UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

MODELLING, VALIDATION, AND CONTROL OF AN INDUSTRIAL FUEL GAS BLENDING SYSTEM

by

Cornelius Jacobus Muller

Submitted in partial fulfilment of the requirements for the degree

Master of Engineering (Electronic)

in the

Faculty of Engineering, Built Environment and Information Technology

Department of Electrical, Electronic and Computer Engineering

UNIVERSITY OF PRETORIA

April 2011

SUMMARY

MODELLING, VALIDATION, AND CONTROL OF AN INDUSTRIAL FUEL GAS BLENDING SYSTEM

by

Cornelius Jacobus Muller

Promoter: Prof. Ian K. Craig

Department: Electrical, Electronic and Computer Engineering

University: University of Pretoria

Degree: Master of Engineering (Electronic Engineering)

Keywords: dynamic model, model predictive control, state-space model, validation, real-time optimisation

In industrial fuel gas preparation, there are several compositional properties that must be controlled within specified limits. This allows client plants to use the fuel gas mixture safely without having to adjust and control the composition themselves. The variables to be controlled are the Higher Heating Value (HHV), Wobbe Index (WI), Flame Speed Index (FSI), and Pressure (P). These variables are controlled by adjusting the volumetric flow rates of several inlet gas streams of which some are makeup streams (always available) and some are wild streams that vary in composition and availability (by-products of plants). The inlet streams need to be adjusted in the correct ratios to keep all the controlled variables (CVs) within limits while minimising the cost of the gas blend. Furthermore, the controller needs to compensate for fluctuations in inlet stream compositions and total fuel gas demand (the total discharge from the header). This dissertation describes the modelling and model validation of an industrial fuel gas header as well as a simulation study of three different Model Predictive Control (MPC) strategies for controlling the system while minimising the overall operating cost.

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

OPSOMMING

MODELLERING, VALIDASIE EN BEHEER VAN 'N INDUSTRIËLE BRANDSTOFGASVERMENGINGSTELSEL

deur

Cornelius Jacobus Muller

Promotor: Prof. Ian K. Craig

Departement: Elektriese, Elektroniese en Rekenaaringenieurswese

Universiteit: Universiteit van Pretoria

Graad: Magister in Ingenieurswese (Elektroniese Ingenieurswese)

Sleutelwoorde: dinamiese model, model voorspellende beheer, toestand-ruimte model, validasie, regte-tyd optimering

By die voorbereiding van industriële brandstofgas is daar verskeie samestellingseienskappe wat binne bepaalde beperkings beheer moet word. Dié beheer verseker veilige gebruik van die brandstofgasmengsel deur kliëntaanlegte sonder om self die samestelling te beheer of aan te pas. Die veranderlikes wat beheer moet word, is die hoër hittewaarde, die Wobbe-indeks, die vlamspoedindeks, en die druk. Hierdie veranderlikes word beheer deur aanpassing van die volumetriese vloeitempo's van verskeie inlaatgasstrome, waarvan sommige aanvulstrome is (altyd beskikbaar) en ander wilde strome is wat wissel in beskikbaarheid en samestelling (byprodukte van aanlegte). Die inlaatstrome moet aangepas word om die regte verhoudings te lewer sodat al die beheerde veranderlikes binne die vasgestelde perke bly, terwyl die koste van die gasmengsel tot 'n minimum beperk word. Die beheerder moet terselfdetyd kompenseer vir fluksuasies in die inlaatstroom-samestelling en die totale gasaanvraag (die totale vloeitempo uit die vermengingsaanleg). Hierdie verhandeling beskryf die modellering en model-validering van 'n industriële brandstofgasaanleg, sowel as 'n simulasiestudie van drie model-voorspellende beheertoepassings vir die beheer van die stelsel wat die totale bedryfskoste van die aanleg tot 'n minimum beperk.

ACKNOWLEDGEMENTS

Thanks to Adolf Wolmarans from Sasol Infrachem for giving permission to use the plant data and to Paul Hughes from Sasol Technology for his advice and guidance (especially with regard to the iterative linearisation). Special thanks to Professor Larry Ricker from the University of Washington, Seattle for all his help and advice (especially regarding the derivation of the first principle model), and finally, thanks to Professor Ian Craig from the University of Pretoria for his supervision and valuable input for this work.

LIST OF ABBREVIATIONS

ARX	Auto Regression with eXogenous inputs
ARMA	Auto Regressive Moving Average
ARMAX	Auto Regressive Moving Average with eXogenous inputs
CV	Controlled Variable
DAE	Differential Algebraic Equation
DMC	Dynamic Matrix Control
DV	Disturbance Variable
ECR	Equal Concern for Relaxation
FF	Feed-Forward
FIC	Flow Indicator Controller
FIR	Finite Impulse Response
FSI	Flame Speed Index
GA	Genetic Algorithm
GNN	Grouped Neural Network
GUI	Graphical User Interface
HHV	Higher Heating Value
IAE	Integral Absolute Error
IIR	Infinite Impulse Response
LP	Linear Programming
LTI	Linear Time-Invariant
MIMO	Multiple Inputs, Multiple Outputs
MPC	Model Predictive Control
MV	Manipulated Variable
NARIMAX	Nonlinear Auto Regressive Integrated Moving Average with eXogenous inputs
NLP	NonLinear Programming
NMPC	Nonlinear MPC
ODE	Ordinary Differential Equation
PID	Proportional, Integral, Derivative
QP	Quadratic Programming
RGA	Relative Gain Array
RTO	Real-Time Optimisation

SA	Simulated Annealing
SG	Specific Gravity
SID	System Identification
SISO	Single Input, Single Output
SQP	Sequential Quadratic Programming
SVA	Singular Value Analysis
SVD	Singular Value Decomposition
TFM	Transfer Function Matrix
WI	Wobbe Index

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

1.1 PROBLEM STATEMENT

Fuel gas is an important utility in industrial processes. It is used in furnaces, catalytic crackers, and other combustion units. The end user's equipment is designed according to certain specifications with regard to the composition of the fuel gas. The specifications may include (as is the case for this study) the higher heating value (HHV or gross calorific value) [1], the Wobbe index (WI), and the flame speed index (FSI, using Weaver's flame speed factor) [2]. Apart from these compositional properties, the fuel gas must be supplied at a prescribed pressure (P). All of these requirements need to be addressed in the preparation of the gas in the fuel gas blending header.

Typically, the header consists of a vessel or piping network which is fed by several inlet streams and discharges from a single outlet point. In the system considered for this study, six feed streams enter the header. Of these six streams, four are makeup streams (always available but often costly) and two are wild streams (tail gasses which are by-products of plants and vary considerably in availability and composition). The feed gas streams need to be mixed in the correct quantities and ratios to ensure that the compositional properties are kept within specified limits while ensuring that the outlet pressure of the header does not fluctuate beyond predetermined bounds.

The header is susceptible to disturbances in the form of load fluctuations (changes in the demand for fuel gas downstream of the header), compositional changes in the feed streams, and availability changes in the tail gas streams. Controlling the header composition and pressure in the presence of these disturbances and noise using SISO (Single Input, Single Output) control is a challenging task, even for the most experienced operator. Ratio control can improve control somewhat but the fluctuations in the feed stream compositions mean that the required ratios change as well. Furthermore, the unit cost of the fuel gas blend must be minimised. Therefore, an automatic control system needs to be designed to take all

the disturbances (and noise) into account and simultaneously adjust the feed rates on the inlet streams to reject disturbances while at the same time driving the system to a state of minimum cost.

1.2 OBJECTIVES

The objectives of this research project are to address the control problem described in the previous section by completing the following tasks:

- Changes cannot be made on the actual plant for the purposes of this study. Therefore, a simulation study needs to be conducted to evaluate the proposed control solution. To legitimise the simulation study, an accurate model of the process needs to be derived. Therefore, the first task is to derive a process model for use in the process simulation which will capture the nonlinear dynamic behaviour of the process. This model will then be used as the "real plant".
- In order to verify that the plant model represents the system adequately, a validation needs to be carried out. Plant data are available and can be used to compare the response of the model with that of the real system. Therefore, the second step is to run a simulation with the feed values of the real plant and compare the response of the model to that of the plant (from recorded output data) to determine the correlation between the two.
- The system is a nonlinear, multivariable, interactive process which makes the use of model predictive control (MPC) an attractive option for control due to its ability to handle constrained, multivariable systems robustly and because of the ease at which the control problem can be formulated into the MPC framework. By designing the controller around a specific operating point, the effect of nonlinearities can be mitigated. MPC uses a model of the process to predict behaviour. If the model is non-linear or linear with constraints, the MPC algorithm requires the use of a non-linear optimiser which complicates matters. Furthermore, the software used for the design of the controller (namely the Matlab Model

Predictive Control Toolbox) only supports linear models. The MPC software packages most widely used in industry (such as Aspentech's DMCplus, Honeywell's RMPCT, Adersa's HIECON, etc.) also use linear algorithms. Therefore, a linear time-invariant (LTI) model needs to be derived from the non-linear simulation model in order to apply linear MPC.

- After the LTI model has been obtained, the MPC controller can be designed subject to constraints on the controlled variables (CVs) and manipulated variables (MVs). This includes the specification of all the tuning parameters such as control horizon, prediction horizon, weights, disturbance handling, constraints and constraint softening, and speed of response.
- The closed loop system must be simulated with some disturbances to evaluate the ability of the controller to keep the CVs within the prescribed ranges while avoiding violations of the MV limits. At the same time, the operating cost of the plant (the unit cost of the fuel gas blend) must be minimised.
- After the simulation is complete, the results need to be analysed to determine the effectiveness of the control system.
- If the controller's performance is far from the theoretical economic optimal (calculated using a nonlinear optimiser), iterative linearisation can be employed to compensate for process nonlinearities at different operating points.
- If the controller's performance is still not close to the economic optimal, real-time optimisation (RTO) can be included to set steady-state targets for the MVs and CVs to drive the process closer to the theoretical optimum.

1.3 CONTRIBUTION OF THIS RESEARCH PROJECT

On a practical industrial level, there is great potential in optimisation and cost saving in the preparation of plant utilities (which often go unnoticed). Steam and electricity minimisation are becoming popular areas of optimisation and enjoy considerable attention. In the same way, optimising on fuel gas preparation can reduce running costs and the use

of expensive makeup streams (such as natural gas) by maximising on the use of process tail gas streams which are often free and would otherwise be flared. This has an obvious environmental benefit as well.

Although there are many papers on MPC, modelling, model validation, and optimisation, there is an absence of literature covering the specific application of these concepts on a fuel gas blending system. Therefore, on an academic level, there is a possibility that this project can make a significant contribution to this field by introducing a novel application of these well studied topics. Furthermore, the iterative linearisation discussed in Section 5.3 and the RTO discussed in Section 5.4 allow for the application of types of nonlinear MPC using linear algorithms. The work described in this document has been accepted for publication in the Journal of Process Control [3] and for presentation at the 18th IFAC world congress in Milan, Italy [4].

1.4 ORGANISATION OF DISSERTATION

The order of this dissertation follows that of the tasks described in Section 1.2. The first part is a discussion of the modelling of the fuel gas blending system which is essential for establishing a solid foundation for the work that follows. The second part contains the validation results, comparing the response of the simulation model to that of the actual plant to identical inputs. The third and fourth parts contain details on the derivation of an LTI model (using system identification or SID) and the design of the MPC controller (which includes three possible control schemes, increasing in complexity and effectiveness). Finally, a simulation of the controlled system (closed-loop simulation) is presented and its results analysed and explained. No dedicated chapter is included for literature review. Instead, each chapter contains several inserts pertaining to the literature for that specific topic.

CHAPTER 2 PROCESS OVERVIEW

2.1 BACKGROUND

Although plant utilities like instrument air, steam, fuel gas, etc. do not receive as much attention as more complex core processes such as reaction and distillation, these processes are strongly dependent on them. As mentioned in Section 1.3, there seems to be a shortage (or even an absence) of literature on the topic of the control of gas blending, in particular fuel gas blending.

A similar application can be found in liquid fuel blending where additives are dosed into the fuel in precise quantities (for which more literature is available). In a recent publication, Chèbre et al. [5] described a control algorithm for blending liquid fuels to produce mixtures with some prescribed properties while minimising the production cost. Although the properties and behaviour of liquid blending differ from the gas blending of this application, there are some similarities and concepts that can be applied to this study. In some cases, real-time composition measurements are not available (mostly due to cost and reliability issues regarding analysers) and observers are developed to estimate the properties of the feed streams. If lab samples are taken, the observer parameters can be updated periodically. For downstream measurements (on the blender outlet), it is recommended to have online measurement using more than one analyser. This is also the case for this study (mentioned in Section 3.3.1).

There are mainly two differences between the gas blending and liquid blending processes. In the first place, the blended gases behave as nearly ideal mixtures and their combustion properties are known functions of composition. Furthermore, periodic measurements of the feed gas compositions are available. Thus, prediction of the impact of MV adjustments on blend properties is easier than in liquid blending. Secondly, for gas blending, the dynamic behaviour is generally faster than that of liquid blending (a residence time of between 2 to

4 minutes in this case, whereas that of liquid blending systems is typically in the order of hours) and therefore requires more frequent control adjustments and quicker sampling.

2.2 FUEL GAS PROCESS DESCRIPTION

Figure 2.1 shows a process diagram of the system. Although the header is depicted as a vessel, it is made up of the volume of the piping network (estimated at 100 m^3 by using the gradient of the pressure increase for a 1 kNm^3/h increase in feed). The flow rates are high so it is assumed that turbulent flows facilitate perfect mixing so that the composition of the exit stream equals the header composition (which is assumed to be uniform across the header). Six gas streams enter the fuel gas header (shown with their fictional tag names in Figure 2.1). These six feed streams are Natural Gas (NG), Reformed Gas (RG, hydrogen to CO ratio of between 1.8:1 and 2:1), Hydrogen (H_2), Nitrogen (N_2), Tail Gas 1 (TG_1), and Tail Gas 2 (TG_2).

The first four streams are make-up streams whereas the two tail gas streams are wild streams, varying in availability and composition. The tail gasses are produced as by-products of plants and can be utilised for heating purposes across the complex. Their compositional properties are, however, unsuitable for the heating requirements of industrial fuel gas. Therefore, these streams need to be mixed with make-up streams in correct ratios and quantities to adjust the output composition to be more suitable for use as a fuel source. In addition, the fuel gas mixture must be supplied at a specific pressure. Table 2.1 shows the ranges for the outputs with their units of measure. These specifications determine the combustion properties of the gas mixture. The HHV and WI give indications of the energy content and density of the gas. The FSI determines how the flame physically behaves when combusting. If the flame speed goes too high, there is a danger of the flame burning back into the burner nozzle. If it is too low on the other hand, the probability of the flame being blown out will increase. The Weaver FSI (or Weaver's flame speed factor) is an indication of the flame speed of a gas with reference to that of pure hydrogen (having a flame speed of 100) [2].

The NG, RG, and N_2 streams have costs associated with them whereas the H_2 and tail gas streams would otherwise be flared and are therefore considered free. Therefore, the use of the NG, RG, and N_2 streams must be minimised in the optimisation problem whereas the use of the tail gas streams and H_2 should be maximised subject to its availability. Natural gas is used continuously to increase the calorific value up to specification due to the typically low heating value of the tail gasses. Nitrogen will only be used when the FSI is too high (which will be the case if the tail gasses are rich in hydrogen). Reformed gas is used as a substitute for the tail gas streams when not available. Hydrogen can be used continuously, but cannot be used if the flame speed is high. Apart from these streams, several disturbances act on the system, including fluctuations in the feed stream compositions and total fuel gas demand (i.e. the discharge flow rate from the header).

Table 2.2 gives the typical compositions and characteristics of the inlet streams. The HHV, WI, and FSI are functions of the molar composition of the fuel gas. Although the inlet streams are depicted as flow controllers (the FIC referring to a Flow Indicator Controller), the controller and actuator dynamics are omitted in this study and perfect manipulation of the flow rates is assumed.

Table 2.1: Controlled variable ranges.

CV	Abbreviation	Range	Units
Higher Heating Value	HHV	16.5 – 18	MJ/Nm^3
Wobbe Index	WI	25 – 27	MJ/Nm^3
Flame Speed Index	FSI	39 – 46	-
Pressure	P	2000 – 2200	kPa

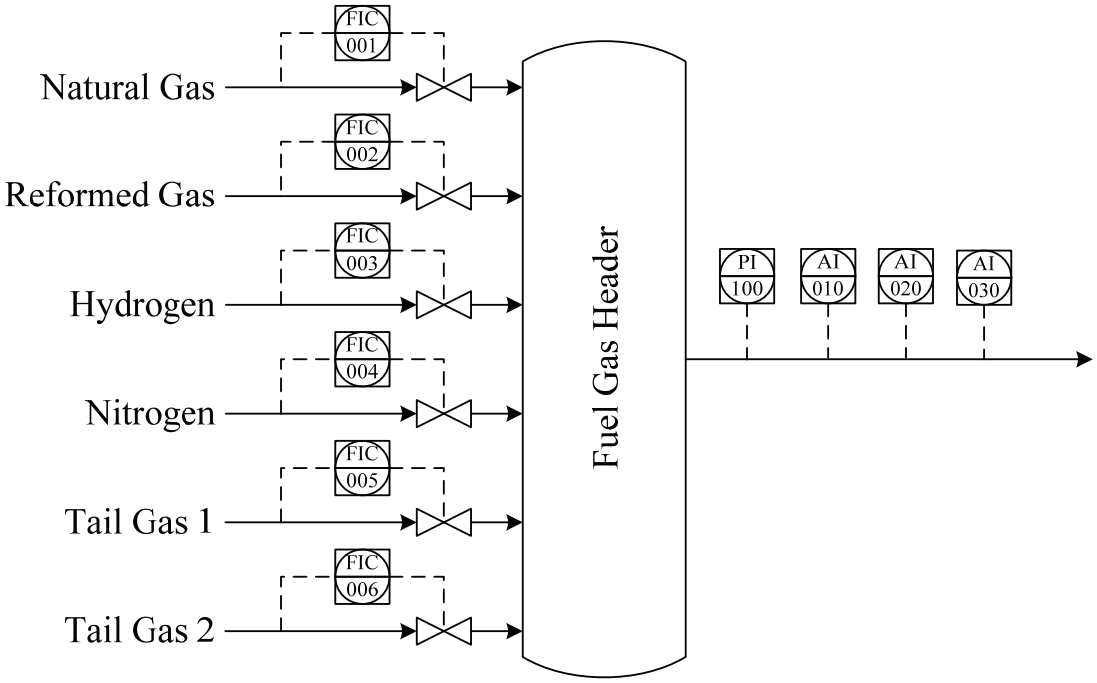

Figure 2.1. Process diagram of blending header.

Table 2.2: Typical inlet compositions (mol %).

	NG	RG	H_2	N_2	TG_1	TG_2
CH_4	91.1	1.5	-	-	5.5	15.0
C2+	6.8	0.0	-	-	1.0	1.0
H_2	0.0	62.0	100.0	-	62.0	57.0
N_2	1.5	0.5	-	100.0	2.5	6.0
CO	0.0	31.0	-	-	26.0	13.0
CO_2	0.6	5.0	-	-	3.0	8.0
HHV	43.02	11.78	12.10	0.0	13.96	15.39
WI	52.62	17.87	45.73	0.0	21.60	22.92

The ranges of the inlet streams are restricted by the typical availability of the streams. Natural gas and reformed gas are available in large quantities whereas the hydrogen and nitrogen streams are more limited. The flow rates on the tail gas streams can also become quite large when they are available. Therefore, in normal operation, it is ideal to use as many of these streams as possible. When one or both of these streams are not available (such as when the upstream plants producing them trip), reformed gas is used to replace them (having a comparable heating value). Table 2.3 shows the ranges used for the inlet streams for this study. These ranges comprise physical constraints for the control problem and cannot be exceeded in any way.

Table 2.3 MV limits and units.

MV	Low limit	High limit	Units
NG	0	15	kNm^3/h
RG	0	20	kNm^3/h
H_2	0	5	kNm^3/h
N_2	0	5	kNm^3/h
TG_1	0	30	kNm^3/h
TG_2	0	30	kNm^3/h

2.3 CONCLUSION

In this section, the fuel gas blending process was described on a practical level, introducing the feed streams, the controlled process variables, predominant components, typical ranges, and limits. Figure 2.1 illustrates that six feed streams of variable composition and availability enter the header and the gas mixture discharges from a single point at which the composition and pressure are measured. In the next chapter, mathematical models are developed for the fuel gas blending system for simulation and control purposes.

CHAPTER 3 DYNAMIC MODELLING

3.1 INTRODUCTION

Mainly two approaches are followed when developing process models. The first is the derivation of a model from the scientific principles of physics, chemistry, and biology (called theoretical, first principle, or physical models) [6,7]. These models can become very complex for some processes (and can be very time-consuming and expensive to derive), especially if the model requires a large number of equations with a considerable number of variables and unknown parameters. On the other hand, first principle models give valuable insight into the behaviour of the process and are usually applicable over a wide operating range [6]. The second method makes use of experimental data to determine the relationship between the inputs and outputs of a process and is known as system identification (SID). The models resulting from SID are called empirical models (also referred to as black-box models) [6,7,8,9]. These models are typically easier to develop than first principle models (and are usually more cost effective). Some disadvantages of empirical models are that they are normally only valid around the operating point at which they were developed and that aspects of the process that does not feature in the data used to derive the model can easily be overlooked [6]. A third, less frequently used modelling type is semi-empirical modelling (also called grey-box models) where one or more parameters of a theoretical model is determined by fitting experimental data [6,8,9].

Both first principle and empirical models can be developed as steady-state or dynamic models. Most linear models used in the process industry are empirical models [6,10,11]. Furthermore, with the development of industrial MPC technology, various vendors also developed proprietary process identification technologies that integrate seamlessly with their MPC development software. These software applications expedite and reduce the cost for the development of dynamic empirical models from test data [11].

In this chapter, a brief overview of some of the most popular model forms, types, and concepts is provided. These are not limited to either empirical or first principle models although some will be more applicable to one than the other. In most cases, one model form can readily be converted to another (for example from state-space to transfer function and vice versa). Although the discussion is focussed on continuous time models, the models can also be represented in discrete time format. Section 3.3 continues to describe the development of a first principle model of the fuel gas blending system and Section 3.4 the development of a linear empirical model for use in the control algorithm.

3.2 MODEL TYPES

3.2.1 Linear versus nonlinear

Linear time invariant (LTI) models are usually sufficient for describing process behaviour around a specific operating point (such as a plant running mostly at nominal operating conditions). The main advantages of using a linear model are its simplicity (especially when considering constrained cases), and the wealth of knowledge regarding linear system analysis. There are some cases, however, where the process is highly non-linear or operate over vastly different operating regions where it might be beneficial to use a nonlinear model. This can allow improved control by improving the accuracy of the behavioural predictions used in model-based control [10]. Linear models usually refer to LTI models which mean that the model does not change with time and conforms to the principle of superposition. LTI models can be derived from non-linear models by linearisation around an operating point. Most industrial MPC applications are linear (Qin et al. considered a total of 4542 linear applications versus a total of 93 nonlinear applications in their survey performed from mid-1999 to early 2000's [11]).

3.2.2 Steady-state versus dynamic

Steady-state (or static) models are concerned with the long term behaviour of the process (where it settles at equilibrium). In these models, the relationships between inputs and

outputs are direct and instantaneous. The output values depend only on current inputs, not on past process behaviour [7]. In the case of this study, a steady-state model is used for real-time optimisation (RTO) as discussed in Section 5.4.

Dynamic models (unsteady-state or transient models) on the other hand describe how the process reacts after input changes or disturbances. Dynamic behaviour occurs at conditions such as plant start-up, shut-down, process disturbances, and transition from one operating point to another [6]. The instantaneous output values of the system are dependent on the current input values and disturbances as well as past system behaviour [8,7]. For this study, dynamic models are used for simulation of the process and for control predictions.

3.2.3 Regression

Regression (also called parameter estimation) is the process of selecting a model form and then determining the unknown model parameters from input-output data by minimising a measure of difference between the model output and the actual output data [6]. The model form may be selected based on some process knowledge or experience. Plotting the input and output data and identifying overall trends in behaviour can help to choose a realistic model form. It is desirable to select the simplest model structure that provides a good fit. A popular model form is ARX (Auto Regression with eXogenous input) and is used by various developers of MPC technology including Honeywell, Adersa, and Invensys [11].

3.2.4 State-space

State-space models describe system behaviour in the form of a set of ordinary differential equations (ODEs) and are not limited to either linear or nonlinear systems [6]. There are several advantages to using state-space models including the intuitive transition from SISO to MIMO (Multiple Inputs, Multiple Outputs) models, ease of analysis of closed-loop properties, computational advantages, and the ability to use the abundance of linear systems theory with this form [10]. The basic form of a linear state-space model is [6,7]

$$\frac{dx(t)}{dt} = A.x(t) + B.u(t) + E.d(t) \tag{3.1}$$

$$y(t) = C.x(t) + D.u(t) \tag{3.2}$$

where x denotes the state vector (and determines the order of the system), y represents the output vector, u is the input vector, and d is the disturbance vector. The outputs have to be observable, i.e. must be measurable or inferable. Typically, the states are chosen to be the output variables (if measurable directly). In such a case, the system is completely portrayed by differential equations describing its dynamic behaviour (if there is no direct feed-forward action from the inputs to the outputs). For an LTI system, the matrices A, B, C, D, and E are constant and describe the system in full [6]. The stability of the system described in Equations (3.1) and (3.2) is solely determined by the A matrix of which the eigenvalues must have negative real parts (corresponding to the roots of the characteristic equation, $|\lambda I - A| = 0$) [6]. The first principle model discussed in Section 3.3 is in the state-space form. Linear state-space models can easily be converted to transfer function models described in the next section.

3.2.5 Transfer function

A transfer function model describes the dynamic relation between a specific input and output in an algebraic expression and is represented in the *s*-domain (or the Laplace domain). When representing systems graphically, these models are very intuitive. Transfer functions are, however, limited to linear systems because the Laplace transform can only be applied to linear equations [6]. Therefore, nonlinear systems need to be linearised in order to be described by transfer functions. The basic form of a transfer function model is

$$G(s) = \frac{Y(s)}{U(s)} \tag{3.3}$$

where $Y(s)$ and $U(s)$ are the Laplace transforms of the output $y(t)$ and input $u(t)$ signals and $G(s)$ is the Laplace transform of $g(t)$, the impulse response from $u(t)$ to

$y(t)$. The Laplace transform for $g(t)$ is calculated as

$$G(s) = \mathcal{L}\{g(t)\} = \int_0^\infty g(t)e^{-st}dt \tag{3.4}$$

where s is a complex independent variable and $\mathcal{L}$ is the Laplace operator. The Laplace transfer of a unit impulse is $\mathcal{L}\{\delta(t)\} = 1$. Therefore, defining the transfer function to be the Laplace transform of the impulse response of the relation between input and output makes it independent of the particular choice of forcing function. Therefore, any response of $Y(s)$, can be calculated by simply substituting the transfer function of the forcing function into $U(s)$ in Equation (3.3). Calculating the steady-state gain of a system described by transfer functions can easily be done by letting $s \to 0$ in $G(s)$ according to the final value theorem (if the gain exists, i.e. if the system is stable) [6]. The Honeywell RMPCT Identifier package is an example of a commercially available SID tool that presents the models in transfer function form [11].

3.2.6 Step response

Step response models describe the system by recording what the output does in response to a unit step change in the input. For discrete calculation, a finite number of coefficients are used to capture the system dynamics. The amount of coefficients required depends on the settling time of the process and the desired accuracy of the model. These models are therefore usually described with tables of coefficients rather than mathematical models. This allows unusual dynamics to be captured without having to use high order parametric models. This is the model form used in Aspentech's DMCplus control software [11].

3.2.7 Impulse response

Impulse response models describe the system by recording what the output does in response to a unit impulse injected at the input. Similar to step response models, these models are often described by tables of coefficients. Adersa uses an FIR (Finite Impulse Response) model in the HIECON package [11].

3.3 FIRST PRINCIPLE MODEL OF FUEL GAS SYSTEM

To obtain a realistic simulation of the fuel gas blending process, an accurate model is required. The fuel gas system is nonlinear and should therefore be represented with a nonlinear model if the nonlinear dynamics are to be captured satisfactorily. This section describes the derivation of a first principle, nonlinear, state-space model based on a molar balance of the components in the header. The HHV, WI, and FSI are functions of the molar composition of the fuel gas. There are six states (the numbers of moles of the six components in the header), six inputs (the volumetric flow rates of the six inlet streams), and four outputs (HHV, FSI, WI, and pressure). The state equations are given by

$$\dot{N}_{fg,i} = u_i - y_{fg,i}.u_T \tag{3.5}$$

where $i = 1$ to 6, $N_{fg,i}$ is the number of moles of component i in the header, u_i is the total molar flow of component i entering the header (summed over all inlet steams), u_T is the total molar discharge rate from the header, and $y_{fg,i}$ is the molar fraction of component i in the header.

The inlet flows are described in terms of volumetric flow rates and compositions. Therefore, these flows need to be converted to molar flow rates of the individual components to get to u_i. This is done by converting the volumetric flow rates to molar flow rates. The volumetric flow rates are measured in kNm^3/h (i.e. under an ideal gas assumption). Therefore, the individual component molar flow rates u_i are

$$u_i = 44.64 \sum_{j=1}^{6} y_{F_j,i}.F_j \tag{3.6}$$

for $i = 1$ to 6 and where F_j is the volumetric flow rate of the j^{th} inlet stream (kNm^3/h) and $y_{F_j,i}$ is the molar fraction of component i in inlet stream j. The j index refers to the sequence shown in Figure 2.1, i.e. $j = 1$ refers to the NG stream and $j = 6$ refers to the TG_2 stream. The factor $1000 / 22.4 = 44.64$ is the amount of litres per cubic meter divided

by the volume (in litres) filled by one mole of gas under ideal conditions and is used to convert the volumetric flow rate to molar flow rate under the ideal gas assumption.

The outputs are calculated according to the molar fractions of the components in the system (and the total number of moles in the case of pressure) [1,2,12,13]. The output calculations are

$$HHV_{fg} = \sum_{i=1}^{6} HHV_i . y_{fg,i} \tag{3.7}$$

$$WI_{fg} = \frac{HHV_{fg}}{\sqrt{\rho_{fg}}} \tag{3.8}$$

$$FSI_{fg} = \frac{\sum_{i=1}^{6} y_{fg,i} . s_i}{\sum_{i=1}^{6} y_{fg,i} . A_i + 5\sum_{j=1}^{2} n_{fg,j} - 18.8 x_{O_2} + 1} \tag{3.9}$$

$$P = \frac{N_T RTZ}{V} \tag{3.10}$$

where s_i is the flame speed factor for component i, A_i is the molar stoichiometric air demand factor (for total combustion) for component i, $n_{fg,j}$ is the molar fraction of inert component j in the fuel gas, and x_{O_2} is the mole fraction of oxygen in the gas (usually zero in this application) [2]. N_T is the total number of moles in the system, $R = 8.314$ is the gas constant, T is the header temperature (Kelvin), V is the header volume (m^3, estimated at $100\,\mathrm{m}^3$), and Z is a real gas correction factor (to compensate for the difference between the fuel gas behaviour and that of an ideal gas; $Z = 1.006$ in this case indicating that the ideal gas assumption is reasonable) [1]. The Fuel Gas specific gravity, ρ_{fg}, is calculated as

$$\rho_{fg} = \frac{\sum_{i=1}^{6} MWt_i . y_{fg,i}}{MWt_{air}} \tag{3.11}$$

where MWt_i is the molar weight of component i and $MWt_{air} = 28.8$ is the standard molar weight of air. Table 3.1 lists some characteristics of the components [1].

