Modeling and Optimal Control of Biomass Power Plant for Better Steady-State Error and Disturbance Rejection

Michael David Janicki

A thesis submitted
in partial fulfillment of the
requirements for the degree of

Masters of Science in Mechanical Engineering

University of Washington

2021

Committee:

Joseph L. Garbini

Ashley Emery

John Kramlich

Program Authorized to Offer Degree:
Mechanical Engineering

University of Washington

Abstract

Modeling and Optimal Control of Biomass Power Plant for Better Steady-State Error and Disturbance Rejection

Michael David Janicki

Chair of the Supervisory Committee:
Professor Joseph L. Garbini
Mechanical Engineering

This thesis describes the development and testing of a dynamic system model and optimal controls for a solid biomass fuel power plant. The motivation for the model is to derive optimal Multi-Input, Multi-Output (MIMO) controllers intended to improve the steady-state output error of the power plant states and help reduce the effect of disturbances in the fuel stream. This model is developed primarily from first principles with some testing on a specific power plant used as a case study. The sensitivity of the model to the parameters is investigated. The model is used to derive Linear Quadratic Gaussian (LQG) servo controllers in a cascade feedback arrangement to control all of the steam states, combustion states, and outputs. The optimal controllers were implemented on a power plant and tested to compare their performance to the original hand-tuned Single-Input, Single-Output (SISO) controllers running the plant. The two control schemes were tested and compared for output error performance and disturbance rejection. The optimal controllers show a marked improvement over the SISO controllers in both categories. The standard deviations of the outputs are computed and compared. The optimal controllers reduce the output errors by 48.9% and 20.6% on average for steady-state and disturbance rejection for the four outputs, respectively.

TABLE OF CONTENTS

LIST OF FIGURES

GLOSSARY

OPTIMAL: Control law/estimator that has been optimized through a cost function

SISO: Single-Input, Single-Output

MIMO: Multi-Input, Multi-Output

LQG: Linear Quadratic Gaussian

LQR: Linear Quadratic Regulator

KALMAN FILTER: Optimal State Estimator

HHV: Higher Heating Value

ACKNOWLEDGMENTS

The author wishes to express sincere appreciation to the following parties:

- University of Washington, where he was given the tools, knowledge, and support required to complete this project.
- Sedron Technologies, for funding the plant testing, and providing the opportunity to use their equipment as the case study for this thesis.
- Professor Joseph Garbini, for his unending guidance, feedback, and patience.
- Professors John Kramlich and Ashley Emery, for their knowledge and advice throughout the project.
- Dr. K.J. Åström, for his research and his help.
- Dr. K. Boyd Fackler, for his technical support regarding questions on the J-OP S250.

DEDICATION

To my parents. For their support, guidance, and encouragement.

Chapter 1
INTRODUCTION

In the United States, steam power plants of one form or another contribute over 80% of the national electricity production total [7]. These power plants range from combined cycle natural gas, coal, nuclear, biomass, geothermal, and more. For these plants, control of the energy production and the system sub-states is critically important for emissions, safety, and maintaining plant power output. For biomass power plants, control is more challenging. Biomass fuel is generally a byproduct of different waste streams, such as wood waste, agricultural waste, or other bio-waste streams. Being derived from waste streams, biomass fuel often has inconsistent energy density, moisture content, and chemical composition. Although the fuel is inconsistent, it is often free or taken at a fee as opposed to expensive natural gas. The fuel creates unique challenges for the controls of power plants, but also makes biomass plants attractive from a renewable and sustainable energy perspective.

1.1 Research Question

Modern biomass power plants provide a carbon neutral source of electricity, but have more complicated controls due to the added difficulty of solid biomass fuel combustion. This difficulty comes from variations in the fuel stream, as well as physical differences in the combustion mechanics involved with solid fuels. Without proper automation and control, these issues require more skilled operators to adjust inputs and operating parameters to maintain plant outputs, thereby making the plants more expensive to operate.

Typically, industrial control schemes incorporate hand-tuned Single-Input, Single-Output (SISO) controllers. These controllers are simple to implement, but time consuming to tune and have limited performance characteristics due to the coupled nature of the system dynam-

ics. Coupled systems with SISO controllers can have difficulty coordinating inputs, which can cause cyclic oscillations in the outputs. Lower bandwidth of the SISO controllers can help mitigate oscillations, but slows the response of the controllers to disturbances, meaning operator oversight is required in the case of rapid fuel input changes.

Based on these issues, the following research question has been formulated:

- Can optimal Multi-Input, Multi-Output (MIMO) controllers be found to help reduce variations in the system outputs during steady-state and from disturbances in the fuel stream?

1.2 Thesis Statement

It is proposed that optimal MIMO controllers could improve the system performance, help reduce the effect of variations in the fuel input stream, and reduce total plant labor. Additionally, these controllers do not need to be tuned on-line by operators, which reduces the cost and time of implementation during the commissioning phase of the power plant.

To derive these controllers, an adequate model of the power plant needs to be developed. This model needs to represent the steady-state output values and dynamics well enough for feedback control. This model needs to be developed from first-principles modeling as much as possible to allow it to represent the outputs and dynamics of different plants based on their parameters.

This thesis investigates the development of this model and the creation and testing of the controllers on a real power plant to determine the validity of the proposal.

Chapter 2
BACKGROUND

This chapter discusses the motivation behind this research and some of the challenges associated with modeling and controls of a solid biomass fuel power plant.

2.1 Research Motivation

This research is concerned with improving the control of of solid biomass fuel power plants. Specifically, it focuses on improving the control of a particular power plant, the Sedron Omniprocessor Version 250 (J-OP S250), in Sedro-Woolley, Washington. The J-OP S250 is a relatively small power plant, with a thermal output of 3.0 MW. While this power plant was used for testing, parameters, and validation, the design of the model and controls is intended to be generalized to any power plant that fits the same basic design and layout.

The fuel for the J-OP S250 is wet fecal sludge taken from the local sewer treatment plant in Sedro-Woolley. The J-OP S250 has an integrated dryer that processes this wet fuel and takes it from approximately 12% solids to 85% solids. The dryer utilizes the waste heat from the Rankine cycle to dry the fuel. The plant was designed for an initial solids content of 50-60%, so the wetter sludge in Sedro-Woolley is augmented with dry wood pellets to decrease the average moisture content of the incoming fuel.

One of the specific challenges with the J-OP S250 is that the drying process is inconsistent, which leads to variations in the fuel moisture content. On a given day, fuel moisture content can vary anywhere from 12% to 18%, which results in a large change in energy density on a mass basis. This change can be observed in some of the data, as shown later in Chapter 6. The moisture content is measured separately from the machine using an independent moisture meter, but is not instantaneously measured or controlled.

2.2 Natural Gas Combustion vs. Solid Biomass Fuel Combustion

In the United States, power generation using combustion is dominated by natural gas. In 2016, 34% of all power generation resulted from natural gas, while only 2% came from all biomass combustion put together, only a fraction of which is solid biomass fuels [15]. Natural gas has several advantages over solid biomass fuels in terms of its ease of use. It has consistent energy density, chemical composition, and physical properties. Additionally, measuring the mass flow rate of a natural gas is relatively easy. This all makes natural gas an excellent energy source from a controls perspective as the input gains are easy to characterize and are nearly constant. Additionally, natural gas is an extremely clean burning fuel. Due to its short chemical structure, CH_4, it burns quickly and cleanly and does not produce many pollutants.

With solid biomass fuels meanwhile, it is more difficult to achieve the complete combustion required to meet emissions standards. An indication of complete combustion for solid fuel is the content of carbon monoxide (CO) measured in the exhaust gasses. For a power plant of the size and type of the J-OP S250 in the United States, the federal requirement for CO is less than 100 parts-per-million (ppm) on a volume basis. A low CO ppm count indicates complete combustion occurred, and also suggests other pollutants and toxins were not generated during the combustion process, as these pollutants are often inversely correlated with CO exhaust concentration [21].

Unlike the combustion of natural gas, which is very consistent and direct, solid fuel biomass combustion is broken up into two stages. The first stage is oxidative pyrolysis (flaming), and the second stage is heterogeneous reduction (char glowing) [6]. Once the fuel is brought up to temperature and dried out, it first starts to smoke and burn, and then it reduces down to charcoal and burns out the rest of the way as small coals. This is observed around a typical campfire - first large flames, then glowing coals. The flaming step happens quickly, but the char glowing step slows down the combustion process significantly. This greatly increases the amount of time required for complete combustion [6, 3]. Plant designers

try to engineer the combustion chambers so there is sufficient time-at-temperature to properly reduce and burn the char particles. The amount of unburned fuel can be correlated with the concentration of CO in the exhaust.

What makes this challenging from a controls perspective is the additional time-at-temperature introduces an added delay in the combustion process, which correlates to a longer time delay between fuel addition to the combustion system and heat addition to the steam system. Time delays in feedback controls systems are extremely problematic as they introduce fundamental limitations on the system stability. If not properly accounted for, these time delays can create unsynchronized application of the control inputs, which can lead to unwanted oscillations in the outputs. Characterizing and handling these time delays is one of the aspects of this research.

2.3 Literature Review

There are many papers in the literature about controls and modeling of steam power plants. This research is primarily inspired by the work of K.J. Åström and R.D. Bell. They develop a dynamic system model for steam power plant from first principles that can be used for controls purposes in their paper *Drum Boiler Dynamics* [1]. Åström and Bell reference several papers that use their model for controls schemes such as [16, 19, 4]. The Åström and Bell model is additionally referenced for control purposes by [17, 10, 11, 18]. More control methods have been attempted that do not incorporate the Åström and Bell model such as [20, 8, 5].

However, this research is primarily concerned with modeling and controls of a solid biomass fuel power plant. All of the above mentioned papers are for liquid or gaseous fuels only, and some of the papers neglect the combustion portion of the controls/modeling altogether.

Few papers investigate modeling and control of solid fuel combustion. In the paper *Dynamic modelling of biomass power plant using micro gas turbine*, the authors S. Barsali et al investigate a dynamic system model for a solid fuel biomass power plant [2]. This analysis is

specific to a single type of fuel, glucose, and the incorporation of the micro gas turbine makes the model difficult to apply to a more generic solid biomass fuel boiler. The paper *Dynamic Model of a Circulating Fluidized Bed Boiler* by Majanne and Kökkä provides a complete but very complicated model for solid biomass fuel combustion [14]. This model includes a few elements not incorporated in the J-OP S250 and a high number of required parameters with no feasible way to determine them. The paper *Modeling and model predictive control of the BioPower combined heat and power (CHP) plant* by Kortela and Jämsä-Jounela also describes a dynamic model for solid fuel biomass combustion [12]. However, this model is specific to a given type of fuel, and makes certain assumptions about the consistency of the fuel input stream which simplifies the equations beyond what can be utilized for the J-OP S250.

After careful analysis of the current state of the literature, there is a gap of research for determining a simple model for a solid biomass fuel power plant for controls purposes. Chapter 3 shows derivation of this model.

Chapter 3
SYSTEM MODEL DESCRIPTION

This chapter provides a detailed description of the mathematical model developed for the J-OP S250. This model can be applied to a generic solid biomass fuel power plant, but the specifics and parameters are fitted to the J-OP S250.

3.1 Plant Description

The purpose of the J-OP S250 power plant is to burn biomass fuel to provide heat to boil water to generate power. This must be accomplished while maintaining complete combustion for proper emissions control. Some of the steam that is generated is throttled down to a lower pressure for generic uses around the plant, but most is fed to a steam turbine, which generates electrical power for the grid and the plant itself.

To generate the heat for boiling water, solid fuel is burned with a fixed amount of sub-stoichiometric air in a fluidized sand bed, and secondary air is added after the sand bed for complete combustion and emissions control. The excess air is monitored with an oxygen sensor in the exhaust stream.

The secondary air is added in a chamber called the freeboard, which is surrounded by walls made of welded steel pipes, as seen in Figure 3.1. These pipes boil water and are called the waterwalls. The combustion temperature in the freeboard is an important part of the system, as the heat provided by the waterwalls makes up about 80 % of the total heat in the system. The heat from the freeboard is proportional to the combustion temperature in the freeboard. After the freeboard there is an evaporator for additional boiling, a superheater, a secondary evaporator, and an economizer.

The water is boiled with a naturally circulating drum boiler. The drum acts as a phase

separator, and separates the two-phase flow coming out of the risers from the evaporator circuits. The saturated steam separated by the steam drum is sent to the superheater, and the liquid water settles due to density differences and circulates around back to the evaporators. The water level in the drum is maintained with a feedwater pump that adds additional water to make up for the steam leaving the system. This cold water is heated with the economizer, but is always kept colder than the saturation temperature of the steam drum to prevent boiling in the economizer. A small amount of water leaves the system through the blowdown valves, which open and close periodically at the bottom of the evaporators to remove sediment buildup left behind by the boiling water.

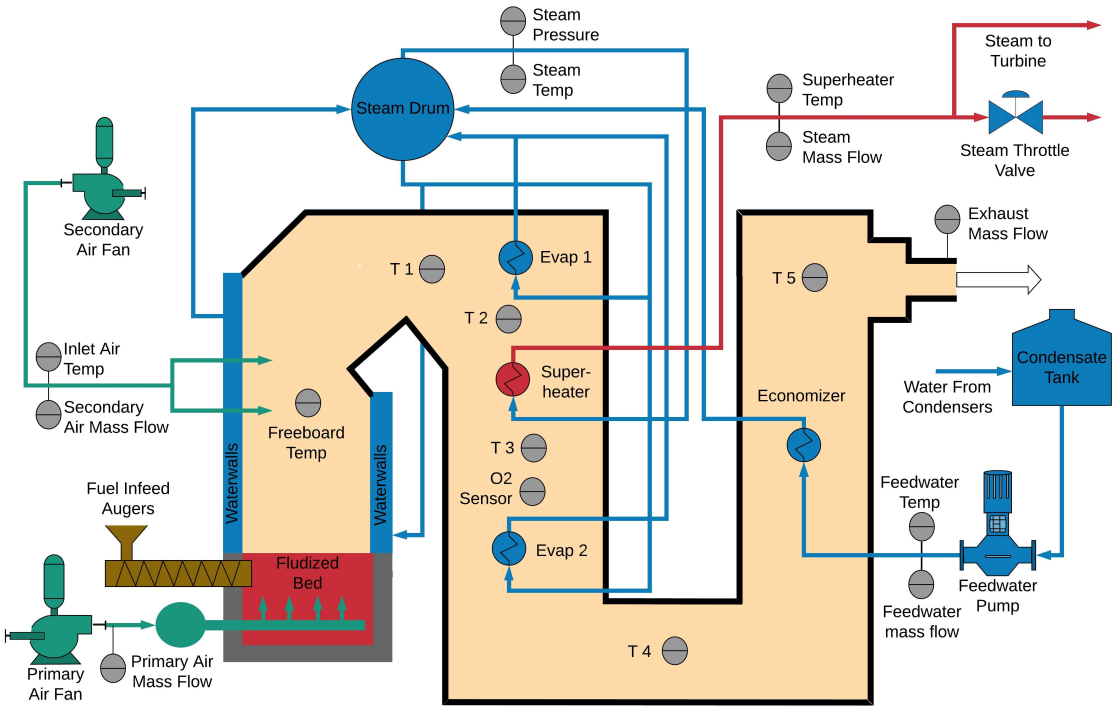

Figure 3.1: Diagram of J-OP S250. Grey circles represent sensor locations.

3.2 Model Motivation

The purpose of this model is to accurately represent the behavior of the J-OP S250 power plant to derive optimal controllers. The derivation of the optimal controllers is shown in Chapter 5. The nonlinear model is used to test the controllers and is compared to the closed-loop plant testing data, which is shown in Chapter 6.

3.3 System Model Overview

The model is split up into two major subsystems: The steam system, and the combustion system. The steam system is modeled almost exclusively using the fourth order non-linear model developed by K.J. Åström and R.D. Bell in their paper titled *Drum Boiler Dynamics*, with a few modifications to match the specifics of the J-OP S250. The combustion system models the fuel and air inputs, combustion dynamics, and heat exchangers, while the steam system models the steam drum dynamics, and system load. The combustion system model is developed mostly from first principles and has been validated with experimental data.

3.3.1 Steam System

The steam system for the J-OP S250 consists of the feedwater pumps, economizer, steam drum, evaporator circuits, superheater, steam load, and pressure control valve. The purpose of the steam model is to represent the dynamic behavior and states of the boiler. A diagram of the steam drum and naturally circulating downcomer/riser loops is shown in Figure 3.2.

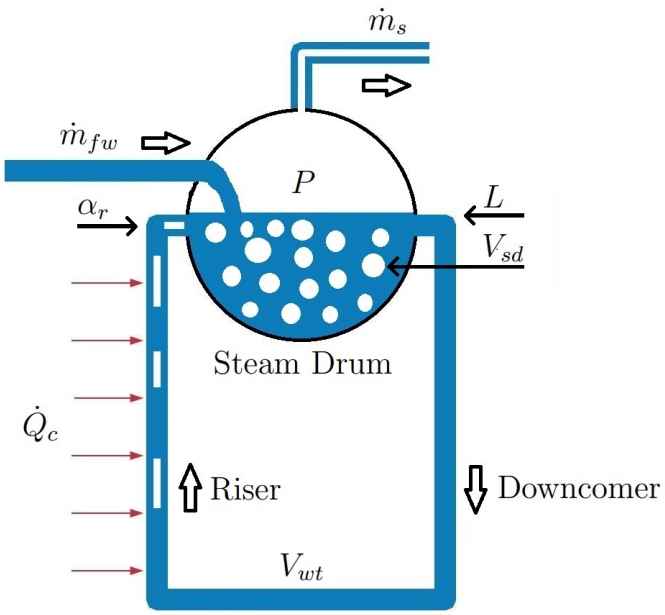

Figure 3.2: Diagram of naturally circulating boiler described by steam system state equations.

Figure 3.2 shows the inputs, the states, and the outputs. The incoming feedwater is on the left, the saturated steam mass flow is on top of the steam drum, and the heat to boil the water next to the riser. The steam drum level is shown on the right next to the steam bubble volume term, V_{sd}. The quality leaving the riser is indicated by α_r. The quality at the top of the riser is a measure of the percentage of water that was boiled going up the riser tubes. The entire system is assumed to operate at the saturated system pressure, P. Lastly, the blue volume of water is the total system water, V_{wt}

Steam System State Equations

The state equations for the steam system model are as follows:

$$e_{11}\frac{dV_{wt}}{dt} + e_{12}\frac{dP}{dt} = \dot{m}_{fw} - \dot{m}_s - \dot{m}_L \tag{3.1}$$

$$e_{21}\frac{dV_{wt}}{dt} + e_{22}\frac{dP}{dt} = \dot{Q}_c + \dot{m}_{fw}h_{fw} - \dot{m}_s h_s - \dot{m}_L h_w \tag{3.2}$$

$$e_{32}\frac{dP}{dt} + e_{33}\frac{d\alpha_r}{dt} = \dot{Q}_c - \alpha_r h_c \dot{m}_{dc} \tag{3.3}$$

$$e_{42}\frac{dP}{dt} + e_{43}\frac{d\alpha_r}{dt} + e_{44}\frac{dV_{sd}}{dt} = \frac{\rho_s}{T_d}(V_{sd}^0 - V_{sd}) + \frac{h_{fw} - h_w}{h_c}\dot{m}_{fw} \tag{3.4}$$

The coefficients e_{ij} are as follows:

$$\begin{aligned}
e_{11} &= \rho_w - \rho_s \\
e_{12} &= V_{wt}\frac{\partial \rho_w}{\partial P} - (V_t - V_{wt})\frac{\partial \rho_s}{\partial P} \\
e_{21} &= \rho_w h_w - \rho_s h_s \\
e_{22} &= V_{wt}(h_w\frac{\partial \rho_w}{\partial P} - h_s\frac{\partial \rho_s}{\partial P} + \rho_w\frac{\partial h_w}{\partial P} - \rho_s\frac{\partial h_s}{\partial P}) \\
&\quad - V_t(h_s\frac{\partial \rho_s}{\partial P} + \rho_s\frac{\partial h_s}{\partial P} - 1) + m_t C_p\frac{\partial t_s}{\partial P} \\
e_{32} &= (\rho_w\frac{\partial h_w}{\partial P} - \alpha_r h_c\frac{\partial \rho_w}{\partial P})(1 - \bar{\alpha}_v)V_r + ((1-\alpha_r)h_c\frac{\partial \rho_s}{\partial P} + \rho_s\frac{\partial h_s}{\partial P})\bar{\alpha}_v V_r \\
&\quad + (\rho_s + (\rho_w - \rho_s)\alpha_r)h_c V_r\frac{\partial \bar{\alpha}_v}{\partial P} - Vr + m_r C_p\frac{\partial t_s}{\partial P} \\
e_{33} &= ((1-\alpha_r)\rho_s + \alpha_r\rho_w)h_c Vr\frac{\partial \bar{\alpha}_v}{\partial \alpha_r} \\
e_{42} &= V_{sd}\frac{\partial \rho_s}{\partial P} + \frac{1}{h_c}(\rho_s V_{sd}\frac{\partial h_s}{\partial P} + \rho_w Vwd\frac{\partial h_w}{\partial P} - V_{sd} - V_{wd} + m_d C_p\frac{\partial t_s}{\partial P}) \\
&\quad + \alpha_r(1+\beta)V_r(\bar{\alpha}_v\frac{\partial \rho_s}{\partial P} + (1-\bar{\alpha}_v)\frac{\partial \rho_w}{\partial P} + (\rho_s - \rho_w)\frac{\partial \bar{\alpha}_v}{\partial P}) \\
e_{43} &= \alpha_r(1+\beta)(\rho_s - \rho_w)Vr\frac{\partial \bar{\alpha}_v}{\partial P}) \\
e_{44} &= \rho_s
\end{aligned}$$

In the e_{ij} coefficients and the state equations, the subscripts s, w, t, and d refer to steam, water, total, and drum, respectively. The h and ρ terms refer to enthalpies and densities, respectively. All of the thermodynamic values, including the partial derivatives, are determined using curve fits as functions of the saturation pressure, P. The parameters in the e_{ij} coefficients are detailed in Section, 4.1. Some variables in the coefficients are omitted for brevity, but can be found in the Åström and Bell paper.

State Variable, Inputs, and Disturbances

The state variables described in this model are as follows:

- V_{wt} (m^3) - Total system water volume
- P (bara) - Steam system pressure
- α_r (dimensionless) - Riser steam quality at exit of risers
- V_{sd} (m^3) - Steam bubble volume below drum water level

The inputs described in this model are as follows:

- $\dot{Q}_c$ (kW) - Heat input
- $\dot{m}_{fw}$ (kg/s) - Feedwater mass flow input
- v (%) - Valve position

The disturbances described in this model are as follows:

- T_{fw} $(^\circ C)$ - Feedwater temperature
- $\dot{m}_L$ (kg/s) - Lost blowdown water mass flow

- P_d (bara) - Dryer pressure after throttle
- $\dot{m}_{st}$ (kg/s) - Turbine steam mass flow

The steam drum level is defined by the following equation:

$$L = \frac{V_{wd} + V_{sd}}{A_d} \tag{3.5}$$

where L is the linearized deviation from the nominal operating level. A_d is the wet surface area at the nominal operating level, V_{sd} is the steam bubble volume below the water level, and V_{wd} is the water volume below the water level in the drum. In Figure 3.2, the bubble volume and drum water volume can be pictured as the white bubbles and blue water in the drum, respectively. The drum water volume is given by the following equation:

$$V_{wd} = V_{wt} + V_{dc} - (1 - \bar{\alpha}_v)V_r \tag{3.6}$$

where V_{wt} is the total water volume, V_{dc} is the fixed downcomer volume, V_r is the fixed riser volume, and $\bar{\alpha}_v$ is the average steam volume fraction in the risers, which is given by the following equation:

$$\bar{\alpha}_v = \frac{\rho_w}{\rho_w - \rho_s}\left(1 - \frac{\rho_s}{(\rho_w - \rho_s)\alpha_r}\ln\left(1 + \frac{\rho_w - \rho_s}{\rho_s}\alpha_r\right)\right) \tag{3.7}$$

Model Description

The first two steam system equations, Equations 3.1 and 3.2, are mass and energy balances, respectively. Since the total volume of the steam drum and evaporator system is fixed, accounting for the two-phase system becomes much easier. By keeping track of the saturation pressure and the total volume of water in the system, all thermodynamic properties can be determined. The saturation pressure and total system volume are the first two state variables. The entire system - fluids and metal piping - is assumed to be at the saturation temperature which is uniquely determined by the saturation pressure. This makes heat transfer calculations much easier.

The second two equations, Equations 3.3 and 3.4, describe the drum level dynamics. The second two state variables are the quality of the steam out of the risers, and the volume of the steam bubbles below the water level in the drum. The quality is the ratio of steam mass to water mass just as the flow enters the steam drum. The quality and the steam bubble volume describe the shrink and swell phenomenon of the boiler system, which is discussed in more detail in the Åström and Bell paper. Additionally, as shown in Equation 3.5, they help relate the state variables to the steam drum level.

The steam exiting the system, $\dot{m}_s$, has two components: Steam that leaves through a throttle valve, and steam demanded by the turbine. The steam flow is modeled as follows:

$$\dot{m}_s = \dot{m}_{st} + \dot{m}_v = \dot{m}_{st} + (P - P_d)(c_1 v + c_2) \tag{3.8}$$

where v is the commanded steam valve position, P is the drum pressure, P_d is the dryer pressure after the valve, and c_1 and c_2 are the valve admittance parameters that are determined experimentally.

The feedwater temperature, blowdown water, dryer pressure, and turbine steam mass flow are all modeled as disturbances. This means they are uncontrolled, but still effect the system dynamics. Controllers operating well should be able to reject disturbances from changes in these variables.

The feedback variables for the steam system are the drum pressure and the drum level. These two variables can be used to control the system. All of the disturbances are measured except for the lost blowdown water mass flow, but it is small in magnitude compared to the other mass flows.

