Murdoch
UNIVERSITY

SCHOOL OF ENGINEERING AND INFORMATION TECHNOLOGY

Integration of MATLAB and LabVIEW with Aspen Plus Dynamics

Using Control Strategies for a High-Fidelity Distillation Column

Thesis submitted to the school of Engineering and Information Technology, Murdoch University in partial fulfillment of the requirements for the degree of

Bachelor of Engineering Honours [BE(Hons)]
Instrumentation and Control, Electrical Power

Word Count: 14,861

Joshua Eggins

Supervisor: Dr. Linh Vu

November 2015

This page has intentionally been left blank

Declaration

I, Joshua Malcolm Eggins, certify that this work contains no material which has been accepted for the award of any other degree or diploma in my name, in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text.

In addition, I certify that no part of this work will, in the future, be used in a submission in my name, for any other degree or diploma in any university or other tertiary institution without the prior approval of the Murdoch University.

I give consent to this copy of my thesis, when deposited in the University Library, being made available for loan and photocopying, subject to the provisions of the Copyright Act 1968.

I also give permission for the digital version of my thesis to be made available on the web, via the University's digital research repository, the Library Search and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.

Signed: ……..

Name: ……..

Date: ……..

This page has intentionally been left blank

Abstract

The energy intensive distillation process has become a widely discussed topic as industry attempts to minimise energy consumption. The implementation of *Model Predictive Control* (MPC) can aid in the reduction of plant energy consumption. However, the leading chemical and petroleum software packages Aspen Plus and Aspen HYSYS do not currently support MPC. This project successfully integrated both MATLAB and LabVIEW with *Aspen Plus Dynamics* (APD), which enables the implementation of MPC schemes. This integration was established using Microsoft's ActiveX Technology. In order to implement MPC from within MATLAB and LabVIEW, their respective MPC toolboxes were explored; these toolboxes possess several major flaws in their functionality. In particular, neither have the ability to perform RGA analysis or determine the model of the plant through data-driven modelling. To overcome these drawbacks a MATLAB script was developed which determines the model of the plant from automatic step tests in Simulink. Once the communication was established, and toolboxes documented, a high-fidelity distillation column was constructed in Aspen Plus before being exported to APD. This plant model was developed as a reference to compare the effectiveness of the PI and MPC control schemes, employing the *Integral of Time-Weighted Absolute Error* (ITAE) performance criterion. MPC outperformed the PI control schemes in all but one scenario. On average the ITAE values were 1000% lower for MPC, due to its ability to quickly track the set point and avoid overshoot. Further research has been highlighted on a number of toolbox features and dynamic communication options. Importantly, the use of the integrated software packages can provide a number of benefits for students and personnel. By developing a dynamic template it will be possible to implement these ideas into university, laboratory and workplace training. This could increase confidence in predictive control schemes, operator plant knowledge and reduce unsafe plant operation.

This page has intentionally been left blank

Contents

Figures

Equations

Tables

Acronyms and Abbreviations

ACCE	Aspen Capital Cost Estimator
AHD	Aspen HYSYS Dynamics
APC	Advanced Process Control

APD	Aspen Plus Dynamics
ASW	Aspen Simulation Workbook
ATV	Auto-Tune Variation
COM	Component Object Model
DLL	Dynamic Linked Library
DMC	Dynamic Matrix Control
DV	Disturbance Variable
Excel	Microsoft Excel
Fortran	Formula Translation
FVT	Final Value Theorem
GMC	Generic Model Control
HYSYS	Aspen HYSYS
ITAE	Integral of Time-Weighted Absolute Error
LabVIEW	Laboratory Virtual Instrument Engineering Workbench
MATLAB	Matrix Laboratory
MIMO	Multiple-Input Multiple-Output
MPC	Model Predictive Control
MV	Manipulated Variable
NI	National Instruments
OLE	Object Linking and Embedding
PFD	Process Flow Diagram
PID	Proportional, Integral, Derivative
PV	Process Variable
RGA	Relative Gain Array
SISO	Single-Input Single-Output

SP	Set Point
SS	State Space
TF	Transfer Function
UOM	Units of Measurement
VB	Visual Basic
VBA	Visual Basic for Applications

Units

$°C$	Degrees Celsius
°F	Degrees Fahrenheit
atm	Atmosphere
bar	
Btu	British thermal unit
ft	Foot
g	Gram
hr	Hour
in	Inch
lb	Pound
m	Meter
mol	Mole
psi	Pound per square inch
s	Second
W	Watt

Acknowledgements

Firstly, I would like to express my gratitude to my supervisor, Dr. Linh Vu. Your door was always open when I needed support and guidance over the last two years. You continued to challenge me to achieve higher than I could have ever imagined. Thank you for helping shape me into the engineer I am. Additionally, I would like to thank Will Stirling for promptly assisting me throughout my research. You had every issue resolved within the day regardless of your prior commitments, I appreciate your time.