Table 3.1: Component characteristics.

	HHV	WI	SG	MWt	A	S
CH_4	37.78	50.72	0.557	16.04	9.55	148
C2-C6	126.5	87.62	2.018	58.12	31.0	514
H_2	12.10	45.88	0.069	2.016	2.39	339
N_2	-	-	0.973	28.02	-	-
CO	11.97	12.17	0.968	28.01	2.39	61
CO_2	-	-	1.528	44.01	-	-

3.3.1 Model validation

The integrity of the process model needs to be determined in order to support the validity of the simulation study. Comparison of calculated data from the model versus data from the real plant outputs using the same input data (flows, compositions, etc.) is the most intuitive way of determining the integrity of the process model [14].

For the validation, a period of operation was identified in which all the flow measurements are reliable (either zero or greater than the turn-down of the transmitters). This ensures that the data used for the validation is a good representation of the flows entering and exiting the header. The inlet flow rates, feed stream compositions, and header discharge rate were used as verification data and the simulation output data compared to the plant measurements (the system is at ambient temperature for which the effects of daily fluctuations on the outputs are negligible). The initial model states were determined by running a simulation using average feed flow rates and compositions as inputs and recording the steady state molar values.

The dead-times on the analysers (for measuring HHV, WI, and FSI) were initially estimated at 2 minutes and adjusted for better data correlation. The final dead-times were 20 seconds for HHV, 1 minute for WI, and 20 seconds for FSI (the FSI is measured by a mass spectrometer which has a small dead-time but only provides a sample every 10 minutes).

As shown in Figure 3.1 to Figure 3.3, the model's open-loop predictions of HHV, WI, and FSI track the observed trends rather well. Computed correlation coefficients for a validation period of 18 hours are shown in Table 3.2. Some factors contributing to the discrepancies between the plant data and the model include infrequent feed stream and fuel gas composition measurements, errors in feed flow measurements (especially when close to the turn-downs of the flow transmitters), and interpolation adjustments made when the plant's data historian recorded the data.

The presence of feedback control on the header pressure for all plant data complicates the validation of the pressure model. Feedback can introduce non-causal effects from input to output (for example an operator who anticipates an event and compensates for it before it actually occurs) which complicates the validation [15]. The model is, however, based on well developed physical models and will be assumed to be adequate for the purposes of this simulation study.

Table 3.2: Correlation coefficients for output data.

Data set	Correlation coefficient (%)
HHV	93.8
WI	84.8
FSI	83.1

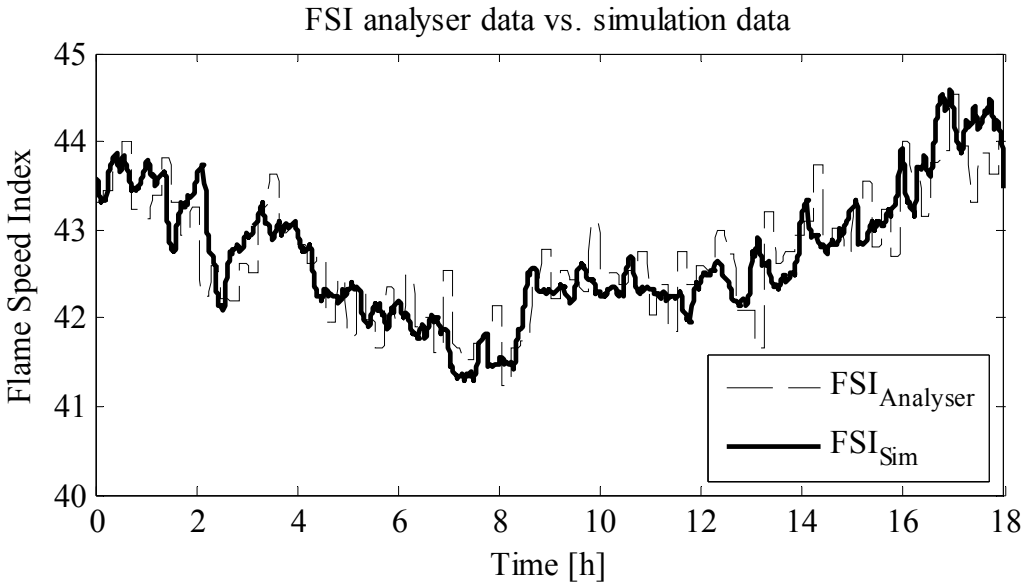

Figure 3.1: FSI analyser data versus simulation data.

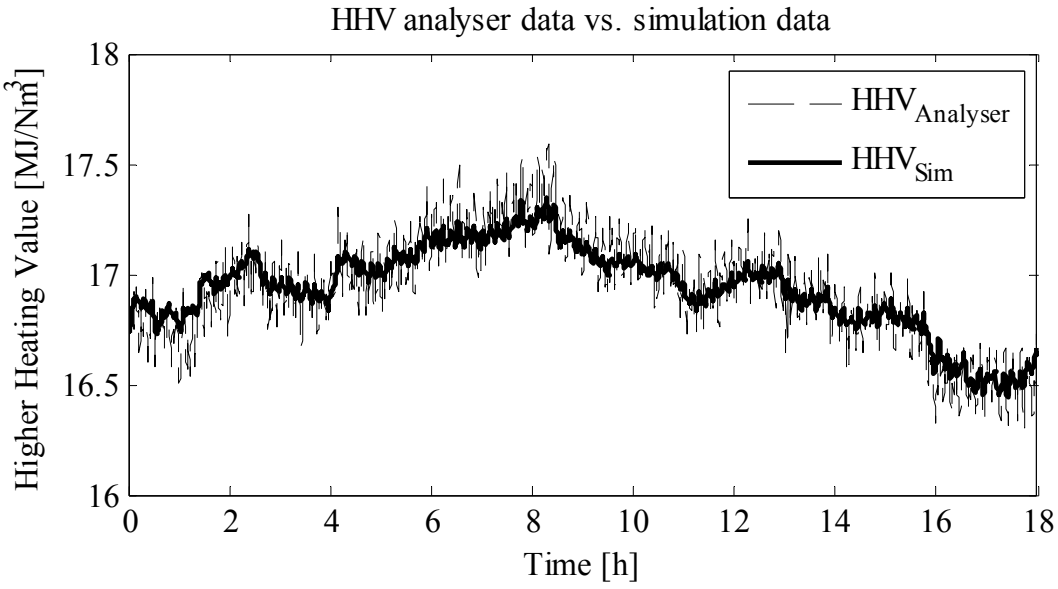

Figure 3.2: HHV analyser data versus simulation data.

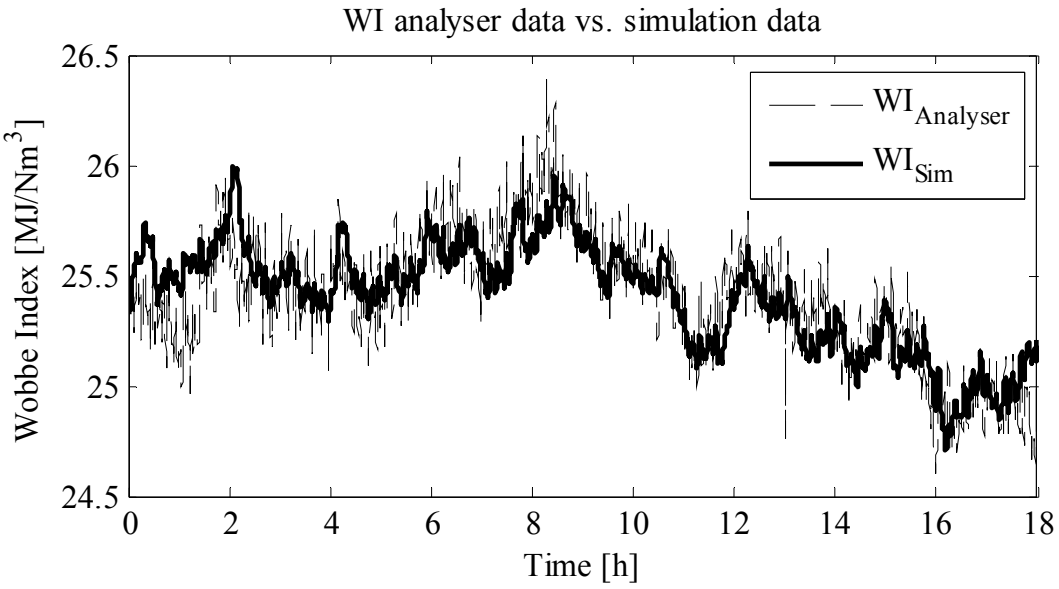

Figure 3.3: WI analyser data versus simulation data.

For the HHV and WI there are in actual fact two analysers each, measuring at the same points in the process (for redundancy reasons). The values are also recorded separately. Therefore, although the plots shown in Figure 3.2 and Figure 3.3 show the comparisons of the simulation data to one plant data set each for the HHV and WI, these are the average values for the two analysers in each case.

3.4 EMPIRICAL MODEL OF FUEL GAS SYSTEM

Standard linear MPC was initially used for control of the plant due to its simplicity and availability of linear MPC development software (such as the Matlab MPC Toolbox used for this case study). In order to apply linear MPC, a linear dynamic model needs to be derived. To derive a linear model for the process, the nonlinear first principle model can be used and linearised around an operating point using theoretical calculations. Alternatively, an empirical model can be derived from input-output data generated by the actual process or (as in this case) from a simulation of the process using the first principle model.

3.4.1 Steps in system identification

Several authors describe the sequence of activities to perform SID. The following is a summary of the most important steps for deriving empirical dynamic models [6,7,8,9,15].

1. *Specify intended purpose/objective of the model*.

   This will clarify what accuracy, computational time, etc. will be acceptable. For example will the model be used in a training simulator or will it be used in a robust controller.

2. *Select model variables*.

   The selection of variables will determine the size of the model. Ideally, the model should be kept concise (depending on the intended use) and should therefore only include variables that have significance. For example, selecting an input variable that has a negligible effect on the selected output variables will only add complexity to the model without adding value.

3. *Measure input and output signals*.

   In this step the actual plant data is collected and evaluated. This will include determining the quality of the data and whether the data captures the dynamic behaviour of the process adequately. Ideally, step testing should be performed on the process to ensure that the captured data contains the necessary dynamic information (with input signals designed to provide maximum information, i.e. to adequately excite the process).

4. *Select model structure and complexity*.

   This step will include an initial model order, type of model (state-space, transfer function, FIR, etc.), and whether the model will be linear of non-linear.

5. *Apply estimation method to estimate values for parameters in candidate model structure*.

   This step will typically involve solving an optimisation problem that will attempt to minimise some measure of error between the behaviour of the candidate model and

the plant data by changing the parameter values. Providing realistic initial values for the parameters can expedite the process.

6. *Evaluate the estimated model.*

   There are several means of determining the quality/validity of the model. As mentioned in Section 3.3.1, the simplest and most intuitive way is to evaluate the correlation between the response of the model and the response of the plant to the same input values. If possible, a different data set should be used for the validation than for the identification. For dynamic models, graphical evaluation can also be used to compare the speed and shape of the responses to identical input changes (typically a step input).

3.4.2 Final LTI model

Several types of LTI (Linear Time-Invariant) models were considered and fitted. Finally, first-order-plus-dead-time transfer function models were selected and fitted for HHV, WI, and FSI and with integrating models for pressure (to prevent numerical problems in Matlab, the pressure models are not pure integrators but have poles close to the origin, not at the origin). Therefore, the model fitted is of the form

$$G(s) = \frac{\beta}{s+\alpha} e^{-s\tau_d} \tag{3.12}$$

where α and β are the parameters to be estimated and τ_d is the time delay (in hours), equal to $1/180$ for the models to HHV and FSI, $1/60$ for WI, and 0 for pressure. An equivalent of Equation (3.12) in difference equation format is

$$y(t) + \alpha . y(t-T) = \beta . u(t-nT) \tag{3.13}$$

where T is the sampling period (20 seconds or $1/180$ hours in this study) and n is the delay in number of sampling periods. The linearisation was performed around an operating point of [HHV, WI, FSI, P] = [16.75, 25.32, 43.47, 2085] which is a typical operating region for the plant. The resulting model matrix is shown in Table 3.3. The time unit for the model is hours.

Table 3.3: Linearised model matrix.

	NG	RG	H_2
HHV	$\frac{24.61}{s+28.62}e^{-s/180}$	$\frac{-4.42}{s+23.12}e^{-s/180}$	$\frac{-5.04}{s+26.44}e^{-s/180}$
WI	$\frac{30.73}{s+28.69}e^{-s/60}$	$\frac{-6.23}{s+23.39}e^{-s/60}$	$\frac{2.25}{s+26.29}e^{-s/60}$
FSI	$\frac{-59.21}{s+28.57}e^{-s/180}$	$\frac{11.05}{s+22.92}e^{-s/180}$	$\frac{32.26}{s+27.14}e^{-s/180}$
P	$\frac{1120}{s}$	$\frac{1120}{s}$	$\frac{1120}{s}$

	N_2	TG_1	TG_2
HHV	$\frac{-13.27}{s+23.09}e^{-s/180}$	$\frac{-3.31}{s+25.78}e^{-s/180}$	$\frac{-1.66}{s+22.64}e^{-s/180}$
WI	$\frac{-30.42}{s+23.00}e^{-s/60}$	$\frac{-4.04}{s+25.62}e^{-s/60}$	$\frac{-2.45}{s+22.85}e^{-s/60}$
FSI	$\frac{-33.59}{s+23.00}e^{-s/180}$	$\frac{10.40}{s+26.72}e^{-s/180}$	$\frac{2.23}{s+22.17}e^{-s/180}$
P	$\frac{1120}{s}$	$\frac{1120}{s}$	$\frac{1120}{s}$

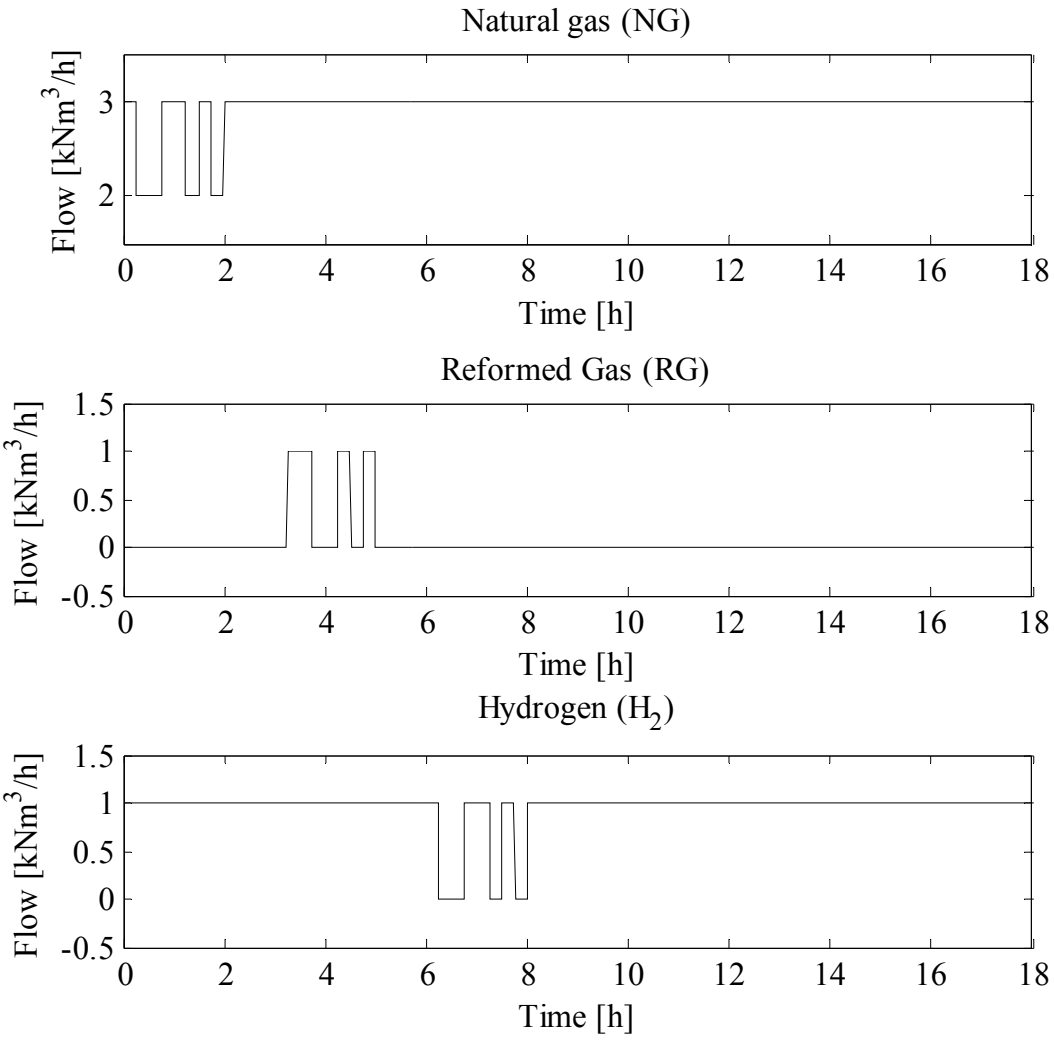

Figure 3.4. SID step changes in the NG, RG, and H_2 streams.

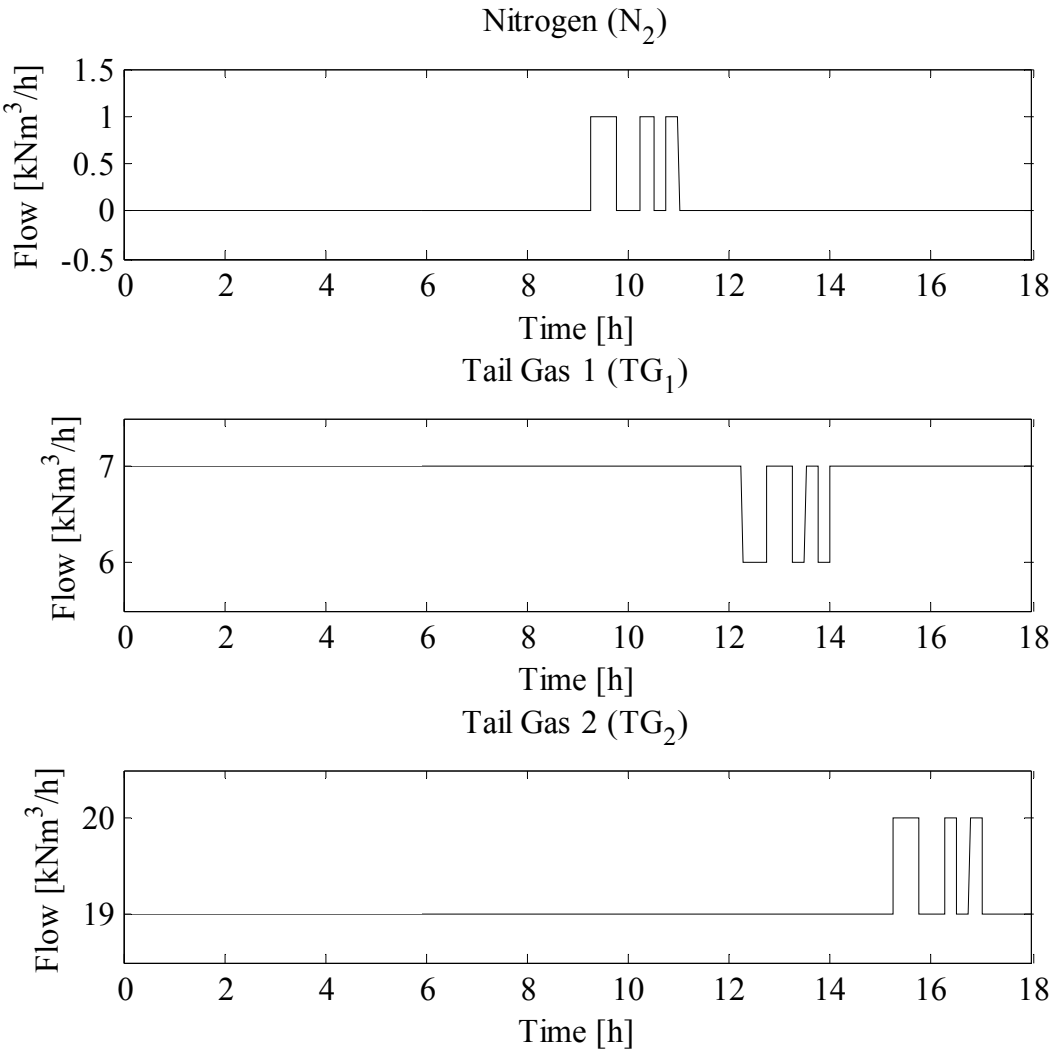

Figure 3.5. SID step changes in the N_2, TG_1, and TG_2 streams.

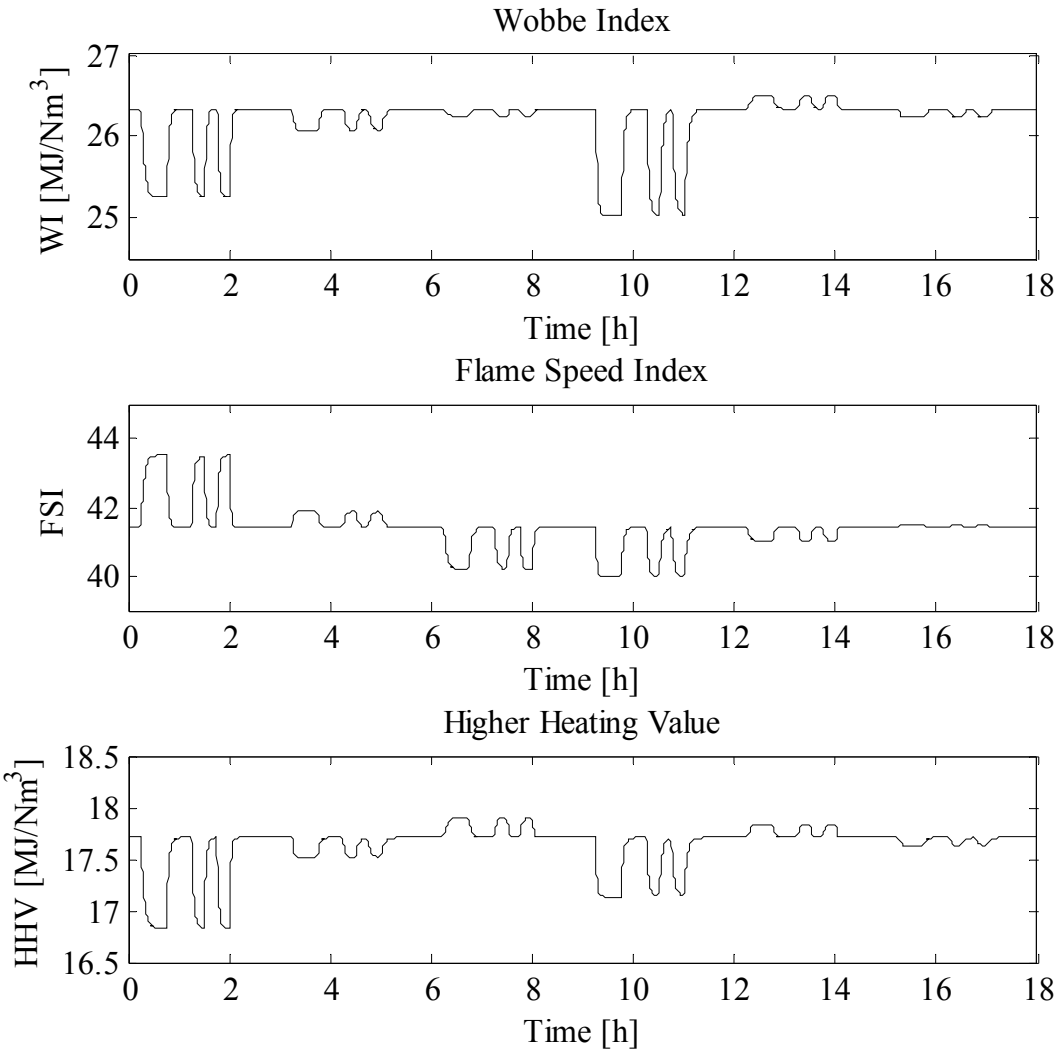

Figure 3.6. Step test results for the WI, FSI, and HHV.

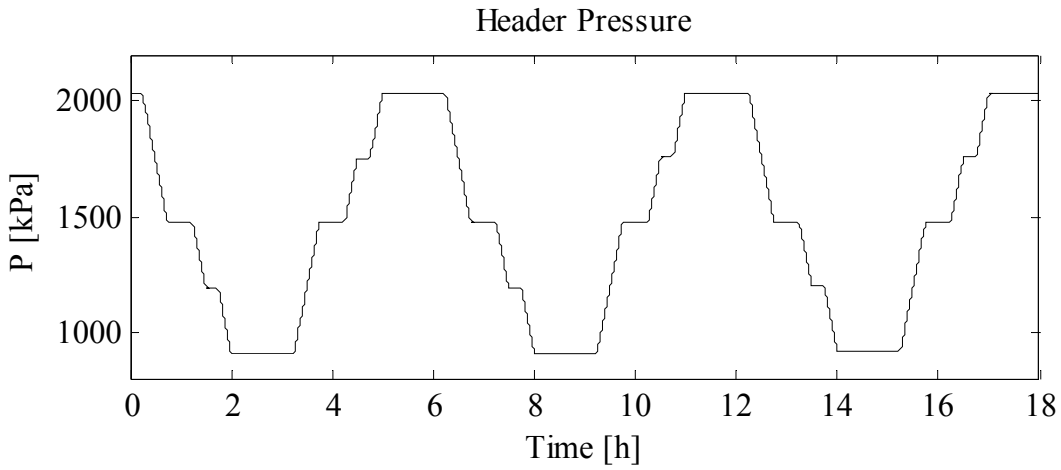

Figure 3.7. Step test results for the header pressure.

The Matlab System Identification Toolbox was used with the data displayed in Figure 3.4 to Figure 3.7 to identify the model parameters. The toolbox requires the specification of the data sets as well as the proposed forms of the candidate models (one form for each of the sub-models). It is also capable of estimating linear and nonlinear dynamic models and has additional functionality to validate the derived models. The user has the option of using the SID GUI (launched with the 'IDENT' command in Matlab) or to use the command line [8].

The step input magnitudes were chosen to excite the system adequately and were executed in a mutually exclusive manner (when changing one MV, all other MVs are kept constant) to ensure that the response data is uncorrelated. Furthermore, the durations of the changes were adequate to capture the important time constants for each sub-model. This is clear from Figure 3.4 and Figure 3.5. The step changes were made to the simulation model (the non-linear first principle model discussed in Section 3.3) and are free of noise or other unmeasured disturbances. Therefore, it is not necessary to be concerned about the signal-to-noise ratio. The simulation input and output data must be combined into an 'IDDATA' object (that also contains the sampling time) and de-trended to remove the steady-state offsets. If bad data sections are present, these need to be removed or interpolated, and if the

data is very noisy, filtering can be considered. The data can be viewed conveniently by using the 'IDPLOT' command [8]. See Addendum A: Matlab code for detail on the implementation in Matlab.

Several options and model types are available. The types of parametric models include ARX (Auto Regression with eXogenous inputs), state-space, transfer function, and non-linear parameterised models. They are referred to as parametric models because they have fixed structures with a number of unknown parameter to be determined by the SID process. Any of these types require specification of the model order of which the definition will vary depending on the type of model selected [8]. Step response and frequency response models are not considered to be parametric models as they are described by data tables rather than compact mathematical formulas with adjustable parameters. As mentioned, for this study, transfer function models were used (specifically first-order-plus-dead-time). The transfer function category is divided into two options. The first is referred to as process models which includes low order models (up to 3 poles) and may contain an integrator, a delay, and a zero. The second category is referred to as generalised transfer functions described in an input-output polynomial form [9].

After identifying the model, the SID toolbox generates an SID model object which can be converted to an LTI model object for use in (among others) the Control Systems Toolbox and the Model Predictive Control Toolbox. Difficulties with poles at or near the origin occur however when converting a high order parametric SID model object to an LTI model object (especially when the models contain dead-times that need to be converted to states). The model quality can be evaluated in several ways including comparing the model response to the measured response, analysing the residuals between the two, and analysis of the model uncertainty. The calculation of the residuals is illustrated in Figure 3.8 where u is the common input signal, y is the measured output data from the plant, $\hat{y}$ is the calculated output from the model, and e is the residual (or error) signal to be analysed.

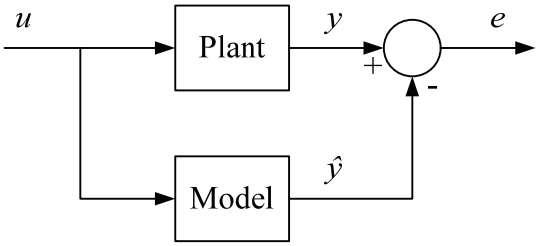

Figure 3.8. Block diagram of residual calculation [16].

To compare the outputs of the model with the measured outputs, the command 'COMPARE' can be used. For the model shown in Table 3.3, the comparison yielded the results shown in Figure 3.9. The percentage fit is summarised in Table 3.4 and indicates that the first-order-plus-dead-time model structure provides a good representation of the dynamic behaviour of the nonlinear first principle model at the specific operating point. In this case, the same data set was used for the modelling and validation due to the artificial nature of the step data. In cases where actual plant data is used, it is better to use separate data sets for the model identification and validation to ensure that the validation is uncorrelated to the identification [16].

Table 3.4. LTI model validation.