This model accurately describes the behavior of the steam dynamics well-enough for model-based control design. However, one of the inputs, the heat input $\dot{Q}_c$ is assumed to be known and controlled directly. For a natural gas power plant, determining the combustion heat input is trivial, and can be estimated as follows:

$$\dot{Q}_{total} = HHV_{ng}\dot{m}_{ng} \tag{3.9}$$

where HHV_{ng} is the higher heating value of natural gas, and $\dot{m}_{ng}$ is the mass flow of natural gas. This total combustion heat is typically combined with a boiler efficiency term, η_b, to determine the heat provided to the steam system for boiler water. This efficiency is usually around 80%. This is shown as follows:

$$\dot{Q}_c = \eta_b \dot{Q}_{total} \tag{3.10}$$

If the plant has a mass flow sensor on the gas line, then the heat can be determined using this method.

For a solid biomass fuel power plant, however, this method is insufficient to model the combustion process. The bulk energy density of the fuel is determined as a function of the dry fuel energy density, the moisture content, and the chemical composition. With solid biomass fuel, these parameters are often changing and not measured, so the simple method shown in Equations 3.9 and 3.10 will not work. Instead, the heat release to the boiler must be estimated based on the temperature profiles of the heat exchangers. This is shown in the next section.

3.3.2 Combustion System

The combustion system models the combustion dynamics. This includes heat release, temperature profiles of the heat exchangers, and excess oxygen content in the exhaust stream. Time delays and dynamic elements have been included represent experimental data taken from the J-OP S250. Figure 3.3 shows a block diagram representation of the combustion model.

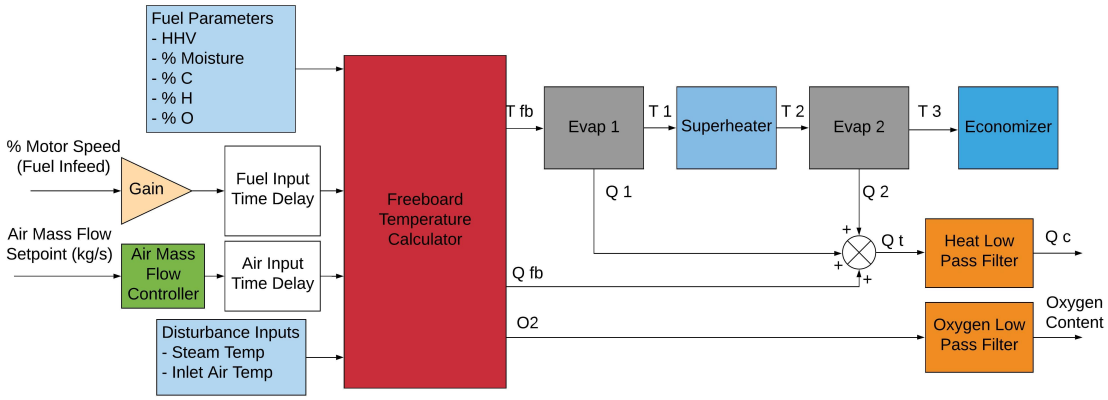

Figure 3.3: Diagram of combustion model.

Combustion Model Objective

The objective of the combustion model is to find differential equations for the heat to the steam system and the exhaust oxygen content as a function of the combustion inputs. The heat exchanger temperature profiles are used to estimate the heat for feedback control purposes.

System Inputs

The inputs to the combustion system are air and fuel. The air is controlled with an electric blower, and the fuel is added to the system with dual metering augers. These augers drop the fuel through a rotary lock to an infeed auger, which conveys the fuel directly into a fluidized sand bed for combustion. The mechanics of the behavior of the fluidized sand bed have been largely ignored for the purpose of this model. This is a reasonable modeling decision since the bed plays a fairly small role in the heat generation dynamics, provided the fuel is pre-processed for particle size and has a moisture content less than 20%.

One of the important characteristics of the combustion system is the process delays associated with the fuel and air inputs. These are very important parameters for accurate

modeling and control. These delays are defined as the time that it takes after a change in an input to observe a change in the outputs. Through experimental tests, the input delays were determined to be approximately 70 seconds for the fuel, and 16 seconds for the air. The reason for these delays is a combination of physical transport delays for the air and fuel, and the mixing, heating, and drying of the fuel that occurs in the fluidized sand bed.

Freeboard Temperature

To calculate the heat, the temperature profiles of the heat exchangers must be known. The initial average combustion temperature in the freeboard must first be determined, which is a function of the fuel composition and fuel and air mass flows. The temperature is calculated by performing an enthalpy balance on the combustion chamber and assuming complete combustion. Since the J-OP S250 has a non-homogeneous fuel input stream, special care has been taken to model the different factors that effect the temperature output, such as chemical composition, Higher Heating Value (HHV), and moisture content.

The calculation of the freeboard temperature has the following list of inputs and parameters:

- *carbon* (%) - Percent carbon in fuel by weight
- *hydrogen* (%) - Percent hydrogen in fuel by weight
- *oxygen* (%) - Percent Oxygen in fuel by weight
- HHV (kJ/kmole) - Fuel higher heating value
- MP (%) - Fuel moisture
- UA_{fb} (kW/K) - Freeboard overall heat transfer coefficient
- T_a (K) - Inlet air temperature

- $T_{sat.}$ (K) - Steam saturation temperature
- $\dot{m}_f$ (kg/s) - Fuel mass flow
- $\dot{m}_a$ (kg/s) - Air mass flow

First, the moles of each element in the fuel molecule are calculated by dividing each elements mass percentage by its respective molar mass. These are then used to balance the chemical equation to the following form:

$$C_xH_yO_z + \phi a_s(O_2 + 2.76N_2) \rightarrow \frac{1}{2}yH_2O + xCO_2 + a_s(\phi - 1)O_2 + 3.76\phi a_s N_2 \tag{3.11}$$

where $a_s = \frac{y+4x-2z}{4}$ is the stochiometric air-to-fuel ratio, and x, y, and z are the molar compositions of each element in the fuel molecule. Note that this equation is not valid for $\phi < 1$, which would imply sub-stochiometric combustion. The chemistry of sub-stochimetric combustion is more complicated, so it is assumed there will only be fuel-lean combustion taking place. This assumption is made because the real system runs fuel-lean to control emissions.

This chemical equation is used with $\phi = 1$ along with the given HHV to determine the enthalpy of formation ($\bar{h}^\circ_f$) for the fuel molecule using the following equation:

$$\bar{h}^\circ_f = x\bar{h}^\circ_{fCO_2} + \frac{y}{2}\bar{h}^\circ_{fH_2O} + 100HHV \tag{3.12}$$

where $\bar{h}^\circ_{fCO_2}$ and $\bar{h}^\circ_{fH_2O}$ are the enthalpies of formation for CO_2 and liquid H_2O, respectively. Next, the theoretical air ratio must be determined. This represents the ratio of the air mass flow to fuel mass flow.

$$A_{theo} = \frac{100\dot{m}_{air}}{4.76a_sMW_{air}(100 - MP)\dot{m}_{fuel}} \tag{3.13}$$

where MP is the percent moisture in the fuel. Once h°_f and A_{theo} have been calculated, a total energy balance is performed for the reaction using the following formula:

$$H_p(T_{fb}) - H_r = UA_{fb}(T_{fb} - T_{sat.}) \tag{3.14}$$

where H_p is the enthalpy of the products as a function of final temperature, H_r is the enthalpy of the reactants, and $UA_{fb}(T_{fb} - T_{sat.})$ is the heat absorbed through the waterwalls in the freeboard to the steam system.

The enthalpy of the reactants can be explicitly calculated if the fuel is assumed to enter the combustion chamber at 298 K:

$$H_r = \bar{h}^\circ{}_f + a_s A_{theo} \Delta h_{O_2} + 3.76 a_s A_{theo} \Delta h_{N_2} + w \bar{h}^\circ{}_{f H_2O_l} \tag{3.15}$$

where $w = MP\dot{m}_{fuel}/(100 MW_{H_2O})$ is the kmoles of liquid in the incoming fuel stream. The enthalpies of CO_2 and H_2O are calculated from integrated specific heat equations, starting at 298 K and ending at the inlet air temperature.

The enthalpy of the products is more difficult to calculate, because it is a function of the final temperature of the freeboard. It can be expressed implicitly as follows:

$$\begin{aligned} H_p = x(\bar{h}^\circ{}_{fCO_2} + \Delta h_{CO_2}(T_{fb})) + (y/2 + d)(\bar{h}^\circ{}_{f H_2O_g} + \Delta h_{H_2O}(T_{fb})) \\ + a_s(A_{theo} - 1.0)\Delta h_{O_2}(T_{fb}) + 3.76 a_s A_{theo} \Delta h_{N_2}(T_{fb}) \end{aligned} \tag{3.16}$$

where T_{fb} is the final reaction temperature. The total implicit enthalpy balance equation is solved using iteration until the residual error is less than 0.1 kW.

Once the freeboard temperature is calculated, the heat to the freeboard can be found by evaluating the heat loss term as follows:

$$\dot{Q}_{fb} = UA_{fb}(T_{fb} - T_{sat.}) \tag{3.17}$$

Heat Exchangers

Once the temperature is calculated for the freeboard, the exhaust gasses are exposed to the rest of the heat exchangers. There are two different types of heat exchangers in the J-OP

S250: Evaporators for boiling water, and the superheater and economizer which are both counterflow heat exchangers. These two types are differentiated by the fact that saturated steam has a constant temperature, $T_{sat.}$, at a given pressure, whereas the other two heat exchangers transfer heat to a variable temperature stream, which needs a more complicated heat transfer equation.

To determine the temperature of the gas after an evaporator, the following heat transfer relation is used:

$$T_2 = T_{sat.} + (T_e - T_{sat.})e^{-\frac{UA_{evap}}{\dot{m}_e C_p(T_e)}} \tag{3.18}$$

where T_e is the initial exhaust gas temperature, UA_{evap} is the evaporator overall heat transfer coefficient, $\dot{m}_e$ is the exhaust mass flow, and $C_p(T_e)$ is the specific heat of the exhaust gas evaluated at the initial exhaust gas temperature. This equation assumes constant specific heat because the relative change in the specific heat is small. Additionally, the exhaust gas, though a multi-component stream, is approximated as 100% Nitrogen (N_2) to simplify the energy balances on the heat exchangers. This assumption is made because the exhaust stream is more than 75% N_2 at all times, and the actual components cannot be measured real-time to perform a more thorough analysis.

Once the temperatures are known before and after the evaporator, the heat given up to the steam system is calculated using the following formula:

$$\dot{Q}_{evap} = \dot{m}_e \Delta h \tag{3.19}$$

where Δh is the change in enthalpy across the heat exchanger. Both evaporators use the same equations to calculate the temperature after the evaporator and the heat transferred to the steam system. The enthalpy is calculated by integrating the specific heat across the temperature difference, as seen in Equation 3.20:

$$\Delta h = \int_{T_e}^{T_2} C_p dT = \frac{R_u(aT + \frac{1}{2}bT^2 + \frac{1}{3}cT^3 + \frac{1}{4}dT^4 + \frac{1}{5}eT^5)}{MM_{N_2}} \tag{3.20}$$

where R_u is the universal gas constant, MM_{N_2} is the molar mass of N_2, and the polynomial coefficients are for the isobaric specific heat of N_2 [22].

The superheater uses a similar method, but with the additional input of $\dot{m}_s$ - the steam mass flow. The superheater also uses a different heat transfer equation as follows:

$$T_2 = \frac{T_{sat.} + C_r T_e + (T_e - T_{sat.})e^{-NTU(1+C_r)}}{1 + C_r} \tag{3.21}$$

where$C_r = \frac{C_{min}}{C_{max}}$ is the ratio of the specific heats. This equation also makes the assumption of constant specific heats over the temperature difference. NTU is the Number of Transfer Units, which is a metric heat exchangers are evaluated for performance characteristics. The equation for NTU is shown as follows:

$$NTU = -\frac{UA_{sh}}{\dot{m}_e C_p(T_e)} \tag{3.22}$$

The NTU value is determined from plant data, which is shown in Chapter 4.

Total combustion Heat

The total amount of heat from the combustion system that went to boil water in the steam system is the sum of the heat from each of the evaporators and the freeboard:

$$\dot{Q}_c = \dot{Q}_{fb} + \dot{Q}_1 + \dot{Q}_2 \tag{3.23}$$

where $\dot{Q}_1$ and $\dot{Q}_2$ are the heats from the first and second evaporators, calculated using Equation 3.19.

To accurately represent the input/output data for the J-OP S250, the total combustion heat is fed through a first order low pass filter with a 50 second time constant to better represent input/output data taken from the J-OP S250.

Excess Oxygen

The excess oxygen in the exhaust stream can be calculated explicitly using the chemical compositions and fuel and air mass flows. It can be found using the following equation:

$$O_2 = \frac{100a_s(A_{theo} - 1.0)}{x + y/2 + w + a_s(A_{theo} - 1.0) + 3.76a_s(A_{theo} - 1.0)} \tag{3.24}$$

where O_2 is the percent excess oxygen that is measured in the exhaust stream. Similar to the combustion heat, the oxygen is fed through a first order low pass filter with a 25 second time constant.

Chapter 4

PARAMETER ESTIMATION AND SENSITIVITY ANALYSIS

This chapter provides a detailed description of the parameter estimation and sensitivity analysis for the system model.

The model contains many parameters, some of which are based on plant geometry such as masses, volumes, or areas, and others are parameters such as heat transfer coefficients, empirical parameters, or time constants. The former set of parameters are determined fairly easily and accurately from geometric data, but the second set can only be estimated or determined experimentally.

The sensitivity analysis is performed to determine which of the parameters the system is sensitive to in a relative sense. The effects of perturbations in the sensitive parameters is investigated on the closed-loop simulation. The sensitivity analysis was not performed for the parameters that are known with a high degree of certainty.

4.1 Parameter Estimation

4.1.1 Steam System Parameters

As follows is a complete list of parameters for the steam system of the plant model:

- V_d (m^3) - Drum volume
- V_r (m^3) - Riser volume
- V_{dc} (m^3) - Downcomer volume
- A_d (m^2) - Drum water/steam surface area

- m_t (kg) - Total metal mass
- m_t (kg) - Riser metal mass
- k (dimensionless) - Downcomer-riser friction coefficient
- V°_{sd} (m^3) - Volume of steam below water level with no condensation
- β (dimensionless) - Empirical parameter
- c_1 (kg/s-bar) - Steam valve admittance slope
- c_2 (kg/s-bar) - Steam valve admittance intercept

Of these parameters, all but the last five are based on plant geometry. The volumes, areas, and masses were determined using CAD model data for the J-OP S250. The friction coefficient and empirical parameter are both taken directly from the Åström and Bell paper [1]. This is because both parameters are not observable from the data and the system is relatively insensitive to them, which is shown later in this chapter.

The volume of steam below the water level, V°_{sd}, is a hypothetical parameter where no steam is condensed in the drum from the cold feedwater, so the quality of the two-phase mixture in the lower half of the drum is equal to the quality coming out of the risers at steady-state. Using this assumption, the volume is estimated with the following relation:

$$V^\circ_{sd} = \frac{\frac{1}{2}\alpha_r \rho_w V_d}{\rho_s(1-\alpha_r) + \alpha_r \rho_w} \tag{4.1}$$

where ρ_s and ρ_w are the densities of steam and water, respectively. This equation comes from the definition of quality being the mass ratio of steam mass to total mass. The quality used is the steady-state desired quality, and the densities of saturated steam and water are evaluated at the steady-state operating system pressure.

To determine the valve admittance parameters, pressure and flow data were taken from the plant over a wide range of operating conditions. The admittance of the valve, defined

as the flow divided by the pressure drop across the valve, is linear with valve position. The parameters c_1 and c_2 were determined from the experimental data using a linear curve fit, defined earlier in Equation 3.8. The data and the corresponding curve fit can be observed in Figure 4.1.

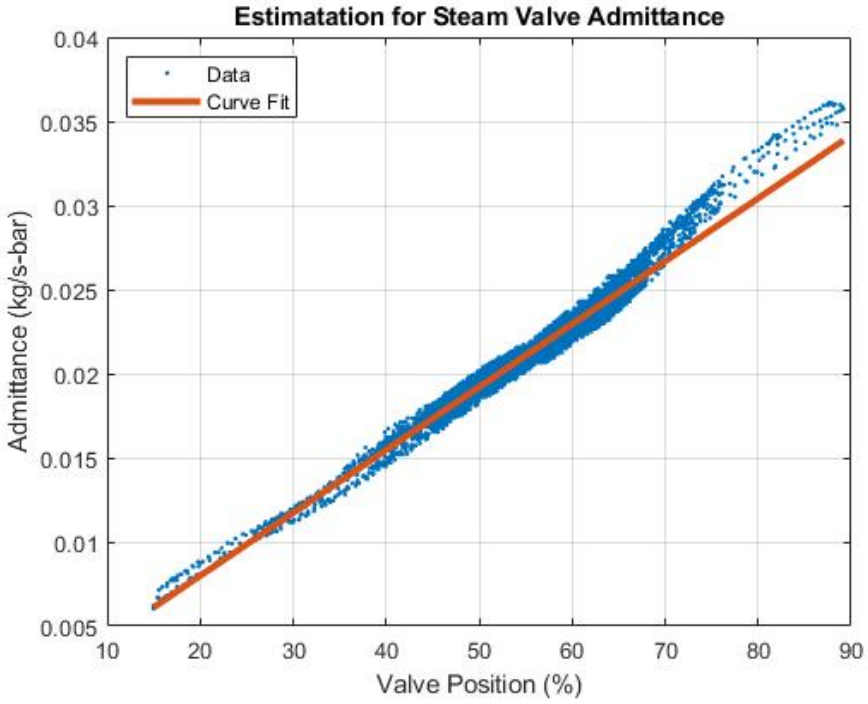

Figure 4.1: Steam valve admittance fit vs. data.

Since the curve fit for the valve admittance agrees well with the data, the relative degree of accuracy for c_1 and c_2 is high. Additionally, since the valve is a precision, electronic, steam globe valve, the repeatability of the data is very high. This means the parameters are unlikely to change over time - provided there is no mechanical degradation of the valve due to high steam flow or condensation in the steam lines.

4.1.2 Combustion System Parameters

As follows is a complete list of parameters for the combustion system of the plant model:

- UA_{fb} (kW/K) - Freeboard overall heat transfer coefficient
- UA_1 (kW/K) - Evaporator 1 overall heat transfer coefficient
- UA_2 (kW/K) - Superheater overall heat transfer coefficient
- UA_3 (kW/K) - Evaporator 2 overall heat transfer coefficient
- $delay_1$ (s) - Air input time delay
- $delay_2$ (s) - Fuel input time delay
- tau_{Q_c} (s) - Heat production time constant
- tau_{O_2} (s) - Oxygen time constant

The first four parameters for the combustion system affect the steady-state values such as the temperature profiles and the heat production. The methods for determining the heat transfer coefficients are detailed in the next two sections.

Freeboard Heat Transfer Coefficient

To estimate the freeboard heat transfer coefficient, the heat transferred through the freeboard must be estimated. It is calculated using a steady-state first law energy balance on the boiler. The first law balance is as follows:

$$\dot{Q}_s = \dot{Q}_c \tag{4.2}$$

where $\dot{Q}_s$ total heat provided to boil the steam, and $\dot{Q}_c$ is the total heat from each of the evaporators. The balance can be further broken down into sub components. The heat terms are defined as follows:

$$\dot{Q}_s = \dot{m}_s h_s - \dot{m}_{fw} h_w \tag{4.3}$$

and

$$\dot{Q}_c = \dot{Q}_{fb} + \dot{Q}_1 + \dot{Q}_2 \tag{4.4}$$

where the enthalpies, heats, and mass flows are defined in Chapter 3. To account for the steady-state first law balance, averages of the sensor data were calculated from extended steady-state operating runs to remove any transients. The heats from evaporators 1 and 2 are calculated by integrating the specific heat of N_2 using the temperature sensors before and after each evaporator, and multiplying them by the exhaust mass flow. Since all active combustion has completed by the time the exhaust gasses reach the first evaporator, the integrated specific heat accounts for all heat lost to the steam. When Equations 4.2, 4.3, and 4.4 are combined, the only unknown variable is the freeboard heat term, $\dot{Q}_{fb}$. This is isolated as follows:

$$\dot{Q}_{fb} = \dot{m}_s h_s - \dot{m}_{fw} h_w - (\dot{Q}_1 + \dot{Q}_2) \tag{4.5}$$

Using Equation 4.5, and the definition of the freeboard heat defined in Chapter 3 ($\dot{Q}_{fb} = UA_{fb}(T_{fb} - T_{sat.})$), the following equation determines the freeboard heat transfer coefficient:

$$UA_{fb} = \frac{\dot{m}_s h_s - \dot{m}_{fw} h_w - (\dot{Q}_1 + \dot{Q}_2)}{T_{fb} - T_{sat.}} \tag{4.6}$$

Evaporator Heat Transfer Coefficients

To determine the heat transfer coefficients for both of the evaporators, the following relations were used. First, the Number of transfer units were calculated for the heat exchangers:

$$NTU_i = \ln\left(\frac{T_i - T_{sat.}}{T_{i+1} - T_{sat.}}\right) \tag{4.7}$$

where the subscript on NTU_i corresponds to the evaporator, T_i corresponds to the temperature before the evaporator, and T_{i+1} corresponds to the temperature after the evaporator.

Equation 4.7 can be used to calculate the heat transfer coefficient as follows:

$$UA_i = \dot{m}_e C_p NTU_i \tag{4.8}$$

where $\dot{m}_e$ is the exhaust mass flow and C_p is the specific heat of N_2 evaluated at the mean temperature over the evaporator.

Superheater Heat Transfer Coefficients

To determine the heat transfer coefficients for the superheater, a similar method to the evaporators is used, but with slightly more complicated equations.

The effectiveness of the superheater must first be calculated:

$$\epsilon_{sh} = \frac{T_{sh,i} - T_{sh,o}}{T_{sh,i} - T_{sat.}} \tag{4.9}$$

where in this equation, i and o correspond to the temperature of the exhaust gasses going into and out of the evaporator, respectively.

The effectiveness is used to calculate the Number of Transfer Units as follows:

$$NTU_{sh} = \frac{1}{C_r - 1} \ln\left(\frac{\epsilon - 1}{\epsilon C_r - 1}\right) \tag{4.10}$$

where C_r is the ratio of the specific heat of the exhaust gasses to the specific heat of the superheated steam.

Finally, the heat transfer coefficient can be calculated as follows:

$$UA_{sh} = \dot{m}_e C_p NTU_{sh} \tag{4.11}$$

where $\dot{m}_e$ and C_p are the same as for the evaporators.

Dynamic Combustion Parameters

The final four parameters all determine the dynamic behavior of the combustion system. These parameters were determined using step responses in the fuel and air inputs to the J-OP S250. The inputs and outputs were recorded, and the parameters were determined by fitting the input/output data dynamics. The experimentally determined parameters fit the input output data well, but also have some degree of uncertainty.

4.2 Sensitivity Analysis

Sensitivity analysis is performed for all the parameters that are not based on plant geometry data. This analysis is intended to determine the sensitivity of the plant outputs - heat production, oxygen content, steam drum pressure, and steam drum level - to changes in the system parameters. Once the sensitive parameters are found, optimal controllers are tested to observe their performance with the perturbed sensitive parameters.

4.2.1 Sensitivity Definition

All sensitivities are calculated using the relative sensitivity function, which is defined in *Introduction to System Sensitivity Theory* by Paul M. Frank as follows [9]:

$$\frac{\Delta y(t,\mathbf{a})}{y(t,\mathbf{a}_0)} = \sum_{j=1}^{n} \bar{\sigma}_{aj}(t,\mathbf{a}_0)\frac{\Delta a_j}{a_{j0}} \tag{4.12}$$

where $y(t,\mathbf{a})$ is the output, $\mathbf{a}$ is the list of parameters, and $\bar{\sigma}_{aj}(t,\mathbf{a}_0)$ is the relative sensitivity function. Since only one parameter is varied at a time for these calculations, the expression is simplified as follows:

$$\frac{\Delta y(t,a)}{y(t,a_0)} = \bar{\sigma}_{a}(t,a_0)\frac{\Delta a}{a_0} \tag{4.13}$$

Equation 4.13 is rearranged to calculate the relative output sensitivity as follows:

$$\bar{\sigma}_a(t, a_0) = \frac{\Delta y(t, a)}{y(t, a_0)} \frac{a_0}{\Delta a} \tag{4.14}$$

Since the model equations are fairly complicated, the relative sensitivity function is evaluated using the simulation to calculate the change in the outputs.

4.2.2 Open Loop Sensitivity

The open-loop sensitivities are calculated by first determining all of the nominal inputs required to maintain the appropriate output feedback variables in the closed loop simulation during steady-state with the nominal parameters. Once these inputs are determined, they are fed into the open loop model. One parameter is varied at a time to determine all of the output sensitivities for that specific parameter.

Steady-State Output Sensitivities

The steady-state output variables are the steam drum pressure, steam drum level, heat production, and the exhaust oxygen content. Figures 4.2, 4.3, and 4.4 show the percent deviation in the pressure, heat, and oxygen output variables for a 20% change in each of the parameters. The time delay and time constant parameters from the combustion system are ignored for these plots because they only affect the dynamics and not the steady-state values of the outputs.

In Figures 4.2 and 4.3, the most sensitive parameters for the pressure and heat are the freeboard heat transfer coefficient and the valve admittance slope parameters. The output deviations for UA_{fb} are approximately 20% and 6% for the pressure and heat, respectively, and the deviations for c_1 are -12% and less than 1% for the pressure and heat, respectively.

Figure 4.4 shows the output deviation in the oxygen content. The oxygen content has zero sensitivity to all of the system parameters because it is uniquely determined by the ratio of the air to fuel mass flows and the fuel chemical composition.

Figure 4.6 shows the output deviations for pressure, heat, and oxygen for a $+/-$ 50%

change the freeboard heat transfer coefficient. This figure illustrates the relatively linear relationship of the sensitivities to the parameters. This analysis was performed to determine if any of the sensitivity slopes go through an inflection point near the operating point. If this were the case, the linearized gains could have the wrong sign, and could lead to controller instability. As seen in Figure 4.6, the slopes remain consistent. Similar analysis was performed for the other parameters with different values but consistent slope relationships.

The steam drum level sensitivity is calculated differently from the other three outputs. Since the level dynamics contain a neutrally stable pole, the open-loop simulation diverges for any change of parameters. Since the system does not converge to a constant final value, the same method of output error sensitivity cannot be used. However, once the transients settle out, the level diverges at a constant rate. Using this fact, a characteristic time of 100 minutes is used to determine the final output value at that time. If a longer time was chosen, the output values would scale with it, but this method is used to observe a relative sensitivity between the different parameters. As can be seen in Figure 4.5, the level is most sensitive to the freeboard heat transfer coefficient and the hypothetical drum volume parameter, V°_{sd}. The deviations are -40% and 50% for these parameters, respectively.

