal(1,0.005)
num3 = N.random.normal(1,0.005)

#Extract dynamic staets at the end of the simulation
```

```
        #If noise is wanted, multiply with these variables
        HrecM = sim_res['process.receiverM.Hrec'][-1]
        HrecE = sim_res['process.receiverE.Hrec'][-1]
        TrecP = sim_res['process.receiverP.Trec'][-1]

        #Environmental variables are extracted in the same way
        Tambient = sim_res['process.Tamb'][-1]
        LNGtemp = sim_res['process.LNGtemp'][-1]
        LNGflow = sim_res['process.LNGflow'][-1]
        alphaM = sim_res['process.alphaM'][-1]
        alphaE = sim_res['process.alphaE'][-1]
        alphaP = sim_res['process.alphaP'][-1]

        #Set new initial conditions for solver to prepare for next step
        solver.set('HrecM',float(HrecM))
        solver.set('HrecE',float(HrecE))
        solver.set('TrecP',float(TrecP))

        #toggle these off for unknown change (Used for robust comparrison)
        solver.set('Tamb_set',float(Tambient))
        solver.set('Tamb',float(Tambient))

        solver.set('LNGtemp_set',float(LNGtemp))
        solver.set('LNGtemp',float(LNGtemp))
        solver.set('LNGflow',float(LNGflow))
        solver.set('alphaM',float(alphaM))
        solver.set('alphaE',float(alphaE))
        solver.set('alphaP',float(alphaP))

        #Set warm start
        solver.set_warm_start(True)
        set_warm_start_options(solver)

    #Plotting of random variables logged from simulation model

    #Code of plotting would be here


#Warm start options
def set_warm_start_options(solver, push=1e-4, mu_init=1e-1):
    solver.set_solver_option('IPOPT', 'warm_start_init_point', 'yes')
    solver.set_solver_option('IPOPT', 'mu_init', mu_init)

    solver.set_solver_option('IPOPT', 'warm_start_bound_push', push)
    solver.set_solver_option('IPOPT', 'warm_start_mult_bound_push', push)
    solver.set_solver_option('IPOPT', 'warm_start_bound_frac', push)
    solver.set_solver_option('IPOPT', 'warm_start_slack_bound_frac', push)
    solver.set_solver_option('IPOPT', 'warm_start_slack_bound_push', push)


if __name__=="__main__":
    run_demo()
```