CV	% Fit
HHV	93.88
WI	93.12
FSI	93.92
P	99.20

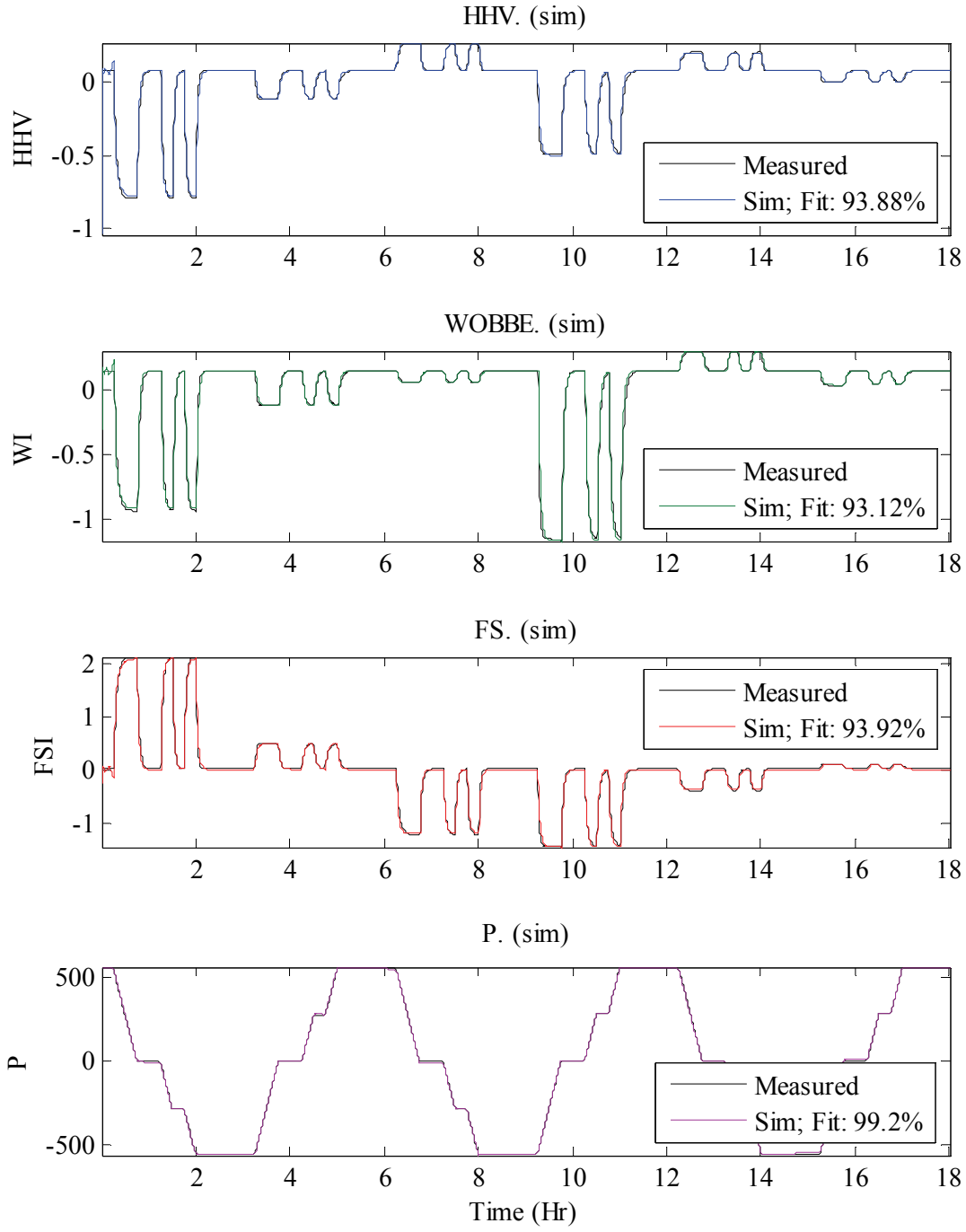

Figure 3.9. Validation of the LTI model.

3.5 CONCLUSION

In this chapter, the two main approaches for model development were introduced, namely first principle modelling and system identification from experimental data. Some important model forms and concepts were discussed for clarity. Thereafter, both a nonlinear first principle model and a linear empirical model were developed for the fuel gas blending system. The former is used for process simulation whereas the latter is used for control purposes (to be discussed in subsequent chapters). The first principle model was validated against real plant data to ensure that it gives a fair representation of the actual process. The empirical model was validated against the data from which it was derived (namely step response data from the nonlinear first principle model simulation).

CHAPTER 4 SIMULATION ENVIRONMENT

4.1 INTRODUCTION

The model described in Equations (3.5) to (3.11) was captured in a Simulink model for use within Matlab. The model receives data from vectors in the Matlab workspace and writes output vectors back into the workspace. The simulation time can be set in Simulink or the simulation can be initiated through Matlab commands. For the simulation in this study, the latter approach is followed. The model receives five vectors containing the compositions of the NG, RG, TG_1, and TG_2 streams as well as the volumetric discharge flow rate from the header (at 20 second intervals). All of these are unmeasured disturbances entering the model. In addition, the model requires an initial state vector when the simulation is initiated.

In turn, the model generates four main output vectors containing the HHV, WI, FSI, and pressure values (also at 20 second intervals). Several other vectors are also returned containing the HHV, WI, FSI and SG (specific gravity) of each of the individual inlet streams (used in the iterative linearisation calculations discussed in Section 5.3). As mentioned, the simulation receives an initial state vector. This vector contains the initial values of the six defined states of the model (the number of moles of each of the components) as well as several other values such as initial output values for the MPC controller, initial values for the delay blocks on the model outputs, etc. The simulation is set to receive this vector from the Matlab workspace and also to return a final state vector at the end of each simulation run. This is required for seamless transition between simulation runs. Therefore, the final state vector of one simulation run becomes the initial state vector for the subsequent run. The configuration of these vectors can be done in the Simulink settings shown in Figure B.1, Addendum B: MATLAB screenshots, where the initial state vector is called ‘xInitial’ and the final state vector is called ‘xFinal’.

4.2 SIMULINK MODEL DESCRIPTION

Figure 4.1 shows the main Simulink model display indicating the input vectors entering on the left and the output vectors on the right. The model contains several subsystems, each containing some more detail and logic (numbered from 1 to 7 in Figure 4.1). The model is shown in its closed loop configuration where its inputs (the volumetric flow rates of the six inlet streams) are generated (and therefore manipulated) by the MPC controller. The controller in turn receives the current controlled variable (CV) values (the HHV, WI, FSI, and pressure) generated by the model and compares them to the reference values, thereby providing the feedback in the control loop.

The first subsystem, Subsystem 1, (labelled 'Inflows' in Figure 4.1) is used to convert the volumetric flow rates and compositions of the inlet streams to molar flow rates of the individual components. Therefore, this subsystem implements Equation (3.6). The detail for this block is shown in Figure 4.2. The Subsystem receives the volumetric flow rates of the six inlet streams (kNm^3/h, depicted as input 1 in Figure 4.2) and multiplies them with a factor $1000/22.4 = 44.64$ (the amount of litres per cubic meter divided by the volume (litres) filled by one mole of gas under ideal conditions) to convert it to molar flow rate under the ideal gas assumption (as mentioned in Section 3.3). This multiplication is done by the gain block labelled 'To kmol' in Figure 4.2. The molar flow rates are then multiplied by the molar compositions of the inlet streams (depicted as inputs 2 to 5 in Figure 4.2) to arrive at the molar flows for the individual components in each stream. Finally the molar flows of the individual components are summed over the inlet streams to produce the total molar flow for each component into the header (u_i, depicted as output 1 in Figure 4.2).

The second subsystem (Subsystem 2, labelled 'Header' in Figure 4.1) solves the state equations to determine the state values over time and is shown in Figure 4.3. Therefore, this subsystem solves Equation (3.5) from the molar flow rates calculated in Subsystem 2 to produce the total moles of each component in the header and converts them to molar

fractions by dividing by the total number of moles in the header. It also calculates the header pressure from the moles, the temperature, the gas constant, the header volume, and the ideal gas correction factor (thereby solving Equation (3.10), output 3 in Figure 4.3). The block also receives the initial molar values at the initiation of the simulation (input 2 in Figure 4.3) in order to solve the initial value integration problem. The gas constant, volume, and real gas correction factor is set as block parameters in the subsystem (as shown in Figure B.3). The differential state equations are solved by numerical integration. The integration algorithm and step size can be specified in the simulation properties (as shown in Figure B.2). For this study, the Bogacki-Shampine numerical integration method is used which is a third order Runge-Kutta algorithm and is calculated as [17]

$$k_1 = f\left(t_n, y_n\right) \tag{4.1}$$

$$k_2 = f\left(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1\right) \tag{4.2}$$

$$k_3 = f\left(t_n + \frac{3}{4}h, y_n + \frac{3}{4}hk_2\right) \tag{4.3}$$

$$y_{n+1} = y_n + \frac{2}{9}hk_1 + \frac{1}{3}hk_2 + \frac{4}{9}hk_3 \tag{4.4}$$

$$k_4 = f\left(t_n + h, y_{n+1}\right) \tag{4.5}$$

$$z_{n+1} = y_n + \frac{7}{24}hk_1 + \frac{1}{4}hk_2 + \frac{1}{3}hk_3 + \frac{1}{8}hk_4 \tag{4.6}$$

where $\dot{y} = f\left(t, y\right)$ is the ordinary differential equation (ODE) to be solved, y_n is the numerical solution at time t_n, and h is the step size. The value calculated for y_{n+1} is a third order approximation of the function whereas z_{n+1} is a second order approximation. The difference between the two can be used to adapt the step size. The value of k_4 in one step equals the value of k_1 in the subsequent step which allows for only three function evaluations per step.

Subsystem 3 (labelled "MPC Controller in Figure 4.1) represents the MPC block for Simulink. The block references an MPC object which is generated in Matlab by either using the MPC GUI (Graphical User Interface) or through the Matlab command line. As mentioned above, the block receives the current CV values from the plant model as well as the CV reference setpoints and produces the manipulated variable (MV) moves (as volumetric flow rates) for controlling the plant. The details of the MPC controller object will be discussed in detail in Section 5.2.

Subsystems 4, 5, and 6 contain the calculations for Equations (3.7), (3.9), and (3.11) to produce the HHV, FSI and SG. The WI is then calculated by taking the HHV (from Subsystem 4 and dividing it by the square root of the SG from Subsystem 5 (i.e. the calculation performed in Equation (3.8)). The analysers measuring the HHV, FSI, and WI have time delays associated with them. Therefore, the simulation contains time delay blocks that delay the results of the HHV and FSI calculations by 1 sampling instant each (20 seconds) whereas that of the WI is delayed by three sampling periods (60 seconds). Integer delay blocks were used instead of transport delays because they allow their initial values to be arrays, thereby enabling the last values of one simulation run to be used as the initial values for a next run. If this is not done, the CVs will have constant values for the first couple of sampling periods (equal to the number of delays of the integer delay blocks) of each simulation (the entire simulation is divided into 10 minute runs in-between which the model is updated as discussed in Section 5.3). The delay values were set by maximising the correlation between the simulation data and the plant data (discussed in Section 3.3.1). The details for Subsystem 4 is shown in Figure 4.4. This subsystem implements Equation (3.7) (the gain block multiplies the molar fractions of the components with their corresponding HHV values as indicated in Table 3.1). Figure 4.5 shows the details for Subsystem 5 for the calculation of the SG. The first gain block multiplies the molar fractions of the components with their respective molar weights and the last divides them by the molar weight of air (see Equation (3.11)). Finally, Figure 4.6

depicts the details for Subsystem 6 that calculates the FSI of the fuel gas according to Equation (3.9).

Subsystem 7 is used to calculate the HHV, WI, FSI, and SG of each inlet stream for use in the iterative linearisation and RTO (Real-Time Optimisation) as discussed in Sections 5.3 and 5.4. Its detail is shown in Figure 4.7 and indicates that it contains Subsystems 4, 5, and 6.

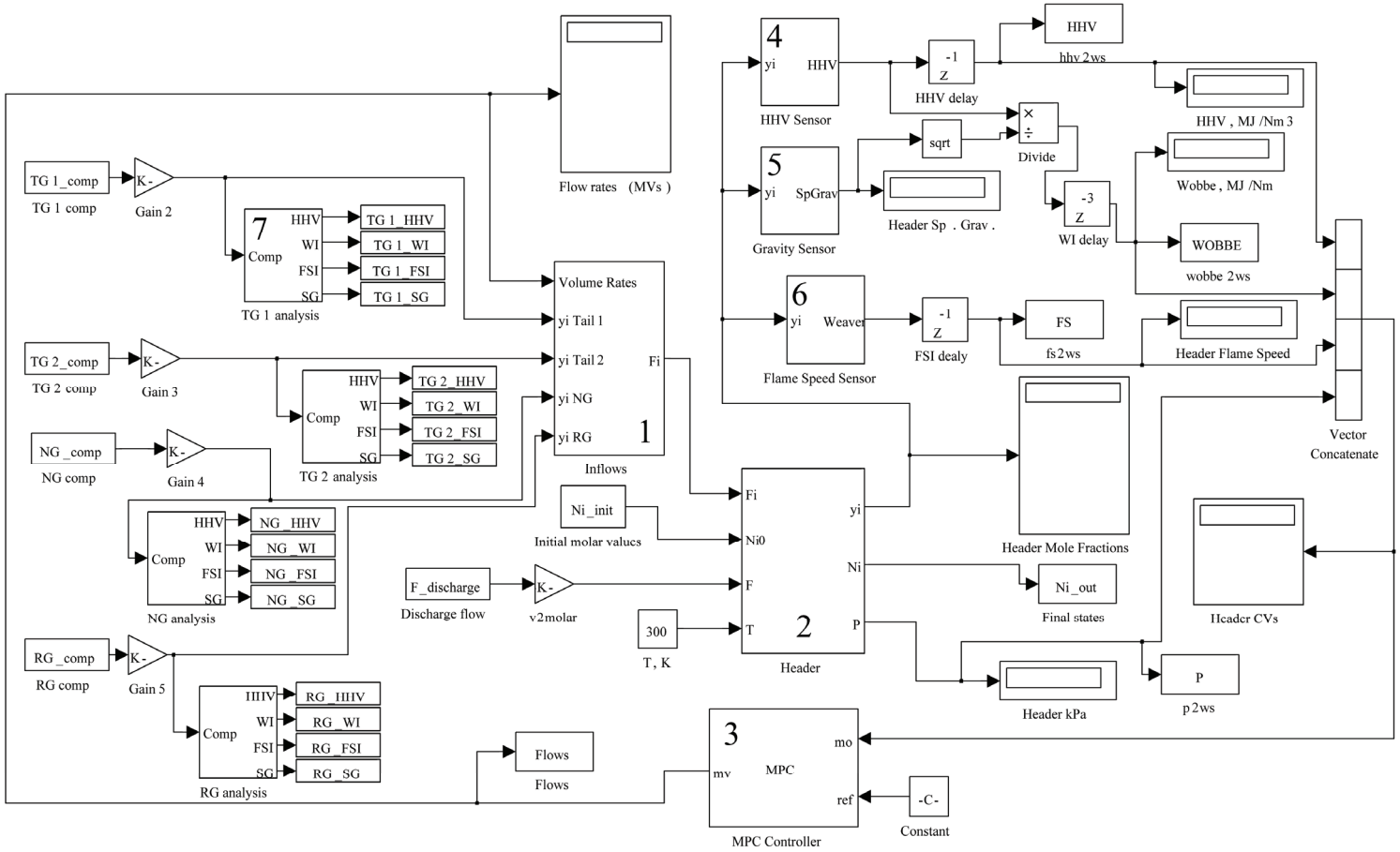

Figure 4.1. Simulink model overview.

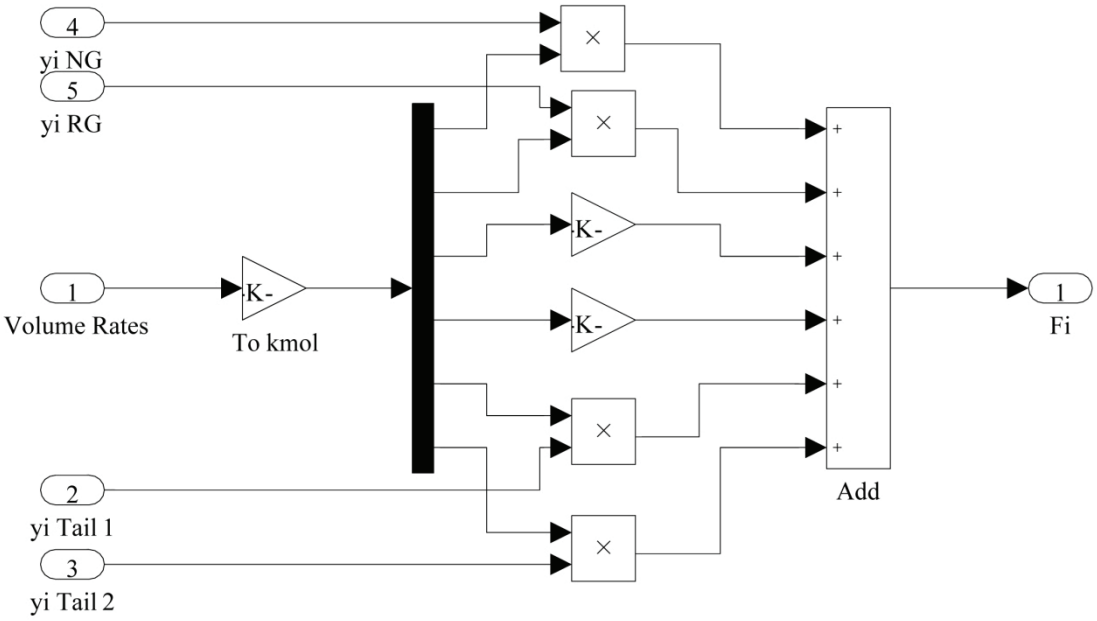

Figure 4.2. Subsystem 1 detail that converts the volumetric flow rates of the inlet streams to molar flow rates of the individual components.

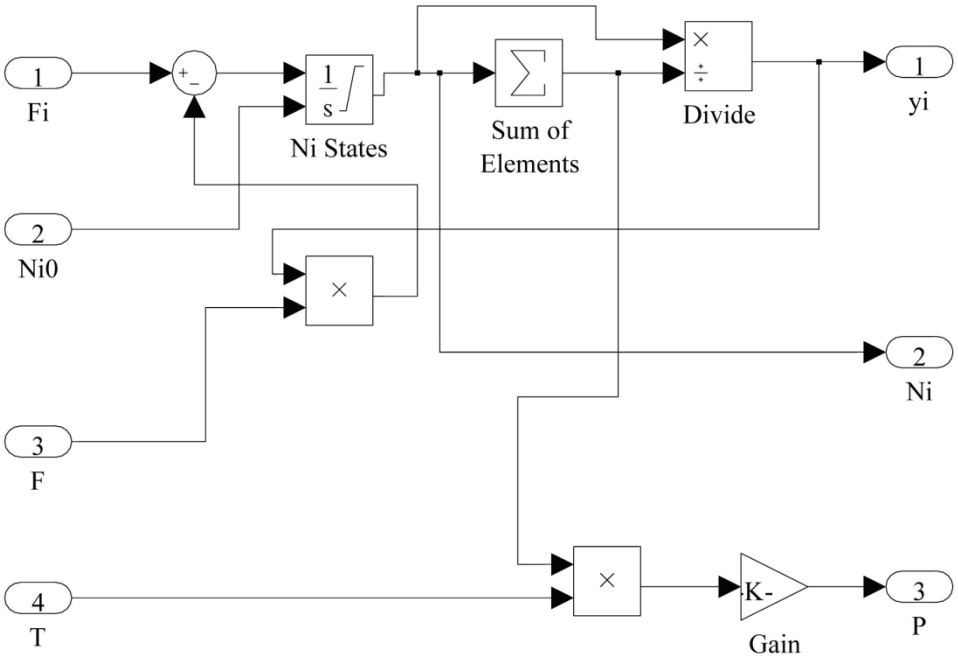

Figure 4.3. Subsystem 2 detail that solves the state equations and calculates the header pressure.

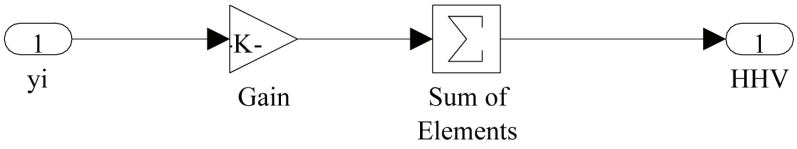

Figure 4.4. Subsystem 4 detail for the calculation of the HHV.

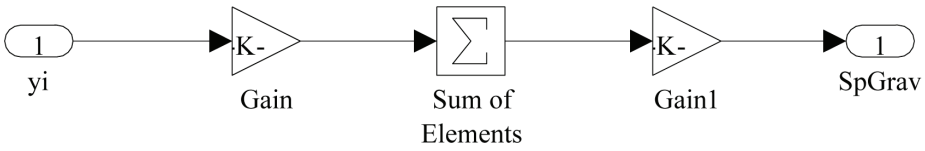

Figure 4.5. Subsystem 5 detail for the calculation of the SG.

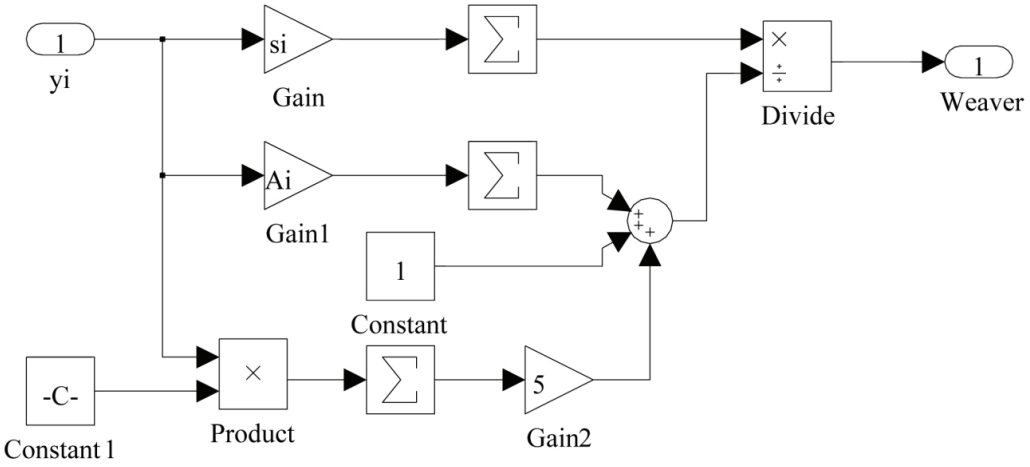

Figure 4.6. Subsystem 6 detail for the calculation of the FSI.

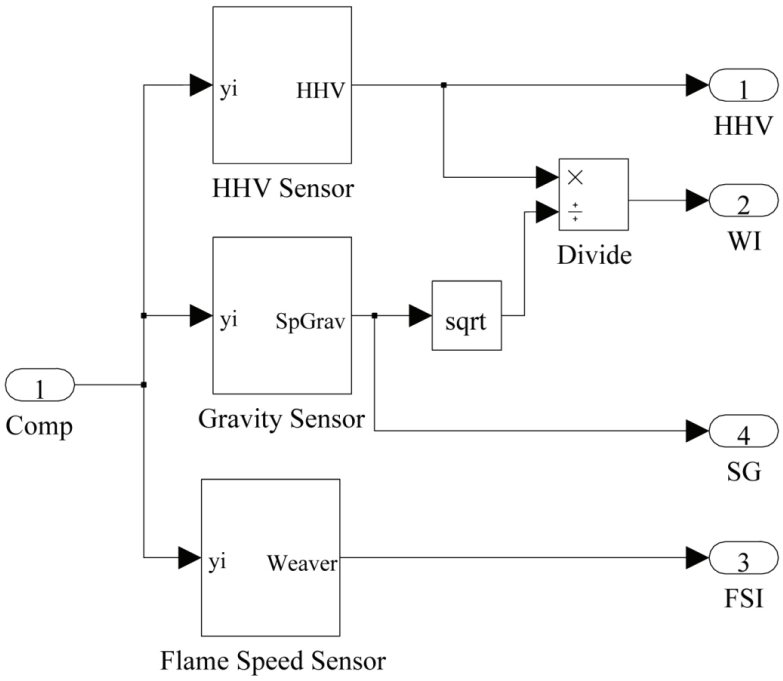

Figure 4.7. Subsystem 7 detail for the calculation of the HHV, WI, FSI, and SG of the individual inlet gas streams.

4.3 CONCLUSION

This chapter described details on the capturing of the first principle model described in Section 3.3 into the simulation environment, namely Simulink. This allows the model to be used in a simulation study to evaluate the efficacy of different control algorithms and to gain insight into the behaviour of the process. The next chapter describes the development of these control strategies.

CHAPTER 5 CONTROL

5.1 MPC OVERVIEW

Since the first description of MPC in the late 1970's by Richalet et al. [18] and its application in the refining industry by Shell Oil [19], significant attention has been given to the development of this powerful advanced control technique. MPC is a model-based control strategy that uses a dynamic model of a system to predict its future behaviour and then calculate the optimal control moves that would drive the states/outputs of a system (described by a mathematical process model) to their desired values (some reference trajectories) in an optimal way (determined by the definition of a performance function), and keep them there. This is done by performing optimal control over a finite time interval into the future, based on current state/output measurements, implementing the first control moves, and repeating the process using the latest measurements (also known as receding horizon control). Only the first moves are implemented due to model-process mismatches and unmeasured disturbances that cause the predicted state trajectories to differ from the actual system behaviour. If perfect process models were available and no disturbances were present, all the control moves calculated by the optimal control problem could have been enforced before repeating the optimisation problem. In essence, MPC aims to optimise the predicted process behaviour by implementing optimal control moves [10].

Although there are many MPC algorithms, the basic structure of MPC is common to most applications. The different algorithms differ according to the model used, the form of the cost function, and the way the controller handles noise and disturbances [20]. An overview of the commercial technologies available for MPC application in industrial plants can be found in [11].

Predicting plant behaviour typically requires solving the state differential equations by numerically integrating them over time. The control horizon is generally shorter than the prediction horizon resulting in the last couple of prediction intervals (equal to the difference between the prediction and control horizon) being subjected to the same

constant control moves. Figure 5.1 gives an illustration of a SISO (single input, single output) MPC-controlled system after a setpoint change in the controlled output variable. The input sequence u is calculated to get the predicted response, $\hat{y}$, as close as possible to the reference trajectory while honouring the constraints on u and y. The control horizon, N, is shorter than the prediction horizon, M. Therefore, for the balance of the sampling instants between the control and prediction horizons, u will stay constant. If the controller stops after calculating the first set of optimal control moves, mismatches between the system and the model will be ignored as well as changes in the process and disturbances. This problem is solved by implementing the first control moves (e.g. at time t_1, calculated with the information available at time t_0), taking new measurements after one sampling interval, recalculating the optimal control vector, and repeating the process. MPC can therefore be seen as a repetition of an optimal control algorithm at every sampling interval using the latest process measurements.

Some of the advantages of using MPC include flexibility in formulating the objective function and defining the process model, the ability to include equality and inequality constraints directly in the control law, accommodation of multivariable systems, and the possibility of dealing with large disturbances quickly (due to its inherent use of feed forward control). The main drawbacks in using MPC are the computational burden associated with it (especially when considering large systems and large control and prediction horizons) and the need for a reliable model of the process [20]. The main components of MPC are discussed in the following sections. One of the most popular applications of MPC is distillation control. Therefore, due to a lack of literature on MPC applications on blending processes, the components of MPC are discussed with reference to its use in control of distillation columns.

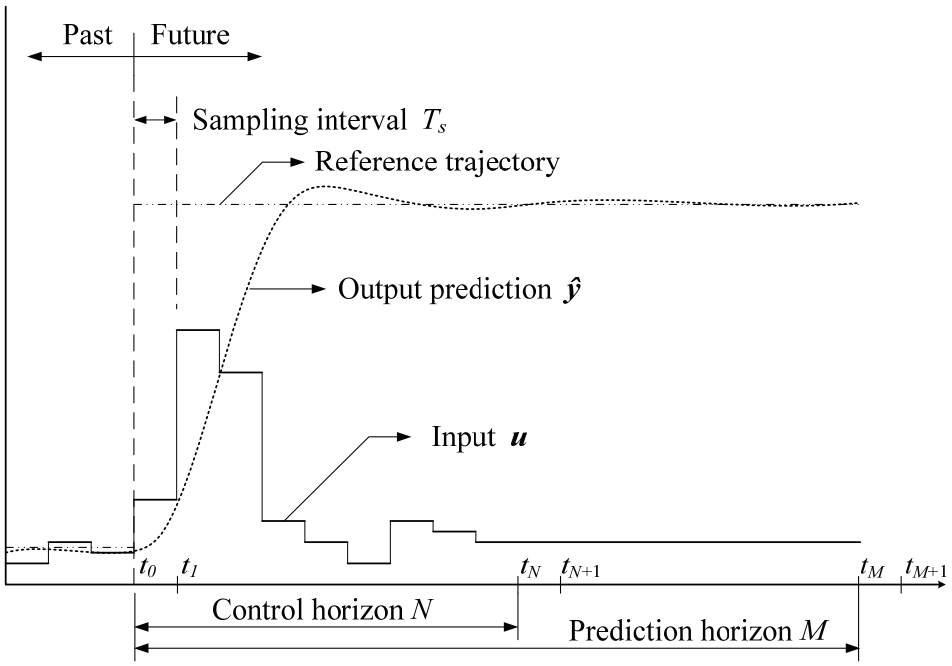

Figure 5.1. Illustration of SISO MPC concept (adapted from [6]).

5.1.1 Process model

The predicted behaviour to be optimised, is based on the model which makes it the most important element of the controller, although there will always be plant-model mismatch that will influence the accuracy of the predictions. Feedback can be used to compensate for these mismatches (although only to a certain extent) [10]. Refer to Sections 3.1 and 3.2 for more information on model types.

5.1.2 Choosing system variables

This task involves choosing the CVs (or outputs), the MVs (or control inputs), the DVs, and the state variables. The number of MVs, DVs, states, and CVs will be responsible for the model size and consequently the size of the control problem (apart from whether the model will be linear or nonlinear and if robustness will be considered). Choosing the wrong variables will affect the controllability and observability of the system.

Furthermore, the measurability and measurement latency have to be considered when choosing the controlled variables. For example, a process temperature can easily be measured with a thermocouple device and the measurement can be available to the control software after a negligible delay. An on-line composition measurement on the other hand might require expensive analyser equipment and the measurement result might only be available after a considerable delay. Therefore, inferentials are often used to predict variables (like composition) from other easily obtainable variables (such as pressures and temperatures). This is common in processes such as distillation control. For example Richalet et al. [18] used key tray temperatures to control the compositions of light and heavy end product streams. Product flow rates were changed to compensate for inlet temperature disturbances to offer a degree of feed-forward action. Karacan [21] manipulated the reflux ratio to control the top product temperature (and therefore the purity) of a packed-type distillation column. Grüner et al. [22] aimed to control the top (distillate) and bottoms compositions by adjusting the reflux ratio and reboiler heat duty. They used the temperatures in the 4th and 60th trays (of a 63 tray column) to estimate the compositions of the top and bottom product streams. Kawathecar and Riggs [23] chose the primary composition control to be the percentage of impurities in the overhead product of an ethyl acetate producing reactive distillation column. They specifically used the temperature in the second tray from the top to infer the impurity level in the overhead product and the temperature in the fifth tray from the bottom to infer the impurity level in the bottom product. They chose these temperatures as their main handles based on data obtained from a steady state model of the column which suggested the strongest correlation between deviation in product purity and tray temperature deviation. Khaledi and Young [24] controlled the percentage conversion and the bottom product purity in a reactive distillation column by manipulating the reflux flow rate and reboiler heat duty respectively. The reaction percentage was determined by measuring the differential temperature over trays 2 to 4. They suggested using a tray temperature close to the bottom tray to infer the purity of the bottom product. By graphing the stage temperature profiles at different reboiler heat duties, the tray with the highest sensitivity to reboiler duty was determined

and used for the inferential. Therefore, a sensitivity study can provide insight into the most appropriate variables to use when considering inferential control.

Several tools can be used to determine which variables will be best suited for the intended control application. RGA (Relative Gain Array) analysis requires only the steady-state gain matrix of the process and gives an indication of the process interactions as well as an idea of the most suitable pairing between controlled and manipulated variables [6]. Another useful tool is the SVA (Singular Value Analysis) which can help in the selection of the CVs, MVs, and DVs, assist with evaluation of the robustness of the system, and (if not using multivariable control) give recommended pairing of variables in a multiloop configuration [6].

5.1.3 System identification

The topic of system identification is covered in Section 3.4. As mentioned, the two main categories for models to be used in MPC are first principle (or theoretical) models and empirical (or experimental) models. Richalet et al. [18] used a discrete-time FIR (Finite Impulse Response) model to describe the relationship between inputs and outputs. The output at any given time depends on a linear combination of past inputs. Cutler and Ramaker [19] from Shell Oil used a linear step response model which relates output changes to a weighted sum of past input changes.