The most sensitive parameters for the steady-state outputs are UA_{fb}, c_1, and V°_{sd}. Effects of perturbations in these parameters is investigated in Section 4.2.3.

Output Settling Time Sensitivities

The output settling time sensitivities are calculated using the same percent difference method as for the steady-state output sensitivities. Figures 4.7, 4.8, 4.9, and 4.10 show the percent difference in the settling time for the outputs for a 20% change in each of the parameters. The dynamic parameters are included for this section.

For the pressure settling time, as seen in Figure 4.7, the freeboard heat transfer coefficient and the valve admittance slope are again the most dominant parameters, with output deviations of 20% and -32%, respectively. This follows with what would be expected from the state equations. The freeboard heat transfer coefficient is the most dominant gain for

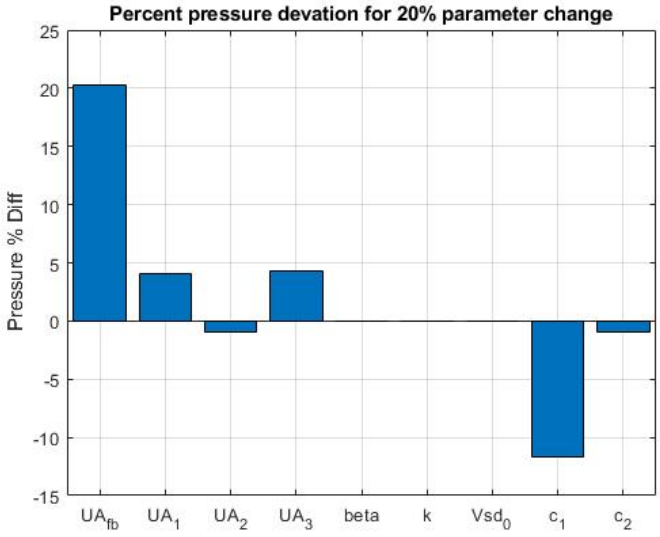

Figure 4.2: Pressure output deviations for 20% change in each parameter.

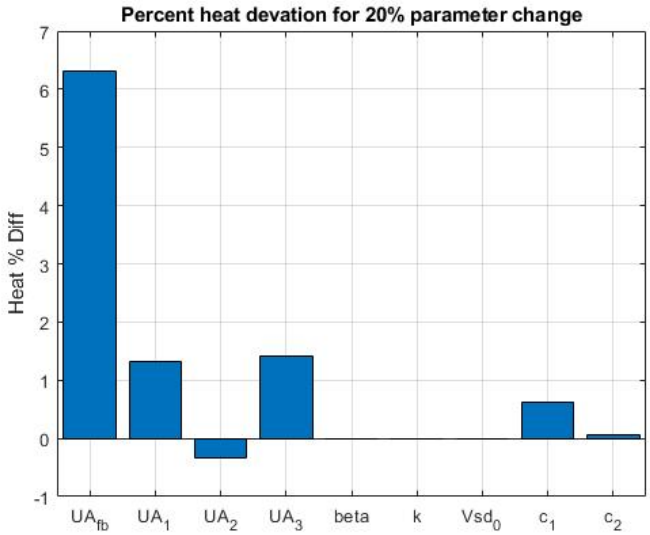

Figure 4.3: Heat output deviations for 20% change in each parameter.

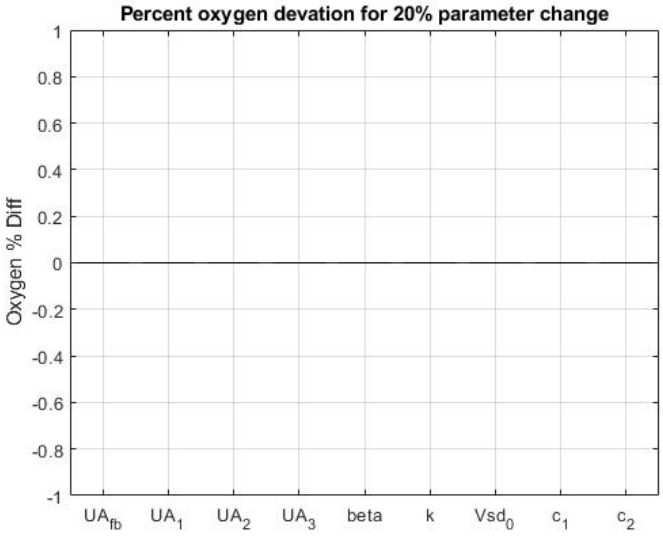

Figure 4.4: Oxygen output deviations for 20% change in each parameter.

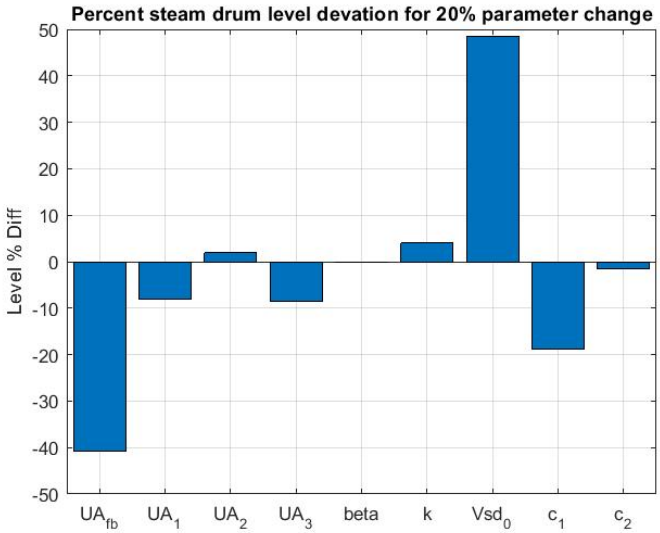

Figure 4.5: Steam drum level deviations for 20% change in each parameter.

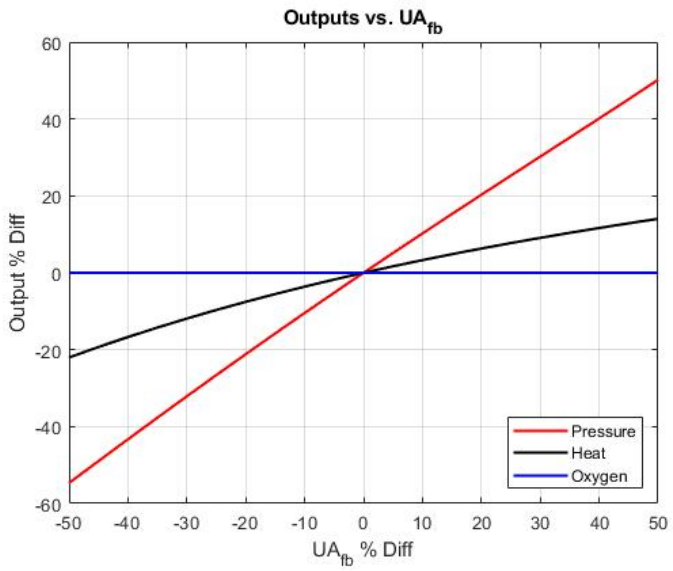

Figure 4.6: Output deviations vs. UA_{fb}.

the heat production and the valve admittance slope directly affects the open-loop pole of the pressure dynamics.

The sensitivity of the heat settling time, shown in Figure 4.8, is mostly dominated by the heat time constant, with an output deviation of 14%. All other parameters result in deviations less than 5% in magnitude.

The oxygen settling sensitivity, similar to the heat, is dominated by its respective time constant, with a deviation of 12%, as shown in Figure 4.9. The fuel time delay also affects the oxygen settling time, with a deviation of 8%.

Settling time sensitivity for the drum level is again handled slightly differently. The level is unstable, but since it diverges at a constant rate, the derivative of the drum level converges to a constant final value. By these means, the drum level settling time is defined as the time for 2% settling of its time derivative. As seen in Figure 4.10, the most dominant parameters for the sensitivity are the heat transfer coefficients and the valve admittance slope parameter. The output deviations for UA_{fb} and c_1 are 44% and 38%, respectively.

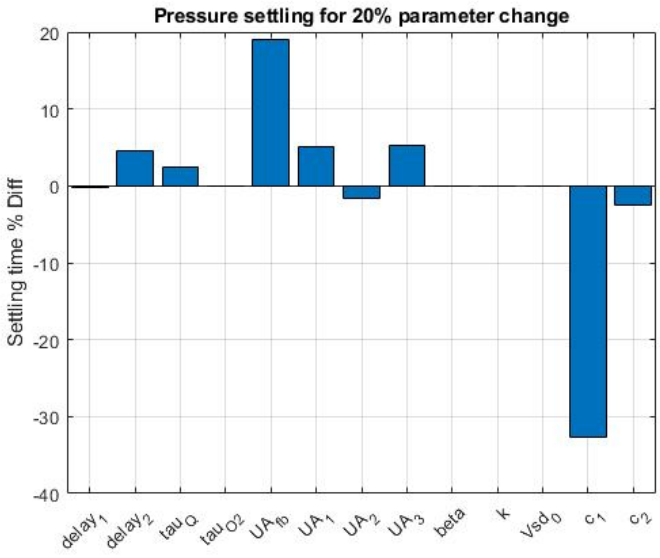

Figure 4.7: Pressure settling time deviations for 20% change in each parameter.

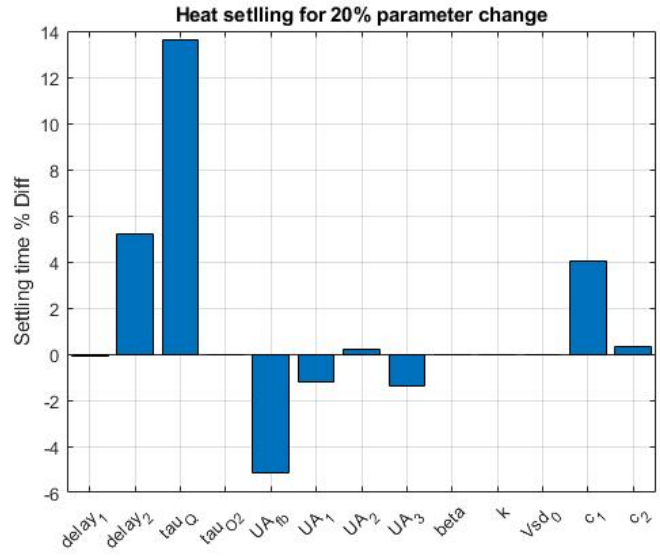

Figure 4.8: Heat settling time deviations for 20% change in each parameter.

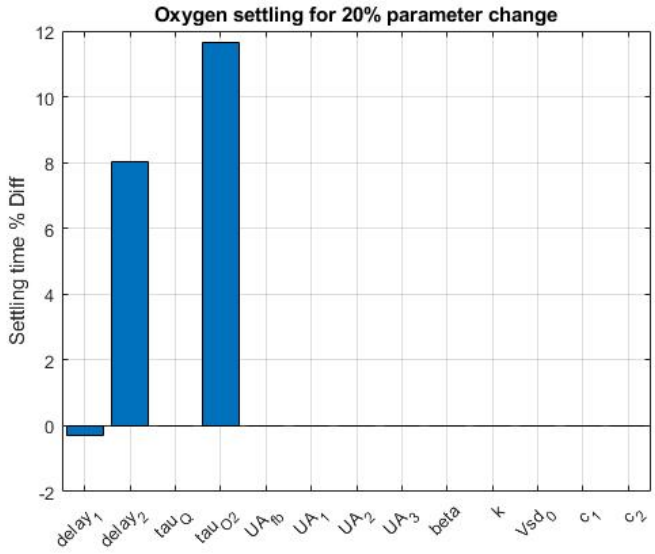

Figure 4.9: Oxygen settling time deviations for 20% change in each parameter.

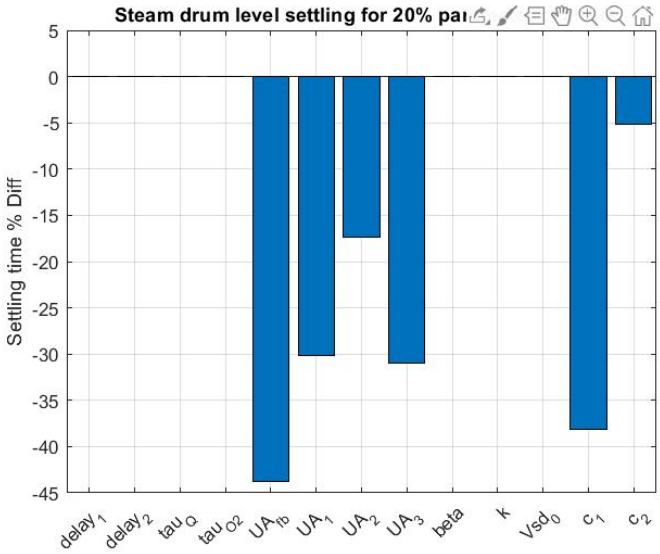

Figure 4.10: Steam drum level settling time deviations for 20% change in each parameter.

This section shows the sensitivity of the settling time to the parameters. As determined by this analysis, the most sensitive parameters are again the freeboard heat transfer coefficient the valve admittance slope parameters. The time delays and time constants also affected the system settling, but with relatively low magnitude output deviations.

4.2.3 Closed-Loop Sensitivities

The open-loop sensitivity analysis provides context to choose the appropriate parameters to perturb for the closed-loop simulations. In these simulations, the parameters in the model are changed, but the optimal controllers derived from the nominal model parameters are unchanged. The derivation of the optimal controllers is shown in Chapter 5, but the closed-loop responses with these controllers are shown here to illustrate the sensitivities.

UA_{fb} Closed-Loop Response

Closed loop simulations were investigated for all of the sensitive parameters, but UA_{fb} is shown because it had the most dominant effect on the closed-loop simulations. Figures 4.11, 4.12, and 4.13 show the closed loop comparison between the nominal response, and the response with UA_{fb} increased by 30%. These figures indicate that even with the perturbed parameter the closed-loop responses are similar, and the simulation is stable and reached the appropriate outputs. These results indicate the control scheme is robust to changes in sensitive parameters and should be able to handle changes in the real system. The notable differences in the Air Input and O_2 Response sub figures, seen in Figure 4.11, is a result of the slow eigenvalue corresponding to the oxygen error integrator. This is discussed further in Chapter 6.

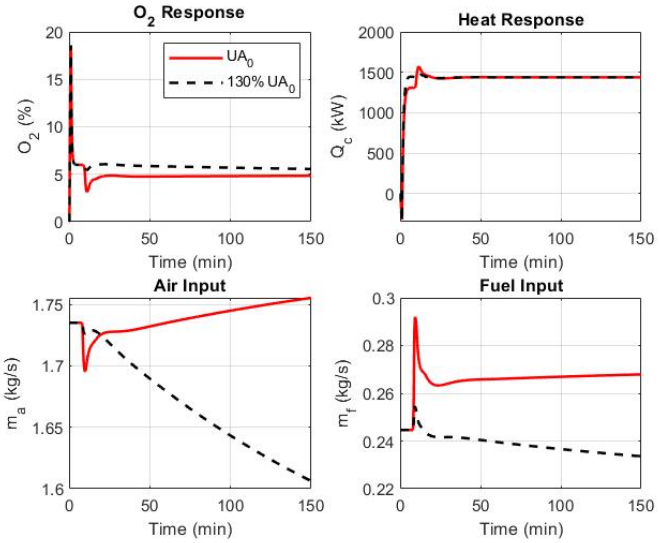

Figure 4.11: Combustion states and inputs.

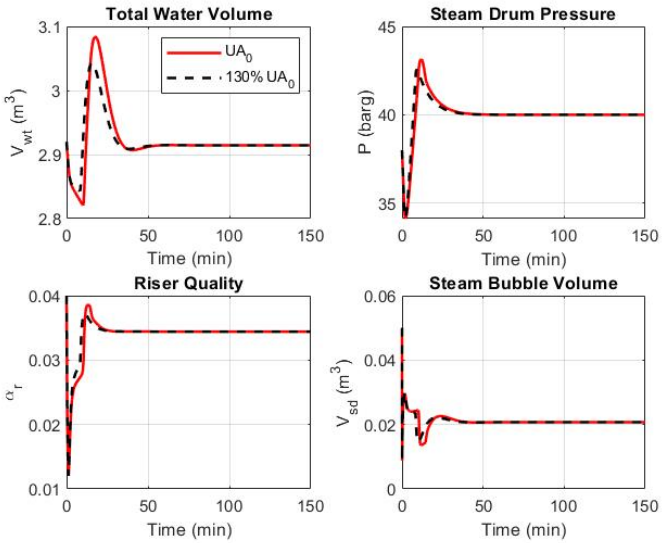

Figure 4.12: Steam system states.

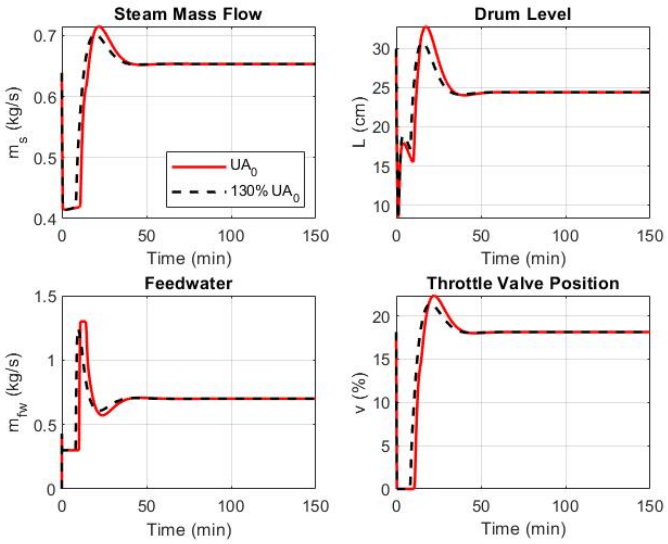

Figure 4.13: Steam system inputs/outputs.

Various Closed-Loop Parameter Changes

The open-loop analysis shows that the steam system pressure is not sensitive to the three steam system parameters, k, β, and V°_{sd}. However, these parameters affect the steam drum level, and also have a high degree of uncertainty. Figures 4.14, 4.15, and 4.16 show the closed-loop steam system output response for a wide range of these parameters. Figure 4.15 shows that the system is minimally sensitive to β. The outputs are more sensitive to k and V°_{sd}, but the settling time is fairly stable and the controllers still achieve the appropriate outputs without excessive oscillation or overshoot.

Figure 4.17 shows the closed loop response to varying the valve admittance slope parameter by $+/-$ 50%. The steam pressure is sensitive to this parameter, but the plots illustrate that even with a wide range of the parameter, the controllers still get the outputs to converge without much longer settling time or output overshoot, especially in the steam drum pressure.

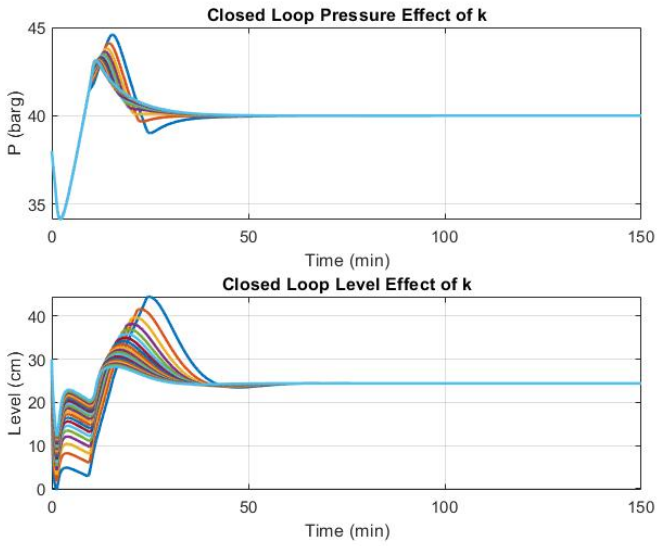

Figure 4.14: k varied from 100 to 2000. Nominal $k_0 = 1000$.

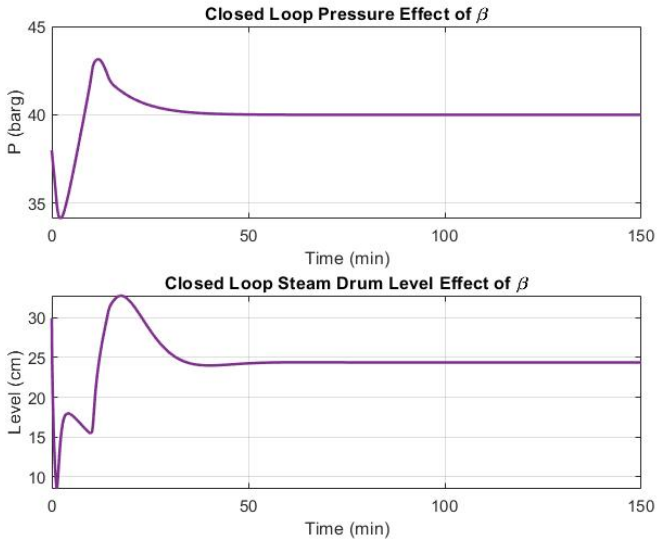

Figure 4.15: β varied from 0.0 by 0.1 to 1.0. Nominal $\beta_0 = 0.3$.

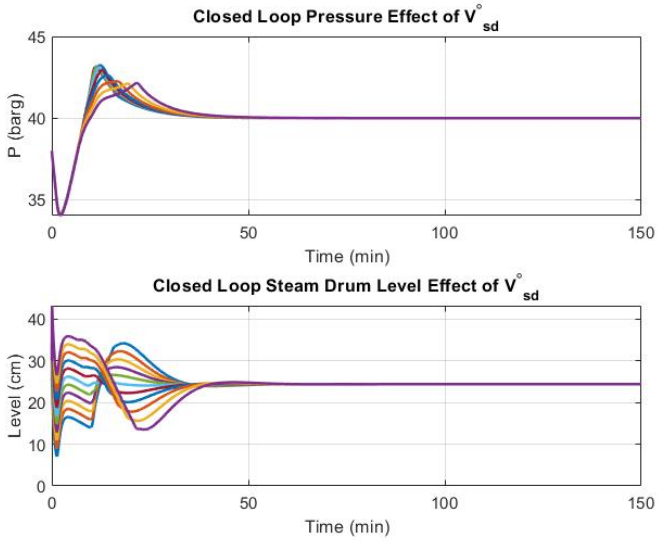

Figure 4.16: V°_{sd} varied from 0.0 by 0.05 to 0.4. Nominal $V^{\circ}_{sd\,0} = 0.03$.

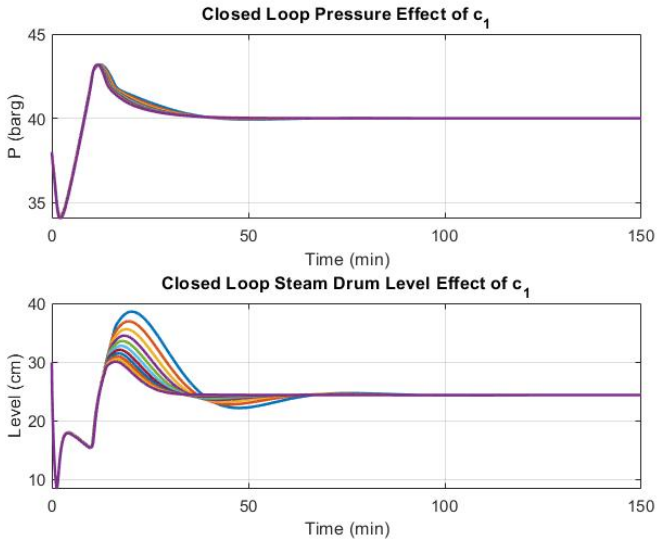

Figure 4.17: c_1 varied by $+/-$ 50%. Nominal $c_{1\,0} = 0.000374$.

4.2.4 Sensitivity Analysis Conclusion

The goal of this analysis is to determine the sensitivity of the model outputs to the parameters. For the parameters that are easy to determine, the sensitivities are not calculated because the parameters are known with a high degree of certainty. For the purposes of system modeling and controller design, two possible outcomes from the sensitivity analysis are desirable. If a parameter has uncertainty, then it would be convenient if the model is insensitive to it. Otherwise, if the model is sensitive to a parameter with uncertainty, then the controllers should be able to compensate for this. Based on the analysis shown above, all of the uncertain parameters are either fairly unimportant, or have small impact on the closed-loop simulations. The most important parameters from an open-loop sensitivity perspective are the freeboard heat transfer coefficient, UA_{fb}, and the valve admittance slope parameter, c_1. Closed loop simulations showed the controllers can compensate for changes in these two parameters.

Chapter 5
MODEL LINEARIZATION AND CONTROLS

This chapter provides a detailed description of the linearization of the non-linear system model, and the subsequent controls derived based on the linear model.

5.1 Linearization

The classical control techniques used in this chapter require linear system model equations, but the system model derived and shown in Chapter 3 is both non-linear and implicit. The system equations must be linearized around an operating point to utilize optimal MIMO control tuning techniques. The system is assumed to always operate at or near the chosen equilibrium points, which makes the linear model suitable for control purposes.

5.1.1 Combustion System

As shown in Chapter 3, the combustion system is composed of input time delays, non-linear equations for oxygen and heat production, and first order low pass filters for the outputs. The equations that determine the heat production are non-linear and implicit in terms of the freeboard temperature, while the equations for the oxygen content in the exhaust are uniquely determined by the ratio of fuel to air and the molecular composition of the fuel molecule.