I am forever indebted to my mother, Eleonora Korljan. You will never realise how much I appreciate everything you have done for me. You taught me the true meaning of hard work. To my partner Esther Healy, thank you for your consistent encouragement. You were always available when I needed advice or feedback and never lost faith in me. You continue to motivate me to push my limits. Finally, to the strong group of friends I am fortunate to have made throughout my studies. Thank you for assisting me in my development, I learned greatly off each of you over the years. In particular, special thanks to Chris McGivern for his advice with my thesis formatting and structure.

This page has intentionally been left blank

1.0 Introduction

From 1991 to 2002 the manufacturing sector accounted for approximately one quarter of the total energy consumption in the U.S. (U.S. Energy Information Administration, 1991; 1994; 1998; 2002; 2006). Within this sector the chemical and petroleum industries contributed the majority of this consumption. Approximately 40-50% of this consumption is due to the distillation process, caused by the energy intensive thermal separation process (Cheremisinoff, 2000; Olujić et al. 2008; Gorak & Sorensen, 2014, p. 226). As a result of this consumption, the U.S. Environmental Protection Agency targeted superior energy efficiency through refined processes as a major goal for saving energy in the industrial sector (Neelis, Worrell, & Masanet, 2008). This instigated a drop in energy consumption within the manufacturing industry to one fifth of the total U.S. consumption (U.S. Energy Information Administration, 2010). This decrease in consumption has plateaued since and additional methods are being investigated to reduce energy consumption. One such method is the implementation of advanced control schemes which can minimise operational costs and energy usage in the distillation process. The implementation of advanced control schemes at the Algyo Gas Plant in Hungary returned a 35% decrease in energy costs over a 12 month period (Emerson, 2011). However, in order to simulate these controller schemes, improvements must be made to current industrial software packages.

AspenTech is the market-leading process software provider in industry (Ma, 2013, p. 15; AspenTech 2015a). Their software packages Aspen Plus and *Aspen HYSYS* (HYSYS) prove useful for modelling and simulating complex chemical and petroleum processes involving distillation columns, reactors and heat exchangers. These packages have dynamic equivalents appropriately named *Aspen Plus Dynamics* (APD) and *Aspen HYSYS Dynamics* (AHD)

respectively. APD and AHD are however limited by their controller selection, which only allows for conventional PID. Given current industrial trends this does not satisfy the growing need for advanced controllers. AspenTech offers an additional software package, DMCPlus, which utilises the *Dynamic Matrix Control* (DMC) algorithm and allow for *Model Predictive Control* (MPC) to be extended to their software packages. However, this package comes at an additional cost to the end user, so expanding the capabilities of the original software packages would be beneficial. In order to achieve these expansions, ActiveX communication servers will be established from MATLAB and LabVIEW to APD. Since Aspen Plus and HYSYS are similar in operation, and the expansions are transferable between packages, only APD will be utilised in this thesis; this software was preferred as the distillation column being modelled is commonly found in chemical plants.

This software amalgamation will provide a number of benefits. The integration with MATLAB will provide a means to test the performance of advanced control schemes on complex plants. This will determine any negative or positive effects that new, or upgraded, advanced control schemes would have on plant operation. Moreover, the integration with LabVIEW creates an educational tool which can be operated as a real time simulator. This simulation package will allow students and personnel to gain invaluable experience operating complex plants without the hazards associated with real plant dynamics, effectively opening up an avenue for more training on advance control schemes and hopefully increased implementation in industry.

The primary aim of this thesis is to enable communication between MATLAB and LabVIEW to APD. Once this integration of software has been validated an additional aim will be to document the capabilities of the MPC toolbox found within each package. Finally, the performance of MPC will be compared against conventional PID control by minimising the

Integral of Time-Weighted Absolute Error (ITAE) performance criterion. This will be to establish the most efficient control scheme for reducing energy consumption.

2.0 Background

2.1 Distillation

2.1.1 Distillation Process

Distillation, through the use of distillation columns, is a common method for separating mixtures of two or more substances in the pharmaceutical, petroleum, food, and chemical industries. This technique exploits the differing boiling points of the input feed substances and separates them into a vapour and liquid. The vapour is rich in the lower boiling point substances while the liquid contains the remaining products (Khoury, 2005, pp. 61-62).