Several authors developed dynamic models from first principles [23,22,25]. Kawathecar and Riggs [23] followed this approach to develop a nonlinear model for a reactive distillation column. Their model consisted of liquid dynamics equations, component material balances, and energy balances (for each tray and the overall process). Grüner et al. [22] and Rueda et al. [25] developed detailed column models based on dynamic mass and energy balances.

Waller and Böling [26] used a quasi-ARMAX (Auto Regressive Moving Average with

eXogenous inputs) modelling scheme which can be seen as a combination of FIR and IIR (Infinite Impulse Response) modelling [27]. The identification process was performed by applying random magnitude input step changes of random duration. The steps were made in both positive and negative directions to ensure that the model accounts for the ill-conditioning. Simultaneous input changes were also made in the high and low gain directions (determined by the ratio of the input signal changes). Their simulations showed that the quasi-ARMAX model adequately captured the nonlinearity and directionality of the process. In a similar fashion, Venkateswarlu and Reddy [28] used a polynomial ARMA (Auto Regressive Moving Average) model and mentioned that the main advantage of these models are that they capture the nonlinearities of the process in a structure with linear parameters. The model parameters can be estimated by applying efficient parameter estimation methods (such as recursive least-squares).

Similar to the approach followed in this study, Khaledi and Young [24] estimated the nonlinear process of an ETBE (ethyl tert-butyl ether) reactive distillation column with a simple linear first-order-plus-dead-time model. The model was obtained by performing a series of step changes in the reflux rate and the reboiler heat duty (the MVs). From the step responses, the first order process gains and time constants were determined. Although the process showed high nonlinearity over wide MV ranges, it had near linear behaviour over the operating range of interest and could therefore be adequately approximated by these simple linear models.

Karacan [21] used a polynomial NARIMAX (Nonlinear Auto Regressive Integrated Moving Average with eXogenous inputs) model (a type of nonlinear difference equation model) to describe the dynamics of a pilot packed distillation column. The empirical model was generated using process input-output data. He made several assumptions in modelling the system to reduce the complexity of the resulting model and used a recursive Gauss-Newton prediction error algorithm to determine the parameters for the NARIMAX model.

Ou and Rhinehart [29] developed a neural network (NN) model suitable for parallel processing. The model consisted of a group of sub-models, each providing prediction for one CV at a future point in time. The sub-models were mutually independent, allowing them to be evaluated separately using parallel processing. Each of the sub-models was created as a separate neural network structure, resulting in a grouped neural network (GNN) process model. To compensate for process-model mismatches, the differences between the current measured outputs and the values predicted for the current outputs at the previous sampling interval were calculated and added to all prediction values in the current prediction trajectory.

Alpbaz et al. [30] developed a set of models governed by ODEs for a packed distillation column by dividing the height of the column into a number of stages. The process dynamics were investigated by making step changes to the reflux ratio (MV) at steady state conditions and observing the effect on the top temperature (CV related to the top product purity). Again similar to the LTI model for this study, the step-response model was expressed as a first-order-plus-dead-time function.

After the model has been developed, it needs to be validated to determine how closely it resembles the actual process. This was discussed in Section 3.3.1.

5.1.4 Constraints and performance index

After obtaining the mathematical representation, the input, output, and state constraints need to be specified and the performance function (also called the cost function, objective function, or performance index) formulated. The performance function typically consists of errors between the desired output/state trajectories and the predicted values. Additional terms may include energy usage in the control signals and outputs, terminal conditions, and time dependency. The performance index has to be minimised subject to the system constraints. The constraints typically consist of both equality and inequality functions describing physical constraints (such as levels having to be greater than zero) and

performance constraints (such as an optimal setpoint). In some cases, the rate of change in the control and state/output variables are also included in the definition of the performance index and constraints, thereby limiting the strain on actuators and process equipment. Some constraints will be obvious (such as level constraints) whereas others may be difficult to determine. The performance index to be minimised can be written in a general form as [31] (see Equations (5.5) to (5.14) for detail on the cost function used in this study)

$$J(\bar{y},\bar{u},t) = S(\bar{y}_M,\bar{u}_M,t_M) + \int_0^{t_M} V(\bar{y},\bar{u},t)\,dt \tag{5.1}$$

where $\bar{y}$ is the output vector (the CVs), $\bar{u}$ is the input vector (the MVs), M is the prediction horizon, S is the terminal cost (at the end of the prediction horizon), and V is the interval cost. In many cases, the states are penalised rather than the actual outputs. In such cases, the penalty function remains the same as Equation (5.1) with $\bar{y}$ substituted by $\bar{x}$, the state vector [31]. The terminal cost function is used if it is important that the system reach a certain condition at the end of the prediction horizon (typically within a certain range of a setpoint). The interval cost is used where it is not only important to get the system to a desired state but also how it gets there. Equations (5.2) to (5.4) indicate the form of V for some popular objectives including minimising time (Equation (5.2)), energy (in control or output variables; Equation (5.3)), and deviation from setpoints (Equation (5.4)).

$$V(\bar{y},\bar{u},t) = 1 \tag{5.2}$$

$$V(\bar{y},\bar{u},t) = \bar{u}^T \cdot \bar{R} \cdot \bar{u} \tag{5.3}$$

$$V(\bar{y},\bar{u},t) = [\bar{y}_r - \bar{y}]^T \cdot \bar{Q} \cdot [\bar{y}_r - \bar{y}] \tag{5.4}$$

The move suppression matrix $\bar{R}$ allows for the individual weighting of the input signals according to their priorities, whereas $\bar{Q}$ allows for giving different weights to the output signals according to their relevant importance.

Richalet et al. [18] defined the performance index as the error between the predicted outputs and first order reference trajectories from the current values to the desired setpoints. Thereby, the speed of the controller could be set by changing the time constants of the reference trajectories.

Cutler and Ramaker [19] implemented a quadratic performance function which penalised output deviations from the desired setpoint values. They also included a penalty term for the MVs to limit the movement of the control signals, resulting in less aggressive output responses. The weighting matrix for the MVs (move suppression matrix) allowed inputs to be penalised by different factors. Waller and Böling [26], Kawathecar and Riggs [23], Venkateswarlu and Reddy [28], Ou and Rhinehart [29], and Alpbaz et al. [30] followed similar approaches. In some cases, a disturbance term was added to the objective function over the prediction horizon to compensate for prediction-measurement mismatches at each sampling interval [23].

Abou-Jeyab et al. [32] formulated the objective function using the absolute errors of the controlled variables with regard to a reference trajectory minus the reflux flow rate (in order to maximise the reflux flow). Upper and lower constraints were imposed on each of the controlled and manipulated variables. For the manipulated variables, rate of change constraints were also considered.

Wojsznis et al. [33] discussed a three-tier objective function with constraint handling as the first priority, maximising economics as the second, and maintaining control as the third. They employed dynamic assignment of weights (slack variables) to control and constraint variables when it was predicted that the values would exceed their limits. The optimiser addressed the basic control functionality objectives in its normal form (with built-in default objectives) and addressed constraint and economic objectives using the penalised slack variables. The slack variables allowed the performance index to be penalised according to the severity of the predicted violations, weighting the errors more as the predicted

deviations became larger. This was done by reformulating the inequality constraints on the controlled variables into equality constraints by adding the values of the predicted violations as slack variables. This incremented the degrees of freedom and helped the optimiser to cope with possible unfeasible solutions (such as after large disturbances). The optimiser was prevented from using the slack variables in normal operation by making the penalty weights for violating constraints significantly higher than the weights on economic and control performances.

5.1.5 Controller parameters

The choice of controller parameters plays an important role in the performance of the controller. The most important parameters for MPC are the control and prediction horizons, the sampling (or iteration) time, and the objective function weighting matrices. The control horizon refers to the number of future control moves calculated for every iteration. The prediction horizon is the number of steps into the future for which the system equations are solved to give estimates of the state/output values at these instances. It is obvious that these parameters play an intricate role in the execution time of the control algorithm.

The execution time of the control algorithm must be less than the required sampling time of the system for the controller to be implementable in practice. Every system has a minimum sampling time associated with it to ensure proper control. Therefore, the control parameters play an important role in the success (or failure) of the controller. If the prediction and control horizons are chosen to be too large, the execution time of the algorithm (on a particular system) may become larger than the required sampling time of the process.

Kawathecar and Riggs [23] used a prediction horizon of 80, a control horizon of 15, and a sampling interval of 20 minutes in their control of a reactive distillation column. Furthermore, they weighted impurities in the overhead product 10 times heavier than

impurities in the bottoms product due to the effect of the overhead impurities on the final product. The move suppression factors were chosen to minimise the IAE (Integral Absolute Error) from the setpoint of the overhead product based on overhead impurity setpoint changes.

Abou-Jeyab et al. [32] used a control horizon of 1 in their formulation of a simplified model predictive control algorithm. To determine the prediction horizon, they calculated the average time it would take the predicted output trajectories to intersect the setpoint trajectories due to a single control vector. The errors were then minimised only at this one point in the future. Therefore, only one optimisation problem was solved and one control vector calculated every sampling interval. By manipulating the optimisation point (prediction horizon), the controller could be made more or less aggressive. Furthermore, this algorithm allowed the independent adjustment of the response times for each of the controlled variables by tuning the respective intervals at which the specific errors were minimised.

Waller and Böling [26] used a control horizon of 3, a prediction horizon of 25, and a sampling time of 1 minute. They weighted the bottom composition error slightly higher than the distillate composition error with equal weights on the inputs.

Khaledi and Young [24] used a prediction horizon of 20 with a sampling time of 1.5 minutes and a control horizon of 2, reasoning that a small control horizon will prevent the controller from being too aggressive.

Bezzo et al. [34] chose to use a prediction horizon of 60, a control horizon of 15, and a sampling time of 5 minutes for control of a middle-vessel continuous distillation column. They weighted the distillate and bottom compositions more heavily than the levels and maintained loose control over the various levels.

5.1.6 Solving the optimisation problem

Some common formulations of the objective function (or performance index) were discussed in Section 5.1.4. Solving the optimisation problem entails minimizing (or maximising, depending on the formulation) the objective function by adjusting the set of input variables (subject to constraints).

Various techniques are used for solving the optimisation problem. An important factor to be considered is whether the system model is linear or nonlinear. There are two main approaches to solving the optimisation problem namely sequentially or concurrently [23,25]. For sequential solution, a reference control profile is selected and the system equations are integrated to obtain a state/output profile. Thereafter, the objective function, as well the sensitivities of the objective function to changes in manipulated variables, are determined. The optimisation problem is then solved (using these gradients) to calculate an updated control profile that will reduce the value of the objective function. These steps are repeated to minimise the objective function value. The drawback of the sequential approach is that the system equations have to be solved at each iteration of optimisation resulting in long execution times. For concurrent solution, the model equations are appended to the optimisation problem in the form of equality constraints (algebraic form) and the optimality and constraints are therefore treated simultaneously. The optimisation problem is then solved subject to these equality constraints as well as constraints on the controlled, manipulated and state variables. The concurrent method will reduce the time spent on numerical integration at the cost of a larger optimisation problem. Therefore, there is a trade-off between these approaches. Generally, for large problems (involving complex models and large prediction horizons) the concurrent solution is preferable whereas the sequential method may be chosen for smaller problems with few states and small prediction horizons. Some of the most common optimisation techniques include linear programming (LP), quadratic programming (QP), sequential quadratic programming (SQP), and nonlinear programming (NLP).

Cutler and Ramaker [19] calculated the optimal control vectors by solving a least squares problem. Their use of a step response model allowed future output variations to be written as a linear combination of future input moves. The matrix relating the inputs and the outputs is known as the dynamic matrix. Therefore, their control algorithm is known as DMC (Dynamic Matrix Control).

Kawathecar and Riggs [23] solved the system equations for their nonlinear model using the Euler method with 0.5s time steps (compared to a 20 minute sampling interval). They formulated the optimization problem as a nonlinear programming (NLP) problem. They used the concurrent approach for performing optimisation and applied orthogonal collocation on finite elements to convert the ODEs (Ordinary Differential Equation) into a set of algebraic equations to be used as equality constraints.

Abou-Jeyab et al. [32] managed to perform a single linear programming solution at each sampling interval with their simplified MPC algorithm. As mentioned in Section 5.1.5, they calculated an average time at which the predicted response for a single control move would intersect the setpoint trajectory (used as the prediction horizon) and minimised the error at only that point in the future. They used the result as the only control vector calculated at each sampling interval (i.e. a control horizon of 1), thereby reducing the size of the optimisation problem drastically.

Grüner et al. [22] made use of asymptotically exact input/output linearisation for aiding in solving the system equations. The control law required that the state of the plant be known, which led to the development of an observer to estimate the state of the plant according to simple temperature measurements. The input/output linearisation involved differentiating the output equations with regard to time until the expression explicitly involved at least one linear input component. The differentiated output equations were then linearised by approximation with a finite differences approach. A new artificial input was then introduced into the equation (together with input gain) that forced a linear (and decoupled)

behaviour from this input to the output. Therefore, by linearising the output equations, linear optimisation techniques could be applied instead of using nonlinear optimisation.

Venkateswarlu et al. [28] investigated the use of stochastic algorithms, including genetic algorithms (GAs) and simulated annealing (SA) for nonlinear optimisation as an alternative to sequential quadratic programming (SQP). These stochastic techniques were combined with a polynomial-type nonlinear empirical process model to arrive at their modified NMPC algorithms, namely GANMPC and SANMPC. The attractiveness of using GAs and SA lies in the ability to deal with constrained, nonlinear, and non-convex optimisation problems without having to calculate the model derivatives. Furthermore, these algorithms have the ability to find solutions in close proximity to the global minima, and not to get stuck at local minima. Their choice of a polynomial input-output model provided one-step-ahead prediction. Therefore, the prediction trajectory to be optimised could be calculated by cascading the model with itself.

Diehl et al. [35,36] proposed a real-time optimisation algorithm based on direct multiple shooting (falling under the class of concurrent optimisation). In this optimisation scheme, the entire prediction horizon is considered and successive linearisation along optimal trajectories is performed. The algorithm can be performed on systems described by differential algebraic equations (DAE) and subject to inequality constraints. Direct multiple shooting is a way of converting a continuous time system into discrete time for use in digital systems. The real-time optimisation scheme starts by solving a Newton-type optimisation problem over the entire prediction horizon, implements the first control move, and proceeds to solving the next optimisation problem, considering only the rest of the original prediction horizon. Therefore, the size of the optimisation problem decreases with every iteration until the end of the prediction horizon is reached where after a new prediction horizon is considered. This method is known as receding horizon control and allows rapid response to disturbances. In a similar discussion, Gerdts [37] formulated a direct shooting method for numerical solution of optimal control problems for systems

described by higher-index DAEs subject to state constraints. The direct shooting method is described as a way of casting a continuous-time optimal control problem into a discrete-time finite-dimensional nonlinear programming (NLP) problem. The problems encountered when using numerical integration schemes to solve DAE systems include ill-conditioning, instability, convergence problems, and calculation of consistent initial values.

Karacan [21] used orthogonal collocation on finite elements employing Legendre polynomials for solution of the NARIMAX model equations. This method implies dividing the column into a number of elements and applying the orthogonal collocation to each element separately. Thereafter, the model partial differential equations are solved using these finite elements.

Ou and Rhinehart [29] formulated their grouped neural network model in such a way that the predicted process values were calculated directly using past inputs and outputs and the future guess inputs. Therefore, there was no need for numerical integration of the process model.

5.2 MPC DESIGN FOR FUEL GAS CONTROL

The interactive and multivariable nature of the fuel gas blending system makes it an ideal candidate for MPC. As mentioned, the MVs for this process were chosen to be the volumetric flow rates (kNm^3/h) of the six inlet gas streams and the CVs are the HHV, WI, FSI, and pressure. No measured disturbances (DVs or FF variables) were included in the model. The LTI model described in Section 3.4.2 was used as an initial model in the MPC algorithm to control the plant CVs within their specified ranges while attempting to minimise the operating cost according to the relative costs of the inlet streams. The relative costs are given in Table 5.1. The MPC was designed using the Model Predictive Control Toolbox in Matlab [15]. The Toolbox includes the ability to add an MPC function block to

a Simulink model and link the block to an MPC object (created with the MPC Toolbox). This approach was followed and is indicated in Section 4.2, specifically in Figure 4.1. The MVs and CVs are set in the MPC Toolbox utility (as shown in Figure B.4). This will happen automatically if the LTI object is configured correctly.

Table 5.1. Relative costs of inlet streams [cost/kNm^3].

Feed stream	Relative cost
NG	0.678
RG	0.254
H_2	0
N_2	0.068
TG_1	0
TG_2	0

The average settling time for the HHV, WI, and FSI is about 12 minutes. This can be determined by taking an average time constant of $\tau = 1/25*60 = 2.6$ minutes and then calculating the settling time $t_s \approx 5.\tau = 12$ minutes [6]. The initial values for the prediction and control horizons were chosen according to the proposed guidelines in [6]. This resulted in a control horizon of 5 and a prediction horizon of 44 (12 (settling time in minutes) times 3 (three samples per minute) plus 5 (the control horizon) plus 3 (the maximum dead-time in samples)). These values did not give the desired results. The values were changed by trial and error to arrive at a final prediction horizon of 39 samples (to cover the settling time (36 samples) plus largest dead-time (3 samples)) and a control horizon of 3 sampling intervals (to prevent the controller from being too aggressive). Furthermore, blocking of the control horizon was used to distribute the control moves more evenly over the prediction horizon. A block is one or more successive sampling periods over which the MVs are kept constant [15]. The blocking parameters were selected to calculate control moves at 2, 6, and 12 sampling intervals (as shown in Figure B.5).

Weights were added to the inputs in relation to the relative costs of the streams (given in Table 5.1) with identical rate weights. The weights are shown in Table 5.2. The MV weights penalise the deviation from nominal values (which are initially set to [0 0 5 0 30 30]). The outputs were given weights (shown in Table 5.3) for deviations from the nominal values which were chosen to be mid-range (i.e. $HHV_{nominal}$ = 17.25, $WI_{nominal}$ = 26, $FSI_{nominal}$ = 42.5, and $P_{nominal}$ = 2100) for the initial two cases (without real-time optimisation as discussed in Section 5.4). The weights were chosen according to priority of the CVs where the highest priority is given to HHV and the lowest to pressure. The weight settings are shown in Figure B.6.

Table 5.2. Weights on MVs.

MV	Weight	Rate weight
NG	67	1
RG	25	1
H_2	0	1
N_2	7	1
TG_1	0	1
TG_2	0	1

Table 5.3. Weights on CVs.

CV	Weight
HHV	100
WI	80
FSI	70
P	20

In addition to the weights, constraints were put on the inputs and outputs with moderate constraint softening on the CVs. The input constraints were chosen to be representative of the availability of each stream and are shown in Table 5.4. The configuration of the constraints and constraint softening are shown in Figure B.7 and Figure B.8.

Table 5.4. Limits on MVs [kNm^3/h].

MV	Low limit	High limit	Units
NG	0	15	kNm^3/h
RG	0	20	kNm^3/h
H_2	0	5	kNm^3/h
N_2	0	5	kNm^3/h
TG_1	0	30	kNm^3/h
TG_2	0	30	kNm^3/h

The output constraints correspond to the controlled variable ranges shown in Table 2.1. The tuning settings (including prediction horizon, control horizon, weights, and rate constraints) were derived by trial and error (using the recommended guidelines from [6]). The form of the cost function is [15]

$$\min_{\Delta u(k|k),\ldots,\Delta u(m-1+k|k),\,\varepsilon} \quad \sum_{i=0}^{p-1}\left(S_{y,i}(k)+S_{\Delta u,i}(k)+S_{u,i}(k)\right)+\ \rho_\varepsilon\varepsilon^2 \tag{5.5}$$

with

$$S_{y,i}(k)=\sum_{j=1}^{n_y}\left|w_j^y\left(y_j(k+i+1\,|\,k)-r_j(k+i+1)\right)\right|^2 \tag{5.6}$$

$$S_{\Delta u,i}(k)=\sum_{j=1}^{n_u}\left|w_j^{\Delta u}\Delta u_j(k+i\,|\,k)\right|^2 \tag{5.7}$$

$$S_{u,i}(k)=\sum_{j=1}^{n_u}\left|w_j^{u}\left(u_j(k+i\,|\,k)-u_{j_{target}}(k+i)\right)\right|^2 \tag{5.8}$$

subject to

$$u_{j_{\min}}(i)-\varepsilon V_{j_{\min}}^u(i)\le u_j(k+i\,|\,k)\le u_{j_{\max}}(i)+\varepsilon V_{j_{\max}}^u(i) \tag{5.9}$$

$$\Delta u_{j_{\min}}(i)-\varepsilon V_{j_{\min}}^{\Delta u}(i)\le \Delta u_j(k+i\,|\,k)\le \Delta u_{j_{\max}}(i)+\varepsilon V_{j_{\max}}^{\Delta u}(i) \tag{5.10}$$

$$y_{j_{\min}}(i)-\varepsilon V_{j_{\min}}^y(i)\le y_j(k+i+1\,|\,k)\le y_{j_{\max}}(i)+\varepsilon V_{j_{\max}}^y(i) \tag{5.11}$$

$$i=0,\ldots,\,p-1 \tag{5.12}$$

$$u(k+h\,|\,k)=0;\quad h=m,\ldots,\,p-1 \tag{5.13}$$

$$\varepsilon\ge 0 \tag{5.14}$$

where $\Delta u(k+i\,|\,k)$ is the input change vector at time $k+i$ based on the information available at time k, m is the control horizon, p is the prediction horizon, ε is a slack variable, ρ_ε is the weight on the slack variable, $r(k)$ is the output reference vector at time k, w_j^y, $w_j^{\Delta u}$, and w_j^u are weighting matrices for the outputs, input increments, and the

inputs, n_y is the number of outputs, and n_u is the number of inputs. The vectors $u_{j_{\min}}$, $\Delta u_{j_{\min}}$, $y_{j_{\min}}$, $u_{j_{\max}}$, $\Delta u_{j_{\max}}$, and $y_{j_{\max}}$ are the minimum and maximum values for the inputs, input changes, and outputs whereas $V^{u}_{j_{\min}}$, $V^{\Delta u}_{j_{\min}}$, $V^{y}_{j_{\min}}$, $V^{u}_{j_{\max}}$, $V^{\Delta u}_{j_{\max}}$, and $V^{y}_{j_{\max}}$ are the ECR (Equal Concern for Relaxation) vectors that govern how much constraint violations can be tolerated [15].

The way the MPC handles noise and disturbances is also specified (as shown in Figure B.9 and Figure B.10). The MPC GUI utility was used for the initial design of the MPC object which was linked into the MPC block in Simulink. All subsequent changes to the model and to the MPC object were done though Matlab commands (see Addendum A: Matlab code). These include changes to the gains of the LTI model used by the MPC, changes to the nominal MV and CV values, and updating of ideal resting values (steady-state targets) on MVs.

5.3 ITERATIVE LINEARISATION

The MPC algorithm used for the control is designed for linear plants. The Fuel Gas system however is nonlinear which reduces the performance of the MPC when moving away from the design operating region. There are several possibilities for dealing with nonlinearities. One option is to use nonlinear MPC which requires the use of a nonlinear model and optimiser (which is more complex and computationally intensive than the linear case). Another option is to continuously linearise the plant at the current operating point [38]. The latter is the technique followed in this study and results in the control scheme becoming a variant of nonlinear MPC (NMPC). The iterative linearisation is performed by calculating the gains of the Transfer Function Matrix (TFM) at every 30th iteration (every 10 minutes) and updating the model used by the MPC accordingly [12]. The compositions of the inlet streams on the plant are measured by the same mass spectrometer mentioned in Section 3.3.1 which samples every 10 minutes. Therefore, the linearisation frequency was chosen to coincide with this sampling frequency. The initial TFM is given in Table 3.3.

The gain calculations are discussed in the following sections. The linearisation involves normalising the TFM, updating it with the newly calculated gain values, and updating the MPC to use the new TFM. The transient behaviour of the plant does not change significantly at different operating regions. Therefore, only the gains of the TFM are updated, leaving the dynamic parameters (the time constants and delays) unchanged.

The steady state values for the controlled variables can be calculated directly from the volumetric flow rates and compositions of the inlet streams. Taking the derivatives of these equations with regard to the individual inlet streams give the instantaneous gains. These gains can then be used to update the model used in the MPC to provide a form of iterative linearisation. The gain calculations are described in the next sections [12,13,39].

5.3.1 Heating value

The fuel gas heating value is calculated as

$$HHV_{fg} = \frac{\sum_{i=1}^{6} F_i \cdot HHV_{F_i}}{F_T} \tag{5.15}$$

where F_i and HHV_{F_i} are the volumetric flow rate (kNm^3/h) and heating value (MJ/Nm^3) of the i^{th} inlet stream and $F_T = \sum_{6}^{i=1} F_i$ is the total inlet volumetric flow rate. The gains are then calculated as

$$\frac{\partial HHV_{fg}}{\partial F_i} = \frac{HHV_{F_i} - HHV_{fg}}{F_T} \tag{5.16}$$

5.3.2 Wobbe index

The Wobbe index is calculated as (same as Equation (3.8))

$$WI_{fg} = \frac{HHV_{fg}}{\sqrt{\rho_{fg}}} \tag{5.17}$$

where HHV_{fg} is given in Equation (5.15) and ρ_{fg} is the relative density of the fuel gas, calculated with regard to inlet flow rates as

$$\rho_{fg} = \frac{\sum_{i=1}^{6} F_i \cdot \rho_{F_i}}{F_T} \tag{5.18}$$

where ρ_{F_i} is the relative density of inlet gas *i*. Taking the derivative of WI_{fg} with regard to F_i gives

$$\frac{\partial WI_{fg}}{\partial F_i} = \frac{1}{\sqrt{\rho_{fg}}} . \frac{\partial HHV_{fg}}{\partial F_i} - \frac{HHV_{fg}}{2.\rho_{fg}^{1.5}} . \frac{\partial \rho_{fg}}{\partial F_i} \tag{5.19}$$

with $\frac{\partial HHV_{fg}}{\partial F_i}$ given in Equation (5.16) and

$$\frac{\partial \rho_{fg}}{\partial F_i} = \frac{\rho_{F_i} - \rho_{fg}}{F_T} \tag{5.20}$$

5.3.3 Flame speed index

The flame speed formula is (also shown in Equation (3.9)),

$$FSI_{fg} = \frac{\sum_{i=1}^{6} y_{fg,i} . s_i}{\sum_{i=1}^{6} y_{fg,i} . A_i + 5\sum_{j=1}^{2} n_{fg,j} - 18.8 x_{O_2} + 1} \tag{5.21}$$

with the values for A_i and s_i given in Table 3.1. To calculate the FSI in terms of the inlet volumetric flow rates, the molar components in the fuel gas are calculated using

$$y_{fg,i} = \frac{\sum_{j=1}^{6} F_j . y_{Fj,i}}{F_T} \tag{5.22}$$

where $y_{Fj,i}$ is the molar fraction of component i in inlet stream j. The derivative can then be determined as

$$\frac{\partial FSI_{fg}}{\partial F_i} = \sum_{x=1}^{4} \frac{\partial FSI_{fg}}{\partial y_{fg,x}} . \frac{\partial y_{fg,x}}{\partial F_i} + \sum_{k=1}^{2} \frac{\partial FSI_{fg}}{\partial n_{fg,k}} . \frac{\partial n_{fg,k}}{\partial F_i} \tag{5.23}$$

where $y_{fg,x}$ refers to the molar fraction of combustible component x in the fuel gas and $n_{fg,k}$ is the molar fraction of inert component k in the fuel gas. The individual terms in Equation (5.23) are given by

$$\frac{\partial FSI_{fg}}{\partial y_{fg,x}} = \frac{s_x - A_x . FSI_{fg}}{\sum_{i=1}^{6} y_{fg,i} . A_i + 5 \sum_{j=1}^{2} n_{fg,j} - 18.8 x_{O_2} + 1} \tag{5.24}$$

$$\frac{\partial y_{fg,x}}{\partial F_i} = \frac{y_{F_i,x} - y_{fg,x}}{F_T} \tag{5.25}$$

$$\frac{\partial FSI_{fg}}{\partial n_{fg,k}} = \frac{-5 . FSI_{fg}}{\sum_{i=1}^{6} y_{fg,i} . A_i + 5 \sum_{j=1}^{2} n_{fg,j} - 18.8 x_{O_2} + 1} \tag{5.26}$$

$$\frac{\partial n_{fg,k}}{\partial F_i} = \frac{n_{F_i,k} - n_{fg,k}}{F_T} \tag{5.27}$$

5.4 REAL-TIME OPTIMISATION (RTO)

5.4.1 Literature review

In order to push the operating cost of a process towards its true minimum (or maximise operating profit) as operating conditions change (such as changes in feed and product costs, variable equipment availability, and process disturbances), an online economic optimisation technique is employed on top of MPC known as real-time optimisation (RTO)

[6,14,38,40]. Most RTO applications involve the use of nonlinear steady-state models that are updated with plant parameters such as product compositions [41]. To increase process economics, efficiency, and product quality, real-time optimisation (RTO) has become an important topic in the control industry (with its relevance already noted in the early 1980's [14]). RTO is applied on top of one or more unit-based multivariable controllers and strives to achieve operation near the economic optimum of the system as a whole by recalculating the optimum operating conditions on a regular basis [6]. Typically, RTO is based on a steady-state model of the plant and calculates the ideal target values for CVs and MVs to reach the steady-state optimal (while honouring constraints). The steady-state RTO calculations are performed at a lower frequency than the MPC execution interval (due to typically long settling times on processes which limits the frequency of RTO execution) [40,41]. It is also possible to apply dynamic RTO at a lower frequency than the execution of the MPC (though at a higher frequency possible with steady-state RTO). This allows for optimisation at a point in the future which is not necessarily the steady-state point. One approach is to use a reduced order dynamic model of the plant that is less sensitive to high frequency disturbances and more focussed on longer term behaviour (which is more applicable to plant economics and unit interaction) [40].

5.4.2 Implementation

The optimal steady-state targets for the MVs and the CVs as well as the ideal resting values for the MVs were calculated using a constrained nonlinear optimiser (the 'FMINCON' function in Matlab) and the nonlinear steady-state model discussed in section 5.3. The code is shown in Section A.6. The use of the nonlinear model for the RTO results in the control scheme being another variant of nonlinear MPC.

5.5 CONCLUSION

In this chapter, a brief overview of MPC was given after which three controllers were developed, starting with a base case MPC strategy that uses a constant LTI model. Thereafter, iterative linearisation was introduced that calculates the model gains at

different operating points from a nonlinear steady-state model and adjusts the model used by the controller accordingly. This results in a form of nonlinear MPC using a linear MPC algorithm. Finally, the concept of real-time optimisation was discussed that calculates the true optimal operating cost and the associated MV and CV values and writes these values as targets to the controller. This is done by using a nonlinear constrained optimiser together with the nonlinear steady-state model discussed in Section 5.3. In the next chapter a simulation study is conducted that evaluates the performance of these controllers amid several process disturbances.

CHAPTER 6 SIMULATION AND RESULTS

6.1 INTRODUCTION

The controller was tested in a simulation study (using the nonlinear model discussed in Section 3.3 to represent the plant) to demonstrate its ability to control the CVs within limits in the presence of noise (the discharge flow signal has a 2% peak to peak noise) and disturbances, while attempting to minimise operating cost. For simplicity, actuator dynamics were not considered and perfect flow manipulation assumed. Three cases were considered. The first uses only the initial LTI model and is referred to as the base case MPC. The second uses iterative linearisation to update the gains of the LTI model. The third utilises RTO in combination with iterative linearisation.