To determine the linear system, the gains between the fuel and air inputs to the heat and oxygen outputs must be determined. This is accomplished by calculating the implicit partial derivatives, then solving for the partial derivatives and evaluating them at the appropriate points. The code for this linearization can be seen in Appendix F. The steam mass flow and steam temperature disturbance gains are also calculated in this manner. This is shown with

matrices in the following equation:

$$\begin{bmatrix} O_2 \\ Q_c \end{bmatrix} = \begin{bmatrix} \frac{\partial O_2}{\partial \dot{m}_a} & \frac{\partial O_2}{\partial \dot{m}_f} & \frac{\partial O_2}{\partial T_{sat}} & \frac{\partial O_2}{\partial \dot{m}_s} \\ \frac{\partial Q_c}{\partial \dot{m}_a} & \frac{\partial Q_c}{\partial \dot{m}_f} & \frac{\partial Q_c}{\partial T_{sat}} & \frac{\partial Q_c}{\partial \dot{m}_s} \end{bmatrix} \begin{bmatrix} \dot{m}_a \\ \dot{m}_f \\ T_{sat} \\ \dot{m}_s \end{bmatrix} \tag{5.1}$$

where the inputs and the outputs are deviations from the nominal values, not the actual values. This equation shows the linearization of the combustion equations, but it does not include the time delays or the low pass filters. The low pass filters are linear and can be added directly the outputs, but the time delays must be linearized first.

To handle the non-linearity associated with the time delays, Padé approximations are used. To determine the necessary order of the approximations, accuracy is balanced with system order to arrive at a second order approximation for both the input time delays. A step response comparison of the true delays and first and second order Padé approximations can be seen in Figure 5.1.

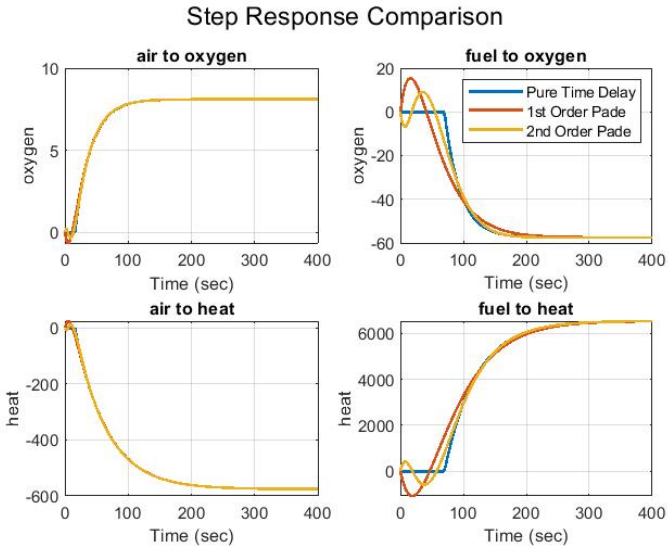

Figure 5.1: Padé approximation step response comparison.

Once the Padé approximations are used for the time delays, the entire system can be combined to form the following state-space system:

$$\dot{x_c} = A_c x_c + B_c u_c \tag{5.2}$$

$$y_c = C_c x_c + D_c u_c \tag{5.3}$$

where x_c is the combustion state vector, u_c is the combustion input vector, and y_c is the combustion output vector. In this system, all of the states have non-physical meaning because the Padé approximations create intermediate states according to the order of their approximation. The input and output vectors are defined as follows:

$$u_c = \begin{bmatrix} \dot{m}_a \\ \dot{m}_f \\ T_{sat} \\ \dot{m}_s \end{bmatrix} \tag{5.4}$$

$$y_c = \begin{bmatrix} O_2 \\ Q_c \end{bmatrix} \tag{5.5}$$

The matrices shown in Equations 5.2 and 5.3 are defined as follows:

$$A_c = \begin{bmatrix} a_{11} & 0 & a_{13} & 0 & a_{15} & 0 \\ 0 & a_{22} & a_{23} & 0 & a_{25} & 0 \\ 0 & 0 & a_{33} & a_{34} & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a_{55} & a_{56} \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{5.6}$$

$$B_c = \begin{bmatrix} b_{11} & b_{12} & 0 & 0 \\ b_{21} & b_{22} & b_{23} & b_{24} \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.7}$$

$$C_c = \begin{bmatrix} c_{11} & 0 & 0 & 0 & 0 & 0 \\ 0 & c_{22} & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.8}$$

$$D_c = \begin{bmatrix} 0 \end{bmatrix} \tag{5.9}$$

In this state space, the variables T_{sat} and $\dot{m}_s$ are treated as inputs, when in reality they are outputs of the steam system. Perturbation analysis was performed to determine how sensitive the combustion system is to these inputs, and it was determined they could be treated as disturbance inputs to decouple the dynamics of the steam and combustion systems.

5.1.2 Steam System

The steam system linearization is much easier than the combustion system since the differential equations are both explicit and do not contain any time delays. The partial derivatives can be calculated explicitly and evaluated at the equilibrium points to determine the linear perturbation state-space. The linearization code for the steam system is shown in Appendix E. The linear steam system state space is shown below:

$$\dot{x_s} = A_s x_s + B_s u_s \tag{5.10}$$

$$y_s = C_s x_s + D_s u_s \tag{5.11}$$

where x_s is the steam state vector, u_s is the steam input vector, and y_s is the steam output vector. The state vector, input vector, and output vector are defined as follows:

$$x_s = \begin{bmatrix} V_{wt} \\ P \\ \alpha_r \\ V_{sd} \end{bmatrix} \tag{5.12}$$

$$u_s = \begin{bmatrix} \dot{m}_{fw} \\ v \\ \dot{Q}_c \\ T_{fw} \\ P_d \\ \dot{m}_{st} \end{bmatrix} \tag{5.13}$$

$$y_s = \begin{bmatrix} P \\ L \end{bmatrix} \tag{5.14}$$

The matrices shown in Equations 5.10 and 5.11 are defined as follows:

$$A_s = \begin{bmatrix} 0 & a_{12} & 0 & 0 \\ 0 & a_{22} & 0 & 0 \\ 0 & a_{32} & a_{33} & 0 \\ 0 & a_{32} & a_{43} & a_{44} \end{bmatrix} \tag{5.15}$$

$$B_s = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} & b_{16} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} & b_{26} \\ b_{31} & b_{32} & b_{33} & b_{34} & b_{35} & b_{36} \\ b_{41} & b_{42} & b_{43} & b_{44} & b_{45} & b_{46} \end{bmatrix} \tag{5.16}$$

$$C_s = \begin{bmatrix} 0 & 1 & 0 & 0 \\ c_{21} & c_{22} & c_{23} & c_{24} \end{bmatrix} \tag{5.17}$$

$$D_s = \begin{bmatrix} 0 \end{bmatrix} \tag{5.18}$$

Unlike the combustion system, the steam system dynamic states are all physical values. While the states are physical, only the steam pressure is measured. The other output variable for the steam system is the steam drum level, which is a linear combination of all four of the dynamic states. The steam pressure and steam drum level feedback make the system observable for a state estimator.

As can be seen from the input vector u_s, shown in Equation 5.13, there are six inputs to the steam system. The first three are controlled inputs, and the last three are disturbances. The feedwater mass flow, $\dot{m}_{fw}$, and the valve position, v, are both explicitly controlled, while the heat input, $\dot{Q}_c$, is one of the outputs of the combustion system.

5.2 Controls

With the linearized versions of the combustion and steam systems available, a clear picture of the structure of the equations can be discerned. The combustion system is almost entirely decoupled from the steam system, except for the steam saturation temperature and steam mass flow. The steam system, however, is entirely dependent on the combustion system because of the heat production term.

Due to the structure of these equations and the relative magnitudes of the eigenvalues, the entire system lended itself well to cascade control, with the combustion system being on the inner loop and the steam system on the outer loop. This control structure can be seen in Figure 5.2.

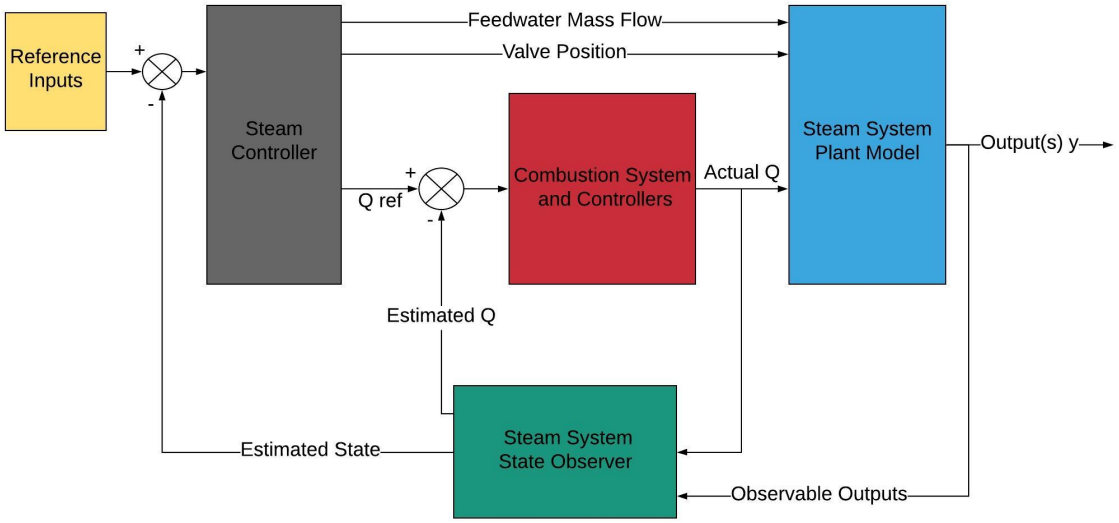

Figure 5.2: Overall system control structure showing cascade control and state estimation for steam and combustion systems.

5.2.1 Combustion System Controls

For the combustion system, the nature of the cascade control combined with the variable reference inputs lead to the following requirements for the controls:

- Zero steady-state error
- Ability to reject disturbances without oscillation
- Ability to reject noise in the feedback sensors

A controller structure that accommodates these requirements must be determined. Given the relatively high system order of six, and the MIMO nature of the system, an optimal Linear Quadratic Regulator (LQR) controller with a Kalman filter estimator is combined with pure integrators on the reference errors to create the full controller. This is referred to as a Linear

Quadratic Gaussian (LQG) servo controller. A diagram of this controller can be seen in Figure 5.3. This controller estimates the states based on the current inputs and feedback signals, and determines the new outputs based on the derived control law. The estimator is tuned by balancing noise in the feedback signals vs. noise in the system inputs. This means that if there is a lot of feedback noise, the estimator relies more heavily on the model to determine the states, and vice versus. The controller is tuned similarly by balancing state error with control inputs.

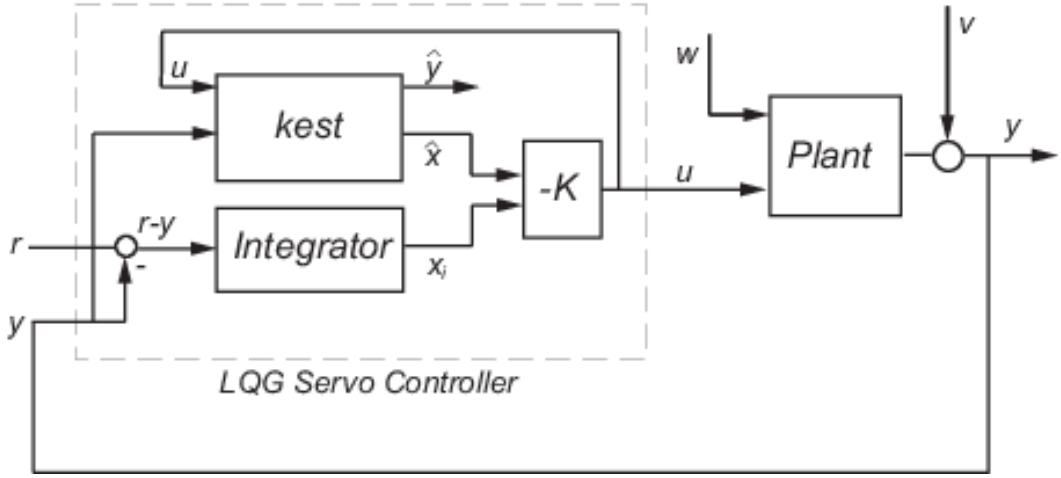

Figure 5.3: LQG controller diagram [13].

For the combustion system, the reference inputs are the desired oxygen content in the exhaust, and the desired heat production. The desired oxygen content is a constant value of 5.0%. This was determined by the testing engineers to optimize for emissions controls. The heat setpoint is determined by the steam system, hence the cascade control structure. The outputs of this controller are the fuel mass flow and the air mass flow. In Figure 5.3, the variables r, y, and u are the reference inputs, feedback variables, and input variables, respectively. The terms $\hat{x}$ and $\hat{y}$ are the predicted states and outputs, respectively. The terms w and v are disturbances to the system model and the feedback sensors, respectively. The labeled blocks *kest*, *Integrator*, and *Plant* are the optimal Kalman filter estimator, the

integrators on the reference errors, and the system plant model, respectively. Finally, the gain K is the optimal controller gain matrix.

The dynamic structure of the controller and the control law can be observed as follows:

$$\frac{d}{dt}\begin{bmatrix}\hat{x}\\ x_i\end{bmatrix} = \begin{bmatrix}A - BK_x - LC & -BK_i\\ 0 & 0\end{bmatrix}\begin{bmatrix}\hat{x}\\ x_i\end{bmatrix} + \begin{bmatrix}0 & I\\ I & -I\end{bmatrix}\begin{bmatrix}r\\ y\end{bmatrix} \tag{5.19}$$

$$u = \begin{bmatrix}-K_x & -K_i\end{bmatrix}\begin{bmatrix}\hat{x}\\ x_i\end{bmatrix} \tag{5.20}$$

where the concatenated vector $\begin{bmatrix}\hat{x} & x_i\end{bmatrix}^T$ is the state vector associated with the controller. The first term, $\hat{x}$, is the estimated system states and the second term, x_i, is the integrated error between the feedback variables and the references. The states are used along with the controller gain matrix to determine the feedback inputs, as shown in Equation 5.20.

The control law (Equation 5.20) is derived using a Linear Quadratic Integrator (LQI) [23]. The law is derived by optimizing the cost function equation shown below:

$$J = \int_0^\infty (x^T Q x + u^T R u)\, dt \tag{5.21}$$

The weighting matrices Q and R are chosen to tune the controller response to balance state deviation with control input. These matrices were chosen experimentally.

The L matrix in Equation 5.19 is the optimal observer gain matrix and is determined using a very similar optimization to the control law. Instead of balancing control input with state deviation, it balances state estimation error with model noise.

Figure 5.4 shows the closed loop response of the controllers showed in Equations 5.19 and 5.20 applied to the linear combustion model with initial condition error and zero reference inputs. The controllers are able to reject the initial condition error and bring the system states to zero. Additionally, the state estimates converge to the actual system outputs, which is expected with an estimator given no disturbances.

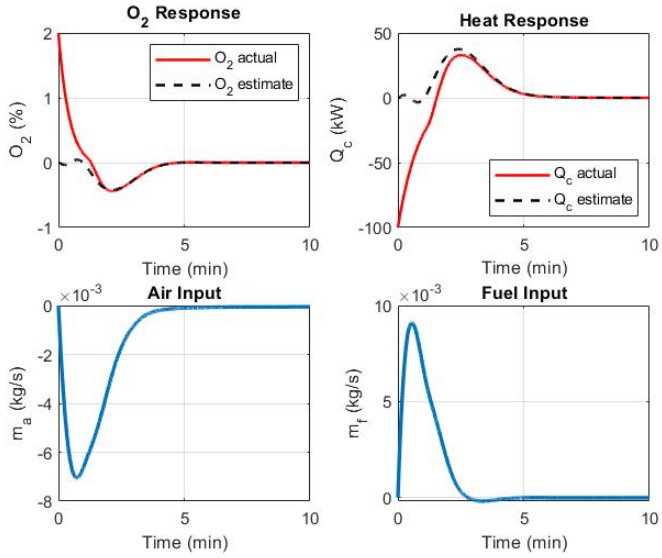

Figure 5.4: Closed-loop linear combustion response.

5.2.2 Steam System Controls

The steam system controls follow the same exact structure of controller as the combustion system, but with the different inputs, outputs, and matrices. The two systems also utilize different weighting matrices for appropriate controller tuning. The steam system controller was tuned such that the reference setpoint changes sent to the combustion system did not change faster than the combustion controller bandwidth. This is one of the limitations of the cascade control structure; the outer loop must operate slower than the inner loop and cannot be tuned arbitrarily fast. For the cascade control system to work, the combustion system response to changes in the heat reference must appear instantaneous to the steam system. Otherwise, the combustion system dynamics will overlap with the steam system and the overall dynamics will be contaminated and the assumption of optimal control cannot be used. The appropriate gain matrices for accomplishing this were tested both in Matlab and on the real plant.

In reality, there is always overlap between the dynamics, from both errors in the model parameters and unmodeled dynamics, which affect the closed-loop behavior of the systems. However, if the gross behavior of the two systems operate on the two different timescales, then the cascade control structure works well. The closed-loop system responses from the simulations and collected data are shown and contrasted in Chapter 6.

5.3 Description of Original Plant Controllers

The original controllers on the power plant consist of a series of Single-Input, Single-Output (SISO) controllers that regulate the four main outputs: oxygen content, steam drum pressure, steam drum level, and steam mass flow. The steam mass flow is used instead of the heat output because the heat was not calculated originally. These controllers were implemented and tuned by hand during the commissioning of the power plant. The four control loops are as follows:

- The steam drum pressure is controlled by the steam throttle valve using a standard PID controller.

- The steam drum level is controlled by setting the feedwater mass flow setpoint equal to the current steam mass flow, plus static offsets determined by discrete level switches on the steam drum.

- The steam mass flow output is controlled by the fuel metering augers using a standard PID controller.

- The exhaust oxygen level is controlled by a timer that adjusts the over fire air by discrete values.

This control scheme was developed by a team of engineers and tuned over several months to find the best tradeoff between steady-state error and disturbance rejection. Under normal operation, this control scheme works well. It meets the setpoints, and adequately manages

to reject disturbances in the fuel stream. However, under some conditions, due to the fundamentally coupled nature of the dynamics, the SISO controllers have difficulty with rejecting cyclic oscillations that arise in the outputs. Without operator oversight, these oscillations can persist for an extended period of time without attenuation. The engineering team tried to remove the oscillations by lowering the controller bandwidth, but this caused the controllers to be unable to respond to disturbances in the fuel input stream. The coupling and cyclic oscillation behavior is further detailed in Appendix B.

Fundamentally, the problem of tuning the four independent control loops is intractable due to the high number of required parameters (12, 3 for each controller), and the slow nature of the dynamics. When a change is made to a parameter, the operator has to wait for the system to settle. And considering the 12 degree-of-freedom parameter space, this requires a tremendous amount of time to iterate by hand to find good control gains. This is one of the reasons the optimal MIMO control scheme was investigated.

Chapter 6

RESULTS AND ANALYSIS

This chapter shows the results of the closed loop plant testing and compares the optimal controller performance to the original SISO controllers using real data from the J-OP S250.

6.1 Testing Objective

The purpose of the plant testing was to satisfy the following list of objectives:

Objective I: Verify optimal controller stability and compare to simulation.

Objective II: Compare optimal to SISO controller performance for output error standard deviation during steady-state.

Objective III: Compare optimal to SISO controller performance for output error standard deviation for disturbance rejection.

If the optimal controllers work well, the data should show an improvement in steady-state output error and disturbance rejection. Additionally, if a good agreement can be made between the simulated closed-loop system and the collected data, the confidence in the model can be increased. This is helpful when making assumptions about better controller tuning in the future. No strict definitions for the quality of the agreement between the simulated and collected data are defined in this thesis. Instead, qualitative comparisons are drawn to illustrate where improvements can be made.

6.2 Implementation

To implement the optimal control scheme on the power plant, the controllers were discretized in Matlab at the 100 millisecond sampling period using Tustin's method. These discrete controllers were then incorporated into the existing automation code running the J-OP S250. The automation for the plant runs on a SIEMENS PLC. Once the controls were written onto the PLC, they were thoroughly tested in software and were tuned appropriately for the combustion and steam systems. The implementation, testing, and tuning, is further detailed in Appendix A.

6.3 Objective I: Stability and Simulation Comparison

6.3.1 Steady-State Comparison

To compare the collected data to the simulation, the plant was tested by operating with the original controllers and then turning on the optimal controllers. During this test, no intentional load changes or disturbances were applied to the system. This was simulated as well, and the comparison is shown in Figures 6.1, 6.2, and 6.3. The difficult part of this simulation was matching the initial conditions. The states in the simulation had to be the same as in the data prior to when the optimal controllers were turned to make a fair comparison. This was accomplished by feeding the combustion data inputs from the SISO controller data into the open loop combustion plant. The optimal combustion controllers were turned on in the simulation at the corresponding time point to the data. There are some discrepancies with the steam system due to the semi-stable steam dynamics. If the open loop steam inputs were fed in, the outputs would drift, so the controllers were left on. The data for the dryer pressure, turbine steam mass flow, and feedwater temperature were fed into the simulation for disturbances.

Important observations concerning the experimental data are as follows:

- The high frequency noise on the heat can attributed to two sources. The first source

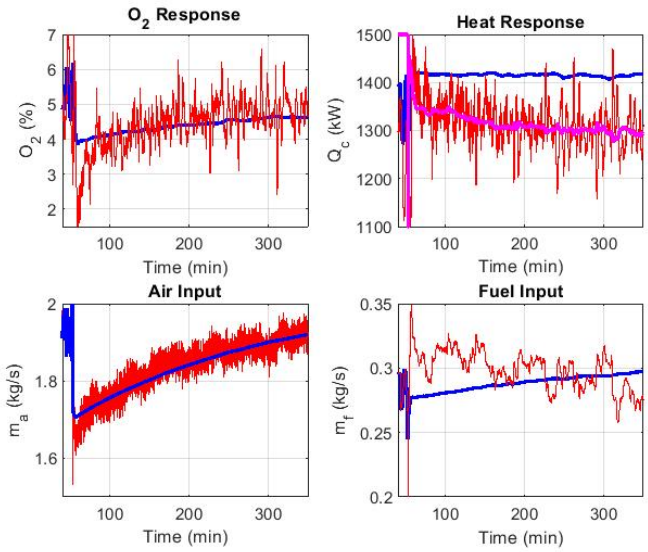

Figure 6.1: Combustion inputs/outputs for simulation vs. data. Data is red and simulation is blue. The magenta curve is the data heat setpoint.

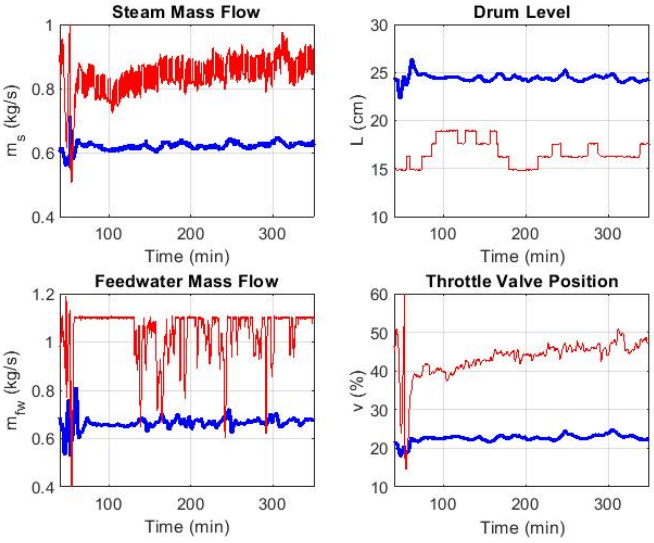

Figure 6.2: Steam inputs/outputs for steady-state closed loop testing of the optimal controllers compared to simulation.

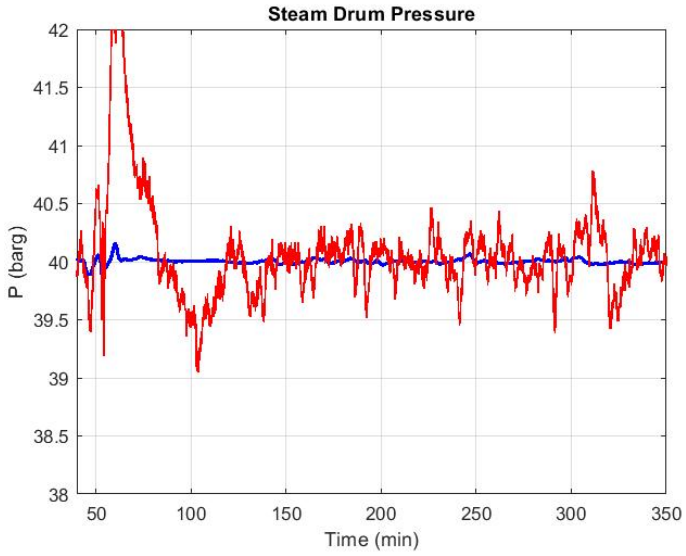

Figure 6.3: Steam drum pressure for steady-state closed loop testing of the optimal controllers compared to simulation.

of noise is unintended disturbances as a result of the fuel. This noise has a period of approximately 6 minutes, and a peak-to-peak amplitude of approximately 100 kW. The other source of noise is from the ID fan creating a disturbance in the exhaust mass flow, which has a period of approximately 20 seconds, and a peak-to-peak amplitude of 20 kW.

- The high frequency noise on the air mass flow input is due to fluctuations in the primary air fan mass flow.

- The long transient in the settling time observed in the O_2 Response is a result of improper tuning on the combustion system controller. This slow eigenvalue is observed in the simulation of the system and controllers when there is a reference change to the heat setpoint.

- The steam mass flow has a square-wave like noise with an amplitude of approximately 0.05 kg/s. This noise is caused by stiction in the steam throttle valve.

- The discrete nature of the drum level feedback is caused by the drum level sensor. This sensor only has a sensitivity of 1.27 cm. This sensor was also attached using very small piping external to the steam drum which causes some attenuation of the level due to stiction in the mechanical float. This poor feedback from the drum level attributes to the strange behavior in the feedwater pump observed in the Feedwater Mass Flow sub figure in Figure 6.2.

Important observations concerning the comparison between the simulation and the closed-loop experimental data are as follows:

- The large disturbances during the transition are due the system being away from the equilibrium points when the optimal controllers were turned on. These mismatches are hard to avoid, due to system noise and general difficulty involving the control of the plant.

- The offsets in the steam outputs in Figure 6.2 are due to unmodeled dynamics of the downstream impedance of the steam flow. The dryer pressure that the throttle valve exhausts to is modeled as a disturbance, but it is actually controlled by an additional valve. This added impedance, along with the pure integrator in the steam system controller, means that the steam mass flow can drift based on different initial conditions. This could be fixed with another loop or setpoint, but is outside the scope of this project.