Two demand disturbances (changes in the total discharge) were introduced after 1 hours and 3 hours respectively, each of a 3 kNm^3/h magnitude. A composition disturbance was also introduced in the NG stream at time 2 hours, changing the composition from [CH_4, C2+, H_2, N_2, CO, CO_2] = [0.911, 0.068, 0.0, 0.015, 0.0, 0.006] to [0.841, 0.088, 0.01, 0.035, 0.01, 0.016]. The composition of the Tail Gas 1 stream was changed at time 4 hours from [CH_4, C2+, H_2, N_2, CO, CO_2] = [0.055, 0.01, 0.62, 0.025, 0.26, 0.03] to [0.075, 0.04, 0.57, 0.025, 0.26, 0.03]. This compositional change causes a change in the HHV of TG_1 from 13.98 to 17.98 MJ/Nm^3 which causes the gain of the model from TG_1 to HHV to change sign (from negative to positive). This illustrates the effectiveness of the iterative linearisation. When a constant LTI model is used, the controller does not detect the change in composition and continues with the same inlet flow rates (not utilising the TG_1 to reduce the cost as it could do). When the gains are updated, the controller detects that the HHV of the TG_1 stream is higher and can therefore be exchanged for some NG, reducing the cost.

6.2 BASE CASE MPC VERSUS MPC WITH ITERATIVE LINEARISATION

The results of using a constant LTI model (the model given in Table 3.3) are shown in Figure 6.1 to Figure 6.4. The results for the iterative linearisation case are shown in Figure

6.6 to Figure 6.8. The operating costs (calculated from the flow rates and normalised costs) for the two cases are shown in Figure 6.5 and Figure 6.9. The same controller was used in both cases (identical tuning settings).

The results illustrate that a seemingly small change in the composition of the feed streams can cause a significant change in the optimal operating costs. In this case, an increase in the HHV of Tail Gas 1 allows for a reduction in the NG usage which gives rise to the significant cost reduction. The total cost (the operating cost integrated over the 6 hour period) for the case of iterative linearisation is 0.2172 units whereas the cost for the constant LTI model is 0.2501 units (a 13.15% cost reduction for this simulation). Considering the composition change at 4 hours alone, the operating cost is reduced from about 0.0415 to 0.0225 units per kNm^3 (a reduction of 45.78%). The true optimal steady state costs before and after this disturbance are 0.0325 and 0.0 units respectively (calculated using a nonlinear optimiser with the steady state equations from Section 5.3 and shown as dashed lines in Figure 6.5 and Figure 6.9), with an ideal total cost of 0.1294 units. This indicates that, although the iterative linearisation improves performance, it still falls short of the true optimal solution. This is mostly due to the formulation of the optimisation problem in the MPC algorithm so as to provide adequate dynamic control. The mid-range targets on the CVs, the rate weights on the MVs, and the weights on the CVs are examples of parameters necessary for proper dynamic performance, which also have an effect on the ability to reach the theoretical optimal cost. The next section provides a solution to this problem by introducing real-time optimisation (RTO).

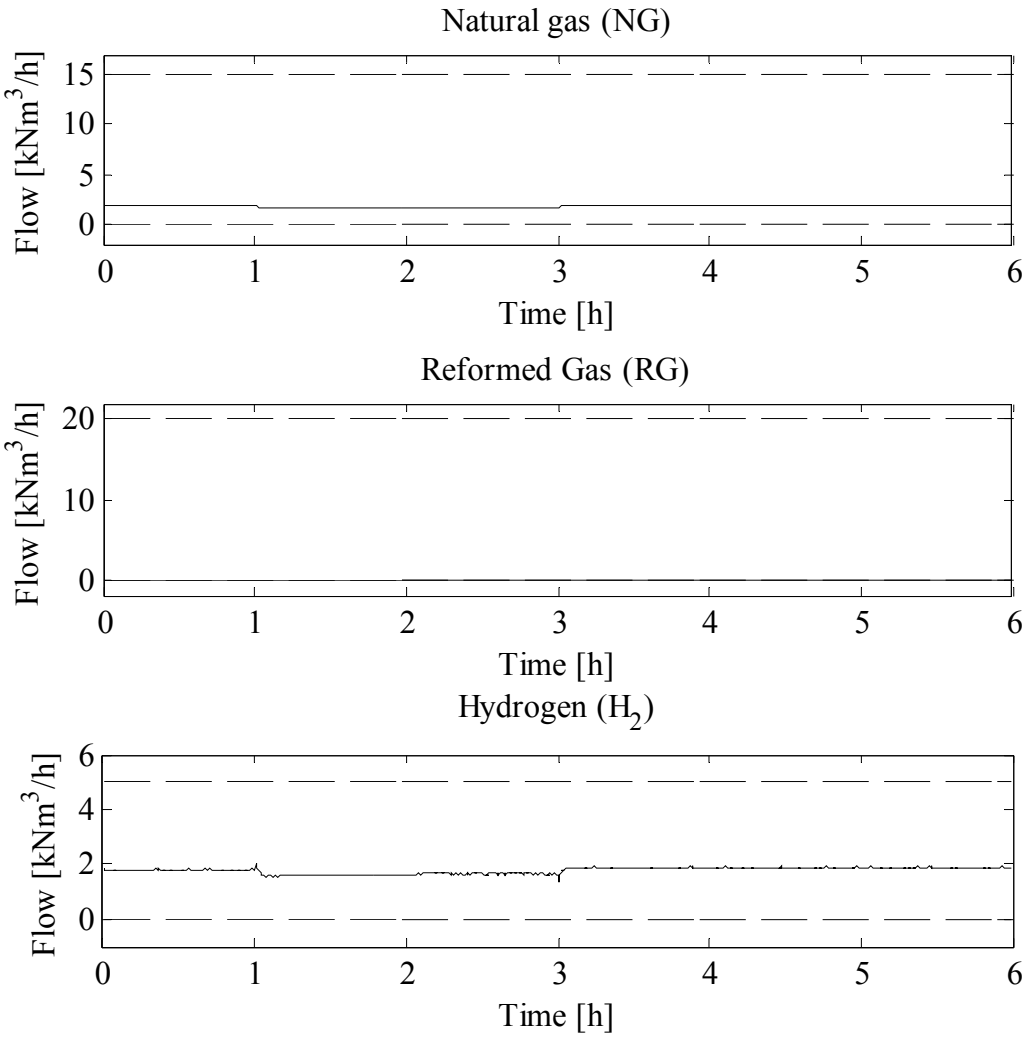

Figure 6.1. NG, RG, and H_2 flows for the constant LTI model. The solid lines are the MV values with the dashed lines indicating the limits.

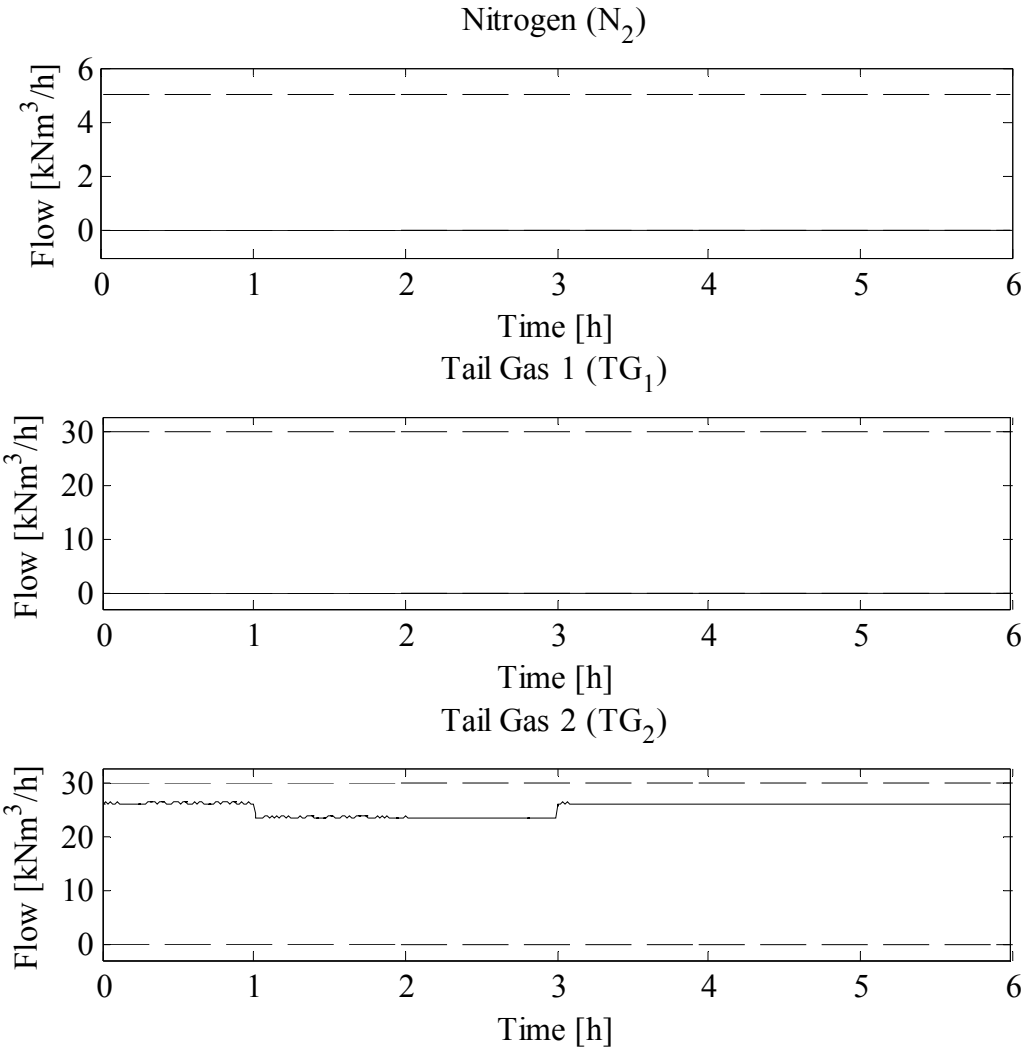

Figure 6.2. N_2, TG_1, and TG_2 flows for the constant LTI model. The solid lines are the MV values with the dashed lines indicating the limits.

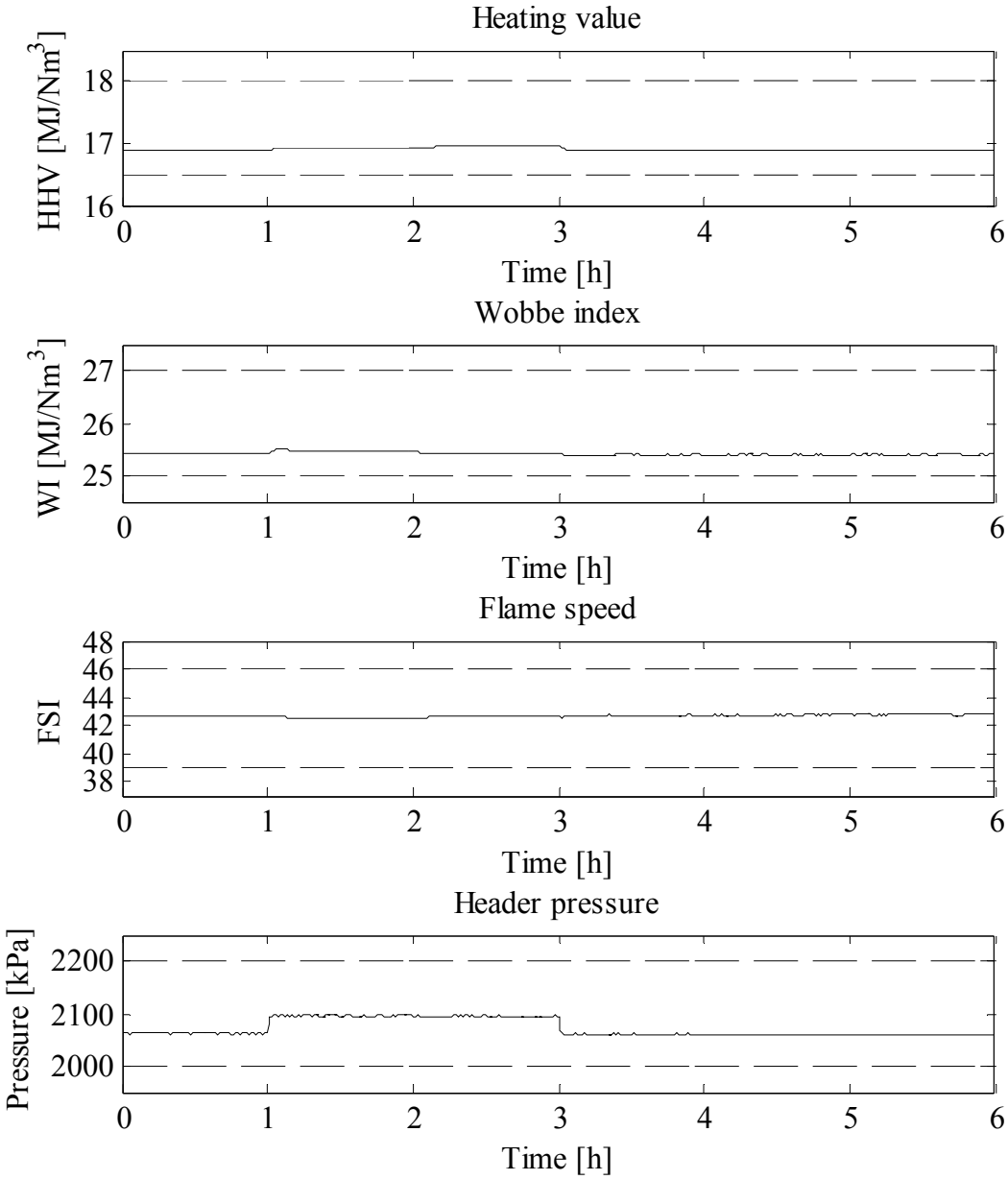

Figure 6.3. Controlled variables for the constant LTI model. The solid lines are the CV values with the dashed lines indicating the limits.

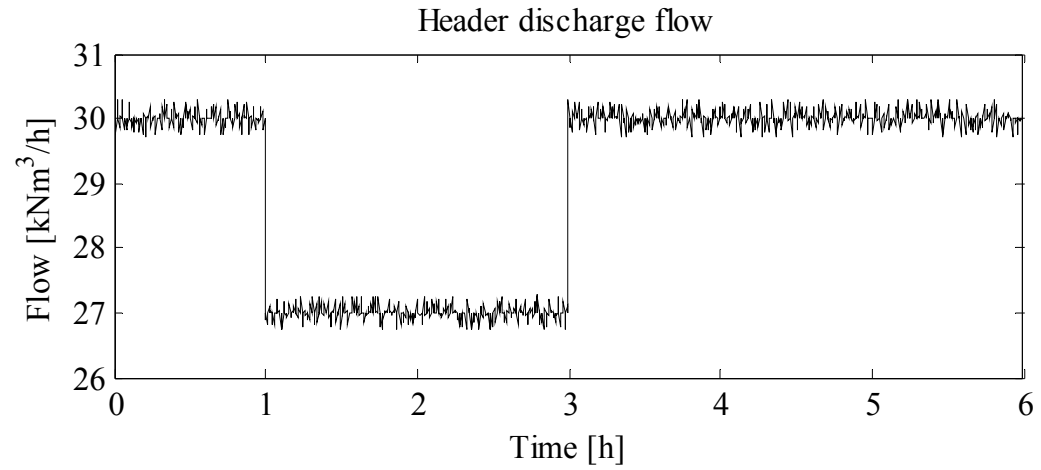

Figure 6.4. Header discharge flow.

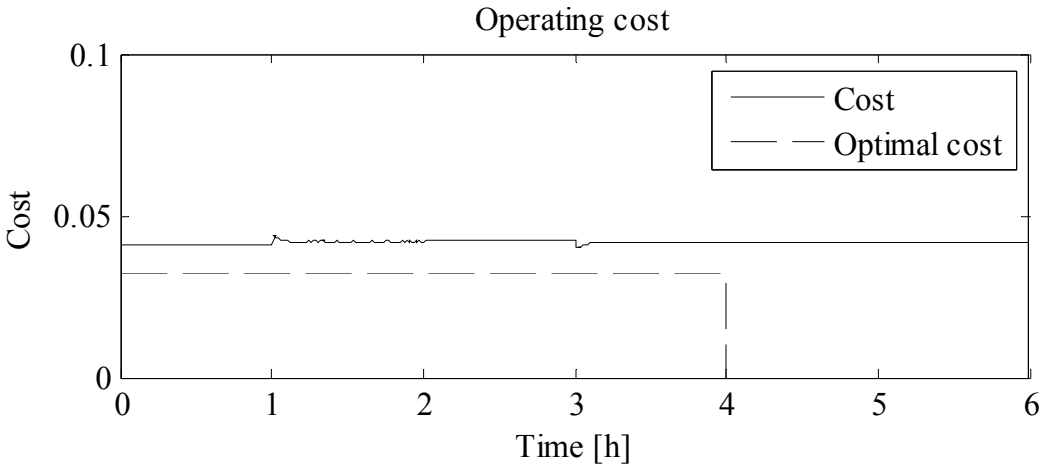

Figure 6.5. Operating cost and ideal optimal cost (dashed line) for the constant LTI model.

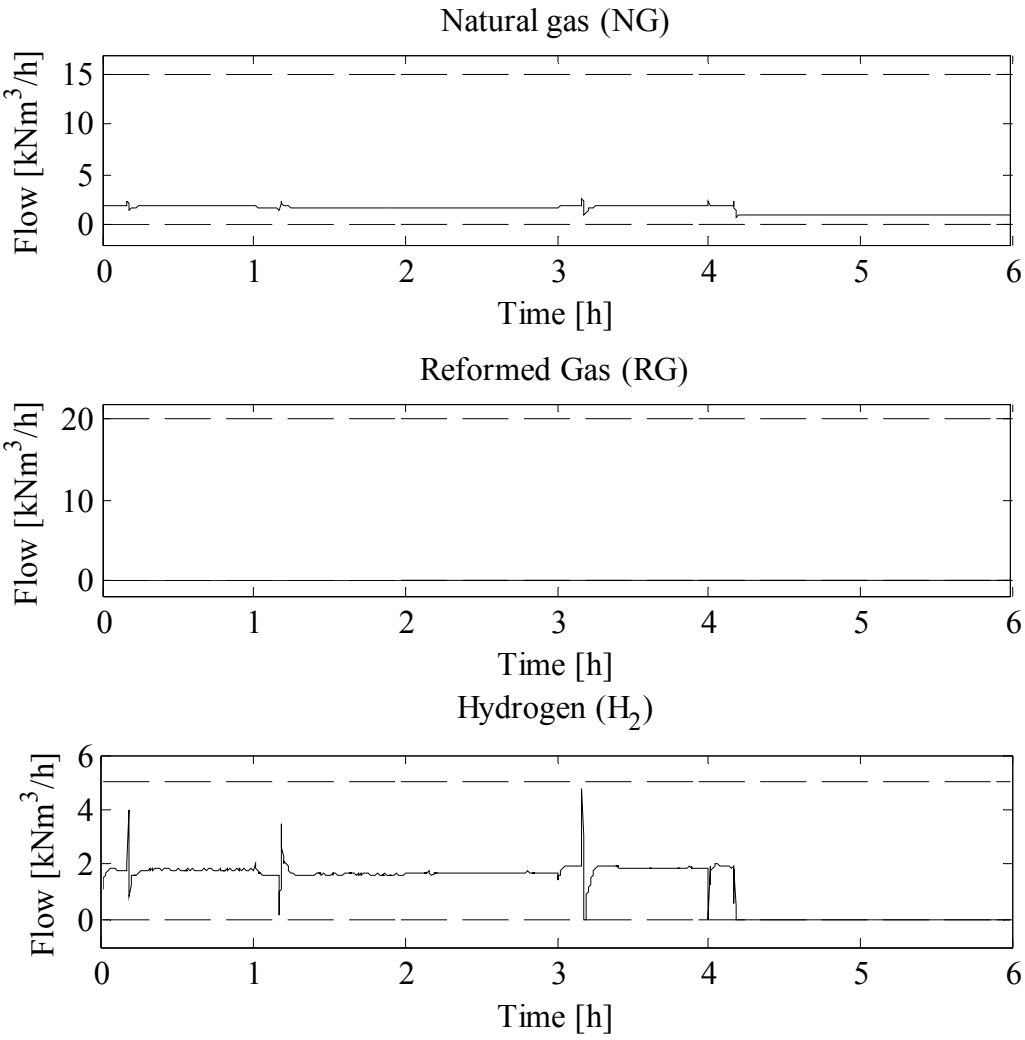

Figure 6.6. NG, RG, and H_2 flows for the iterative linearisation case. The solid lines are the MV values with the dashed lines indicating the limits.

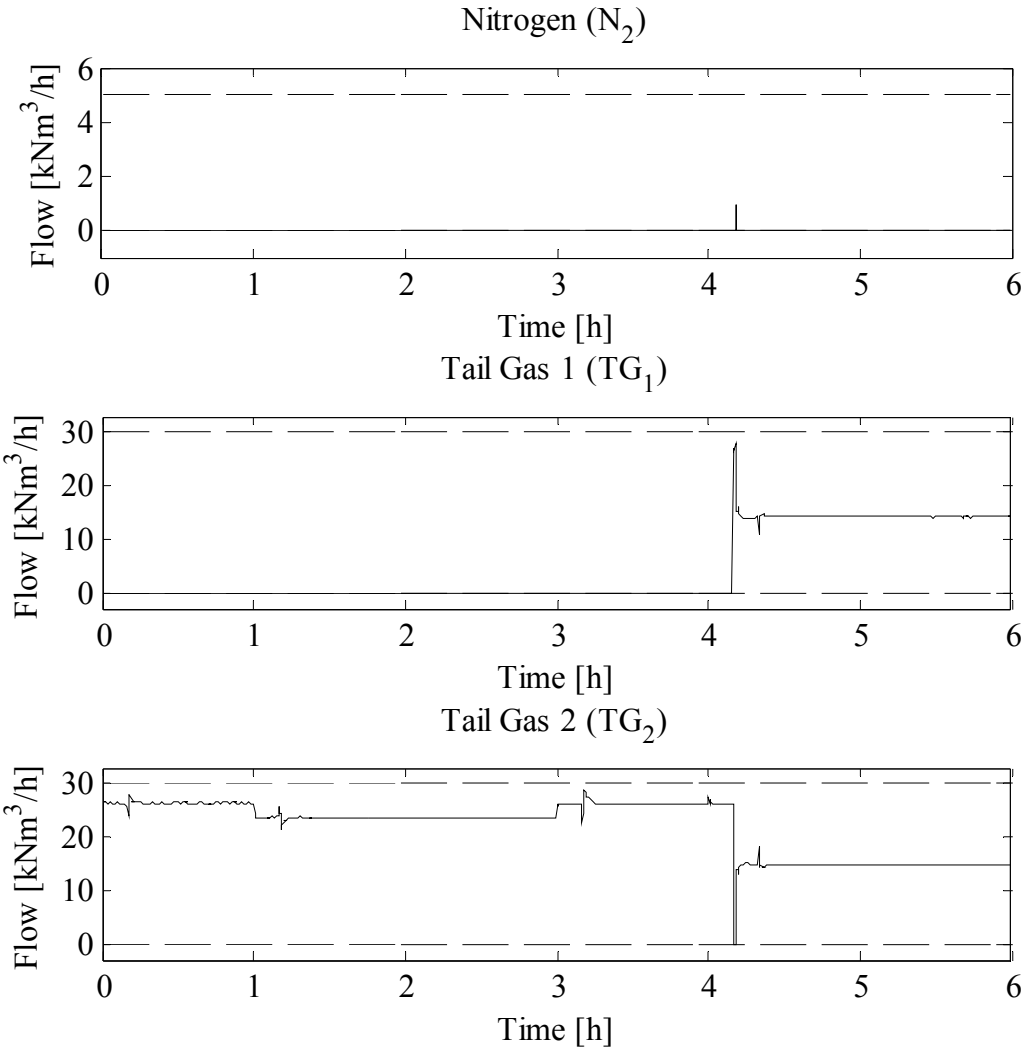

Figure 6.7. N_2, TG_1, and TG_2 flows for the iterative linearisation case. The solid lines are the MV values with the dashed lines indicating the limits.

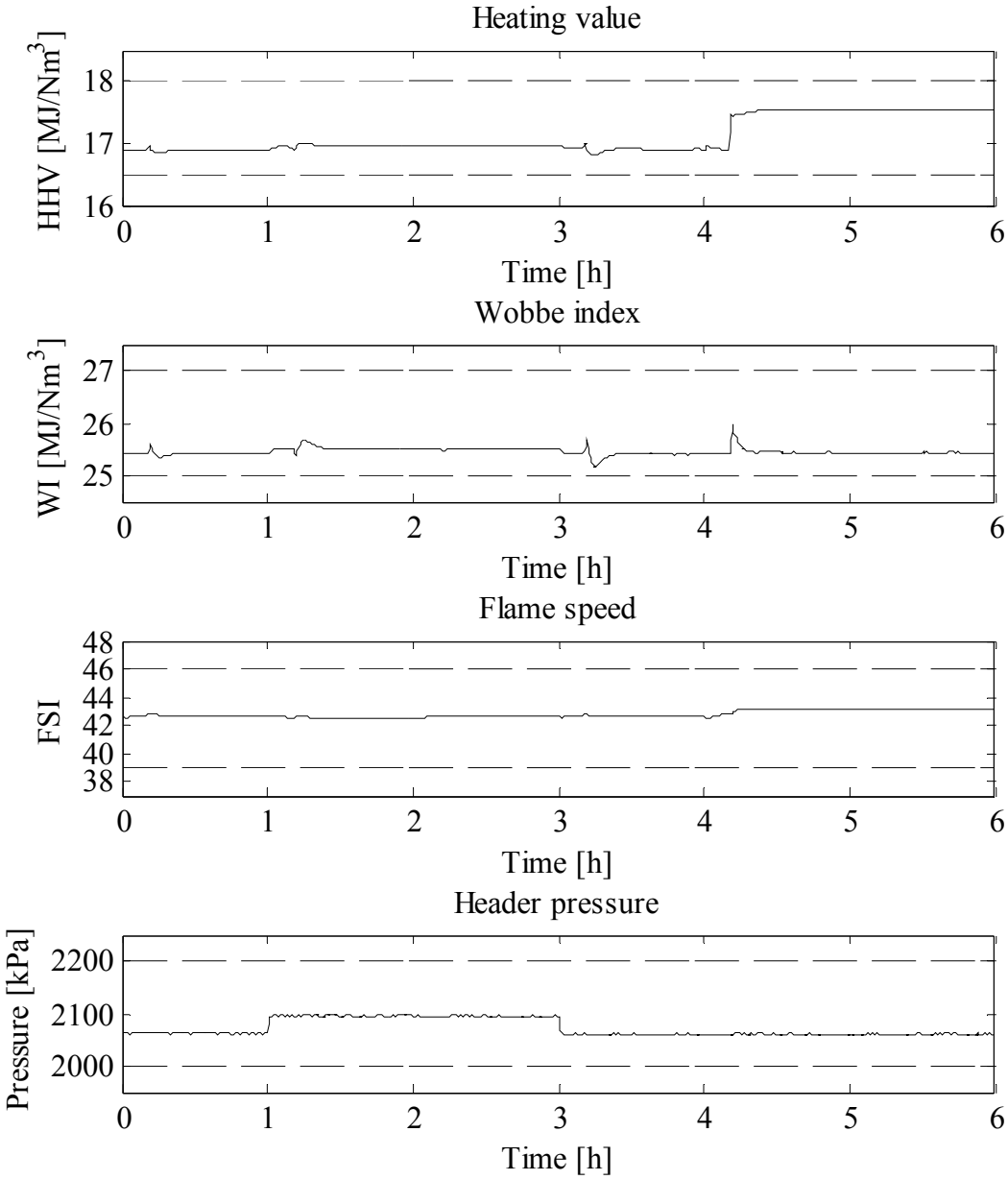

Figure 6.8. Controlled variables for the iterative linearisation case. The solid lines are the CV values with the dashed lines indicating the limits.

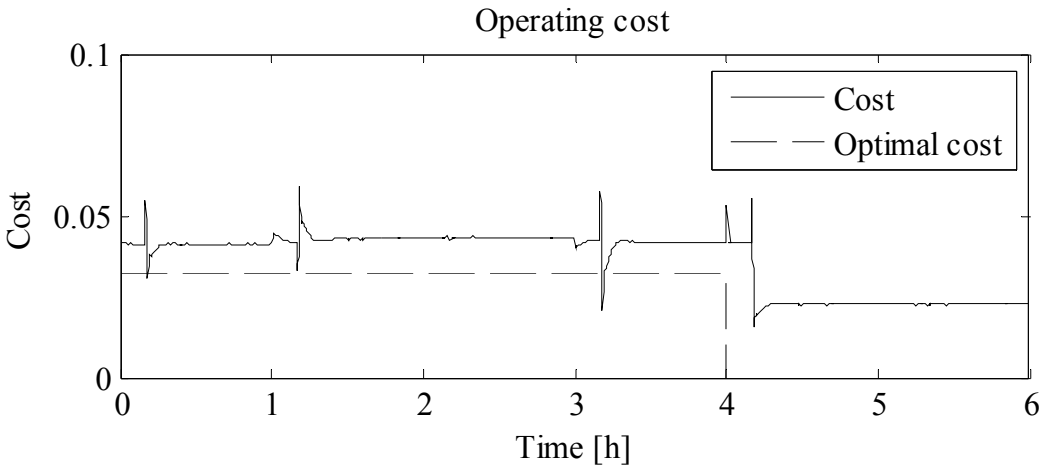

Figure 6.9. Operating cost (solid line) and ideal optimal cost (dashed line) for the iterative linearisation case.

6.3 REAL-TIME OPTIMISATION RESULTS

As discussed in Section 5.4, to get closer to the theoretical optimum, the ideal steady-state values for the MVs must be calculated and pursued by the controller. These steady-state values are calculated by means of a constrained nonlinear optimiser using the nonlinear steady-state model stated in Section 5.3 (at a rate lower than the execution of the MPC). The optimiser also provides the steady-state CV values resulting from the optimal MV values. The ideal MV values are written as ideal MV resting values (steady-state targets) to the MPC and as nominal values for the MVs. The ideal CV values become the setpoint values in the MPC (different from the mid-range values previously used).

This results in the controller approaching the true optimal values much more effectively than its previous counterpart. Figure 6.10 to Figure 6.12 show the results. The effect is a reduction in the total cost from 0.2172 to 0.1369 units (a reduction of 36.97%). As indicated in Section 6.2, the ideal optimal cost is 0.1294 units. Therefore, introducing the RTO enables operation close to optimal cost (also evident from Figure 6.13).

From Figure 6.12 it is clear that the dynamic control is still excellent with hardly any constraint violations. The RTO drives one or more of the CVs to their limits in order to reach the lower operating cost. Although a steady-state RTO approach was followed in this study, it is also possible to apply a dynamic RTO strategy (using the dynamic model discussed in Section 3.3) which will allow for a higher frequency of optimisation [40]. Table 6.1 compares the average costs per hour interval for the three cases and the optimal steady-state cost. These values are calculated by adding the cost values in each one hour interval and dividing by the number of samples in the interval.

Table 6.1: Time averaged costs per 1 hour interval for the constant LTI case, the iterative linearisation case, the RTO case, and the steady-state optimal values.