- The offset between the Heat Responses in Figure 6.1 is due to a variety of sources. First, the fuel energy density and moisture content are not known instantaneously throughout the course of the test, and can only be estimated. Additionally, the thermocouples

report back depressed values due to heat loss from radiation and conduction which will always make collected data heat lower than simulated.

The testing on the plant took place over a short schedule due to limited plant availability which restricted the amount of tuning and optimization for the controllers. Even with these restrictions, the system and controllers are dynamically stable and the outputs become asymptotically constant. The results of the closed-loop testing data help illustrate where improvements can be made in the future, both to the physical plant and the controllers. A better drum level sensor could help improve the steam system control, and better tuning of the combustion system could improve the oxygen response and steady-state error.

6.3.2 Disturbance Comparison

The Figures 6.4, 6.5, and 6.6 show the comparison between the closed loop optimal simulation to the plant data for a -10% offset of the fuel metering augers. For these figures, the comparison is much closer because of better initial condition lineup. Regardless, there are some observed issues because it was not possible to isolate the system noise or other disturbances from the test. Some of the data does not line up exactly, but a comparison can still be made.

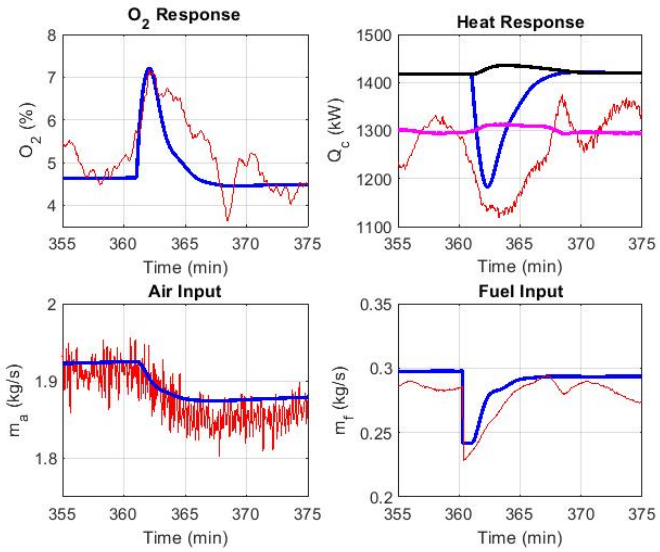

Figure 6.4: Combustion inputs/outputs for simulation vs. data during disturbance rejection. Data is red and simulation is blue. The magenta curve is the data heat setpoint.

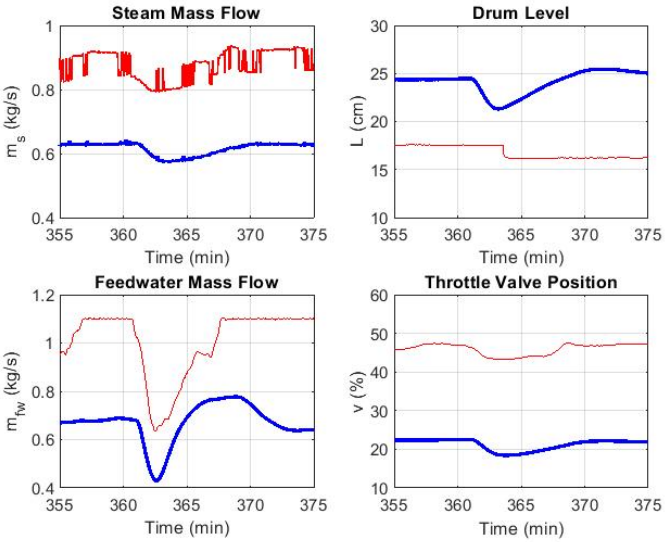

Figure 6.5: Steam inputs/outputs for simulation vs. data during disturbance rejection.

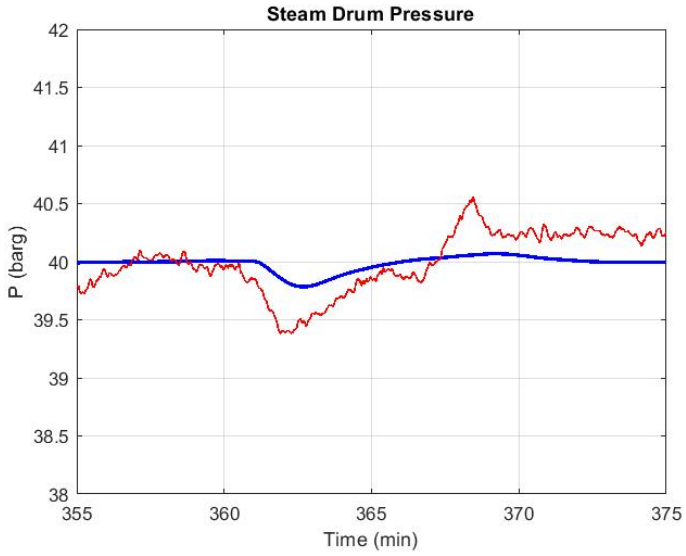

Figure 6.6: Steam drum pressure for simulation vs. data during disturbance rejection.

The disturbance comparison data illustrate the good agreement between the simulated and collected data. There are no specific metrics used to define *good agreement*, but this qualitative observation shows that dynamics of the responses are captured very well. There is some disagreement with the steady-state values, but the controls were designed to compensate for these discrepancies.

6.4 Objective II: Steady-State Controller Comparison

For the steady-state output error comparison, data was collected for both controllers, and the standard deviation from the output setpoints was computed and compared. The data for the SISO controllers can be seen in Figures 6.7, 6.8, and 6.9. These figures show the large output oscillations. The average values are very close to the appropriate setpoints, but the SISO controllers are unable to attenuate the oscillations.

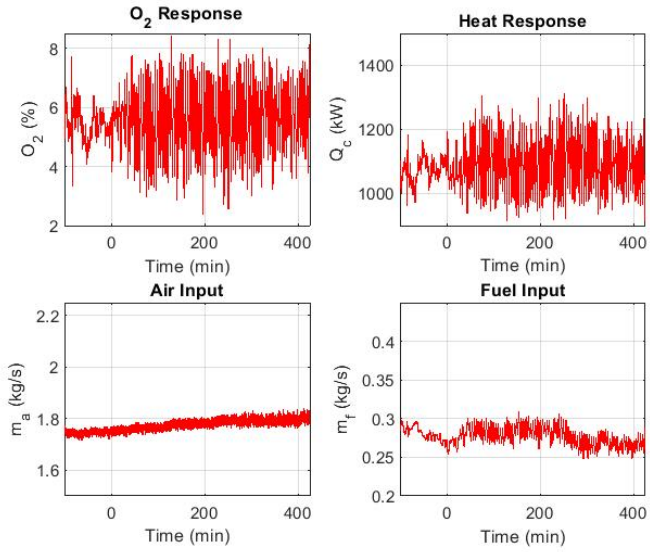

Figure 6.7: Combustion input/output data from SISO controllers.

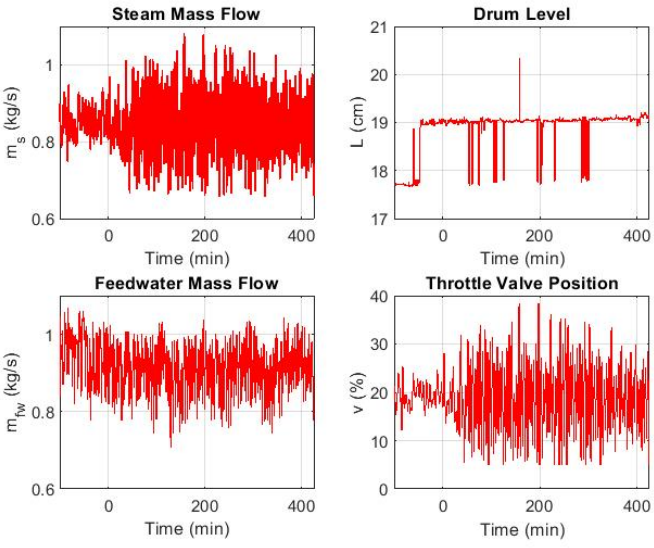

Figure 6.8: Steam input/output data from SISO controllers.

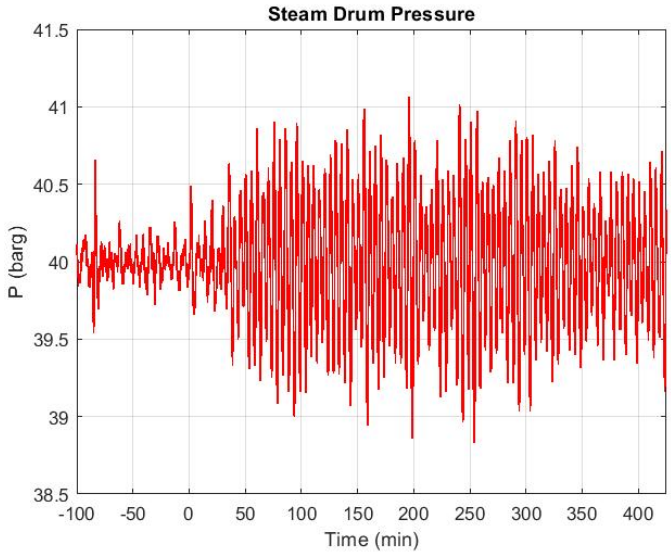

Figure 6.9: Steam drum pressure data from SISO controllers.

Using the data from Figures 6.7, 6.8, and 6.9 and the data from the extended optimal run shown above, Figure 6.10 shows the percentage decrease in Standard Deviation from the output setpoints for the optimal controllers compared to the SISO controllers.

When Figures 6.7, 6.8, and 6.9 are compared to the optimal controller data shown in Figures 6.1, 6.2, and 6.3, the difference is immediately apparent. All of the outputs visually appear to have much smaller oscillations. Quantitatively, the optimal data output error standard deviations all show more than a 40% decrease when compared to the SISO data, as shown in Figure 6.10. The oxygen content, heat production, steam pressure, and steam mass flow had 52.4%, 41.4%, 52.7% and 49.3% reductions, respectively.

The data and figures presented in this section illustrate that the optimal controllers significantly improve the steady-state performance when compared to the SISO controllers. The optimal controllers attenuate the cyclic oscillations in the outputs much better than the SISO controllers, and keep the outputs generally closer to their setpoints.

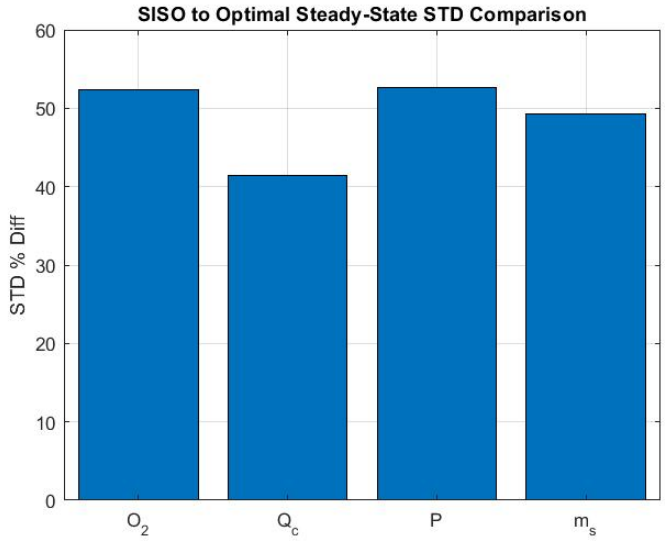

Figure 6.10: Percent decrease in the standard deviation of the outputs from the setpoints when comparing the optimal controllers to the SISO data during steady-state.

6.5 Objective III: Disturbance Rejection Controller Comparison

This section will observe and compare the disturbance rejection of the SISO and optimal controllers. These disturbances were introduced by adding or subtracting 10% from the fuel metering auger speed. The controllers could not measure this disturbance, and had to adjust the inputs based on output feedback alone. This method of disturbance is used to simulate a change in the fuel, whether it be moisture content, energy density, or chemical composition. A decrease in the metering auger speeds would represent either a decrease in energy density, or an increase in fuel moisture content. A disturbance to the speeds was deemed easier than trying to manipulate the fuel composition or moisture content during a testing run, which could not be done in a consistent or repeatable fashion.

Figures 6.11, 6.12, and 6.13, show the comparison between the responses of the two controllers when a 10% disturbance is subtracted from the fuel metering auger speeds.

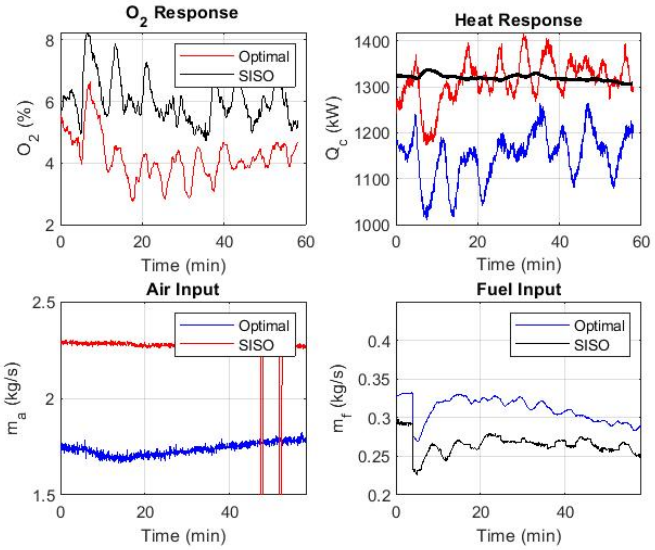

Figure 6.11: Optimal vs. SISO disturbance rejection comparison for combustion input/output data.

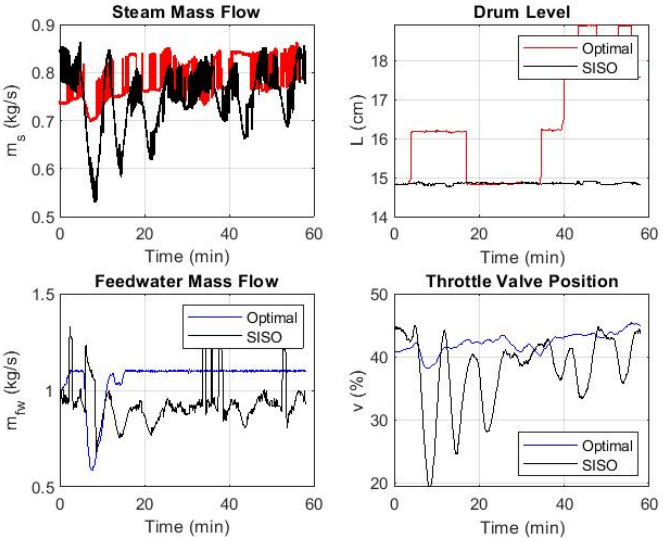

Figure 6.12: Optimal vs. SISO disturbance rejection comparison for steam input/output data.

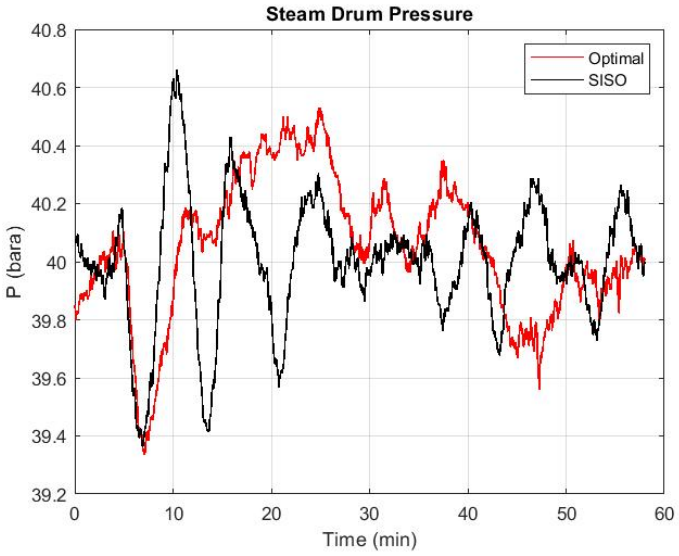

Figure 6.13: Optimal vs. SISO disturbance rejection comparison for steam drum pressure data.

As can be seen qualitatively, in Figures 6.11, 6.12 and 6.13, the two controllers do a comparable job of maintaining some of the states such as oxygen content and steam pressure, but they do a vastly better job of maintaining the steam mass flow output after the disturbance. This is a highly desireable feature of the controls, as maintaining the steam boiler power output is ultimately the principle goal of the entire power plant. The only substate with inferior control is the steam drum level, but it is the least reliable sensor, and therefore difficult to make an accurate comparison. For this reason, it has been neglected from the standard deviation calculations shown in Figures 6.14 and 6.15.

The percentage difference for the standard deviations in the outputs is shown and compared in Figures 6.14 and 6.15. Two tests were performed for each controller, and the these four tests are compared in Figure 6.14, and the mean of the differences is computed in Figure 6.15.

In Figure 6.14, the relative change between all of the different states can be observed.

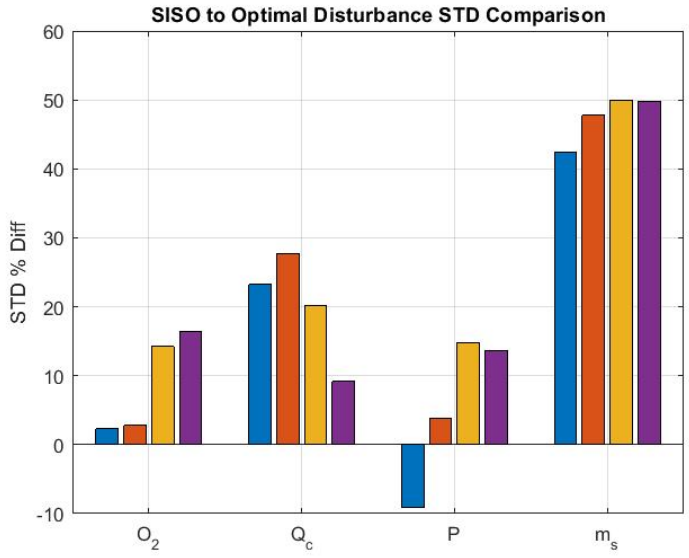

Figure 6.14: Percentage decrease in the standard deviation of the outputs for a disturbance comparing the SISO to the optimal controllers.

While most of the changes are relatively consistent, there is a noticeable difference in the first two test comparisons as opposed to the second two. One of the reasons for this appreciable difference in the data is although all of the tests involved a 10% change in the fuel speed, these tests occurred on three separate days with different fuel conditions. During the testing, keeping the fuel consistent was one of the goals, but there were inevitable differences. This again illustrates the necessity for better disturbance rejection from the inconsistent fuel source.

The optimal controllers improve all of the outputs on average. The oxygen content, heat production, and steam pressure show 8.9%, 20.0%, and 5.7% reductions, respectively. Meanwhile, the most pronounced improvement is on the steam mass flow - with a 47.5% decrease in the standard deviation. This is extremely beneficial in removing disturbances to the steam load after the boiler, and thereby reducing the amount of disturbance seen by the electrical generation systems.

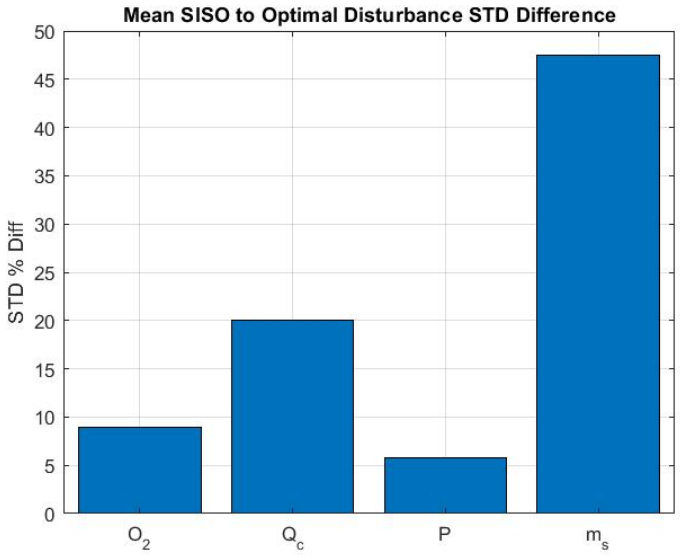

Figure 6.15: Mean percentage decrease in the standard deviation of the outputs for a disturbance comparing the SISO to the optimal controllers.

The data and figures presented in this section show that the optimal controllers also significantly improve the disturbance rejection capabilities of the system when compared to the SISO controllers. The optimal controllers respond to the disturbances without unwanted oscillations, and vastly improve the stability of the boiler power output given a disturbance.

Chapter 7

CONCLUSION

The purpose of this thesis is to investigate a new method for the controls of solid biomass fuel power plants to improve the performance and disturbance rejection capabilities of the controllers. This need for better control arises from the inconsistency of the incoming fuel stream, as well as some fundamental difficulties of burning solid biomass fuel.

The proposed method for improving the control is optimal MIMO controllers derived from a dynamic system model. This model is developed based on an extensive literature review, as well as with some first principles modeling and testing of an individual power plant used as a case study, the J-OP S250. The parameters for this model are estimated based on the physical and testing data from the J-OP S250. A thorough sensitivity analysis is performed on the system model based on these parameters to identify if any issues would occur in the event of a poorly estimated parameter. The parameters the system is most sensitive to are the freeboard heat transfer coefficient, and the valve admittance slope parameter. The effect of perturbations to these parameters on the closed-loop system is investigated. The controllers are still stable and bring the outputs to their reference values.

The non-linear model developed for the plant is linearized about the system operating points and used to derive a series of optimal cascade controllers. These controllers are designed to reject disturbances in the fuel, hold the system outputs constant, and to tolerate some noise in the feedback signals.

These controllers were incorporated into the automation code running the J-OP S250, and tested for steady-state output performance as well as disturbance rejection. The optimal data are compared to the simulated system to observe how well the two data sets agree. A comparison is made, though some issues with initial conditions and sensory feedback (such

as the steam drum level sensor) make it difficult to get a better agreement.

Additionally, the optimal data are compared to the original SISO controller data to observe if an improvement in the system performance is achieved. After careful analysis of the data, the tests show that the optimal system significantly improves the steady-state output error performance. The standard deviations for the output errors are calculated for the optimal and SISO data sets, and the percent decrease in the standard deviations for the oxygen content, heat production, steam pressure, and steam mass flow outputs are 52.4%, 41.4%, 52.7%, and 49.3%, respectively.

The disturbance rejection capabilities of the optimal controllers also show an improvement over the SISO controllers. When the fuel metering augers are disturbed with a 10% reduction, the mean output error standard deviation reductions in the oxygen content, heat production, steam pressure, and steam mass flow outputs are 8.9%, 20.0%, 5.7%, and 47.5%, respectively. The 47.5% improvement in the steam mass flow was noted to be significant, since the steam mass flow is the primary power output variable from the boiler. Holding this output more constant when a disturbance is introduced helps to avoid propagating disturbances into the electrical generation systems on the steam plant.

The positive results from the testing indicate the merits of the modeling and optimal MIMO control methods presented in this thesis.

BIBLIOGRAPHY

[1] Karl Johan Åström and Rodney D Bell. "Drum-boiler dynamics". In: *Automatica* 36.3 (2000), pp. 363–378.

[2] Stefano Barsali et al. "Dynamic modelling of biomass power plant using micro gas turbine". In: *Renewable Energy* 80 (2015), pp. 806–818.

[3] Prabir Basu. *Combustion and gasification in fluidized beds*. CRC press, 2006.

[4] CM Cheng and NW Rees. "Fuzzy model based control of steam generation in drum-boiler power plant". In: *IFAC Proceedings Volumes* 30.17 (1997), pp. 143–149.

[5] R Cori and Claudio Maffezzoni. "Practical-optimal control of a drum boiler power plant". In: *Automatica* 20.2 (1984), pp. 163–173.

[6] Samar Das, Pranay Kumar Sarkar, and Sadhan Mahapatra. "Single particle combustion studies of coal/biomass fuel mixtures". In: *Energy* 217 (2021), p. 119329.

[7] *Electricity Explained*. 2020. URL: `https://www.eia.gov/energyexplained/electricity/electricity-in-the-us.php` (visited on 02/22/2021).

[8] H Emara and A Bahgat. "Boiler Drum Level Control using FOPID Controller with DCS SW Application". In: ().

[9] Paul M. Frank. *Introduction to System Sensitivity Theory*. New York San Francisco London: Academic Press, 1978.

[10] Ahmed El-Guindy, Simon Rünzi, and Kai Michels. "Optimizing drum-boiler water level control performance: A practical approach". In: *2014 IEEE Conference on Control Applications (CCA)*. IEEE. 2014, pp. 1675–1680.

[11] Mihai Iacob and Gheorghe-Daniel Andreescu. "Drum-boiler control system employing shrink and swell effect remission in thermal power plants". In: *2011 3rd International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*. IEEE. 2011, pp. 1–8.

[12] J Kortela and SL Jämsä-Jounela. "Modeling and model predictive control of the BioPower combined heat and power (CHP) plant". In: *International Journal of Electrical Power & Energy Systems* 65 (2015), pp. 453–462.

[13] *lqgtrack*. 2020. URL: `https://www.mathworks.com/help/control/ref/ss.lqgtrack.html` (visited on 02/23/2021).

[14] Yrjö Majanne and Petri Köykkä. "Dynamic model of a circulating fluidized bed boiler". In: *IFAC Proceedings Volumes* 42.9 (2009), pp. 255–260.

[15] Fred Mayes. *Biomass and waste fuels made up of 2% of total U.S. electricity generation in 2016*. 2017. URL: `https://www.eia.gov/todayinenergy/detail.php?id=33872#:~:text=Biomass%5C%20and%5C%20waste%5C%20fuels%5C%20generated,based%5C%20(biogenic)%5C%20energy%5C%20sources` (visited on 02/10/2021).

[16] Norman Miller et al. "Control of steam generation processes". In: *Advances in Instrumentation, Proceedings* 45.pt 3 (1990), pp. 1265–1279.

[17] Farideh Mohammadhassani, Amin Ramezani, and Mohsen Razzazan. "Drum Boiler Control with Output Constraints using Model Predictive Control Method". In: (2014).

[18] Nishant Parikh et al. "Control of a nuclear steam generator using feedback-feedforward LQG controller". In: *2011 International Symposium on Advanced Control of Industrial Processes (ADCONIP)*. IEEE. 2011, pp. 415–420.