Interval	Constant LTI	Iterative linearisation	RTO	Steady-state optimal
0 – 1h	0.0411	0.0411	0.0324	0.0322
1 – 2h	0.0422	0.0430	0.0330	0.0322
2 – 3h	0.0425	0.0433	0.0327	0.0325
3 – 4h	0.0415	0.0413	0.0326	0.0325
4 – 5h	0.0415	0.0259	0.0060	0.0000
5 – 6h	0.0415	0.0225	0.0002	0.0000

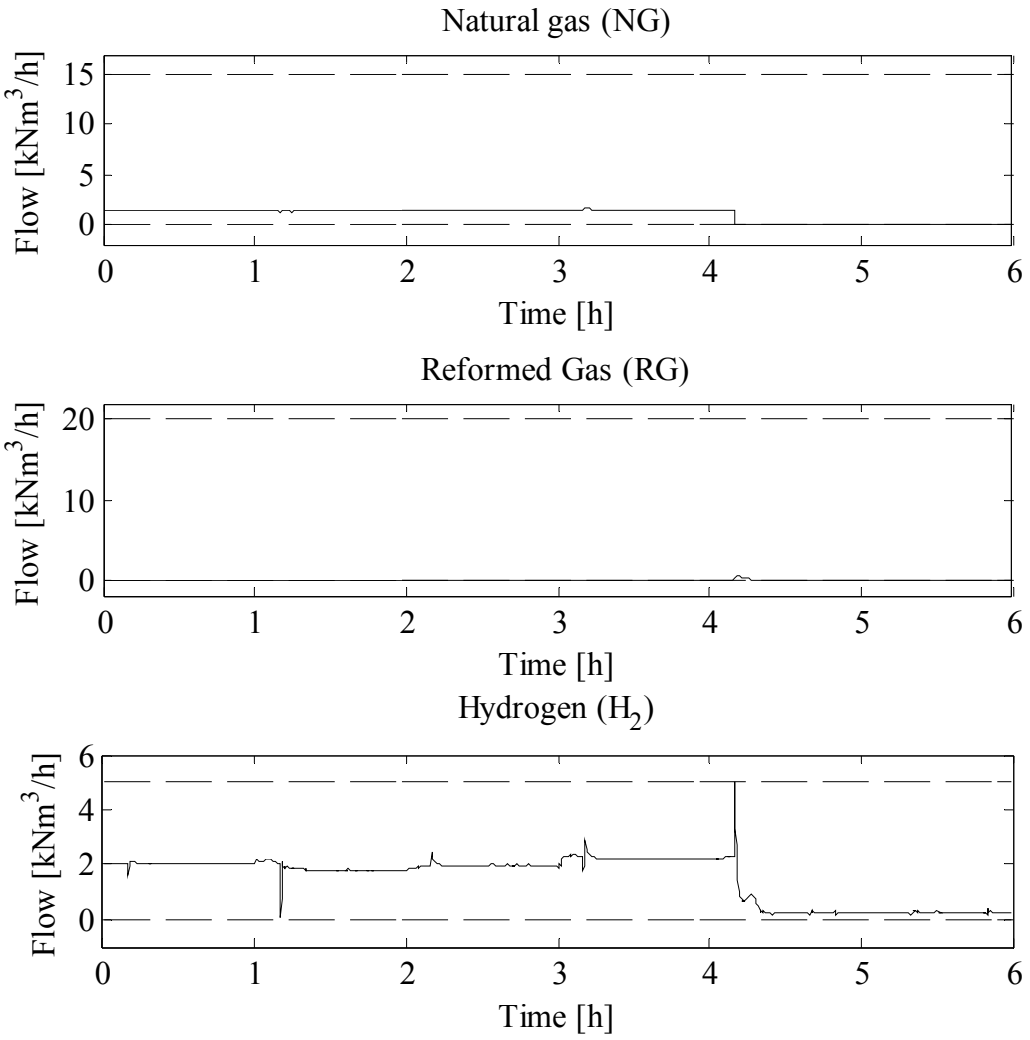

Figure 6.10. NG, RG, and H_2 flows with RTO. The solid lines are the MV values with the dashed lines indicating the limits.

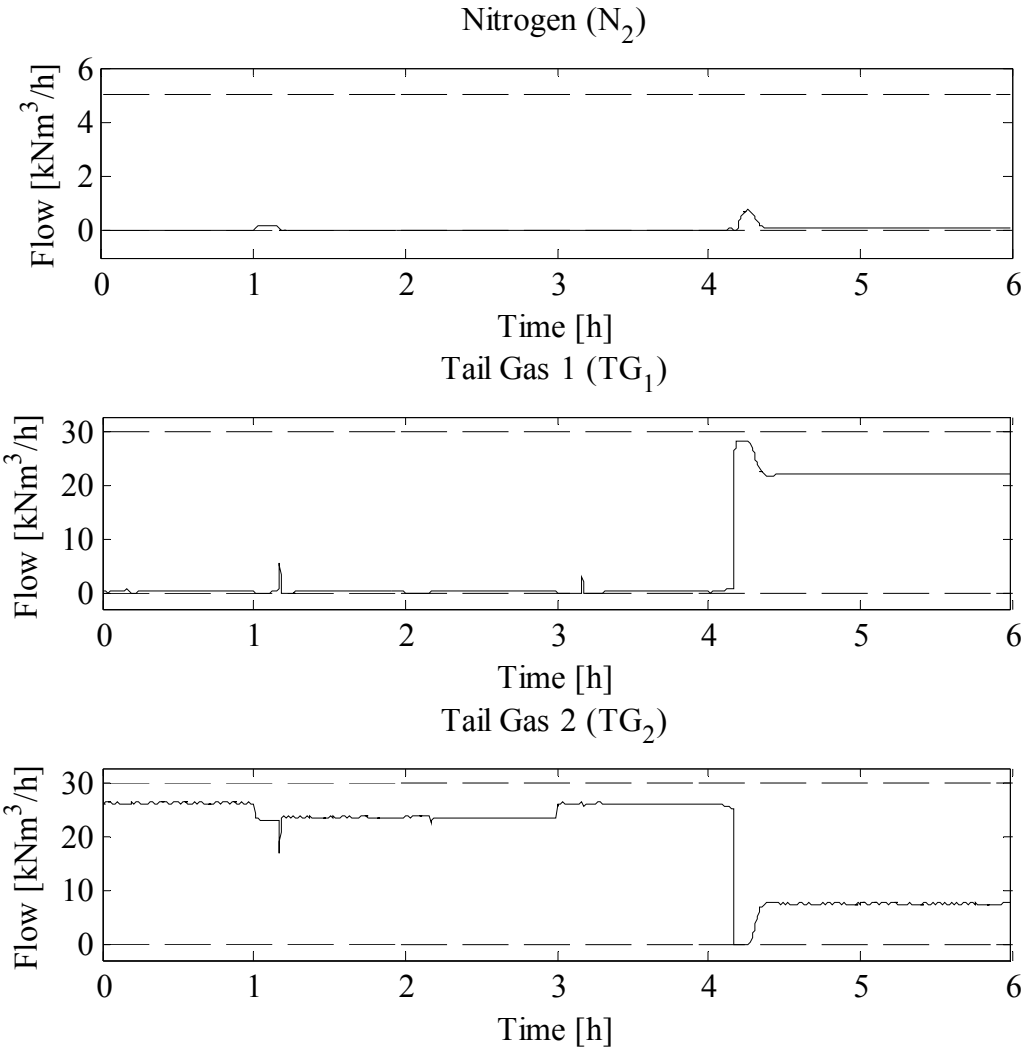

Figure 6.11. N_2, TG_1, and TG_2 flows with RTO. The solid lines are the MV values with the dashed lines indicating the limits.

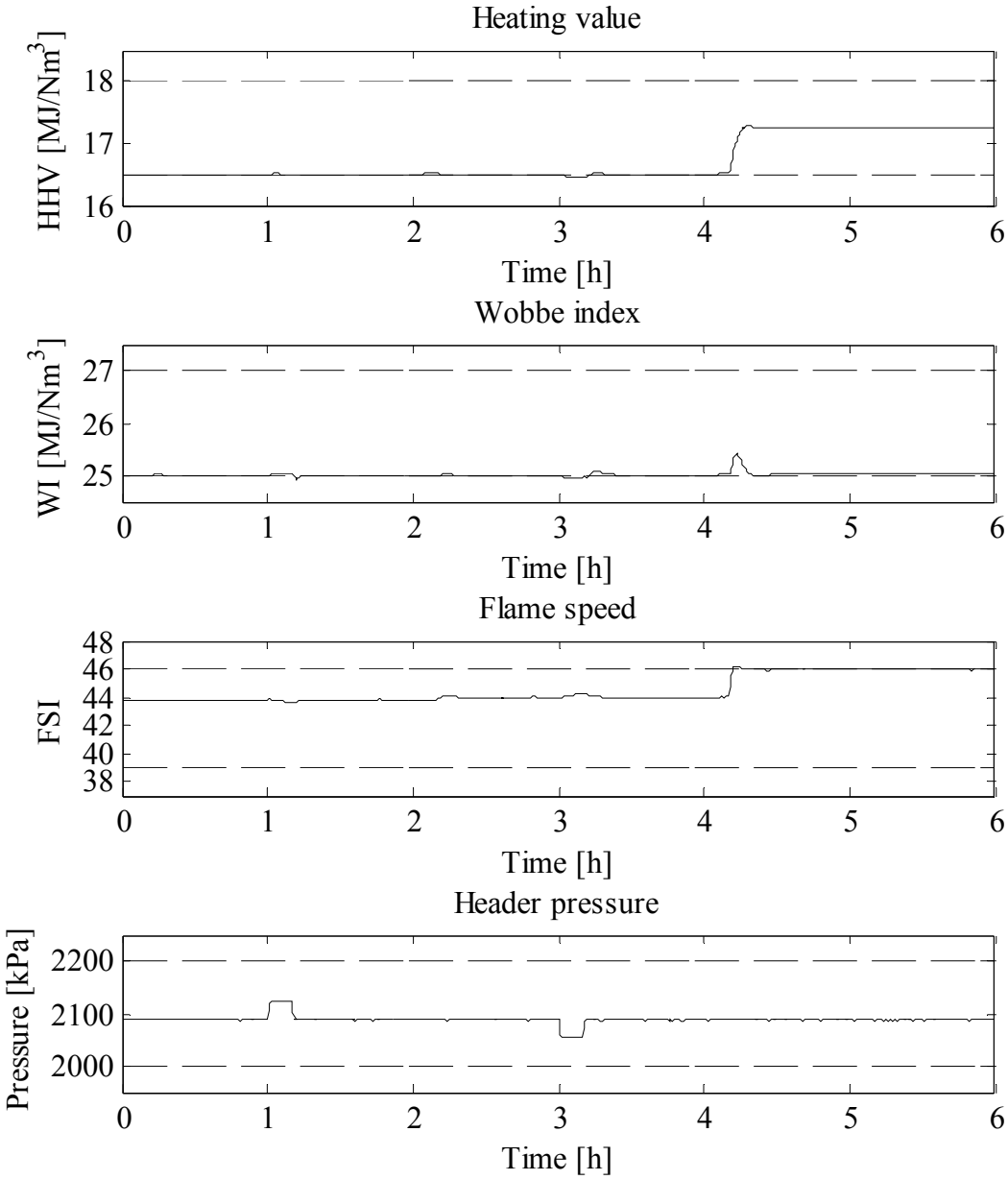

Figure 6.12. Controlled variables with RTO. The solid lines are the CV values with the dashed lines indicating the limits.

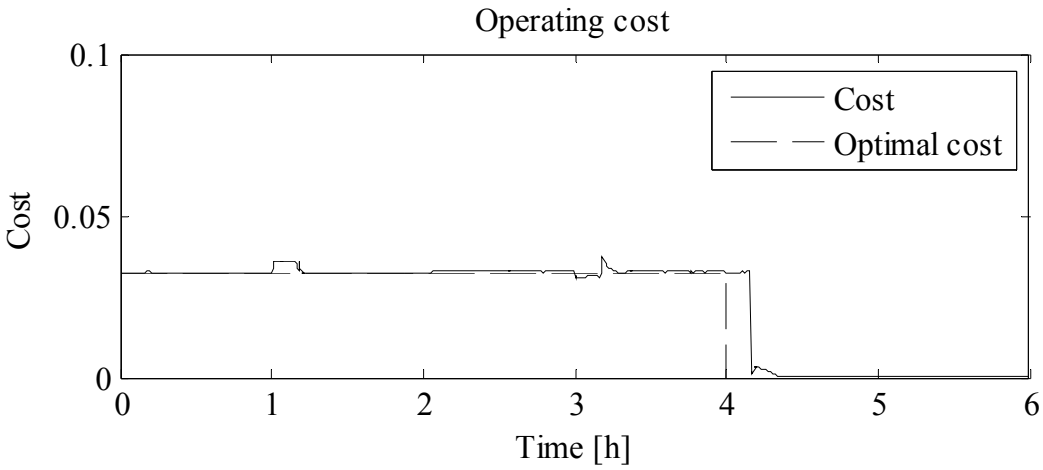

Figure 6.13. Operating cost and ideal optimal cost (dashed line) with RTO.

6.4 CONCLUSION

In this chapter, three controllers were evaluated in a simulation study to evaluate its ability to provide dynamic control during process disturbances while attempting to minimise the operating cost. The base case provided good dynamic control and prevented constraint violations but failed to adapt to feed composition changes that allow cost reduction. The introduction of iterative linearisation improved matters somewhat, allowing the controller to detect feed composition changes and reduce cost while still providing good dynamic control. Finally, the use of real-time optimisation drives the process close to the theoretical optimal cost without compromising dynamic control performance.

CHAPTER 7 CONCLUSION

7.1 RESULTS AND DISCUSSION

This dissertation described the modelling and model validation of an industrial fuel gas header as well as a simulation study of three different Model Predictive Control (MPC) strategies for controlling the system while minimising the overall operating cost.

The first principle model provides an adequate representation of the system to gain insight in the behaviour of the fuel gas blending header for simulation purposes. The correlations between the simulation and plant values indicate that the assumptions made (of ideal gas and perfect mixing) are reasonable.

Despite the nonlinear and interactive nature of the fuel gas system, MPC is very effective in controlling the outputs within the specified ranges while minimising the operating cost. The controller is able to handle significant disturbances in demand and fluctuation in feed compositions. Furthermore, the iterative linearisation allows the MPC to compensate for model changes resulting from feed flow and composition changes. Introducing RTO further increases the MPC performance, allowing the operating cost to approach the ideal optimal cost while providing adequate dynamic control.

For this process, the CVs are allowed to be at the limits considering that small, short duration violations of the constraints can be tolerated (the constraints are not hard). In processes where the constraints may not be violated at all, safety margins can be included to prevent the controller from driving the CVs right to their limits.

7.2 RECOMMENDATION FOR FUTURE WORK

In this work, three MPC algorithms were developed and tested. The first is a base case MPC that employs a constant model and constant tuning settings. The second is the base case with added iterative linearisation which allows the controller to adjust the model gains

at different operating points to compensate for the nonlinearities of the process. The final MPC is the base case with added iterative linearisation with the addition of real-time optimisation which uses a nonlinear constrained optimiser and nonlinear steady-state model to calculate the CV and MV values to drive the process to its steady-state economic optimal. The latter two approaches, although using a linear MPC algorithm, are forms of nonlinear MPC. Therefore, in future work, other forms of nonlinear MPC can be used and evaluated in an attempt to further improve performance. Other multivariable control strategies may also be evaluated.

The LTI model used in the controller uses first-order-plus-dead-time models and integrators to describe the process (see Table 3.3) which result in the fit given in Table 3.4. Other model forms may be evaluated in an attempt to achieve better fit and improve controller performance.

The simulation study uses some typical conditions and disturbances to evaluate the controller's performance. These include composition changes in the NG and TG_1 streams and a demand disturbance downstream of the header. Other scenarios can be included to further evaluate performance such as changes in the availability of one or more of the inlet streams, actuator or transmitter failures on the inlet streams, analyser failures, etc. The simulation model can be further adjusted to include the actuator and sensor dynamics to evaluate the effect of these on the controller performance.

REFERENCES

[1] D.W. Green et al., *Perry's Chemical Engineers' Handbook*, 7th ed.: McGraw-Hill, 1997.

[2] F. Johnson and D.M. Rue, "Gas Interchangeability Tests: Evaluating the Range of Interchangeability of Vaporized LNG and Natural Gas," Gas Technology Institute for Gas Research Institute, April 2003.

[3] C.J. Muller, I.K. Craig, and N.L. Ricker, "Modelling, Validation, and Control of an Industrial Fuel Gas Blending System," *Journal of Process Control*, vol. 21, no. 6, pp. 852-860, July 2011.

[4] C.J. Muller, I.K. Craig, and N.L. Ricker, "Modelling, Validation, and Control of an Industrial Fuel Gas Blending System," in *the 18th IFAC world congress (accepted for presentation)*, Milan, 2011.

[5] M. Chèbre, Y. Creff, and N. Petit, "Feedback control and optimisation for the production of commercial fuels by blending," *Journal of Process Control*, vol. 20, no. 4, pp. 441–451, April 2010.

[6] D.E. Seborg, T.F. Edgar, and D.A. Mellichamp, *Process Dynamics and Control*, 2nd ed.: Wiley, 2004.

[7] L. Ljung and T. Glad, *Modeling of Dynamic Systems*, T.Kailath, Ed. Englewood Cliffs, New Jersey: Prentice-Hall, 1994.

[8] L. Ljung, "System Identification Toolbox 7: Getting Started Guide," The MathWorks, Inc., 2010.

[9] L. Ljung, "System Identification Toolbox 7: User's Guide," The MathWorks, Inc., 2010.

[10] J.B. Rawlings, "Tutorial Overview of Model Predictive Control," *IEEE Control System Magazine*, vol. 20, no. 3, pp. 38–50, June 2000.

[11] S.J. Qin and T.A. Badgwell, "A survey of industrial model predictive control technology," *Control Engineering Practice*, vol. 11, no. 7, pp. 733–764, July 2003.

[12] P. Hughes, "Sasol Fuel Gas Optimiser: APC Controller Feasibility Study," Sasol

Technology, Sasolburg, 2008.

[13] P. Hughes, "SCI Fuel Gas Optimiser: Engineering Documentation," Sasol Technology, Sasolburg, 2010.

[14] C.R. Cutler and R.T. Perry, "Real Time Optimization with Multivariable Control is Required to Maximize Profits," *Computers and Chemical Engineering*, vol. 7, no. 5, pp. 663–667, February 1983.

[15] A. Bemporad, M. Morari, and N.L. Ricker, "Model Predictive Control Toolbox 3: User's guide," The MathWorks, Inc., 2010.

[16] I.K. Craig, Automation EBT410 SID problem set (Lecture notes), University of Pretoria, Department of Electrical, Electronic, and Computer Engineering, 2010.

[17] P. Bogacki and L.F. Shampine, "A 3(2) pair of Runge - Kutta formulas," *Applied Mathematics Letters*, vol. 2, no. 4, pp. 321–325, 1989.

[18] J. Richalet, A. Rault, J.L. Testud, and J. Papon, "Model predictive heuristic control: Applications to industrial processes," *Automatica*, vol. 14, no. 5, pp. 413–428, September 1978.

[19] C.R. Cutler and B.L. Ramaker, "Dynamic Matrix Control - A computer control algorithm," in *Procedings of the Joint Automation and Control Conference*, Houston, 1980.

[20] E.F. Camacho and C. Bordons, *Model Predictive Control*, 2nd ed. London: Springer, 2007.

[21] S. Karacan, "Application of a non-linear long range predictive control," *Chemical Engineering and Processing*, vol. 42, no. 12, pp. 943–953, December 2003.

[22] S. Grüner et al., "Nonlinear control of a reactive distillation column," *Control Engineering Practice*, vol. 11, no. 8, pp. 915–925, August 2003.

[23] R. Kawathekar and J.B. Riggs, "Nonlinear model predictive control of a reactive distillation column," *Control Engineering Practice*, vol. 15, no. 2, pp. 231–239, February 2007.

[24] R. Khaledi and B.R. Young, "Modeling and Model Predictive Control of Composition

and Conversion in an ETRE Reactive Distillation Column," *Industrial and Engineering Chemistry Research*, vol. 44, no. 9, pp. 3134–3145, March 2005.

[25] A. Rueda, S. Cristea, C. de Prada, and R. De Keyser, "Non-linear Predictive Control for a Distillation Column," in *44th IEEE Conference on Decision and Control*, 2005, pp. 5156–5161.

[26] J.B. Waller and J.M. Böling, "Multi-variable nonlinear MPC of an ill-conditioned distillation column," *Journal of Process Control*, vol. 15, no. 1, pp. 23–29, February 2005.

[27] J. Hu, K. Kumamaru, and K. Hirasawa, "A Quasi-ARMAX approach to modelling of non-linear systems," *International Journal of Control*, vol. 74, no. 18, pp. 1754–1766, December 2001.

[28] C. Venkateswarlu and A.D. Reddy, "Nonlinear Model Predictive Control of Reactive Distillation Based on Stochastic Optimization," *Industrial and Engineering Chemistry Research*, vol. 47, no. 18, pp. 6949–6960, September 2008.

[29] J. Ou and R.R. Rhinehart, "Grouped neural network model-predictive control," *Control Engineering Practice*, vol. 11, no. 7, pp. 723–732, July 2003.

[30] M. Alpbaz, S. Karacan, Y. Cabbar, and H. Hapglu, "Application of model predictive control and dynamic analysis to a pilot distillation column and experimental verification," *Chemical Engineering Journal*, vol. 88, no. 1-3, pp. 163–174, September 2002.

[31] D.E. Kirk, *Optimal Control Theory: An Introduction*. New Jersey: Prentice-Hall, 1970.

[32] R.A. Abou-Jeyab, Y.P. Gupta, J.R. Gervais, P.A. Branchi, and S.S. Woo, "Constrained multivariable control of a distillation column using a simplified model predictive control algorithm," *Journal of Process Control*, vol. 11, no. 5, pp. 509–517, October 2001.

[33] W. Wojsznis, A. Mehta, P. Wojsznis, D. Thiele, and T. Blevins, "Multi-objective optimization for model predictive control," *ISA Transactions*, vol. 46, no. 3, pp. 351–361, June 2007.

[34] F. Bezzo, F. Micheletti, R. Muradore, and M. Barolo, "Using MPC to control middle-vessel continuous distillation columns," *Journal of Process Control*, vol. 15, no. 8, pp. 925–930, December 2005.

[35] M. Diehl, H.G. Bock, and J.P. Schlöder, "A Real-Time Iteration Scheme for Nonlinear Optimization in Optimal Feedback Control," *SIAM Journal on Control and Optimization*, vol. 43, no. 5, pp. 1714–1736, March 2005.

[36] M. Diehl et al., "Real-time optimization and nonlinear model predictive control of process governed by differential-algebraic equations," *Journal of Process Control*, vol. 12, no. 4, pp. 577–585, June 2002.

[37] M. Gerdts, "Direct Shooting Method for the Numerical Solution of Higher-Index DAE Optimal Control Problems," *Journal of Optimization Theory and Applications*, vol. 117, no. 2, pp. 267–294, May 2003.

[38] V. Adetola and M. Guay, "Integration of real-time optimization and model predictive control," *Journal of Process Control*, vol. 20, no. 2, pp. 125–133, 2010.

[39] P. Hughes, Private communication, 2010.

[40] J.H. Lee, J.M. Lee, T.Tosukhowong, and J. Lu, "On Interfacing Model Predictive Controllers with a Real-Time Optimizer," *Computer Aided Chemical Engineering*, vol. 15, no. C, pp. 910–915, 2003.

[41] G. De Souza, D. Odloak, and A.C. Zanin, "Real time optimization (RTO) with model predictive control (MPC)," *Computers and Chemical Engineering*, vol. 34, no. 12, pp. 1999–2006, July 2010.

ADDENDUM A: MATLAB CODE

A.1 SYSTEM IDENTIFICATION (HEADERCONTROL.M)

```
%% ---------------------------------------------------------------------%
%                          RUN SIMULATION FOR SID
%----------------------------------------------------------------------%


    Ni_init = [16.0974    1.2514   43.6239    3.6697   11.5725    4.7168];
            % Initial number of moles of each component
        Feed = xlsread('Test\SID_flows.xls');
    P_downstream = xlsread('Test\P_downstream.xls');
    F_discharge = xlsread('Test\F_discharge_SID.xls');

    HHVi=16.5833;
    WOBBEi=ones(1,3)*25.884;
    FSi=41.1172;

    sim('Gas_Blending_SID', [0 18]) %Rub simulation for SID steps

    SIDCVs = figure;
    set(SIDCVs,'Position',[300,300,500,500])

    subplot(3,1,1);
    plot(WOBBE.time, WOBBE.signals.values,'k');
    set(get(gcf,'CurrentAxes'),'FontName','Times New
       Roman','FontSize',10);
    xlabel('Time [h]');
    ylabel('WI [MJ/Nm^3]');
    title('Wobbe Index');
    xlim([0 18]);ylim([24.5 27]);

    subplot(3,1,2);
    plot(FS.time, FS.signals.values,'k');
    set(get(gcf,'CurrentAxes'),'FontName','Times New
       Roman','FontSize',10);
    xlabel('Time [h]');
    ylabel('FSI');
    title('Flame Speed Index');
    xlim([0 18]);ylim([39 45]);

    subplot(3,1,3);
    plot(HHV.time, HHV.signals.values,'k');
    set(get(gcf,'CurrentAxes'),'FontName','Times New
       Roman','FontSize',10);
    xlabel('Time [h]');
    ylabel('HHV [MJ/Nm^3]');
    title('Higher Heating Value');
    xlim([0 18]);ylim([16.5 18.5]);

    SIDP = figure;
    set(SIDP,'Position',[300,300,500,200])
    plot(P.time, P.signals.values, 'k')
    set(get(gcf,'CurrentAxes'),'FontName','Times New
       Roman','FontSize',10);
    xlabel('Time [h]');
```

```
ylabel('P [kPa]');
title('Header Pressure');
xlim([0 18]);ylim([800 2200]);

SIDMVs1 = figure;
set(SIDMVs1,'Position',[300,300,500,500])
subplot(3,1,1);
plot(P.time, Feed(:,2),'k');
set(get(gcf,'CurrentAxes'),'FontName','Times New
   Roman','FontSize',10);
xlabel('Time [h]');
ylabel('Flow [kNm^3/h]');
title('Natural gas (NG)');
xlim([0 18]);ylim([1.5 3.5]);

subplot(3,1,2);
plot(P.time, Feed(:,3),'k');
set(get(gcf,'CurrentAxes'),'FontName','Times New
   Roman','FontSize',10);
xlabel('Time [h]');
ylabel('Flow [kNm^3/h]');
title('Reformed Gas (RG)');
xlim([0 18]);ylim([-0.5 1.5]);

subplot(3,1,3);
plot(P.time, Feed(:,4),'k');
set(get(gcf,'CurrentAxes'),'FontName','Times New
   Roman','FontSize',10);
xlabel('Time [h]');
ylabel('Flow [kNm^3/h]');
title('Hydrogen (H_2)');
xlim([0 18]);ylim([-0.5 1.5]);

SIDMVs2 = figure;
set(SIDMVs2,'Position',[300,300,500,500])
subplot(3,1,1);
plot(P.time, Feed(:,5),'k');
set(get(gcf,'CurrentAxes'),'FontName','Times New
   Roman','FontSize',10);
xlabel('Time [h]');
ylabel('Flow [kNm^3/h]');
title('Nitrogen (N_2)');
xlim([0 18]);ylim([-0.5 1.5]);

subplot(3,1,2);
plot(P.time, Feed(:,6),'k');
set(get(gcf,'CurrentAxes'),'FontName','Times New
   Roman','FontSize',10);
xlabel('Time [h]');
ylabel('Flow [kNm^3/h]');
title('Tail Gas 1 (TG_1)');
xlim([0 18]);ylim([5.5 7.5]);

subplot(3,1,3);
plot(P.time, Feed(:,7),'k');
set(get(gcf,'CurrentAxes'),'FontName','Times New Roman',
   'FontSize', 10);
xlabel('Time [h]');
ylabel('Flow [kNm^3/h]');
title('Tail Gas 2 (TG_2)');
xlim([0 18]);ylim([18.5 20.5]);
```

```
%-------------------------------------------------------------------------%
%                                   SID
%-------------------------------------------------------------------------%
%{
- Retrieve data by DAT.y, DAT.u and DAT.Ts.
- Select portions by DAT1 = DAT(1:300) etc.

- Properties can be set and retrieved by SET and GET or by subfields:
       GET(DAT,'OutputName') or DAT.OutputName
       SET(DAT,'OutputName','Current') or DAT.OutputName = {'Current'};
- Type SET(IDDATA) for a complete list of properties.
- DAT(SAMPLES,OUTPUTS,INPUTS) to select submodels/intervals
      ex. idplot(IODAT(1:540,'HHV','NG'))
- Can dtrend whole iddata object.
- Use advice(DAT) to get some advice on data.
%}

    Y = [HHV.signals.values, WOBBE.signals.values, FS.signals.values, ...
        P.signals.values];
    U = Feed(:,2:7);
    Ts = 1/180;                 % Sampling time of 20s
    IODAT = iddata(Y,U,Ts);     % Create IDDATA object
    set(IODAT, 'OutPutName', {'HHV','WOBBE','FS','P'}, 'InputName', ...
        {'NG','RG','H2','N2','T1','T2'}, 'Domain', 'Time', 'Name', ...
        'Fuel Gas', 'TimeUnit', 'Hr');
    IODATdet = dtrend(IODAT);   % Remove means


%{
- Fit 1st order plus deadtime models to plant
The reason for fitting tf models in stead of ARX models is that,
when transforming ARX models from SID objects to LTI objects
required for use in MPC, the resulting converted models are
untidy, high order models which cause problems in the MPC
functions due to poles close to the origin etc. By fitting
fixed-structure tf models, the converted LTI models are
identical in structure.
%}

    %- 1st order plus DT structures
    G1struc = idproc({'P1D','P1D','P1D','P1D','P1D','P1D'}, ...
        'Td',{'value',[1/180 1/180 1/180 1/180 1/180 1/180]},...
            'Td',{'status','fixed', 'fixed', 'fixed', 'fixed', 'fixed',
'fixed'});
    G1 = pem(IODATdet(:,1,:),G1struc);

    G2struc = idproc({'P1D','P1D','P1D','P1D','P1D','P1D'},...
        'Td',{'value',[1/60 1/60 1/60 1/60 1/60 1/60]},...
            'Td',{'status','fixed', 'fixed', 'fixed', 'fixed', 'fixed',
'fixed'});
    G2 = pem(IODATdet(:,2,:),G2struc);

    G3struc = idproc({'P1D','P1D','P1D','P1D','P1D','P1D'},...
        'Td',{'value',[1/180 1/180 1/180 1/180 1/180 1/180]},...
            'Td',{'status','fixed', 'fixed', 'fixed', 'fixed', 'fixed',
'fixed'});
    G3 = pem(IODATdet(:,3,:),G3struc);

    G4struc = idproc({'P1','P1','P1','P1','P1','P1'});
    G4 = pem(IODATdet(:,4,:),G4struc);

%- Convert SID model to LTI object for use in MPC toolbox.
```

```
%- 'min' discards the additional noise models
%  created by the SID toolbox.
    G1tf = tf(G1,'min');
    G2tf = tf(G2,'min');
    G3tf = tf(G3,'min');
    G4tf = tf(G4,'min');

    Gtf = [G1tf; G2tf; G3tf; G4tf];
```