[19] Gerdon Pellegrinetti, Joseph Bentsman, and Kameshwar Poolla. "Control of nonlinear steam generation processes using H_∞ design". In: *1991 American Control Conference*. IEEE. 1991, pp. 1292–1297.

[20] Wen Tan and Fang Fang. “Linear analysis and control of a boiler-turbine unit”. In: *IFAC Proceedings Volumes* 41.2 (2008), pp. 10832–10837.

[21] Joseph R Visalli. “A comparison of dioxin, furan and combustion gas data from test programs at three MSW incinerators”. In: *JAPCA* 37.12 (1987), pp. 1451–1463.

[22] Kenneth Wark. *Advanced thermodynamics for engineers*. McGraw-Hill New York, 1995.

[23] Peter Colin Young and JC Willems. “An approach to the linear multivariable servomechanism problem”. In: *International journal of control* 15.5 (1972), pp. 961–979.

Appendix A
TESTING NOTES

Once the system simulations and controllers were working, the optimal controllers had to be implemented on the J-OP S250. This appendix details the implementation process and testing.

The controls for the J-OP S250 run on a Siemens S1500 PLC. This PLC is programmed in the Structured Control Language (SCL), which is a basic text-based programming language (as opposed to Ladder Logic, which is typical with PLCs).

One of the challenges of this project was implementing the LQG servo controllers in the basic SCL language. Additionally, the controller gains and matrices had to be updated easily from Matlab for tuning purposes. This was accomplished by first discretizing the controllers in Matlab. Then, since the PLC executes the control code on a constant 100 millisecond interrupt, some simple code was used to step through discretized dynamic systems to simulate the dynamic controller states forward in time. Data types were created for all the required parameters and matrices on the PLC. The matrices were implemented using 2-D arrays. The templates for these data types were exported as a text file, which was used as a template to create a script in Matlab that assigned the appropriate values to all of the matrices and parameters in the data types. The text file was used to load all of the values into the PLC. Additionally, it could be readily updated with new values, to re-tune the controllers.

Once the code for implementing the controllers was written on the PLC, it was tested to verify proper implementation. This was accomplished by first verifying that the zero input response with non-zero initial conditions on the PLC matched an identical simulation in Matlab. This was done for the combustion system and the steam system independently. Once the zero-input simulations agreed, a forced response with static inputs was tested as

well.

To test the closed-loop system, some modifications to the combustion system were made. In the simulations, the inputs to the combustion system are air mass flow and fuel mass flow, while in the physical system the real inputs are the speed of the air fan and the speed of the fuel metering augers.

The air mass flow was handled by incorporating a PID controller between the air mass flow sensor and the air fan speed. The setpoint for the air was determined by subtracting the air mass flow being supplied by the primary air fan from the total setpoint requested by the combustion controller. This smaller setpoint was fed to the air mass flow controller.

Determining a method for the fuel mass flow was more difficult and much less accurate since the system does not have a fuel mass flow meter. This was overcome by creating a curve fit between the steady-state heat production and fuel metering auger speed. This curve fit was used to establish a static gain between the fuel metering auger speed and the correlated mass flow. This calculation is sensitive to fuel moisture content, fuel energy content, and the physical mechanics of the metering augers. This is an unfortunate and unavoidable feature of the J-OP S250, but also one of the features the controllers were designed to be robust to.

Once the controller implementation was verified to be correct, the system was ready for closed loop testing. One of the advantages of the cascade control structure is that the inner loop can be tested independent from the outer loop. In this case, this was accomplished by leaving the original steam system controls for the steam drum pressure and level intact and replacing the control loops that specify the speeds of the fuel metering augers and the secondary air fan. The PLC code was written in such a way that if something went wrong, or the system started to go unstable, the optimal controllers could be swapped back to the original controllers by flipping a boolean. This was also useful during the startup procedure as all the original automation code for startup had to be left intact to get the system up to the operating point before the optimal controllers could be tested.

The combustion system controls took a few days of testing to get working. Due to the labor required to keep the plant running, the plant had to be started every morning and

shut down every night, so after the required heat up period, there was a limited amount of testing time to run the plant at steady-state. The first few days of testing were concerned with ensuring the controls were implemented correctly, and all the inputs and outputs were being assigned properly. Once this was finished, some fine tuning in terms of the controller and Kalman filter gains was required to properly attenuate the system noise, while still responding to disturbances and ensuring zero steady state error in the outputs.

Once the combustion system was operating at a satisfactory level, the two control loops were tested together. Like the combustion system, it took a few days of testing to get the full cascade control system properly tuned and debugged. After the system was operating well, some small tuning adjustments were made, but not optimized very much. Since the closed-loop system has eigenvalues that last minutes and hours, it took almost a full day to observe how well the system operated for a specific set of weighting matrices. Because of the limited schedule, once the system was operating at a satisfactory level, the tuning was stopped and the tests were run so all of the tests had consistent controllers throughout.

Appendix B

SISO CONTROLLER COUPLING NOTES

This appendix provides some context for why the SISO controllers create cyclic oscillations in the event of a input disturbance.

The coupling of the SISO system controllers is difficult to understand, but a scenario can be pictured as follows:

Assume that the system is operating at steady state and at the desired setpoints. Then, a disturbance in the fuel stream causes a drop in steam production. This also produces a drop in steam drum pressure. This drop in steam production causes three responses from the controllers; the fuel augers start to wind up, the steam throttle valve starts to close, and the feedwater pumps decrease in speed to match the steam flow. The steam throttle valve closes to raise the pressure of the steam drum back to the setpoint. As the valve closes, this further depresses the steam production. This additional depression in steam mass flow artificially causes the fuel metering augers to wind up even further but the real production is higher. Since there is a significant delay in the response of the combustion system to the input of fuel, this artificial depression causes the metering augers to wind up long past the appropriate level. This creates a surge in steam production once the system responds, and the process continues just in the opposite fashion, thus creating the cyclic oscillations.

Appendix C

STEAM MODEL CODE

The code included in this appendix executes the required calculations for the non-linear steam model. This code is written in c, and executes in Simulink using an S-function. The code outputs the time derivatives of the state variables as a function of the inputs and current state variables. The functions above the main section of code are the fourth-order polynomial fits to the steam table data for the required thermodynamic properties.

```
/*
 * Include Files
 *
 */
#if defined(MATLAB_MEX_FILE)
#include "tmwtypes.h"
#include "simstruc_types.h"
#else
#include "rtwtypes.h"
#endif



/* %%%-SFUNWIZ_wrapper_includes_Changes_BEGIN --- EDIT HERE TO _END */
#include <math.h>
/* %%%-SFUNWIZ_wrapper_includes_Changes_END --- EDIT HERE TO _BEGIN */
#define u_width 1
#define y_width 1

/*
 * Create external references here.
 *
 */
/* %%%-SFUNWIZ_wrapper_externs_Changes_BEGIN --- EDIT HERE TO _END */
/* extern double func(double a); */
/* %%%-SFUNWIZ_wrapper_externs_Changes_END --- EDIT HERE TO _BEGIN */

// Define functions for thermodynamic properties

// Returns evaluation of fourth order polynomial
// Accepts value, and polynomial coefficients
real_T fourthOrderPoly(real_T value, real_T a, real_T b, real_T c, real_T d, real_T ↙
e) {
    return a*pow(value, 4.0) + b*pow(value, 3.0) + c*pow(value, 2.0) + d*value + e;
}

// Returns feedwater enthalpy hfw [kJ/kg]
// Accepts temperature [C]
real_T feedwaterEnthalpy(real_T temperature) {
    return fourthOrderPoly(temperature, 0.0, 0.0000103, -0.0034125, 4.6368129, ↙
-21.1973334);
}

// Returns saturation temperature [C]
// Accepts pressure [kPa]
real_T waterSatTemp(real_T pressure) {
    return fourthOrderPoly(pressure, -0.0000072, 0.0013521, -0.1069261, 5.3916614, ↙
137.6766227);
```

```
}

// Returns Sat Water Enthalpy
// Accepts pressure [kPa]
real_T satWaterEnthalpy(real_T pressure) {
    return fourthOrderPoly(pressure, -0.0000303, 0.0056984, -0.4494708, 23.5873466, ↙
576.0088667);
}

// Returns Sat Water Density
// Accepts pressure [kPa]
real_T satWaterDensity(real_T pressure) {
    return fourthOrderPoly(pressure, 0.0000060, -0.0011492, 0.0920733, -5.6015142, ↙
933.2120782);
}

// Returns Sat Vapor Enthalpy
// Accepts pressure [kPa]
real_T satVaporEnthalpy(real_T pressure) {
    return fourthOrderPoly(pressure, -0.0000109, 0.0020175, -0.1578148, 5.2523578, ↙
2742.0973787);
}

// Returns Sat Vapor Density
// Accepts pressure [kPa]
real_T satVaporDensity(real_T pressure) {
    return fourthOrderPoly(pressure, -0.0000001, 0.0000144, -0.0003492, 0.4893630, ↙
0.2877700);
}

// Returns partial saturation temperature
// with respect to p
// Accepts pressure [kPa]
real_T dWaterSatTempdP(real_T pressure) {
    return fourthOrderPoly(pressure, 0.0, -0.0000072*4.0, 0.0013521*3.0, -0.1069261 ↙
*2.0, 5.3916614);
}

// Returns partial Sat Water Enthalpy
// with respect to p
// Accepts pressure [kPa]
real_T dSatWaterEnthalpydP(real_T pressure) {
    return fourthOrderPoly(pressure, 0.0, -0.0000303*4.0, 0.0056984*3.0, -0.4494708 ↙
*2.0, 23.5873466);
}

// Returns partial Sat Water Density
// with respect to p
// Accepts pressure [kPa]
```

```
real_T dSatWaterDensitydP(real_T pressure) {
    return fourthOrderPoly(pressure, 0.0, 0.0000060*4.0, -0.0011492*3.0, 0.0920733*2.↙
0, -5.6015142);
}

// Returns partial Sat Vapor Enthalpy
// with respect to p
// Accepts pressure [kPa]
real_T dSatVaporEnthalpydP(real_T pressure) {
    return fourthOrderPoly(pressure, 0.0, -0.0000109*4.0, 0.0020175*3.0, -0.1578148↙
*2.0, 5.2523578);
}

// Returns partial Sat Vapor Density
// with respect to p
// Accepts pressure [kPa]
real_T dSatVaporDensitydP(real_T pressure) {
    return fourthOrderPoly(pressure, 0.0, -0.0000001*4.0, 0.0000144*3.0, -0.0003492↙
*2.0, 0.4893630);
}

/*
 * Output function
 *
 */
void steamDrum_rev_2_Outputs_wrapper(const real_T *heatInput_kW,
            const real_T *feedwaterFlow_kg_p_s,
            const real_T *valvePosition_pct,
            const real_T *feedwaterTemp_C,
            const real_T *dryerPressure_bar,
            const real_T *steamTurbineMassFlow_kg_p_s,
            const real_T *lostMassFlow_kg_p_s,
            const real_T *totalWaterVolume_m3,
            const real_T *drumPressure_barg,
            const real_T *riserQuality,
            const real_T *steamVolumeInDrum_m3,
            const real_T *frictionCoefficient,
            const real_T *betaCoefficient,
            const real_T *Vsd0Coefficient,
            const real_T *valveSlope,
            const real_T *valveIntercept,
            real_T *totalWaterVolume_dot_m3_p_s,
            real_T *drumPressure_dot_barg_p_s,
            real_T *riserQuality_dot,
            real_T *steamVolumeInDrum_dot_m3_p_s,
            real_T *drumWaterHeight_m,
            real_T *steamDrumSaturationTemp_C,
            real_T *steamMassFlow_kg_p_sec)
{
```

```
// Define Dummy Variables to be used instead of the input pointer data
    real_T mf    = *feedwaterFlow_kg_p_s;
    real_T v     = *valvePosition_pct;
    real_T mst   = *steamTurbineMassFlow_kg_p_s;
    real_T mLoss = *lostMassFlow_kg_p_s;
    real_T Q     = *heatInput_kW;
    real_T Vwt   = *totalWaterVolume_m3;
    real_T p     = *drumPressure_barg;
    real_T ar    = *riserQuality;
    real_T Vsd   = *steamVolumeInDrum_m3;
    real_T Tf    = *feedwaterTemp_C;
    real_T pd    = *dryerPressure_bar;
    real_T c1    = *valveSlope;
    real_T c2    = *valveIntercept;

    // Now define thermodynamic variables:
    real_T hs       = satVaporEnthalpy(p);
    real_T hw       = satWaterEnthalpy(p);
    real_T rs       = satVaporDensity(p);
    real_T rw       = satWaterDensity(p);
    real_T drsdp    = dSatVaporDensitydP(p);
    real_T drwdp    = dSatWaterDensitydP(p);
    real_T dhsdp    = dSatVaporEnthalpydP(p);
    real_T dhwdp    = dSatWaterEnthalpydP(p);
    real_T Ts       = waterSatTemp(p);
    real_T dTsdp    = dWaterSatTempdP(p);

    real_T hf       = feedwaterEnthalpy(Tf);
    real_T hc       = hs - hw;

    real_T Cp = 0.51; // kJ/kg - C

    real_T beta = *betaCoefficient;//0.3;
    real_T mt = 8809.0;
    real_T mr = 7041.0;
    real_T md = mt - mr;
    real_T k = *frictionCoefficient;//1000.0;
    real_T Adc = 0.074555;
    real_T Ad = 2.05;

    real_T Vdc = 1.3;
    real_T Vd = 1.00;
    real_T Vsd0 = *Vsd0Coefficient;//0.06*0.5*Vd;
    real_T Td = 1.0;
    real_T Vr = 1.8;
    real_T Vt =  Vd + Vr + Vdc;
    real_T g = 9.81;

    // Define intermediate variables
```

```
    real_T eta = ar*(rw - rs)/rs;

    real_T av = rw/(rw - rs)*(1 - rs/((rw - rs)*ar)*log(1 + (rw - rs)/rs*ar));
    real_T davdp = 1/((rw - rs)*(rw - rs))*(rw*drsdp - rs*drwdp)*(1 + rw/(rs*(1 + ↙
eta)) - (rs + rw)/(eta*rs)*log(1 + eta));
    real_T davdar = rw/(rs*eta)*(log(1 + eta)/eta - 1/(1 + eta));

    real_T Vwd = Vwt - Vdc - (1 - av)*Vr;
    real_T Vst = Vt - Vwt;
    real_T mdc = pow(2*rw*Adc*(rw - rs)*g*av*Vr/k, 0.5);

    // Define coefficients
    real_T e11 = rw - rs;
    real_T e12 = Vwt*drwdp + Vst*drsdp;
    real_T e21 = rw*hw + rs*hs;
    real_T e22 = Vwt*(hw*drwdp + rw*dhwdp) + Vst*(hs*drsdp+rs*dhsdp) - Vt + ↙
mt*Cp*dTsdp;
    real_T e32 = (rw*dhwdp - ar*hc*drwdp)*(1-av)*Vr + ((1 - ar)*hc*drsdp + rs*dhsdp) ↙
*av*Vr + (rs + (rw - rs)*ar)*hc*Vr*davdp - Vr + mr*Cp*dTsdp;
    real_T e33 = ((1 - ar)*rs + ar*rw)*hc*Vr*davdar;
    real_T e42 = Vsd * drsdp + 1/hc*(rs*Vsd*dhsdp + rw*Vwd*dhwdp - Vsd + md*Cp*dTsdp) ↙
+ ar*(1 + beta)*Vr*(av*drsdp + (1 - av)*drwdp + (rs - rw)*davdp);
    real_T e43 = ar*(1 + beta)*(rs - rw)*Vr*davdar;
    real_T e44 = rs;

    // Define steam mass flow
    real_T ms = (p - pd)*(c1*v + c2) + mst;

    // Determine Output State Variables
    real_T denominator  = e12*e21 - e11*e22;
    real_T dVwtdt       = (e12*Q + (e12*hf - e22)*mf - (e12*hs - e22)*ms - (e12*hw - ↙
e22)*mLoss)/(denominator);
    real_T dpdt         = (mf - ms - mLoss - e11*dVwtdt)/e12;
    real_T dardt        = (Q - ar*hc*mdc - e32*dpdt)/e33;
    real_T dVsddt       = (rs/Td*(Vsd0 - Vsd) + (hf - hw)*mf/hc - e42*dpdt - ↙
e43*dardt)/e44;

    // Determine Feedback Variables
    *drumWaterHeight_m  = (Vwd + Vsd)/Ad;

    // Set Outputs
    *totalWaterVolume_dot_m3_p_s    = dVwtdt;
    *drumPressure_dot_barg_p_s      = dpdt;
    *riserQuality_dot               = dardt;
    *steamVolumeInDrum_dot_m3_p_s   = dVsddt;
    *steamDrumSaturationTemp_C      = Ts;
    *steamMassFlow_kg_p_sec         = ms;
}
```

Appendix D

COMBUSTION FLAME TEMPERATURE CODE

The code included in this appendix calculates the average freeboard temperature as a function of the fuel properties, fuel and air mass flows, and other inputs. This code solves the implicit equations for the flame temperature by iteration. This code also uses an S-function in Simulink.

```
/*
 * Include Files
 *
 */
#if defined(MATLAB_MEX_FILE)
#include "tmwtypes.h"
#include "simstruc_types.h"
#else
#include "rtwtypes.h"
#endif



/* %%%-SFUNWIZ_wrapper_includes_Changes_BEGIN --- EDIT HERE TO _END */
#include <math.h>
/* %%%-SFUNWIZ_wrapper_includes_Changes_END --- EDIT HERE TO _BEGIN */
#define u_width 1
#define y_width 1

/*
 * Create external references here.
 *
 */
/* %%%-SFUNWIZ_wrapper_externs_Changes_BEGIN --- EDIT HERE TO _END */
/* extern double func(double a); */
/* %%%-SFUNWIZ_wrapper_externs_Changes_END --- EDIT HERE TO _BEGIN */


// Change in enthalpy function for ideal gas starting at 298 K
real_T deltaH(real_T T, int_T count) {
    // T is in K
    // dh is equal to h(T) - h(T = 298)
    real_T dh = 0.0;
    if (count == 1) {
        // N2
        dh = 28.9*T - 7.855/10000.0*T*T + 2.6937/1000000.0*T*T*T - 7.1825 ↙
/10000000000.0*T*T*T*T - 8.6081*1000.0;
        return dh;
    } else if (count == 2) {
        // O2
        dh = 25.48*T + (1.52/2.0/100.0)*T*T - (0.7155/3.0/100000.0)*T*T*T + (1.312/4. ↙
0/1000000000.0)*T*T*T*T - 8.2074*1000.0;
        return dh;
    } else if (count == 3) {
        // CO2
        dh = 22.26*T + 2.9905/100.0*T*T - 1.167/100000.0*T*T*T + 1.8673/1000000000. ↙
0*T*T*T*T - 8.9951*1000.0;
        return dh;
    } else if (count == 4) {
```

```
        // H2O
        dh = 32.24*T + 9.615/10000.0*T*T + 3.5167/1000000.0*T*T*T - 8.9875 ↙
/10000000000.0*T*T*T*T - 9.7789*1000.0;
        return dh;
    }
    return 0.0;
}


/*
 * Output function
 *
 */
void flame_temp_Outputs_wrapper(const real_T *carbon,
            const real_T *hydrogen,
            const real_T *oxygen,
            const real_T *HHV,
            const real_T *fuelFlow,
            const real_T *airFlow,
            const real_T *moisturePercent,
            const real_T *airT,
            const real_T *bedT,
            const real_T *steamTemp,
            const real_T *UA_freeboard,
            real_T *exhaustFlow,
            real_T *excessO2,
            real_T *phi,
            real_T *Q_freeboard,
            real_T *T_freeboard,
            real_T *massFlowO2,
            real_T *massFlowN2,
            real_T *massFlowCO2,
            real_T *massFlowH2O)
{
    // Define local constants here

    // Molecular weights
    real_T mwH2O = 18.01528;     // kg/kmole
    real_T mwO2 = 31.998;        // kg/kmole
    real_T mwN2 = 28.013;        // kg/kmole
    real_T mwCO2 = 44.01;        // kg/kmole
    real_T mwC = 12.0107;        // kg/kmole
    real_T mwH = 1.0079;         // kg/kmole
    real_T mwO = 15.999;         // kg/kmole
    real_T mwAir = 28.84;

    // Formation enthalpies
    real_T hfCO2 = -393520.0;      // kJ/kmole
    real_T hfH2OL = -285820.0;     // kJ/kmole
```

```
    real_T hfH2Og = -241820.0;     // kJ/kmole

    // Enthalpy at 298K
    real_T h298H2O = 9904.0;  // kJ/kmole
    real_T h298O2 = 8682.0;      // kJ/kmole
    real_T h298N2 = 8669.0;      // kJ/kmole
    real_T h298CO2 = 9364.0;      // kJ/kmole

    // find total fuel comp mass sum
    // compSum = carbon + hydrogen + oxygen;

    // average the compositions by total mass basis
    real_T a = *carbon/mwC;          // kg C/ 1kg fuel
    real_T b = *hydrogen/mwH;        // kg H/ 1kg fuel
    real_T c = *oxygen/mwO;          // kg C/ 1kg fuel

    real_T w = (2.0*a + b/2.0 - c)/2.0;

    // Find Enthalpy of Formation
    real_T hf_fuel = a*hfCO2 + b/2*hfH2OL + 100.0 * *HHV;

    // Determine Theoretical Air. ratio is a pure ratio, not percent.
    real_T ratio = 100 * *airFlow/(mwAir*(1 - *moisturePercent)**fuelFlow*w*4.76);

    if (ratio < 1.0) {
        ratio = 1.0;
    }

    // Find kmoles of water in fuel
    real_T d = *moisturePercent**fuelFlow/mwH2O;

    // Find Enthalpy of Reactants
    real_T Hr  = hf_fuel + ratio*w*deltaH(*airT, 2) + 3.76*ratio*w*deltaH(*airT, 1) + ↙
d*hfH2OL;

    // Set bounds on the temperature range, and then take fist guess to be the ↙
average of the temp bounds
    real_T Tl = 0.0;
    real_T Th = 3000.0;
    real_T T0 = (Th + Tl)/2.0;

    // Use the temperature guess to calculate the enthalpy of the products => Sum(N* ↙
(hf + h(T) - h298))
    real_T Hp = a*(hfCO2 + deltaH(T0, 3)) + (b/2.0 + d)*(hfH2Og + deltaH(T0, 4)) + w* ↙
(ratio - 1.0)*deltaH(T0, 2) + 3.76*ratio*w*deltaH(T0, 1);

    // Set allowable error tolerance
    real_T tol = 0.01;
    int_T count = 0;
```

```
    // Get UA into appropriate units
    real_T UA = *UA_freeboard*100.0/((1.0 - *moisturePercent)**fuelFlow);

    // Iterating over T to get hProducts = hReactants
    while ( fabs(Hp - Hr + UA * (T0 - *steamTemp - 273.15) ) > tol ) {
        //Tbar = (4.0*(T0 - 273.15) + *bedT)/5.0 + 273.15;
        if (count > 1000) {
            break;
        }
        if ( (Hp - Hr + UA * (T0 - *steamTemp - 273.15) ) < 0 ) {
            Tl = T0;
            T0 = (Th + Tl)/2.0;

        } else {
            Th = T0;
            T0 = (Th + Tl)/2.0;
        }

        Hp = a*(hfCO2 + deltaH(T0, 3)) + (b/2.0 + d)*(hfH2Og + deltaH(T0, 4)) + w* ↙
(ratio - 1.0)*deltaH(T0, 2) + 3.76*ratio*w*deltaH(T0, 1);
        count++;
    }

    // Assign outputs
    *T_freeboard = T0;
    *excessO2 = 100.0*w*(ratio - 1.0)/(a + b/2.0 + d + w*(ratio - 1.0) + 3.76 ↙
*ratio*w);
    *exhaustFlow = *fuelFlow + *airFlow;
    *phi = ratio;
    *Q_freeboard = *UA_freeboard * (T0 - *steamTemp - 273.15);
    *massFlowO2  =  w*(ratio - 1)**fuelFlow/100.0;
    *massFlowN2  =  3.76*ratio*w**fuelFlow/100.0;
    *massFlowCO2 = (b/2 + d)**fuelFlow/100.0;
    *massFlowH2O =  a**fuelFlow/100.0;
}
```

Appendix E

STEAM SYSTEM LINEARIZATION CODE

The code included in this appendix determines the linear steam system model. This code outputs the A, B, C, and D matrices required for control design.

Model Derivation For Steam Drum

Part 1: Thermodynamic Curve Fit Equations

In[1]:=
```
rw = 0.0000060 * p^4 - 0.0011492 * p^3 + 0.0920733 * p^2 - 5.6015142 * p + 933.2120782;
drwdp = 0.0000060 * 4.0 * p^3 - 0.0011492 * 3.0 * p^2 + 0.0920733 * 2.0 * p - 5.6015142;

rs = -0.0000001 * p^4 + 0.0000144 * p^3 - 0.0003492 * p^2 + 0.4893630 * p + 0.2877700;
drsdp = -0.0000001 * 4.0 * p^3 + 0.0000144 * 3.0 * p^2 - 0.0003492 * 2.0 * p + 0.4893630;

hw = -0.0000303 * p^4 + 0.0056984 * p^3 - 0.4494708 * p^2 + 23.5873466 * p + 576.0088667;
dhwdp = -0.0000303 * 4.0 * p^3 + 0.0056984 * 3.0 * p^2 - 0.4494708 * 2.0 * p + 23.5873466;

hs = -0.0000109 * p^4 + 0.0020175 * p^3 - 0.1578148 * p^2 + 5.2523578 * p + 2742.0973787;
dhsdp = -0.0000109 * 4.0 * p^3 + 0.0020175 * 3.0 * p^2 - 0.1578148 * 2.0 * p + 5.2523578;

Ts = -0.0000072 * p^4 + 0.0013521 * p^3 - 0.1069261 * p^2 + 5.3916614 * p + 137.6766227;
dTsdp = -0.0000072 * 4.0 * p^3 + 0.0013521 * 3.0 * p^2 - 0.1069261 * 2.0 * p + 5.3916614;
hf = 0.0000103 * Tf^3 - 0.0034125 * Tf^2 + 4.6368129 * Tf - 21.1973334;
hc = hs - hw;
```

Part 2: Model Parameters

In[13]:=
```
Cp = 0.51;
beta = 0.3;
mt = 8809.0;
mr = 7041.0;
md = mt - mr;
k = 1000.0;
Adc = 0.074555;
Ad = 2.05;

Vdc = 1.3;
Vd = 1.00;
Vsd0 = 0.06 * 0.5 * Vd;
Td = 1.0;
Vr = 1.8;
Vt = Vd + Vr + Vdc;
g = 9.81;
a = 0.000374;
b = 0.000503;
```

Part 3: System Model Coefficients and Differential Equations

In[30]:=
```
eta = ar * (rw - rs) / rs;

av = rw / (rw - rs) * (1 - rs / ((rw - rs) * ar) * Log[1 + (rw - rs) / rs * ar]);
davdp = 1 / ((rw - rs) * (rw - rs)) * (rw * drsdp - rs * drwdp) *
    (1 + rw / (rs * (1 + eta)) - (rs + rw) / (eta * rs) * Log[1 + eta]);
davdar = rw / (rs * eta) * (Log[1 + eta] / eta - 1 / (1 + eta));

Vwd = Vwt - Vdc - (1 - av) * Vr;
Vst = Vt - Vwt;
mdc = Sqrt[2 * rw * Adc * (rw - rs) * g * av * Vr / k];

e11 = rw - rs;
e12 = Vwt * drwdp + Vst * drsdp;
e21 = rw * hw + rs * hs;
e22 = Vwt * (hw * drwdp + rw * dhwdp) + Vst * (hs * drsdp + rs * dhsdp) - Vt + mt * Cp * dTsdp;
e32 = (rw * dhwdp - ar * hc * drwdp) * (1 - av) * Vr + ((1 - ar) * hc * drsdp + rs * dhsdp) * av * Vr +
    (rs + (rw - rs) * ar) * hc * Vr * davdp - Vr + mr * Cp * dTsdp;
e33 = ((1 - ar) * rs + ar * rw) * hc * Vr * davdar;
e42 = Vsd * drsdp + 1 / hc * (rs * Vsd * dhsdp + rw * Vwd * dhwdp - Vsd + md * Cp * dTsdp) +
    ar * (1 + beta) * Vr * (av * drsdp + (1 - av) * drwdp + (rs - rw) * davdp);
e43 = ar * (1 + beta) * (rs - rw) * Vr * davdar;
e44 = rs;
denominator = e12 * e21 - e11 * e22;

ms = (p - pd) (a v + b) + mst;
dVwtdt =
  (e12 * Q + (e12 * hf - e22) * mf - (e12 * hs - e22) * ms - (e12 * hw - e22) * mL) / (denominator);
dpdt = (mf - ms - mL - e11 * dVwtdt) / e12;
dardt = (Q - ar * hc * mdc - e32 * dpdt) / e33;
dVsddt = (rs / Td * (Vsd0 - Vsd) + (hf - hw) * mf / hc - e42 * dpdt - e43 * dardt) / e44;

H = (Vwd + Vsd) / Ad;
```

Part 4: Equilibrium Values

In[83]:=
```
p0 = 41;
mst0 = 0.4;
mL0 = 0.0466;
mf0 = 0.70;
ms0 = mf0 - mL0 - mst0;
Tf0 = 150;
Q0 = (ms0 + mst0) hs + mL0 * hw - mf0 hf /. {p → p0, Tf → Tf0};
equilibriumExpression = ar * hc * Sqrt[2 * rw * Adc * (rw - rs) * g * (rw / (rw - rs) *
        (1 - rs / ((rw - rs) * ar) * Log[1 + (rw - rs) / rs * ar])) * Vr / k] /. {p → p0, Tf → Tf0};
sln = NSolve[Q0 == equilibriumExpression, ar, Reals];
ar0 = 0.03440818058454636;
Vsd00 = Vsd0 - Td (hw - hf) / (rs hc) mf0 /. {p → p0, Tf → Tf0};
Vwt0 = 0.5 * Vd + Vdc + (1 - av) * Vr - Vsd00 /. {p → p0, Tf → Tf0, ar → ar0};
H0 = (Vwd + Vsd) / Ad /.
   {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0, ms → ms0, Q → Q0, Tf → Tf0};
pd0 = 6.2;
v0 = (ms0 / (p0 - pd0) - b) / a;
```

In[98]:=
```
a11 = D[dVwtdt, Vwt] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
a12 = D[dVwtdt, p] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
a13 = D[dVwtdt, ar] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
a14 = D[dVwtdt, Vsd] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;

a21 = D[dpdt, Vwt] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
a22 = D[dpdt, p] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
a23 = D[dpdt, ar] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
a24 = D[dpdt, Vsd] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;

a31 = D[dardt, Vwt] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
a32 = D[dardt, p] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
a33 = D[dardt, ar] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
```

```
a34 = D[dardt, Vsd] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, ms → ms0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;

a41 = D[dVsddt, Vwt] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
a42 = D[dVsddt, p] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
a43 = D[dVsddt, ar] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
a44 = D[dVsddt, Vsd] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;

b11 = D[dVwtdt, mf] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
b12 = D[dVwtdt, v] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
b13 = D[dVwtdt, Q] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
b14 = D[dVwtdt, Tf] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
b15 = D[dVwtdt, pd] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
b16 = D[dVwtdt, mst] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;

b21 = D[dpdt, mf] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
b22 = D[dpdt, v] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
b23 = D[dpdt, Q] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
b24 = D[dpdt, Tf] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
b25 = D[dpdt, pd] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
b26 = D[dpdt, mst] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;

b31 = D[dardt, mf] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
b32 = D[dardt, v] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
b33 = D[dardt, Q] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
b34 = D[dardt, Tf] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
```

```
b35 = D[dardt, pd] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
b36 = D[dardt, mst] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;

b41 = D[dVsddt, mf] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
b42 = D[dVsddt, v] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
b43 = D[dVsddt, Q] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
b44 = D[dVsddt, Tf] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
b45 = D[dVsddt, pd] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
b46 = D[dVsddt, mst] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;

c21 = D[H, Vwt] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
c22 = D[H, p] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0, Q → Q0,
      Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
c23 = D[H, ar] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0, Q → Q0,
      Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
c24 = D[H, Vsd] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0, Q → Q0,
      Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;

d21 = D[H, mf] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0,
      Q → Q0, Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
d22 = D[H, v] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0, Q → Q0,
      Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
d23 = D[H, Q] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0, Q → Q0,
      Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
d24 = D[H, Tf] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0, Q → Q0,
      Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
d25 = D[H, pd] /. {Vwt → Vwt0, p → p0, ar → ar0, Vsd → Vsd00, mf → mf0, Q → Q0,
      Tf → Tf0, v → v0, pd → pd0, mst → mst0, mL → mL0} // Simplify;
```

In[147]:=
```
A = {{a11, a12, a13, a14},
        {a21, a22, a23, a24},
        {a31, a32, a33, a34},
        {a41, a42, a43, a44}};
B = {{b11, b12, b13, b14, b15, b16},
        {b21, b22, b23, b24, b25, b26},
        {b31, b32, b33, b34, b35, b36},
        {b41, b42, b43, b44, b45, b46}};
Cmat = {{0, 1, 0, 0},
       {c21, c22, c23, c24}};

eqValues = {{Vwt0, p0, ar0, Vsd00, mf0, v0, Q0, Tf0, pd0, H0, mst0, mL0}};
```

In[ ]:=
```
SetDirectory[NotebookDirectory[]];
<< ToMatlab.m

ToMatlab[A, "A"];
ToMatlab[B, "B"];
ToMatlab[Cmat, "C"];
ToMatlab[eqValues, "eqValues"];
```

Appendix F

COMBUSTION SYSTEM LINEARIZATION MATHEMATICA CODE

The code included in this appendix determines the linear gains associated with the combustion system. The gains from this code are used by the code in Appendix G to determine the linear A, B, C, and D matrices for the combustion system.

Model Derivation For Combustion System

Part 1: Parameters and Inputs

In[151]:=
```
x0 = 42.99;
y0 = 5.25;
z0 = 20.95;
MP0 = 0.15;
mf0 = 0.2446;
ma0 = 1.7350;
me0 = ma0 + mf0;
HHV0 = 17531.062;
Ta0 = 75;
UA0 = 1.5804;
Tsat0 = 250.3637 + 273.15;
T0 = 923.1830 + 273.15;
T10 = 755.6964 + 273.15;
T20 = 703.7497 + 273.15;
T30 = 584.7166 + 273.15;
T40 = 531.5424 + 273.15;
ms0 = 0.6518;
cps = 2.7927;
UA1 = .2515;
UA2 = 0.9158;
UA3 = 0.3897;

mwH2O = 18.01528;
mwO2 = 31.998;
mwN2 = 28.013;
mwCO2 = 44.01;
mwC = 12.0107;
mwH = 1.0079;
mwO = 15.999;
mwAir = 28.84;
```

100

Part 2: Model Equations

In[180]:=
```
hfCO2 = -393520.0;
hfH2OL = -285820.0;
hfH2Og = -241820.0;

a = x / mwC;
b = y / mwH;
c = z / mwO;
w = (2.0 * a + b / 2.0 - c) / 2.0;
hffuel = a * hfCO2 + b / 2 * hfH2OL + 100.0 * HHV;
d = MP * mf / mwH2O;
dhO21 = 25.48 * Ta + (1.52 / 2.0 / 100.0) * Ta^2 -
    (0.7155 / 3.0 / 100000.0) * Ta^3 + (1.312 / 4.0 / 1000000000.0) * Ta^4 - 8.2074 * 1000.0;
dhN21 = 28.9 * Ta - 7.855 / 10000.0 * Ta^2 + 2.6937 / 1000000.0 * Ta^3 -
    7.1825 / 10000000000.0 * Ta^4 - 8.6081 * 1000.0;

dhO2 = 25.48 * T[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat] +
    (1.52 / 2.0 / 100.0) * T[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat]^2 -
    (0.7155 / 3.0 / 100000.0) * T[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat]^3 +
    (1.312 / 4.0 / 1000000000.0) * T[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat]^4 - 8.2074 * 1000.0;
dhN2 = 28.9 * T[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat] -
    7.855 / 10000.0 * T[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat]^2 +
    2.6937 / 1000000.0 * T[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat]^3 -
    7.1825 / 10000000000.0 * T[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat]^4 - 8.6081 * 1000.0;
dhCO2 = 22.26 * T[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat] +
    2.9905 / 100.0 * T[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat]^2 -
    1.167 / 100000.0 * T[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat]^3 +
    1.8673 / 1000000000.0 * T[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat]^4 - 8.9951 * 1000.0;
dhH2O = 32.24 * T[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat] +
    9.615 / 10000.0 * T[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat]^2 +
    3.5167 / 1000000.0 * T[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat]^3 -
    8.9875 / 10000000000.0 * T[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat]^4 - 9.7789 * 1000.0;

ratio = 100 * ma / (mwAir * (1 - MP) * mf * w * 4.76);
Hr = hffuel + ratio * w * dhO21 + 3.76 * ratio * w * dhN21 + d * hfH2OL;
Hp = a * (hfCO2 + dhCO2) + (b / 2.0 + d) * (hfH2Og + dhH2O) +
    w * (ratio - 1.0) * dhO2 + 3.76 * ratio * w * dhN2;
Q = UA * 100.0 / ((1.0 - MP) * mf) * (T[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat] - Tsat);
excessO2 = 100.0 * w * (ratio - 1.0) / (a + b / 2.0 + d + w * (ratio - 1.0) + 3.76 * ratio * w);
```

Part 3: Calculate Partial Derivatives

In[200]:=
```
D[Hp - Hr == -Q, x];
kx = T^(1,0,0,0,0,0,0,0,0,0) [x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat] /.
```

```
     Solve[%, T^(1,0,0,0,0,0,0,0,0,0)[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat]] /.
    {x → x0, y → y0, z → z0, MP → MP0, mf → mf0, ma → ma0, HHV → HHV0, Ta → Ta0,
     UA → UA0, Tsat → Tsat0, T[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat] → T0};

D[Hp - Hr == -Q, y];
ky = T^(0,1,0,0,0,0,0,0,0,0)[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat] /.
     Solve[%, T^(0,1,0,0,0,0,0,0,0,0)[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat]] /.
    {x → x0, y → y0, z → z0, MP → MP0, mf → mf0, ma → ma0, HHV → HHV0, Ta → Ta0,
     UA → UA0, Tsat → Tsat0, T[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat] → T0};

D[Hp - Hr == -Q, z];
kz = T^(0,0,1,0,0,0,0,0,0,0)[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat] /.
     Solve[%, T^(0,0,1,0,0,0,0,0,0,0)[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat]] /.
    {x → x0, y → y0, z → z0, MP → MP0, mf → mf0, ma → ma0, HHV → HHV0, Ta → Ta0,
     UA → UA0, Tsat → Tsat0, T[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat] → T0};

D[Hp - Hr == -Q, MP];
kMP = T^(0,0,0,1,0,0,0,0,0,0)[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat] /.
     Solve[%, T^(0,0,0,1,0,0,0,0,0,0)[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat]] /.
    {x → x0, y → y0, z → z0, MP → MP0, mf → mf0, ma → ma0, HHV → HHV0, Ta → Ta0,
     UA → UA0, Tsat → Tsat0, T[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat] → T0};

D[Hp - Hr == -Q, mf];
kmf = T^(0,0,0,0,1,0,0,0,0,0)[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat] /.
     Solve[%, T^(0,0,0,0,1,0,0,0,0,0)[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat]] /.
    {x → x0, y → y0, z → z0, MP → MP0, mf → mf0, ma → ma0, HHV → HHV0, Ta → Ta0,
     UA → UA0, Tsat → Tsat0, T[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat] → T0};

D[Hp - Hr == -Q, ma];
kma = T^(0,0,0,0,0,1,0,0,0,0)[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat] /.
     Solve[%, T^(0,0,0,0,0,1,0,0,0,0)[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat]] /.
    {x → x0, y → y0, z → z0, MP → MP0, mf → mf0, ma → ma0, HHV → HHV0, Ta → Ta0,
     UA → UA0, Tsat → Tsat0, T[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat] → T0};

D[Hp - Hr == -Q, HHV];
kHHV = T^(0,0,0,0,0,0,1,0,0,0)[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat] /.
     Solve[%, T^(0,0,0,0,0,0,1,0,0,0)[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat]] /.
    {x → x0, y → y0, z → z0, MP → MP0, mf → mf0, ma → ma0, HHV → HHV0, Ta → Ta0,
     UA → UA0, Tsat → Tsat0, T[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat] → T0};

D[Hp - Hr == -Q, Ta];
kTa = T^(0,0,0,0,0,0,0,1,0,0)[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat] /.
     Solve[%, T^(0,0,0,0,0,0,0,1,0,0)[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat]] /.
    {x → x0, y → y0, z → z0, MP → MP0, mf → mf0, ma → ma0, HHV → HHV0, Ta → Ta0,
     UA → UA0, Tsat → Tsat0, T[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat] → T0};
```

```
D[Hp - Hr == -Q, UA];
kUA = T^(0,0,0,0,0,0,0,0,0,1,0)[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat] /.
    Solve[%, T^(0,0,0,0,0,0,0,0,0,1,0)[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat]] /.
   {x → x0, y → y0, z → z0, MP → MP0, mf → mf0, ma → ma0, HHV → HHV0, Ta → Ta0,
    UA → UA0, Tsat → Tsat0, T[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat] → T0};

D[Hp - Hr == -Q, Tsat];
kTsat = T^(0,0,0,0,0,0,0,0,0,0,1)[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat] /.
    Solve[%, T^(0,0,0,0,0,0,0,0,0,0,1)[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat]] /.
   {x → x0, y → y0, z → z0, MP → MP0, mf → mf0, ma → ma0, HHV → HHV0, Ta → Ta0,
    UA → UA0, Tsat → Tsat0, T[x, y, z, MP, mf, ma, HHV, Ta, UA, Tsat] → T0};

ko2ma = D[excessO2, ma] /. {x → x0, y → y0, z → z0, MP → MP0,
    mf → mf0, ma → ma0, HHV → HHV0, Ta → Ta0, UA → UA0, Tsat → Tsat0};
ko2mf = D[excessO2, mf] /. {x → x0, y → y0, z → z0, MP → MP0, mf → mf0,
    ma → ma0, HHV → HHV0, Ta → Ta0, UA → UA0, Tsat → Tsat0};
```

Part 4: Heat Transfer Equations

In[222]:=

```
cp = (1.0316 - 5.608 × 10^(-05) Tbefore +
    2.8847 × 10^(-07) * Tbefore^2 - 1.0256 × 10^(-10) * Tbefore^3);
Cr = (me cp)/(ms cps);
Tafter = Tsat + (Tbefore - Tsat) e^(-UA/(me cp));
Tafter2 = (Tsat + Cr Tbefore + (Tbefore - Tsat) e^(-UA (1 + Cr)/(me cp)))/(1 + Cr);

Qi = cp me (Tbefore - Tafter1);
```

Part 3: Define Gains for all Temperatures and Inputs

In[227]:=
```
k1 = 0.86;
k2 = D[Tafter, Tbefore] /. {Tsat → Tsat0, Tbefore → T10, me → me0, UA → UA1};
k3 = D[Tafter, me] /. {Tsat → Tsat0, Tbefore → T10, me → me0, UA → UA1};
k4 = D[Tafter, Tsat] /. {Tsat → Tsat0, Tbefore → T10, me → me0, UA → UA1};

k5 = D[Tafter2, Tbefore] /. {Tsat → Tsat0, Tbefore → T20, me → me0, UA → UA2, ms → ms0};
k6 = D[Tafter2, Tsat] /. {Tsat → Tsat0, Tbefore → T20, me → me0, UA → UA2, ms → ms0};
k7 = D[Tafter2, ms] /. {Tsat → Tsat0, Tbefore → T20, me → me0, UA → UA2, ms → ms0};
k8 = D[Tafter2, me] /. {Tsat → Tsat0, Tbefore → T20, me → me0, UA → UA2, ms → ms0};

k9 = D[Tafter, Tbefore] /. {Tsat → Tsat0, Tbefore → T30, me → me0, UA → UA3};
k10 = D[Tafter, me] /. {Tsat → Tsat0, Tbefore → T30, me → me0, UA → UA3};
k11 = D[Tafter, Tsat] /. {Tsat → Tsat0, Tbefore → T30, me → me0, UA → UA3};

k12 = D[Qi, me] /. {me → me0, Tbefore → T10, Tafter1 → T20};
k13 = D[Qi, Tbefore] /. {me → me0, Tbefore → T10, Tafter1 → T20};
k14 = D[Qi, Tafter1] /. {me → me0, Tbefore → T10, Tafter1 → T20};

k15 = D[Qi, me] /. {me → me0, Tbefore → T10, Tafter1 → T20};
k16 = D[Qi, Tbefore] /. {me → me0, Tbefore → T10, Tafter1 → T20};
k17 = D[Qi, Tafter1] /. {me → me0, Tbefore → T10, Tafter1 → T20};
```

In[244]:=
```
Kmatrix = {kma, kmf, kTsat, ko2ma, ko2mf, k1, k2,
    k3, k4, k5, k6, k7, k8, k9, k10, k11, k12, k13, k14, k15, k16, k17};
```

In[ ]:=
```
SetDirectory[NotebookDirectory[]];
<< ToMatlab.m
ToMatlab[Kmatrix, "Kvalues"]
```

Appendix G

COMBUSTION SYSTEM LINEARIZATION AND CONTROL MATLAB CODE

The code included in this appendix takes the linear gains determined in Appendix F and combines them with linear approximations for time delays and the dynamic time constants and determines the linearized A, B, C, and D matrices. Additionally, the combustion controllers are determined.

```
close all; clear all; clc
%% Pade Approximation
% Freeboard UA Parameter
UA = 1.5804;

% Linearization Gains
kvalues=[(-0.303277E3),0.324485E4,0.397713E0,0.810465E1,( ...
  -0.574897E2),0.86E0,0.905845E0,0.249129E2,0.103102E0,0.754919E0, ...
  0.263407E0,(-0.392456E2),0.500538E2,0.849438E0,0.24701E2, ...
  0.159746E0,0.606509E2,0.233309E1,(-0.23113E1),0.606509E2, ...
  0.233309E1,(-0.23113E1)];

kma = kvalues(1);
kmf = kvalues(2);
kTsat = kvalues(3);
ko2ma = kvalues(4);
ko2mf = kvalues(5);
k1 = kvalues(6);
k2 = kvalues(7);
k3 = kvalues(8);
k4 = kvalues(9);
k5 = kvalues(10);
k6 = kvalues(11);
k7 = kvalues(12);
k8 = kvalues(13);
k9 = kvalues(14);
k10 = kvalues(15);
k11 = kvalues(16);
k12 = kvalues(17);
k13 = kvalues(18);
k14 = kvalues(19);
k15 = kvalues(20);
k16 = kvalues(21);
k17 = kvalues(22);

% Dynamic Empirical Parameters
tauO2  = 25;
tauQ   = 50;
delay1 = 16;
delay2 = 70;

% Pade Delay Approximation Orders
s1 = 2;
s2 = 2;
s = tf('s');

[n1,d1] = pade(delay1,s1);
[A1,B1,C1,D1] = tf2ss(n1,d1);
sys1 = ss(A1, B1, C1, D1);
```

```
[n2,d2] = pade(delay2,s2);
[A2,B2,C2,D2] = tf2ss(n2,d2);
sys2 = ss(A2, B2, C2, D2);

[A3, B3, C3, D3] = tf2ss([1],[tauO2, 1]);
[A4, B4, C4, D4] = tf2ss([1],[tauQ, 1]);

% This creates the gain matrices for the linear heat output
me = [1,1,0,0];
Tsat = [0,0,1,0];
ms = [0,0,0,1];
Tfb = [kma, kmf, kTsat, 0];
Qfb = UA*(Tfb - Tsat);

T1 = k1*Tfb;
T2 = k2*T1 + k4*Tsat + k3*me;
T3 = k5*T2 + k6*Tsat + k7*ms + k8*me;
T4 = k9*T3 + k11*Tsat + k10*me;

Q1 = k12*me + k13*T1 + k14*T2;
Q3 = k15*me + k16*T3 + k17*T4;

Heat = Qfb + Q1 + Q3;

% Gain matrix for linear oxygen content
O2 = [ko2ma, ko2mf, 0, 0];


Bo2 = B3*O2;
Bq = B4*Heat;

Do2 = D3*O2;
Dq = D4*Heat;


Ahx = [A3, 0;
       0, A4];
Bhx = [Bo2;
       Bq];

Chx = [C3, 0;
       0, C4];
Dhx = [Do2;
       Dq];

% Linear combustion system without delays
sysHx = ss(Ahx, Bhx(:,1:2), Chx, Dhx(:,1:2));
%% Input System (Linearized delay dynamics)
```

```
Ain = [A1, zeros(s1,s2);
       zeros(s2,s1), A2];
Bin = [B1, zeros(s1,1);
       zeros(s2,1),B2];
Cin = [C1, zeros(1,s2);
       zeros(1,s1), C2];
Din = [D1, 0;
       0, D2];
sysIn = ss(Ain, Bin, Cin, Din);

sysSmall = series(sysIn, sysHx);
%% Combined System (Contains disturbance inputs)
A = sysSmall.A;
B = [sysSmall.B,G];
C = sysSmall.C;
D = [sysSmall.D, zeros(2,2)];

sysFull = ss(A, B, C, D);
%% Determine Controllers
nx = 2 + s1 + s2;    %Number of states
ny = 2;    %Number of outputs
stateDiag =.1*[1,0,0,0,0,0;
             0,1,0,0,0,0;
             0,0,1,0,0,0;
             0,0,0,1,0,0;
             0,0,0,0,1,0;
             0,0,0,0,0,1];
integratorDiag =1*[4,0;
                   0,1];
% Weighting Matrix for Controller Gains
Q = 0.00000001*blkdiag(stateDiag,integratorDiag);
R = 10*[10,0;%20
     0,20];
[Ki,ss2, ee2] = lqi(sysSmall,Q,R);

% Weighting Matrices For Estimator
Qn = 0.0001*[1,0;
            0,1];
Rn = 100*[1,0;%0.1
          0,1];
[kest,Lkf,Pkf] = kalman(sysFull,Qn,Rn);
eigs(kest.A);

trksys = lqgtrack(kest,Ki);
A_cmb = [A - Lkf*C, zeros(6,2);
         zeros(2,8)];
B_cmb = [zeros(6,2), Lkf, B(:,1:2);
         eye(2), zeros(2,4)];
C_cmb = eye(8,8);
```

```
D_cmb = zeros(8,6);
sys_cmb = ss(A_cmb, B_cmb, C_cmb, D_cmb);

% Determine A,B,C,D matrices for controller
Akf = kest.A;
Bkf = kest.B;
Ckf = eye(2 + s1 + s2);
Dkf = zeros(2 + s1 + s2,4);

save('hxControllers.↙
mat','tauO2','tauQ','Ki','Akf','Bkf','Ckf','Dkf','Lkf','ICKf','delay1','delay2','sys_↙
cmb')
```