A.2 GAIN CALCULATIONS (HEADERCONTROL.M)

```
%-------------------------------------------------------------------------%
%%                              GAIN CALCULATIONS
%-------------------------------------------------------------------------%
%{
 For gain scheduling, it will be necessary to have models of unity gain
 and multiply them with the calculated gains at each execution. To get
 a tf object to unity gain requires some work.
%}

% Load initial state values for when the simulation is set not to
% use the initial state vector xInitial.
HHVi= 16.59 ;
WOBBEi= [25.81 25.81 25.81];
FSi= 41.03;
Ni_init = [14.1355    0.9777   48.2353    3.9576    8.4169    5.2088];

% Create initial gain matrix.
Goriginal = ones(3,6);

% Normalise the transfer function matrix (TFM)
for j=1:3
for k=1:6
    [Gn, Gd] = tfdata(Gtf(j,k),'v');
    Gp = Gn(2)/Gd(2); % Calculates the ss gain.
    Gtf(j,k) = Gtf(j,k)/Gp; % Normalise.
    Goriginal(j,k)=Gp; % Save gain in initial matrix.
end
end

% Add unity gain multipliers for pressure models.
Ginitial = vertcat(Goriginal,[1 1 1 1 1 1]);
```

A.3 CONTROL AND OPTIMISATION (HEADERCONTROL.M)

```
%% -----------------------------------------------------------------------
%                                 MPC
%  -------------------------------------------------------------------------%

%% Initialise simulation
% Load the initial state vecotr which includes values for the dead-times
% and initialisation values for the MPC object.

% Simulation type   - 3 for full with SS optimisation
```

```
%                   - 2 for model update only
%                   - 1 for constant LTI model
SimType = 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Load teh initial state vector
if SimType == 3
    load xInit_better;
elseif (SimType == 2) | (SimType == 1)
    load xInit_noSSopt;
end
xInitial = xFinal;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Load the controller object and simulation data.
% The simulation data is stored in Microsoft Excel files.
load HeaderMPC_perfect_2.mat % Created with MPCtool.
NG_comp = xlsread('Test\NGcomp.xls');
RG_comp = xlsread('Test\RGcomp.xls');
TG1_comp = xlsread('Test\TG1comp.xls');
TG2_comp = xlsread('Test\TG2comp.xls');

Add noise to the discharge flow.
   F_discharge = xlsread('Test\F_discharge.xls');
    for i=1:2:length(F_discharge)
        F_discharge(i,2) = F_discharge(i,2)+(0.02*F_discharge(i,2)*...
            random('unif',0,1) - 0.01*F_discharge(i,2));
    end

% Initial values for SS optimum calculation
u0 = [1.8 0 4.2 0 0 24];
y0 = [16.59 25.81 41.03];
Fdist0 = 30;
y_NG0 = [91.1 6.8 0 1.5 0 0.6]/100;
y_NG = y_NG0;
y_RG0 = [1.5 0 62 0.5 31 5]/100;
y_RG = y_RG0;
y_TG10 = [5.5 1 62 2.5 26 3]/100;
y_TG1 = y_TG10;
y_TG20 = [15 1 57 6 13 8]/100;
y_TG2 = y_TG20;
HHV0 = [43.16 11.78 12.1 0 13.98 15.41];
Feed_HHV = HHV0;
SG0 = [0.6684 0.4345 0.07 0.9729 0.4172 0.4507];
Feed_SG = SG0;
Ptarget = 2100;
% Calculate SS optimum (true)
[OptimVal, MinCost] = GetSSoptimum(u0,y0,Fdist0,y_NG0,y_RG0,
       y_TG10,y_TG20, HHV0, SG0);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Set the initial ideal resting values for MVs.
if SimType == 3
    F_optimal = OptimVal(1:6);
    for i = 1:6
        HeaderMPC.MV(i).Target = F_optimal(i);
    end
    Targets = [OptimVal(7:9), Ptarget]; % Set ideal CV values
    HeaderMPC.Model.Nominal.U = F_optimal; % Set nominal MC values
elseif (SimType == 2)  | (SimType == 1)
    F_optimal = [0 0 5 0 30 30]; % Use if SS
    for i = 1:6
```

```
        HeaderMPC.MV(i).Target = F_optimal(i);
    end
    Targets = [17.25 26 42.5 2100];     %Use if SS
                                        % optimisation is not required
    HeaderMPC.Model.Nominal.U = [3 0 1 0 9 17];
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

HeaderMPC.Model.Nominal.Y = Targets; % Set nominal CV values

% Initialise the total vectors
HHV_tot=[];
WI_tot=[];
FSI_tot=[];
P_tot=[];
t_tot=[];
Flows_tot=[];
Cost = 0;
Optimal_Cost=[];

% Set gain matrix to initial values
Gain = Ginitial;

T = 10/60; % Time between model updates

Count = 0; % Iteration counter

%% Start simulating
for t=0:T:(6-T)

    % Normalise the transfer function matrix (TFM)
    load Gtf_perfect.mat
    for j=1:3
    for k=1:6
        [Gn, Gd] = tfdata(Gtf(j,k),'v');
        Gp = Gn(2)/Gd(2); % Calculates the ss gains
        Gtf(j,k) = Gtf(j,k)/Gp;
    end
    end
    % Write new gain values to TFM
    Gtf = Gtf.*Gain;

    %**********************************************************************
    if SimType == 3
        HeaderMPC.Model.Plant = ss(Gtf);
    elseif SimType == 2
        HeaderMPC.Model.Plant = ss(Gtf);
    else
        SimType = 1;
    end
    %**********************************************************************

    %Run simulation for one period
    sim('Gas_Blending_MPC', [t (t+T-1/180)]);

    Count = Count+1

    %Record outputs for period
    HHV_tot=vertcat(HHV_tot,HHV.signals.values);
    WI_tot=vertcat(WI_tot,WOBBE.signals.values);
    FSI_tot=vertcat(FSI_tot,FS.signals.values);
    P_tot=vertcat(P_tot,P.signals.values);
```

```
t_tot=vertcat(t_tot,P.time);
Flows_tot=vertcat(Flows_tot,Flows.signals.values);

xInitial=xFinal; %Save final state of last period as
                 initial state for next period

%- Do the actual gain calcs according to the compositional data at
%- the end of the previous execution period.

% Take the latest flow control values
Feed_flow = Flows.signals.values(size((Flows.signals.values),1),:);

% Get the latest feed stream compositions
y_NG = NG_comp(round((t+T)*180)+1,2:7)/100;
y_RG = RG_comp(round((t+T)*180)+1,2:7)/100;
y_TG1 = TG1_comp(round((t+T)*180)+1,2:7)/100;
y_TG2 = TG2_comp(round((t+T)*180)+1,2:7)/100;
y_H2 = [0 0 1 0 0 0];
y_N2 = [0 0 0 1 0 0];

% Get the latest feed stream HHV, WI, FSI, and SG values
Feed_HHV = [NG_HHV.signals.values(length(NG_HHV.signals.values)),...
            RG_HHV.signals.values(length(RG_HHV.signals.values)),...
            12.1,...
            0,...
            TG1_HHV.signals.values(length(TG1_HHV.signals.values)),...
            TG2_HHV.signals.values(length(TG2_HHV.signals.values))];
Feed_WI = [NG_WI.signals.values(length(NG_WI.signals.values)),...
            RG_WI.signals.values(length(RG_WI.signals.values)),...
            45.73,...
            0,...
            TG1_WI.signals.values(length(TG1_WI.signals.values)),...
            TG2_WI.signals.values(length(TG2_WI.signals.values))];
Feed_FSI = [NG_FSI.signals.values(length(NG_FSI.signals.values)),...
            RG_FSI.signals.values(length(RG_FSI.signals.values)),...
            100,...
            0,...
            TG1_FSI.signals.values(length(TG1_FSI.signals.values)),...
            TG2_FSI.signals.values(length(TG2_FSI.signals.values))];
 Feed_SG = [NG_SG.signals.values(length(NG_SG.signals.values)),...
            RG_SG.signals.values(length(RG_SG.signals.values)),...
            0.069,...
            0.973,...
            TG1_SG.signals.values(length(TG1_SG.signals.values)),...
            TG2_SG.signals.values(length(TG2_SG.signals.values))];

% HHV calcualtions
for i=1:6
   HHV_gains(i)=(Feed_HHV(i)/sum(Feed_flow))-...
       (dot(Feed_flow, Feed_HHV)/(sum(Feed_flow))^2);
end

% WI calculations
HHV_fg = dot(Feed_flow, Feed_HHV)/(sum(Feed_flow)); % ss HHV of FG
SG_fg = dot(Feed_flow, Feed_SG)/(sum(Feed_flow));   % ss SG of FG
WI_fg = HHV_fg/sqrt(SG_fg);
for i=1:6
    dSGdF(i)= (Feed_SG(i) - SG_fg)/sum(Feed_flow);
    WI_gains(i) = (HHV_gains(i)/sqrt(SG_fg)) - ...
        (HHV_fg/(2*(SG_fg)^1.5))*dSGdF(i);
end
```

```
    % FSI calculations
    s = [148 514.4 339 0 61 0];
    A = [9.55 31 2.39 0 2.39 0];
    for i=1:6 % Calculate the mole fractions of each comp in FG
        y_fg(i) = (Feed_flow(1)*y_NG(i) + Feed_flow(2)*y_RG(i) +...
            Feed_flow(3)*y_H2(i) + Feed_flow(4)*y_N2(i) + ...
            Feed_flow(5)*y_TG1(i) + Feed_flow(6)*y_TG2(i))/sum(Feed_flow);
    end
    FSI_fg = (dot(y_fg,s))/(dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1); % Calc ss FSI

    FSI_gains(1) = ((s(1)-
A(1)*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_NG(1)-
y_fg(1))/sum(Feed_flow))+...
        ((s(2)-A(2)*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_NG(2)-
y_fg(2))/sum(Feed_flow))+...
        ((s(3)-A(3)*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_NG(3)-
y_fg(3))/sum(Feed_flow))+...
        ((s(5)-A(5)*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_NG(5)-
y_fg(5))/sum(Feed_flow))+...
        ((-5*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_NG(4)+y_NG(6)-
(y_fg(4)+y_fg(6)))/sum(Feed_flow));

    FSI_gains(2) = ((s(1)-
A(1)*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_RG(1)-
y_fg(1))/sum(Feed_flow))+...
        ((s(2)-A(2)*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_RG(2)-
y_fg(2))/sum(Feed_flow))+...
        ((s(3)-A(3)*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_RG(3)-
y_fg(3))/sum(Feed_flow))+...
        ((s(5)-A(5)*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_RG(5)-
y_fg(5))/sum(Feed_flow))+...
        ((-5*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_RG(4)+y_RG(6)-
(y_fg(4)+y_fg(6)))/sum(Feed_flow));

    FSI_gains(3) = ((s(1)-
A(1)*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_H2(1)-
y_fg(1))/sum(Feed_flow))+...
        ((s(2)-A(2)*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_H2(2)-
y_fg(2))/sum(Feed_flow))+...
        ((s(3)-A(3)*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_H2(3)-
y_fg(3))/sum(Feed_flow))+...
        ((s(5)-A(5)*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_H2(5)-
y_fg(5))/sum(Feed_flow))+...
        ((-5*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_H2(4)+y_H2(6)-
(y_fg(4)+y_fg(6)))/sum(Feed_flow));

    FSI_gains(4) = ((s(1)-
A(1)*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_N2(1)-
y_fg(1))/sum(Feed_flow))+...
        ((s(2)-A(2)*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_N2(2)-
y_fg(2))/sum(Feed_flow))+...
        ((s(3)-A(3)*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_N2(3)-
y_fg(3))/sum(Feed_flow))+...
        ((s(5)-A(5)*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_N2(5)-
y_fg(5))/sum(Feed_flow))+...
        ((-5*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_N2(4)+y_N2(6)-
(y_fg(4)+y_fg(6)))/sum(Feed_flow));

    FSI_gains(5) = ((s(1)-
A(1)*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_TG1(1)-
y_fg(1))/sum(Feed_flow))+...
```

```
        ((s(2)-A(2)*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_TG1(2)-
y_fg(2))/sum(Feed_flow))+...
        ((s(3)-A(3)*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_TG1(3)-
y_fg(3))/sum(Feed_flow))+...
        ((s(5)-A(5)*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_TG1(5)-
y_fg(5))/sum(Feed_flow))+...
        ((-5*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_TG1(4)+y_TG1(6)-
(y_fg(4)+y_fg(6)))/sum(Feed_flow));

    FSI_gains(6) = ((s(1)-
A(1)*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_TG2(1)-
y_fg(1))/sum(Feed_flow))+...
        ((s(2)-A(2)*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_TG2(2)-
y_fg(2))/sum(Feed_flow))+...
        ((s(3)-A(3)*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_TG2(3)-
y_fg(3))/sum(Feed_flow))+...
        ((s(5)-A(5)*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_TG2(5)-
y_fg(5))/sum(Feed_flow))+...
        ((-5*FSI_fg)/((dot(y_fg,A)+5*(y_fg(4)+y_fg(6))+1)))*((y_TG2(4)+y_TG2(6)-
(y_fg(4)+y_fg(6)))/sum(Feed_flow));

    % Create the combined gain matrix
    Gain = vertcat(HHV_gains, WI_gains, FSI_gains, ones(1,6));

    % Calculate SS optimum values
    u0 = [1.8 0 4.2 0 0 24];
    y0 = [16.59 25.81 41.03];
    Fdist0 = F_discharge(round((t+T)*180),2);
    y_NG0 = y_NG;
    y_RG0 = y_RG;
    y_TG10 = y_TG1;
    y_TG20 = y_TG2;
    HHV0 = Feed_HHV;
    SG0 = Feed_SG;

    [OptimVal, MinCost] = GetSSoptimum(u0,y0,Fdist0,y_NG0,y_RG0,
       y_TG10,y_TG20, HHV0, SG0);

    Optimal_Cost=vertcat(Optimal_Cost,MinCost);

    %F_optimal = [2.5527   0.0000    5.0000    0.0    0.0000   21.7390];
    F_optimal = abs(OptimVal(1:6));

    % Write the latest SS optimal values for the MVs and CVs
    % to the MPC object.
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if SimType == 3
        Targets = [OptimVal(7:9), Ptarget];
        HeaderMPC.Model.Nominal.U = F_optimal;
        HeaderMPC.Model.Nominal.Y = Targets;
        for i = 1:6
            HeaderMPC.MV(i).Target = F_optimal(i);
        end
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

end
```

A.4 COST CALCULATIONS (HEADERCONTROL.M)

```
%% ---------------------------------------------------------------------
%                            Cost calculations
%  ---------------------------------------------------------------------%

Tot_cost=0;
% Calculate the operating cost
for k=1:length(Flows_tot)
    Cost(k) = (0.68*Flows_tot(k,1) + ...
                0.2*Flows_tot(k,2) + ...
                0.07*Flows_tot(k,4))/sum(Flows_tot(k,:));
    Tot_cost=Tot_cost + Cost(k);
end
Tot_cost=Tot_cost/180

Bench_Cost = [0 0 0 0 0 0];
% Calculate the benchmark costs per time period
for k=1:6
    for i=1:180
        Bench_Cost(k) = Bench_Cost(k) + Cost((k-1)*180+i);
    end
    Bench_Cost(k) = Bench_Cost(k)/180;
end
Bench_Cost

Tot_opt_cost = 0;
% Calculate the operating cost
for k=1:length(Optimal_Cost)
    Tot_opt_cost=Tot_opt_cost + Optimal_Cost(k);
end
Tot_opt_cost=Tot_opt_cost/6

Bench_Opt_Cost = [0 0 0 0 0 0];
% Calculate the benchmark costs per time period
for k=1:6
    for i=1:6
        Bench_Opt_Cost(k) = Bench_Opt_Cost(k) + Optimal_Cost((k-1)*6+i);
    end
    Bench_Opt_Cost(k) = Bench_Opt_Cost(k)/6;
end
Bench_Opt_Cost
```

A.5 PLOT RESULTS (HEADERCONTROL.M)

```
%% ---------------------------------------------------------------------
%                              Plot results
%  ---------------------------------------------------------------------%

Plot the results
if SimType == 3
    GraphDrive = ['C:\Documents and Settings\mullecj1\Desktop',...
        '\Meesters\Documents\Dissertation\Final results\With SS opt\'];
    fid = fopen([GraphDrive,'Cost.txt'],'wt');
        fprintf(fid,'Actual cost: %2.4f\nOptimal cost: %2.4f\n',...
            Tot_cost,Tot_opt_cost);
```

```
        fprintf(fid,'Benchmark
costs:\n%2.4f\n%2.4f\n%2.4f\n%2.4f\n%2.4f\n%2.4f\n', ...
            Bench_Cost(1),Bench_Cost(2),Bench_Cost(3),Bench_Cost(4),...
            Bench_Cost(5),Bench_Cost(6));
    fclose(fid);
elseif (SimType == 2)
    GraphDrive = ['C:\Documents and Settings\mullecj1\Desktop',...
        '\Meesters\Documents\Dissertation\Final results\Model Update\'];
    fid = fopen([GraphDrive,'Cost.txt'],'wt');
        fprintf(fid,'Actual cost: %2.4f\nOptimal cost: %2.4f\n',...
            Tot_cost,Tot_opt_cost);
        fprintf(fid,'Benchmark
costs:\n%2.4f\n%2.4f\n%2.4f\n%2.4f\n%2.4f\n%2.4f\n', ...
            Bench_Cost(1),Bench_Cost(2),Bench_Cost(3),Bench_Cost(4),...
            Bench_Cost(5),Bench_Cost(6));
    fclose(fid);
else
    GraphDrive = ['C:\Documents and Settings\mullecj1\Desktop',...
        '\Meesters\Documents\Dissertation\Final results\Constant LTI\'];
    fid = fopen([GraphDrive,'Cost.txt'],'wt');
        fprintf(fid,'Actual cost: %2.4f\nOptimal cost: %2.4f\n',...
            Tot_cost,Tot_opt_cost);
        fprintf(fid,'Benchmark
costs:\n%2.4f\n%2.4f\n%2.4f\n%2.4f\n%2.4f\n%2.4f\n', ...
            Bench_Cost(1),Bench_Cost(2),Bench_Cost(3),Bench_Cost(4),...
            Bench_Cost(5),Bench_Cost(6));
    fclose(fid);
end

CVs = figure;
set(CVs,'Position',[200,200,500,600])
subplot(4,1,1);
plot(t_tot, HHV_tot,'k','LineWidth',1);   % Plot HHV, Wobbe, and FS
set(get(gcf,'CurrentAxes'),'FontName','Times New Roman','FontSize',10);
xlabel('Time [h]');
ylabel('HHV [MJ/Nm^3]');
title('Heating value');
ylim([16 18.5]);
xlim([0 6]);
hold;
HHVConst = [];
for i = 1:length(t_tot)
    HHVConst = [HHVConst, [18; 16.5]];
end;
plot(t_tot, HHVConst, 'k--');

subplot(4,1,2);
plot(t_tot, WI_tot,'k','LineWidth',1)
set(get(gcf,'CurrentAxes'),'FontName','Times New Roman','FontSize',10);
xlabel('Time [h]');
ylabel('WI [MJ/Nm^3]');
title('Wobbe index');
ylim([24.5 27.5]);
xlim([0 6]);
hold;
WIConst = [];
for i = 1:length(t_tot)
    WIConst = [WIConst, [27; 25]];
end;
plot(t_tot, WIConst, 'k--');

subplot(4,1,3);
```

```
plot(t_tot, FSI_tot,'k','LineWidth',1);
set(get(gcf,'CurrentAxes'),'FontName','Times New Roman','FontSize',10);
ylabel('FSI');
xlabel('Time [h]');
title('Flame speed');
ylim([37 48]);
xlim([0 6]);
hold;
FSIConst = [];
for i = 1:length(t_tot)
    FSIConst = [FSIConst, [39; 46]];
end;
plot(t_tot, FSIConst, 'k--');

subplot(4,1,4);
plot(t_tot, P_tot,'k','LineWidth',1)      % Plot Pressure
set(get(gcf,'CurrentAxes'),'FontName','Times New Roman','FontSize',10);
hold;
PConst = [];
for i = 1:length(t_tot)
    PConst = [PConst, [2000; 2200]];
end;
plot(t_tot, PConst, 'k--');
ylim([1950 2250]);
xlim([0 6]);
xlabel('Time [h]');
ylabel('Pressure [kPa]');
title('Header pressure');

set(gcf,'PaperPositionMode','auto');

if SimType == 3
    saveas(CVs, [GraphDrive,'Outputs_SS.eps']);
    saveas(CVs, [GraphDrive,'Outputs_SS.emf']);
    saveas(CVs, [GraphDrive,'Outputs_SS.fig']);
elseif SimType == 2
    saveas(CVs, [GraphDrive,'Outputs.eps']);
    saveas(CVs, [GraphDrive,'Outputs.emf']);
    saveas(CVs, [GraphDrive,'Outputs.fig']);
else
    saveas(CVs, [GraphDrive,'Outputs_LTI.eps']);
    saveas(CVs, [GraphDrive,'Outputs_LTI.emf']);
    saveas(CVs, [GraphDrive,'Outputs_LTI.fig']);
end

DV = figure;
set(DV,'Position',[300,300,500,200])
plot(0:6/length(F_discharge):(6-1/length(F_discharge)),...
    F_discharge(:,2),'k');
set(get(gcf,'CurrentAxes'),'FontName','Times New Roman','FontSize',10);
ylim([26 31]);
xlim([0 6]);
xlabel('Time [h]');
ylabel('Flow [kNm^3/h]');
title('Header discharge flow');

set(gcf,'PaperPositionMode','auto');
if SimType == 3
    saveas(DV, [GraphDrive,'Fdischarge_SS.eps']);
    saveas(DV, [GraphDrive,'Fdischarge_SS.emf']);
    saveas(DV, [GraphDrive,'Fdischarge_SS.fig']);
elseif SimType == 2
```

```
    saveas(DV, [GraphDrive,'Fdischarge.eps']);
    saveas(DV, [GraphDrive,'Fdischarge.emf']);
    saveas(DV, [GraphDrive,'Fdischarge.fig']);
else
    saveas(DV, [GraphDrive,'Fdischarge_LTI.eps']);
    saveas(DV, [GraphDrive,'Fdischarge_LTI.emf']);
    saveas(DV, [GraphDrive,'Fdischarge_LTI.fig']);
end
MVs1 = figure;
set(MVs1,'Position',[300,300,500,500])
subplot(3,1,1);
plot(t_tot, Flows_tot(:,1),'k');
set(get(gcf,'CurrentAxes'),'FontName','Times New Roman','FontSize',10);
xlabel('Time [h]');
ylabel('Flow [kNm^3/h]');
title('Natural gas (NG)');
xlim([0 6]);ylim([-2 17]);
NGConst = [];
for i = 1:length(t_tot)
    NGConst = [NGConst, [0; 15]];
end;
hold;
plot(t_tot, NGConst, 'k--');

subplot(3,1,2);
plot(t_tot, Flows_tot(:,2),'k');
set(get(gcf,'CurrentAxes'),'FontName','Times New Roman','FontSize',10);
xlabel('Time [h]');
ylabel('Flow [kNm^3/h]');
title('Reformed Gas (RG)');
xlim([0 6]);ylim([-2 22]);
RGConst = [];
for i = 1:length(t_tot)
    RGConst = [RGConst, [0; 20]];
end;
hold;
plot(t_tot, RGConst, 'k--');

subplot(3,1,3);
plot(t_tot, Flows_tot(:,3),'k');
set(get(gcf,'CurrentAxes'),'FontName','Times New Roman','FontSize',10);
xlabel('Time [h]');
ylabel('Flow [kNm^3/h]');
title('Hydrogen (H_2)');
xlim([0 6]);ylim([-1 6]);
H2Const = [];
for i = 1:length(t_tot)
    H2Const = [H2Const, [0; 5]];
end;
hold;
plot(t_tot, H2Const, 'k--');

set(gcf,'PaperPositionMode','auto');
if SimType == 3
    saveas(MVs1, [GraphDrive,'Inputs1_SS.eps']);
    saveas(MVs1, [GraphDrive,'Inputs1_SS.emf']);
    saveas(MVs1, [GraphDrive,'Inputs1_SS.fig']);
elseif SimType == 2
    saveas(MVs1, [GraphDrive,'Inputs1.eps']);
    saveas(MVs1, [GraphDrive,'Inputs1.emf']);
    saveas(MVs1, [GraphDrive,'Inputs1.fig']);
else
```

```
    saveas(MVs1, [GraphDrive,'Inputs1_LTI.eps']);
    saveas(MVs1, [GraphDrive,'Inputs1_LTI.emf']);
    saveas(MVs1, [GraphDrive,'Inputs1_LTI.fig']);
end
MVs2 = figure;
set(MVs2,'Position',[300,300,500,500])
subplot(3,1,1);
plot(t_tot, Flows_tot(:,4),'k');
set(get(gcf,'CurrentAxes'),'FontName','Times New Roman','FontSize',10);
xlabel('Time [h]');
ylabel('Flow [kNm^3/h]');
title('Nitrogen (N_2)');
xlim([0 6]);ylim([-1 6]);
N2Const = [];
for i = 1:length(t_tot)
    N2Const = [N2Const, [0; 5]];
end;
hold;
plot(t_tot, N2Const, 'k--');

subplot(3,1,2);
plot(t_tot, Flows_tot(:,5),'k');
set(get(gcf,'CurrentAxes'),'FontName','Times New Roman','FontSize',10);
xlabel('Time [h]');
ylabel('Flow [kNm^3/h]');
title('Tail Gas 1 (TG_1)');
xlim([0 6]);ylim([-3 33]);
TG1Const = [];
for i = 1:length(t_tot)
    TG1Const = [TG1Const, [0; 30]];
end;
hold;
plot(t_tot, TG1Const, 'k--');

subplot(3,1,3);
plot(t_tot, Flows_tot(:,6),'k');
set(get(gcf,'CurrentAxes'),'FontName','Times New Roman','FontSize',10);
xlabel('Time [h]');
ylabel('Flow [kNm^3/h]');
title('Tail Gas 2 (TG_2)');
xlim([0 6]);ylim([-3 33]);
TG2Const = [];
for i = 1:length(t_tot)
    TG2Const = [TG2Const, [0; 30]];
end;
hold;
plot(t_tot, TG2Const, 'k--');

set(gcf,'PaperPositionMode','auto');
if SimType == 3
    saveas(MVs2, [GraphDrive,'Inputs2_SS.eps']);
    saveas(MVs2, [GraphDrive,'Inputs2_SS.emf']);
    saveas(MVs2, [GraphDrive,'Inputs2_SS.fig']);
elseif SimType == 2
    saveas(MVs2, [GraphDrive,'Inputs2.eps']);
    saveas(MVs2, [GraphDrive,'Inputs2.emf']);
    saveas(MVs2, [GraphDrive,'Inputs2.fig']);
else
    saveas(MVs2, [GraphDrive,'Inputs2_LTI.eps']);
    saveas(MVs2, [GraphDrive,'Inputs2_LTI.emf']);
    saveas(MVs2, [GraphDrive,'Inputs2_LTI.fig']);
end
```

```
Cplot = figure;
set(Cplot,'Position',[300,300,500,200])
plot(t_tot, Cost,'k');
set(get(gcf,'CurrentAxes'),'FontName','Times New Roman','FontSize',10);
ylim([0 0.1]);
xlim([0 6]);
xlabel('Time [h]');
ylabel('Cost');
title('Operating cost');
hold;
stairs(0:1/6:6-1/6,Optimal_Cost,'k--');
legend('Cost','Optimal cost');

set(gcf,'PaperPositionMode','auto');
if SimType == 3
    saveas(Cplot, [GraphDrive,'Cost_SS.eps']);
    saveas(Cplot, [GraphDrive,'Cost_SS.emf']);
    saveas(Cplot, [GraphDrive,'Cost_SS.fig']);
elseif SimType == 2
    saveas(Cplot, [GraphDrive,'Cost.eps']);
    saveas(Cplot, [GraphDrive,'Cost.emf']);
    saveas(Cplot, [GraphDrive,'Cost.fig']);
else
    saveas(Cplot, [GraphDrive,'Cost_LTI.eps']);
    saveas(Cplot, [GraphDrive,'Cost_LTI.emf']);
    saveas(Cplot, [GraphDrive,'Cost_LTI.fig']);
end
```

A.6 CALCULATE STEADY-STATE OPTIMUM (GETSSOPT.M)

```
function [OptimVal, MinCost] = ...
    GetSSoptimum(u0,y0,Fdis0,y_NG0,y_RG0,y_TG10,y_TG20, HHV0, SG0)

lb = [0 0 0 0 0 0 16.5 25 39 0];
ub = [15 20 5 5 30 30 18 27 46 50];

%Opts = optimset('fmincon');
Opts.Display = 'none';
Opts.Algorithm = 'active-set';
x0 = [u0, y0, Fdis0];
[x,fval] = fmincon(@OptimalCost,x0,[],[],[],[],lb,ub,...
    @ (x) myconst(x,Fdis0,y_NG0,y_RG0,y_TG10,y_TG20, HHV0, SG0), Opts);
OptimVal = abs(x);
MinCost = fval;
end

function OptCost = OptimalCost(x)
     OptCost = (0.68*x(1) + ...
                0.2*x(2) + ...
                0.07*x(4))/sum(x(1:6));
end

function [c,ceq] = myconst(x,Fdis0,y_NG0,y_RG0,y_TG10,y_TG20, HHV0, SG0)

    s = [148 514.4 339 0 61 0];
    A = [9.55 31 2.39 0 2.39 0];
```

```
    y_RG = y_RG0;
    y_NG = y_NG0;
    y_TG1 = y_TG10;
    y_TG2 = y_TG20;
    y_N2 = [0 0 0 1 0 0];
    y_H2 = [0 0 1 0 0 0];

    Feed_flow = x(1:6);
    Feed_HHV = HHV0;
    Feed_SG = SG0;

    % Write the outputs as equality constraints
    ceq7 = dot(Feed_flow, Feed_HHV)/(sum(Feed_flow)) - x(7); % ss HHV of FG
    HHV_fg = dot(Feed_flow, Feed_HHV)/(sum(Feed_flow));
    SG_fg = dot(Feed_flow, Feed_SG)/(sum(Feed_flow));   % ss SG of FG
    ceq8 = HHV_fg/sqrt(SG_fg) - x(8);

    for i=1:6 % Calculate the mole fractions of each comp in FG
        y_fg(i) = (Feed_flow(1)*y_NG(i) + Feed_flow(2)*y_RG(i) +...
            Feed_flow(3)*y_H2(i) + Feed_flow(4)*y_N2(i) + ...
            Feed_flow(5)*y_TG1(i) + Feed_flow(6)*y_TG2(i))/sum(Feed_flow);
    end
    ceq9 = (dot(y_fg,s))/(dot(y_fg,A)+5*...
        (y_fg(4)+y_fg(6))+1) - x(9); % Calc ss FSI
    ceq10 = sum(Feed_flow) - Fdis0;

    c = zeros(9);
    ceq = [0
           0
           0
           0
           0
           0
           ceq7
           ceq8
           ceq9
           ceq10];

end
```

A.7 PLOT VALIDATION RESULTS (VALIDATIONDATA.M)

```
valid = xlsread('Validation.xls');

% Plot HHV plots
HHVplot = figure;
plot(0:1/180:18, valid(:,2), 'k--', 'LineWidth', 0.5);
hold;
plot(0:1/180:18, valid(:,3), 'k-', 'LineWidth', 2);
set(HHVplot,'Position',[200,200,500,250])
set(get(gcf,'CurrentAxes'),'FontName','Times New Roman','FontSize',10);
title('HHV analyser data vs. simulation data');
xlabel('Time [h]');
ylabel('Higher Heating Value [MJ/Nm^3]');
legend('HHV_{Analyser}', 'HHV_{Sim}');
```

```
% Plot WI data
WIplot = figure;
plot(0:1/180:18, valid(:,4), 'k--', 'LineWidth', 0.5);
hold;
plot(0:1/180:18, valid(:,5), 'k-', 'LineWidth', 2);
set(WIplot,'Position',[200,200,500,250])
set(get(gcf,'CurrentAxes'),'FontName','Times New Roman','FontSize',10);
title('WI analyser data vs. simulation data');
xlabel('Time [h]');
ylabel('Wobbe Index [MJ/Nm^3]');
legend('WI_{Analyser}', 'WI_{Sim}');

FSIplot = figure;
plot(0:1/180:18, valid(:,6), 'k--', 'LineWidth', 0.5);
hold;
plot(0:1/180:18, valid(:,7), 'k-', 'LineWidth', 2);
set(FSIplot,'Position',[300,300,500,250])
set(get(gcf,'CurrentAxes'),'FontName','Times New Roman','FontSize',10);
title('FSI analyser data vs. simulation data');
xlabel('Time [h]');
ylabel('Flame Speed Index');
legend('FSI_{Analyser}', 'FSI_{Sim}');
ylim([40 45]);
```

ADDENDUM B: MATLAB SCREENSHOTS

B.1 SIMULINK ENVIRONMENT

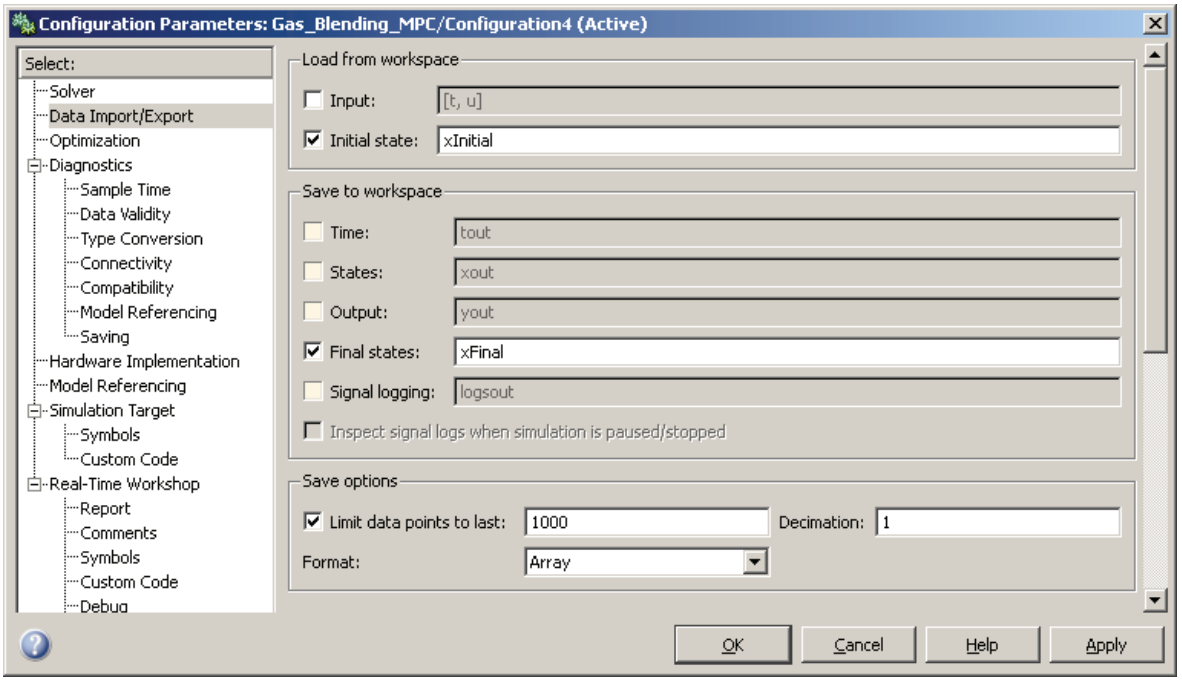

Figure B.1. Configuration settings for initial and final state vectors from and to the Matlab workspace.