Appendix H

PLC CODE

The code included in this appendix is the PLC code written in Structured Control Language (SCL), which is a SIEMENS specific language. This code was written on the PLC and executes the optimal controllers. All matrices in this code are structured as 2-dimensional arrays.

```
REGION Assign Inputs
    // Statement section REGION
    //
    REGION Combustion Values
        IF NOT #Config.ManualControl.CMB.manuallyInput_IO THEN
            #Get.CombustionValues.bedAirMassFlow_kg_p_sec :=
            "CMB_bedBlower".Get.outputData.primaryAirMassFlow_kg_per_s;
            #Get.CombustionValues.exhaustMassFlow_kg_p_sec :=
            "CMB_inducedDraftFan".Get.outputData.exhaustMassFlow_kg_per_s;
            #Get.CombustionValues.fbAirMassFlow_kg_p_sec :=
            "CMB_freeboardBlower".Get.outputData.overFireAir_kg_per_s;
            #Get.CombustionValues.fuelSpeed_pct :=
            "CMB_fuelMeteringAugers".Get.outputData.fuelMeteringAugersSpeed_PCT;
            #Get.CombustionValues.O2_pct :=
            "CMB_boilerTubes_IO".Get.O2Sensor_PCT_vol.Value;
            #Get.CombustionValues.T_fb_average_C :=
            ("CMB_freeboard".Get.outputData.middleFreeboardTemp_C +
            "CMB_freeboard".Get.outputData.upperFreeboardTemp_C +
            "CMB_freeboard".Get.outputData.evap_1_inletTemp_C) / 3.0;
            #Get.CombustionValues.T1_C :=
            "CMB_freeboard".Get.outputData.evap_1_inletTemp_C;
            #Get.CombustionValues.T2_C := "CMB_boilerTubes_IO".Get.evap1ExitTemp_C.Value;
            #Get.CombustionValues.T3_C :=
            "CMB_boilerTubes_IO".Get.superheaterExhaustTemp_C.Value;
            #Get.CombustionValues.T4_C := "CMB_boilerTubes_IO".Get.evap2ExitTemp_C.Value;
            #Get.CombustionFeedback.O2_content := #Get.CombustionValues.O2_pct;
        END_IF;

        IF NOT #Set.observerMode THEN

            #Get.CombustionInputs.massFlowAir := #airMassFlowOld;
            #Get.CombustionInputs.augerSpeed := #fuelSpeedOld;
        ELSE
            #Get.CombustionInputs.augerSpeed := #Get.CombustionValues.fuelSpeed_pct;
            #Get.CombustionInputs.massFlowAir :=
            #Get.CombustionValues.bedAirMassFlow_kg_p_sec +
            #Get.CombustionValues.fbAirMassFlow_kg_p_sec;
        END_IF;

    END_REGION

    REGION Rankine Values
        // Statement section REGION
        IF NOT #Config.ManualControl.RKN.manuallyInput_IO THEN
            #Get.RankineValues.feedwaterMassFlow_kg_p_sec :=
            "RKN_feedwaterPumps".Get.outputData.actualFeedwaterFlow_kg_per_sec;
            #Get.RankineValues.feedwaterTemperature_C :=
            "RKN_economizerHeatExchanger_IO".Get.economizerExitTemp_C.Value;
            #Get.RankineValues.Ldrum_m :=
            ("RKN_steamDrum".Get.outputData.analogDrumLevel -
            #Config.RKNControllerConfigs.L0_inches)*0.0254 + #MATLAB.RKN.eqValues.L0;
            #Get.RankineValues.Pdrum_barg :=
            "RKN_steamDrum".Get.outputData.steamDrumPress_barg;
            #Get.RankineValues.steamMassFlow_kg_p_sec :=
            "RKN_steamPressureAndFlowControl".Get.outputData.steamMassFlow_kg_per_sec;
            #Get.RankineValues.steamSatTemp_C :=
            "RKN_steamDrum".Get.outputData.steamDrumTemp_C;
            #Get.RankineValues.valvePosition_pct :=
            "RKN_steamPressureAndFlowControl_IO".Set.engineBypassThrottleValve.In_pct;
            #Get.RankineValues.dryerPressure_barg :=
            "RKN_desuperHeater".Get.outputData.lowPressSteamMassFlowPress_barg;
            #Get.RankineValues.turbineMassFlow_kg_p_sec :=
            "RKN_steamPressureAndFlowControl".Get.outputData.steamMassFlow_kg_per_sec -
            "RKN_desuperHeater".Get.outputData.lowPressSteamMassFlow_kg_per_s;
        END_IF;

        REGION Assign Inputs
            // Statement section REGION
            IF NOT #Set.observerMode THEN
```

```
47                  #Get.RankineInputs.massFlowFeedwater := #feedwaterFlowOld;
48                  #Get.RankineInputs.valvePosition := #valvePositionOld;
49              ELSE
50                  #Get.RankineInputs.massFlowFeedwater :=
                    #Get.RankineValues.feedwaterMassFlow_kg_p_sec;
51                  #Get.RankineInputs.valvePosition := #Get.RankineValues.valvePosition_pct;
52              END_IF;
53              #Get.RankineFeedback.DrumPressure := #Get.RankineValues.Pdrum_barg + 1.0;
54              #Get.RankineFeedback.DrumLevel := #Get.RankineValues.Ldrum_m;
55              #Get.RankineInputs.dryerPressure := #Get.RankineValues.dryerPressure_barg +
                1.0;
56              #Get.RankineInputs.feedwaterTemp :=
                #Get.RankineValues.feedwaterTemperature_C;
57              #Get.RankineInputs.turbineMassFlow :=
                #Get.RankineValues.turbineMassFlow_kg_p_sec;
58          END_REGION
59
60      END_REGION
61
62  END_REGION
63
64
65  // This determines QCombustion
66  REGION Calculate Heat
67      // This section is for calculating the heat from the Combustion System
68      #Qfb := #MATLAB.CMB.UA_fb * (#Get.CombustionValues.T_fb_average_C -
        #Get.RankineValues.steamSatTemp_C); // kW
69      "dh_N2_kJ_p_kg"(tempIn_C:=#Get.CombustionValues.T1_C,
70                      tempOut_C:=#Get.CombustionValues.T2_C,
71                      dh_kJ_p_kg:=#dh1);
72
73      "dh_N2_kJ_p_kg"(tempIn_C := #Get.CombustionValues.T3_C,
74                      tempOut_C := #Get.CombustionValues.T4_C,
75                      dh_kJ_p_kg := #dh3);
76
77      #Q1 := #Get.CombustionValues.exhaustMassFlow_kg_p_sec * #dh1;
78      #Q3 := #Get.CombustionValues.exhaustMassFlow_kg_p_sec * #dh3;
79      #Qc := #Qfb + #Q1 + #Q3;
80
81      IF NOT #Config.ManualControl.CMB.manuallyInput_IO THEN
82          #Get.CombustionFeedback.Qcombustion := #Qc;
83      END_IF;
84      IF NOT #Config.ManualControl.RKN.manuallyInput_IO THEN
85          #Get.RankineInputs.Qrankine := #Qc;
86      END_IF;
87
88  END_REGION
89
90  REGION Rankine Kalman Filter
91      // Statement section REGION
92      //
93
94      REGION Reset x_hat IC
95          IF #Config.ManualControl.RKN.resetIC THEN
96              // Statement section IF
97              FOR #i := 1 TO 6 DO
98                  // Statement section FOR
99
100                 #x_hat_rkn_old[#i] := #Config.ManualControl.RKN.x_hat_IC;
101             END_FOR;
102
103
104             #Config.ManualControl.RKN.resetIC := FALSE;
105         END_IF;
106     END_REGION
107
108     REGION Find x_hat
109         // States are as follows
110         // x1 = Vwt
```

```
111         // x2 = P
112         // x3 = ar
113         // x4 = Vsd
114         //
115         // Inputs are as follows:
116         // u1 = feedwater
117         // u2 = valve position
118         // u3 = Heat Input
119         // The rest of the inputs are disturbances
120         // u4 = Feedwater Temp
121         // u5 = dryer temp
122         // u6 = turbine mass flow
123         //
124         IF NOT #Config.ManualControl.RKN.manuallyInputPressure THEN
125             #Get.RKN.LQG.setpoints[1] := #MATLAB.RKN.eqValues.P0;
126         END_IF;
127
128         IF NOT #Config.ManualControl.RKN.manuallyInputLevel THEN
129             #Get.RKN.LQG.setpoints[2] := #MATLAB.RKN.eqValues.L0;
130         END_IF;
131
132         #pressureError := #Get.RKN.LQG.setpoints[1] - #Get.RankineFeedback.DrumPressure;
133         #levelError := #Get.RKN.LQG.setpoints[2] - #Get.RankineFeedback.DrumLevel;
134
135         IF ABS(#levelError) < #Config.RKNControllerConfigs.deadband.level THEN
136             #levelError := 0.0;
137         END_IF;
138
139
140         #Get.RKN.LQG.u[1] := #pressureError;
141         #Get.RKN.LQG.u[2] := #levelError;
142         #Get.RKN.LQG.u[3] := #Get.RankineFeedback.DrumPressure - #MATLAB.RKN.eqValues.P0;
143         #Get.RKN.LQG.u[4] := #Get.RankineFeedback.DrumLevel - #MATLAB.RKN.eqValues.L0;
144         #Get.RKN.LQG.u[5] := #Get.RankineInputs.massFlowFeedwater -
            #MATLAB.RKN.eqValues.mfw0;
145         #Get.RKN.LQG.u[6] := #Get.RankineInputs.valvePosition - #MATLAB.RKN.eqValues.v0;
146         #Get.RKN.LQG.u[7] := #Get.RankineInputs.Qrankine - #MATLAB.RKN.eqValues.Q0;
147         #Get.RKN.LQG.u[8] := #Get.RankineInputs.feedwaterTemp -
            #MATLAB.RKN.eqValues.Tfw0;
148         #Get.RKN.LQG.u[9] := #Get.RankineInputs.dryerPressure - #MATLAB.RKN.eqValues.Pd0;
149         #Get.RKN.LQG.u[10] := #Get.RankineInputs.turbineMassFlow -
            #MATLAB.RKN.eqValues.mst0;
150
151         FOR #i := 1 TO 6 DO
152             // Statement section FOR
153             #Ax_hat_rkn := 0.0;
154             FOR #j := 1 TO 6 DO
155                 // Statement section FOR
156                 #Ax_hat_rkn := #Ax_hat_rkn + #MATLAB.RKN.LQG.A_rkn[#i, #j] *
                    #x_hat_rkn_old[#j];
157             END_FOR;
158
159             #Bu_hat_rkn := 0.0;
160             FOR #j := 1 TO 10 DO
161                 // Statement section FOR
162                 #Bu_hat_rkn := #Bu_hat_rkn + #MATLAB.RKN.LQG.B_rkn[#i, #j] *
                    #Get.RKN.LQG.u[#j];
163             END_FOR;
164
165             #x_hat_rkn_new[#i] := #Ax_hat_rkn + #Bu_hat_rkn;
166         END_FOR;
167
168         FOR #i := 1 TO 6 DO
169             // Statement section FOR
170             #Cx_hat_rkn := 0.0;
171             FOR #j := 1 TO 6 DO
172                 // Statement section FOR
173                 #Cx_hat_rkn := #Cx_hat_rkn + #MATLAB.RKN.LQG.C_rkn[#i, #j] *
                    #x_hat_rkn_old[#j];
```

```
174             END_FOR;
175
176             #Du_hat_rkn := 0.0;
177             FOR #j := 1 TO 10 DO
178                 // Statement section FOR
179                 #Du_hat_rkn := #Du_hat_rkn + #MATLAB.RKN.LQG.D_rkn[#i, #j] *
                    #Get.RKN.LQG.u[#j];
180             END_FOR;
181
182             #Get.RKN.LQG.x_hat[#i] := #Cx_hat_rkn + #Du_hat_rkn;
183         END_FOR;
184
185         #x_hat_rkn_old := #x_hat_rkn_new;
186
187     END_REGION
188
189
190     REGION Determine RKN Feedbacks Inputs
191         // Statement section REGION
192         FOR #i := 1 TO 3 DO
193             #ufbConstant_rkn := 0.0;
194             FOR #j := 1 TO 6 DO
195                 // Statement section FOR
196                 #ufbConstant_rkn := #ufbConstant_rkn - #MATLAB.RKN.LQG.K_rkn[#i,#j] *
                    #Get.RKN.LQG.x_hat[#j];
197             END_FOR;
198             #Get.RKN.LQG.u_fb[#i] := #ufbConstant_rkn;
199         END_FOR;
200
201         #Get.Outputs.feedwaterMassFlow := #Get.RKN.LQG.u_fb[1] +
            #MATLAB.RKN.eqValues.mfw0;
202         #Get.Outputs.throttleValvePosition := #Get.RKN.LQG.u_fb[2] +
            #MATLAB.RKN.eqValues.v0;
203         #rankineHeat_setpoint := #Get.RKN.LQG.u_fb[3] + #MATLAB.RKN.eqValues.Q0;
204
205         #feedwaterFlowOld := #Get.Outputs.feedwaterMassFlow;
206         #valvePositionOld := #Get.Outputs.throttleValvePosition;
207
208         REGION Saturate Outputs
209             // Statement section REGION
210             IF #Get.Outputs.feedwaterMassFlow >
                #Config.RKNControllerConfigs.feedwaterSaturations.max THEN
211                 #Get.Outputs.feedwaterMassFlow :=
                    #Config.RKNControllerConfigs.feedwaterSaturations.max;
212             ELSIF #Get.Outputs.feedwaterMassFlow <
                #Config.RKNControllerConfigs.feedwaterSaturations.min THEN
213                 #Get.Outputs.feedwaterMassFlow :=
                    #Config.RKNControllerConfigs.feedwaterSaturations.min;
214             END_IF;
215
216             IF #Get.Outputs.throttleValvePosition >
                #Config.RKNControllerConfigs.valveSaturations.max THEN
217                 #Get.Outputs.throttleValvePosition :=
                    #Config.RKNControllerConfigs.valveSaturations.max;
218             ELSIF #Get.Outputs.throttleValvePosition <
                #Config.RKNControllerConfigs.valveSaturations.min THEN
219                 #Get.Outputs.throttleValvePosition :=
                    #Config.RKNControllerConfigs.valveSaturations.min;
220             END_IF;
221
222             IF #rankineHeat_setpoint > #Config.RKNControllerConfigs.heatSaturations.max
                THEN
223                 #rankineHeat_setpoint :=
                    #Config.RKNControllerConfigs.heatSaturations.max;
224             ELSIF #rankineHeat_setpoint <
                #Config.RKNControllerConfigs.heatSaturations.min THEN
225                 #rankineHeat_setpoint :=
                    #Config.RKNControllerConfigs.heatSaturations.min;
226             END_IF;
```

```
227
228         END_REGION
229
230     END_REGION
231
232     IF #Set.totalControllers_Run THEN
233         #Set.controlFeedwater := TRUE;
234         #Set.controlValve := TRUE;
235         #Set.switchOnCombControllers := TRUE;
236     ELSE
237         #Set.controlFeedwater := FALSE;
238         #Set.controlValve := FALSE;
239     END_IF;
240 END_REGION
241
242 REGION Combustion Kalman Filter
243     // Statement section REGION
244
245
246     REGION Reassign IC
247         // Statement section REGION
248         IF #Config.ManualControl.CMB.resetIC THEN
249             // Statement section IF
250             FOR #i := 1 TO #n_cmb DO
251                 // Statement section FOR
252
253                 #x_hat_cmb_old[#i] := #Config.ManualControl.CMB.x_hat_IC;
254             END_FOR;
255
256
257             #Config.ManualControl.CMB.resetIC := FALSE;
258         END_IF;
259     END_REGION
260
261     REGION Manually Control CMB Setpoints
262         // Statement section REGION
263         IF NOT #Config.ManualControl.CMB.manuallyInputO2 THEN
264             // Statement section IF
265             #Get.CMB.LQG.setpoints[1] :=  #MATLAB.CMB.eqValues.O20;
266         END_IF;
267
268         IF NOT #Config.ManualControl.CMB.manuallyInputHeat THEN
269             // Statement section IF
270             #Get.CMB.LQG.setpoints[2] := #rankineHeat_setpoint;
271         END_IF;
272     END_REGION
273     // The inputs to this system are as follows:
274     // u1 = O2 Setpoint error
275     // u2 = Q setpoint error
276     // u3 = O2 Actual
277     // u4 = Q Actual
278     // u5 = Mass Flow Air
279     // u6 = Mass Flow Fuel
280
281     #Get.CMB.LQG.u[1] := #Get.CMB.LQG.setpoints[1] - #Get.CombustionFeedback.O2_content;
282     #Get.CMB.LQG.u[2] := #Get.CMB.LQG.setpoints[2] - #Get.CombustionFeedback.Qcombustion;
283     #Get.CMB.LQG.u[3] := #Get.CombustionFeedback.O2_content - #MATLAB.CMB.eqValues.O20;
284     #Get.CMB.LQG.u[4] := #Get.CombustionFeedback.Qcombustion - #MATLAB.CMB.eqValues.Q0;
285     #Get.CMB.LQG.u[5] := #Get.CombustionInputs.massFlowAir - #MATLAB.CMB.eqValues.ma0;
286     #Get.CMB.LQG.u[6] := #Get.CombustionInputs.augerSpeed /
        #MATLAB.CMB.pctAugerSpeedToMassFlow - #MATLAB.CMB.eqValues.mf0;
287
288     REGION Find x_hat
289         FOR #i := 1 TO #n_cmb DO
290             // Statement section FOR
291             #Ax_hat_cmb := 0.0;
292             FOR #j := 1 TO #n_cmb DO
293                 // Statement section FOR
294                 #Ax_hat_cmb := #Ax_hat_cmb + #MATLAB.CMB.LQG.A_cmb[#i, #j] *
```

```
                #x_hat_cmb_old[#j];
            END_FOR;

            #Bu_hat_cmb := 0.0;
            FOR #j := 1 TO #p_cmb DO
                // Statement section FOR
                #Bu_hat_cmb := #Bu_hat_cmb + #MATLAB.CMB.LQG.B_cmb[#i, #j] *
                #Get.CMB.LQG.u[#j];
            END_FOR;

            #x_hat_cmb_new[#i] := #Ax_hat_cmb + #Bu_hat_cmb;
        END_FOR;

        FOR #i := 1 TO #n_cmb DO
            // Statement section FOR
            #Cx_hat_cmb := 0.0;
            FOR #j := 1 TO #n_cmb DO
                // Statement section FOR
                #Cx_hat_cmb := #Cx_hat_cmb + #MATLAB.CMB.LQG.C_cmb[#i, #j] *
                #x_hat_cmb_old[#j];
            END_FOR;

            #Du_hat_cmb := 0.0;
            FOR #j := 1 TO #p_cmb DO
                // Statement section FOR
                #Du_hat_cmb := #Du_hat_cmb + #MATLAB.CMB.LQG.D_cmb[#i, #j] *
                #Get.CMB.LQG.u[#j];
            END_FOR;

            #Get.CMB.LQG.x_hat[#i] := #Cx_hat_cmb + #Du_hat_cmb;
        END_FOR;

        #x_hat_cmb_old := #x_hat_cmb_new;
        IF "testBlock".forceO2Integrator THEN
            #x_hat_cmb_old[7] := "testBlock".O2IntegratorValue;
            "testBlock".forceO2Integrator := FALSE;
        END_IF;

    END_REGION

    REGION Determine Feedback Inputs
        // Statement section REGION
        FOR #i := 1 TO 2 DO
            // Statement section FOR
            #ufbConstant_cmb := 0.0;
            FOR #j := 1 TO #n_cmb DO
                // Statement section FOR
                #ufbConstant_cmb := #ufbConstant_cmb - #MATLAB.CMB.LQG.K_cmb[#i, #j] *
                (#Get.CMB.LQG.x_hat[#j]);
            END_FOR;
            #Get.CMB.LQG.u_fb[#i] := #ufbConstant_cmb;
        END_FOR;

        #Get.Outputs.airMassFlow := #Get.CMB.LQG.u_fb[1] + #MATLAB.CMB.eqValues.ma0;
        #Get.Outputs.fuelSpeed := (#Get.CMB.LQG.u_fb[2] + #MATLAB.CMB.eqValues.mf0) *
        #MATLAB.CMB.pctAugerSpeedToMassFlow;

        #fuelSpeedOld := #Get.Outputs.fuelSpeed;
        #airMassFlowOld := #Get.Outputs.airMassFlow;

        IF #Get.Outputs.airMassFlow > #Config.CMBControllerConfigs.airSaturations.max
        THEN
            #Get.Outputs.airMassFlow := #Config.CMBControllerConfigs.airSaturations.max;
        ELSIF #Get.Outputs.airMassFlow <
        #Config.CMBControllerConfigs.airSaturations.min THEN
            #Get.Outputs.airMassFlow := #Config.CMBControllerConfigs.airSaturations.min;
        END_IF;

        IF #Get.Outputs.fuelSpeed > #Config.CMBControllerConfigs.fuelSaturations.max THEN
```

```
356             #Get.Outputs.fuelSpeed := #Config.CMBControllerConfigs.fuelSaturations.max;
357         ELSIF #Get.Outputs.fuelSpeed < #Config.CMBControllerConfigs.fuelSaturations.min
            THEN
358             #Get.Outputs.fuelSpeed := #Config.CMBControllerConfigs.fuelSaturations.min;
359         END_IF;
360
361     END_REGION
362
        #bedAirMassFlow_Filter_firstOrderLowPassFilter_Instance(yFilteredOutput:=#filteredBed
        AirMassFlow,
363                                                                xInput:=#Get.CombustionValues
                                                                .bedAirMassFlow_kg_p_sec,
364
                                                                filter:=#Config.bedAirMassFlo
                                                                wFilter);
365
366     "CMB_freeboardBlower".Config.overFireAirMassFlowController.SetPoint_kg_s :=
        #Get.Outputs.airMassFlow - #filteredBedAirMassFlow;
367      IF #Set.switchOnCombControllers THEN
368         #Set.observerMode := FALSE;
369         #Set.airMassFlowController_run := TRUE;
370         "CMB_fuelMeteringAugers".Config.ManualControl.manuallyControl_IO := TRUE;
371         "CMB_fuelMeteringAugers_IO".Set.drySludgeMeteringAuger_1.FreqReference_PCT :=
            #Get.Outputs.fuelSpeed + "testBlock".offset;
372         "CMB_fuelMeteringAugers_IO".Set.drySludgeMeteringAuger_2.FreqReference_PCT :=
            #Get.Outputs.fuelSpeed + "testBlock".offset;
373         #manualControlTurnOff := TRUE;
374     ELSE
375         //#Set.observerMode := TRUE;
376         #Set.airMassFlowController_run := FALSE;
377         IF #manualControlTurnOff THEN
378             "CMB_fuelMeteringAugers".Config.ManualControl.manuallyControl_IO := FALSE;
379             #manualControlTurnOff := FALSE;
380         END_IF;
381
382     END_IF;
383
384 END_REGION
385
386 "testBlock".Qtest := #Get.CMB.LQG.x_hat[2] * 0.02 + #MATLAB.CMB.eqValues.Q0;
387 "testBlock".O2Test := #Get.CMB.LQG.x_hat[1] * 0.04 + #MATLAB.CMB.eqValues.O20;
388
389 REGION Logging
390     REGION Combustion
391         #Config.loggingData.CMB.CMB_u1.Value := #Get.CMB.LQG.u[1];
392         #Config.loggingData.CMB.CMB_u2.Value := #Get.CMB.LQG.u[2];
393         (* #Config.loggingData.CMB.CMB_u3.Value := #Get.CMB.LQG.u[3];
394         #Config.loggingData.CMB.CMB_u4.Value := #Get.CMB.LQG.u[4];
395         #Config.loggingData.CMB.CMB_u5.Value := #Get.CMB.LQG.u[5];
396         #Config.loggingData.CMB.CMB_u6.Value := #Get.CMB.LQG.u[6]; *)
397
398         #Config.loggingData.CMB.O2_feedback.Value := #Get.CombustionFeedback.O2_content;
399         #Config.loggingData.CMB.Q_feedback.Value := #Get.CombustionFeedback.Qcombustion;
400         #Config.loggingData.CMB.fuelSpeed_output.Value := #Get.Outputs.fuelSpeed;
401         #Config.loggingData.CMB.airMassFlow_output.Value := #Get.Outputs.airMassFlow;
402
403         (* #Config.loggingData.CMB.CMB_x1.Value := #Get.CMB.LQG.x_hat[1];
404         #Config.loggingData.CMB.CMB_x2.Value := #Get.CMB.LQG.x_hat[2];
405         #Config.loggingData.CMB.CMB_x3.Value := #Get.CMB.LQG.x_hat[3];
406         #Config.loggingData.CMB.CMB_x4.Value := #Get.CMB.LQG.x_hat[4];
407         #Config.loggingData.CMB.CMB_x5.Value := #Get.CMB.LQG.x_hat[5];
408         #Config.loggingData.CMB.CMB_x6.Value := #Get.CMB.LQG.x_hat[6];
409         #Config.loggingData.CMB.CMB_x7.Value := #Get.CMB.LQG.x_hat[7];
410         #Config.loggingData.CMB.CMB_x8.Value := #Get.CMB.LQG.x_hat[8]; *)
411
412         #Config.loggingData.CMB.overFireAirMassFlow_kg_p_sec.Value :=
            "CMB_freeboardBlower".Get.outputData.overFireAir_kg_per_s;
413         #Config.loggingData.CMB.exhaustMassFlow_kg_p_sec.Value :=
```

```
        "CMB_inducedDraftFan".Get.outputData.exhaustMassFlow_kg_per_s;
414
415     "updateTCPLogReal"(#Config.loggingData.CMB.CMB_u1);
416     "updateTCPLogReal"(#Config.loggingData.CMB.CMB_u2);
417     (* "updateTCPLogReal"(#Config.loggingData.CMB.CMB_u3);
418     "updateTCPLogReal"(#Config.loggingData.CMB.CMB_u4);
419     "updateTCPLogReal"(#Config.loggingData.CMB.CMB_u5);
420     "updateTCPLogReal"(#Config.loggingData.CMB.CMB_u6); *)
421
422     "updateTCPLogReal"(#Config.loggingData.CMB.O2_feedback);
423     "updateTCPLogReal"(#Config.loggingData.CMB.Q_feedback);
424     "updateTCPLogReal"(#Config.loggingData.CMB.airMassFlow_output);
425     "updateTCPLogReal"(#Config.loggingData.CMB.fuelSpeed_output);
426
427     (* "updateTCPLogReal"(#Config.loggingData.CMB.CMB_x1);
428     "updateTCPLogReal"(#Config.loggingData.CMB.CMB_x2);
429     "updateTCPLogReal"(#Config.loggingData.CMB.CMB_x3);
430     "updateTCPLogReal"(#Config.loggingData.CMB.CMB_x4);
431     "updateTCPLogReal"(#Config.loggingData.CMB.CMB_x5);
432     "updateTCPLogReal"(#Config.loggingData.CMB.CMB_x6);
433     "updateTCPLogReal"(#Config.loggingData.CMB.CMB_x7);
434     "updateTCPLogReal"(#Config.loggingData.CMB.CMB_x8); *)
435     "updateTCPLogReal"(#Config.loggingData.CMB.overFireAirMassFlow_kg_p_sec);
436     "updateTCPLogReal"(#Config.loggingData.CMB.exhaustMassFlow_kg_p_sec);
```

VITA

Michael Janicki is a graduate student in the Mechanical Engineering Department at the University of Washington. His research was motivated by his time working as a controls engineer on the J-OP S250 at Sedron Technologies.

He welcomes your comments and questions at `mjanicki@uw.edu`.