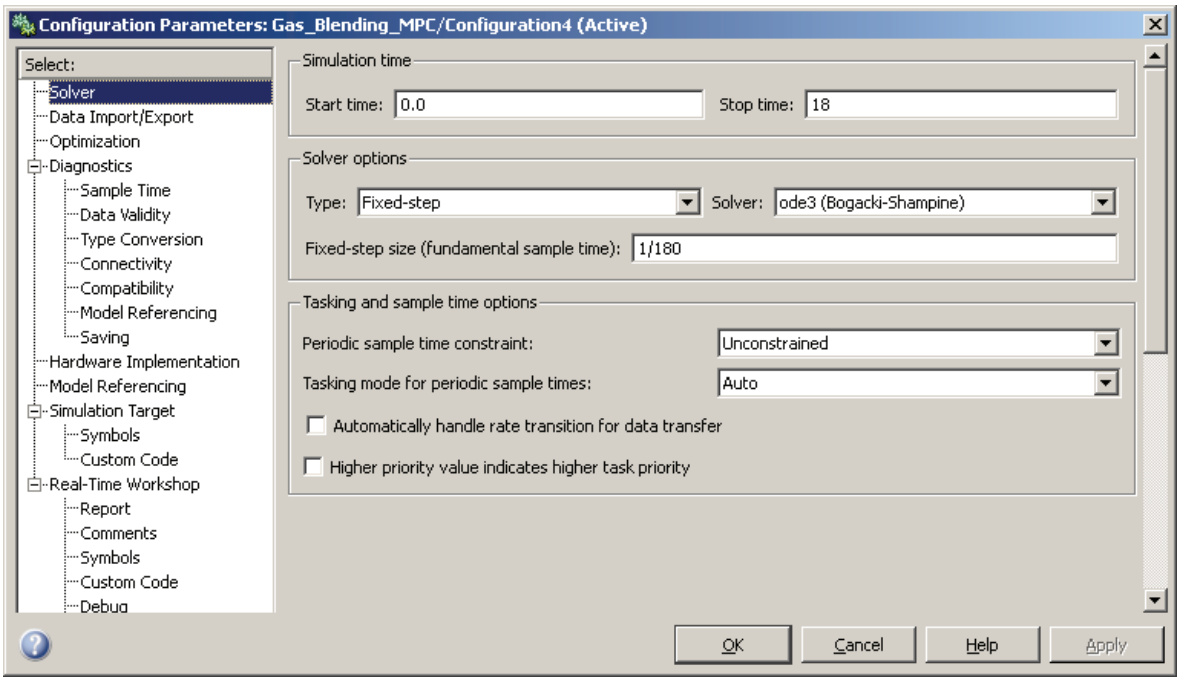

Figure B.2. Setting of the solver parameters in Simulink.

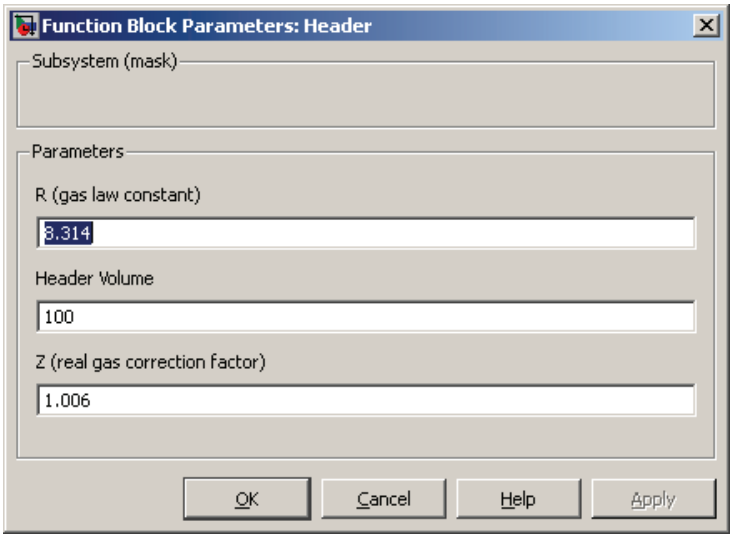

Figure B.3. Subsystem 2 parameters.

B.2 MPC TOOLBOX GUI

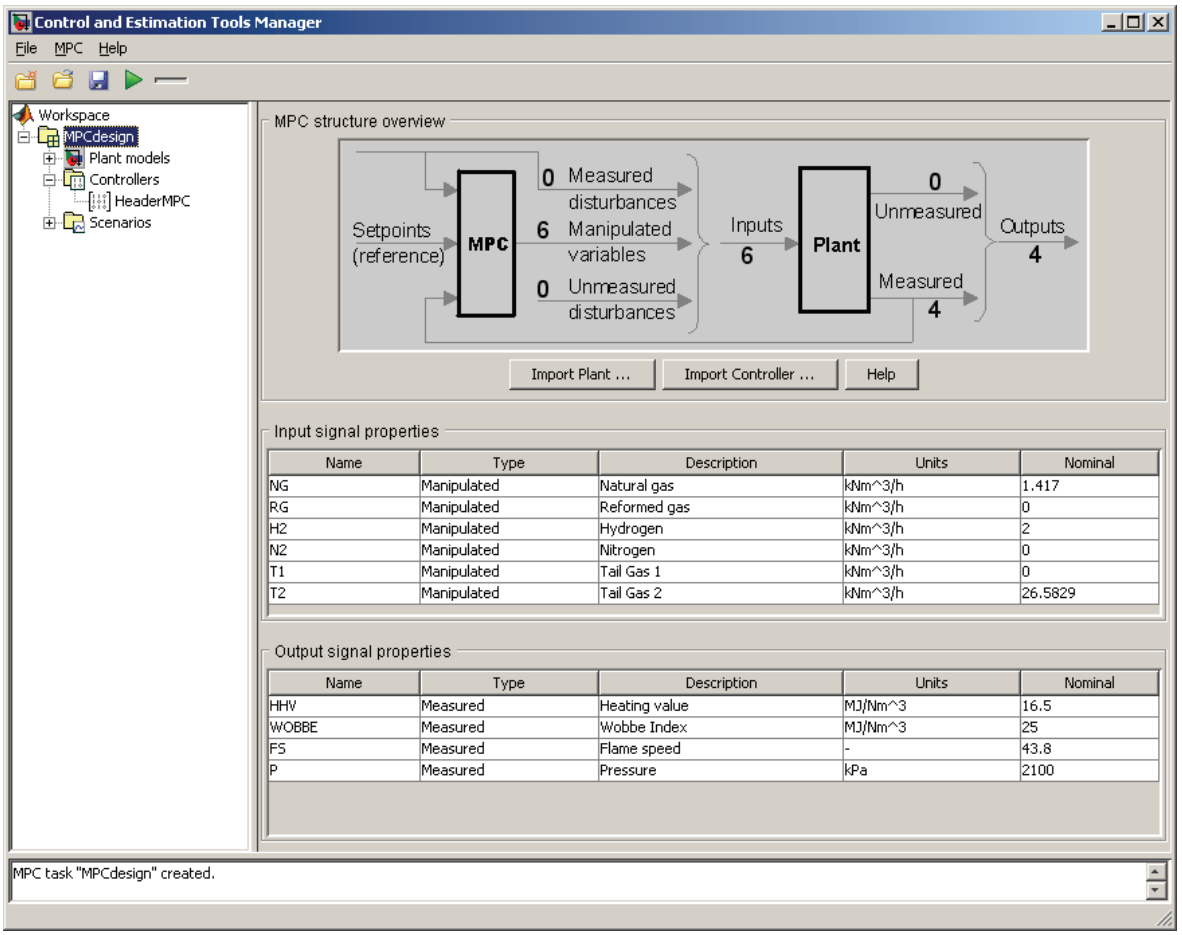

Figure B.4. Declaration of MVs and CVs in the MPC Toolbox utility.

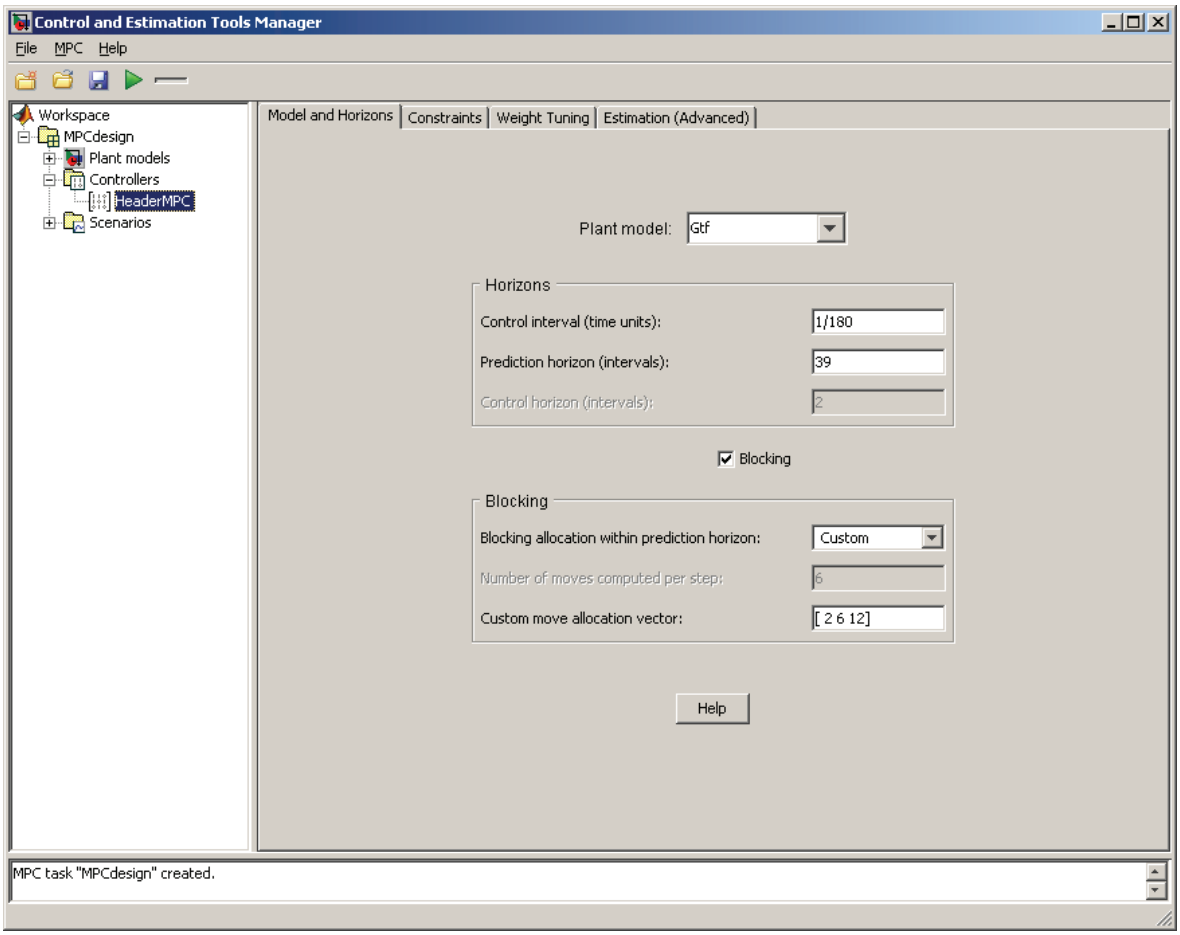

Figure B.5. Control and prediction horizon settings (including blocking).

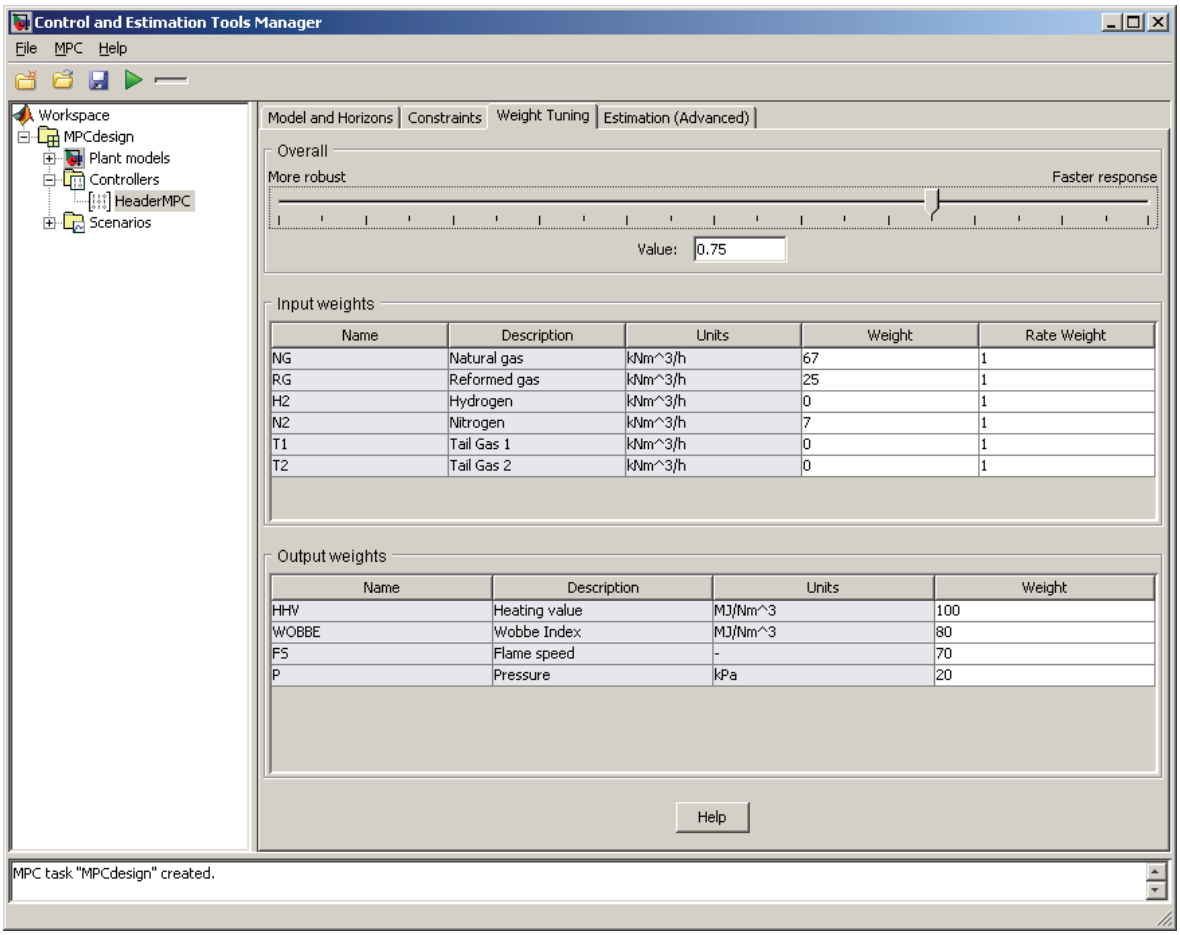

Figure B.6. Weights on MVs and CVs as configured in the MPC utility.

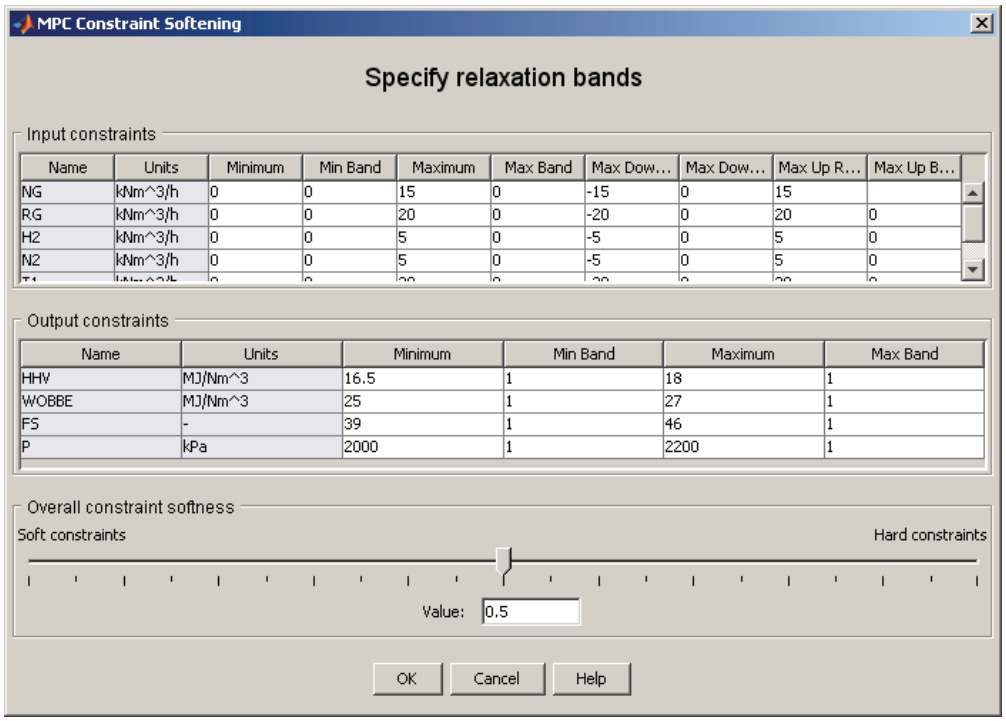

Figure B.7. Configuration of constraint softening.

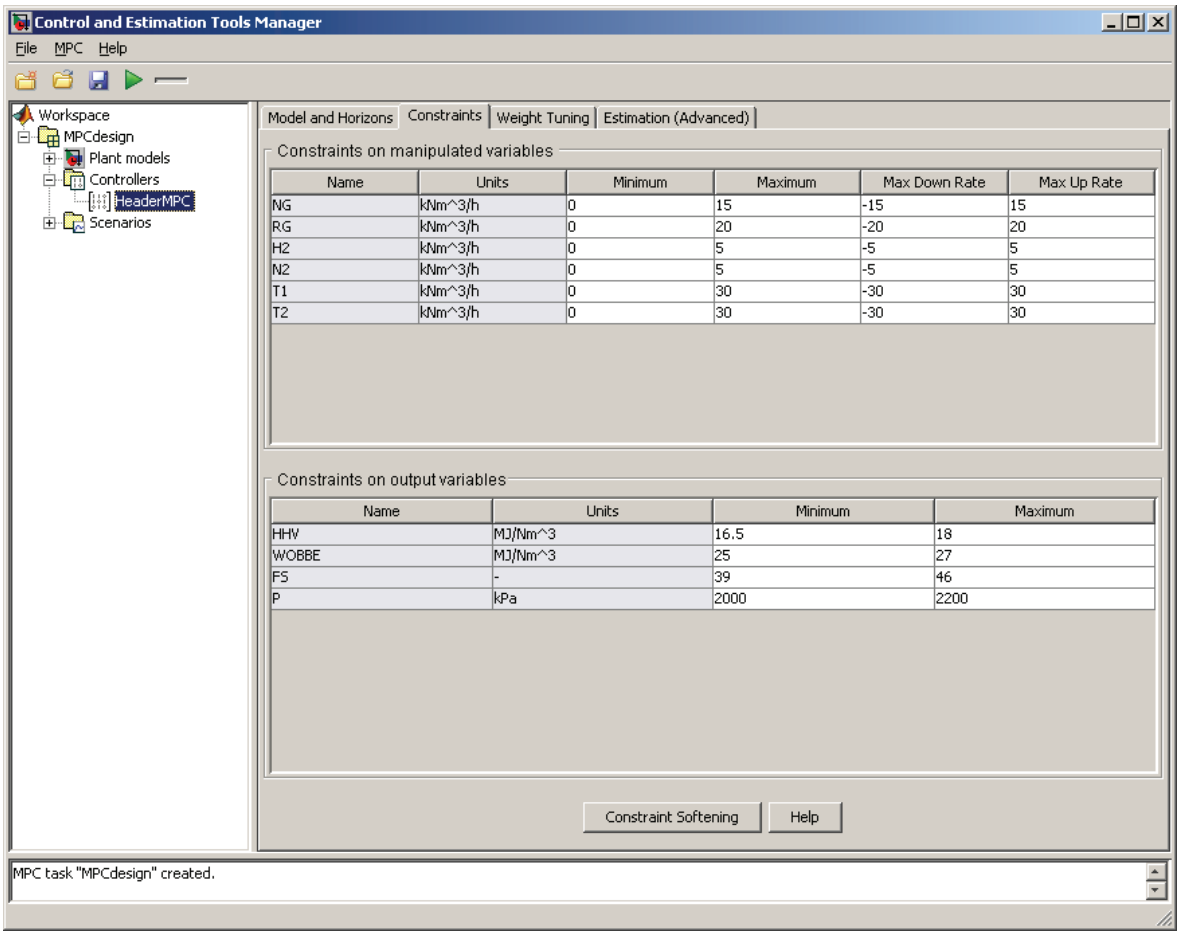

Figure B.8. Configuration of MV and CV constraints.

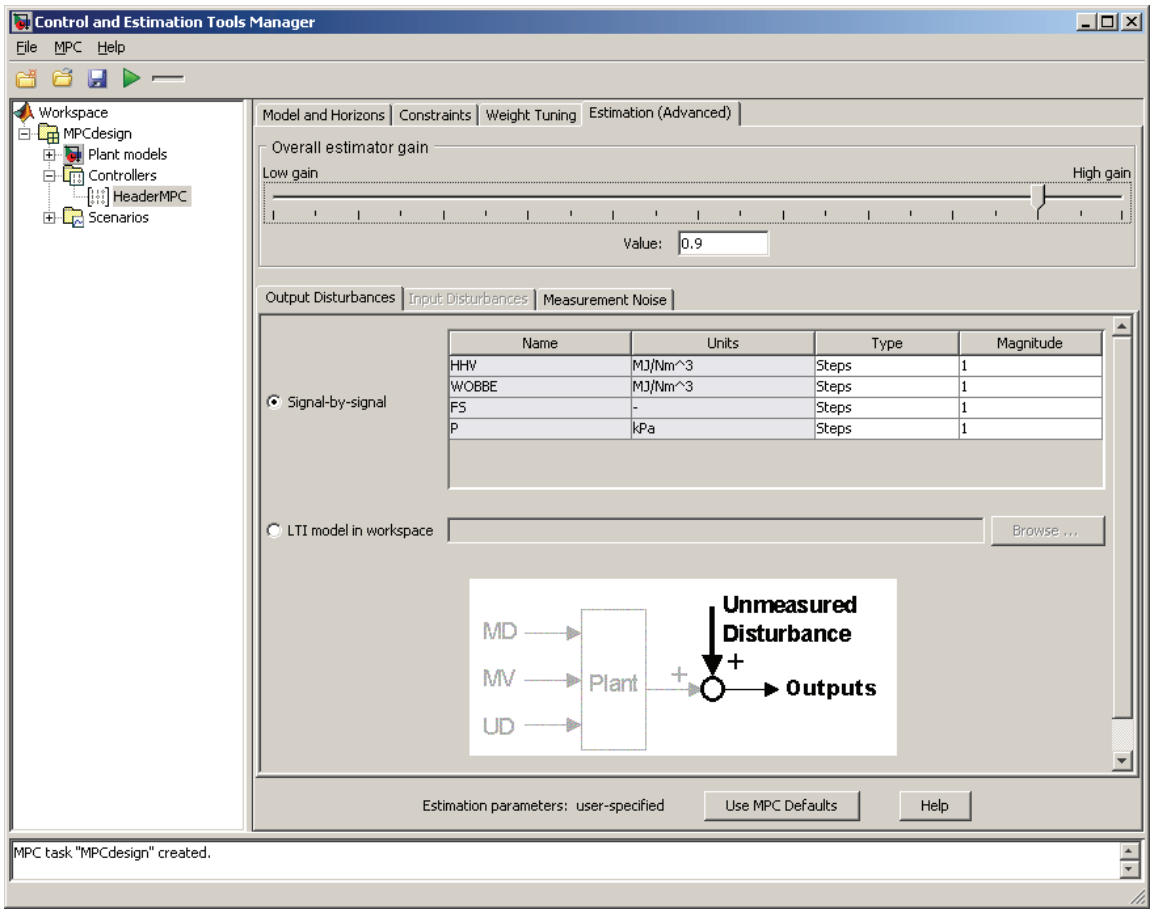

Figure B.9. Configuration of disturbance handling in MPC Toolbox utility.

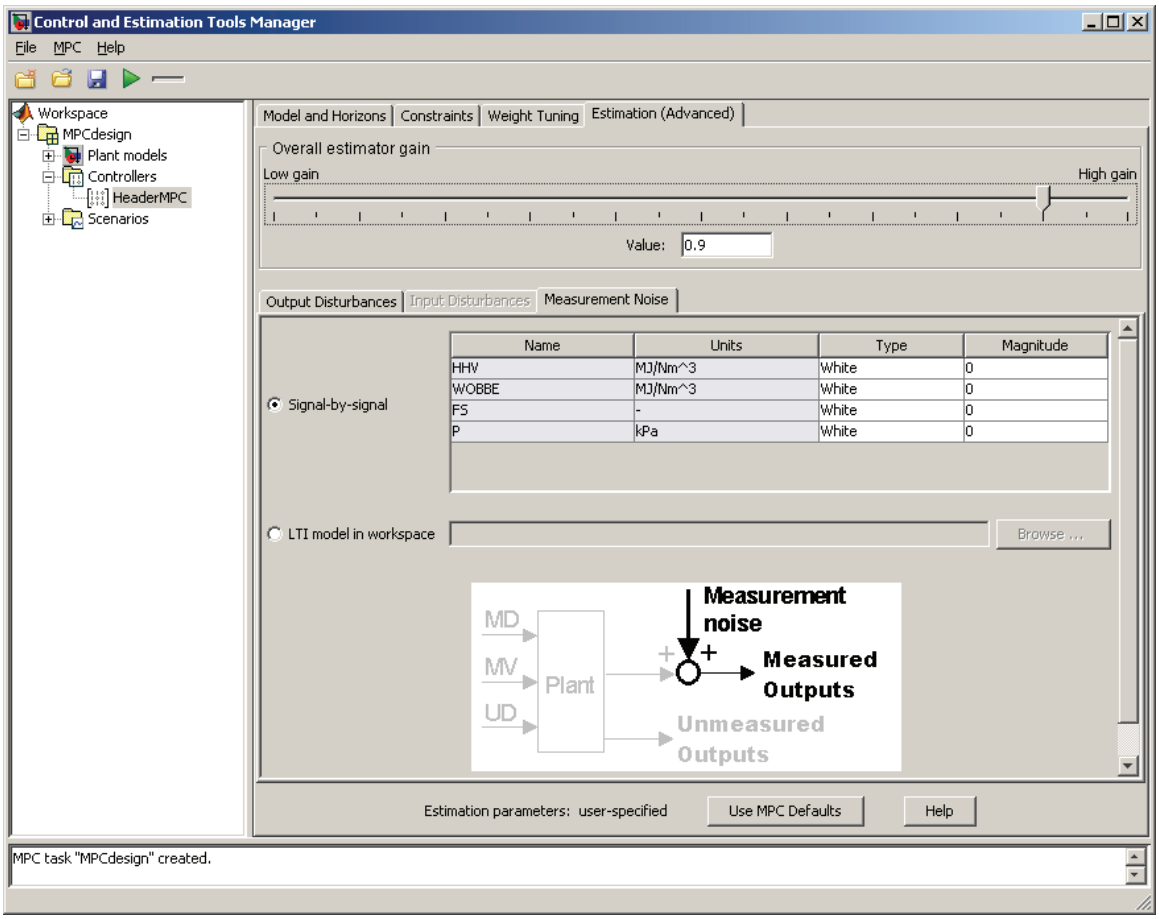

Figure B.10. Configuration of noise handling in MPC Toolbox utility.