2.1.2 Distillation Columns

Distillation columns are designed from several major components: column; condenser; reflux drum; and reboiler. Figure 1 presents the make-up of a standard distillation column. The condenser cools the vapour which leaves the column through the top stage. This vapour is condensed and sent to the reflux drum before some is recycled, through the reflux stream, back into the column and the remainder extracted in the distillate stream (Green & Perry, 2007, p. 13.4). The reboiler, located at the bottom stage, provides vaporisation

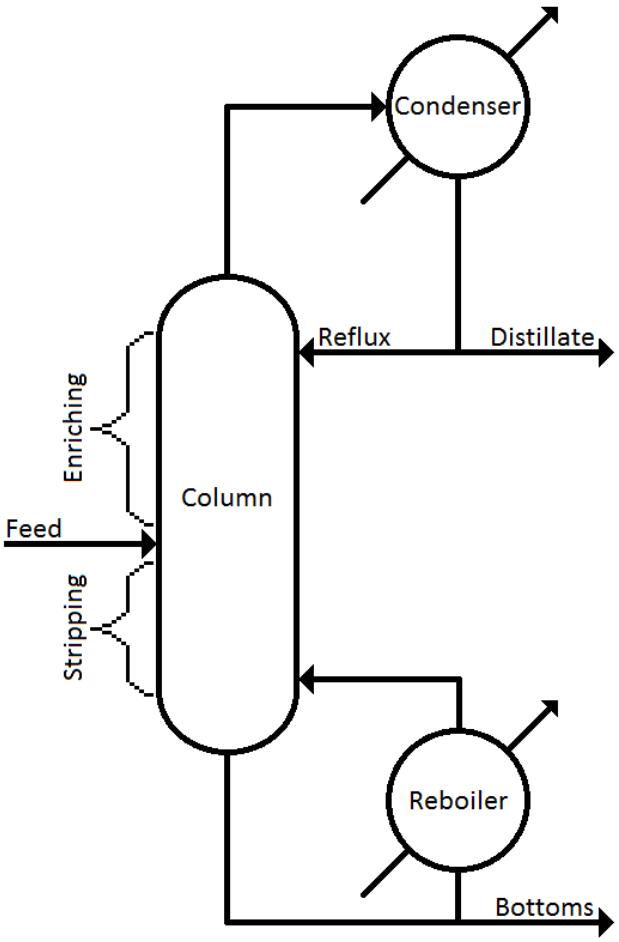

Figure 1: Distillation Column.

for the process. It boils the liquid which is leaving the column in the final stage and reintroduces it in to the column (Green & Perry, 2007, p. 13.4). The remaining liquid is removed in the bottoms stream. The constant heating and cooling of the mixture is the major contributor to energy expenditure in distillation columns (Green & Perry, 2007, p. 13.5). The liquid mixture fed into the column around the middle tray, known as the feed tray, divides the column into the stripping section, below the feed tray, and enriching section, above the feed tray; as shown in Figure 1.

2.1.3 Column Types

The most commonly found distillation column in industry is a continuous column (Mujtaba, 2004, p. 3). These columns are capable of high throughput and, under normal operation, are fed a continuous stream (Green & Perry, 2007, p. 13.4). Another column is batch fed where the feed is input batch-wise and the process completed. Once the process completes the batch is extracted before the next batch introduced (Mujtaba, 2004, pp. 3-5). If the separated material is high in solids, a batch separation should be employed (Mujtaba, 2004, p. 8).

This is however not the only identifying factor of a distillation column. The nature of the feed also plays a role; if two components are fed to the column it is referred to as binary whilst more than two components present in the feed is a multi-component column (Green & Perry, 2007, p. 13.4). Furthermore, it is possible for the column to have multiple product streams (Green & Perry, 2007, p. 13.6). These distillation techniques are useful when the components have boiling point limitations or do not separate during standard distillation processes (Douglas, 1988, p. 185). Crude oil distillation is an example of this, there are many product streams consisting of components with similar volatility.

2.1.4 Column Design

Additionally, the internal operation of the column is dependent on the design employed to enable contact between the vapour and liquid inside the column (Green & Perry, 2007, p. 13.4). A packed column is typically divided into three types (Kister, 1992, p. 421). The first two packing types are structured mesh and grids. These are corrugated sheets arranged within the column in either a wire mesh arrangement or open lattice grid. Random packings however, are discrete geometrical shapes which are randomly packed into the column shell; this is the most common practice in industry (Kister, 1992, p. 421). The overall aim of packing is to maximise the surface area per unit volume, essentially increasing the vapour-liquid contact area and the columns overall efficiency (Kister, 1992, pp. 422-423).

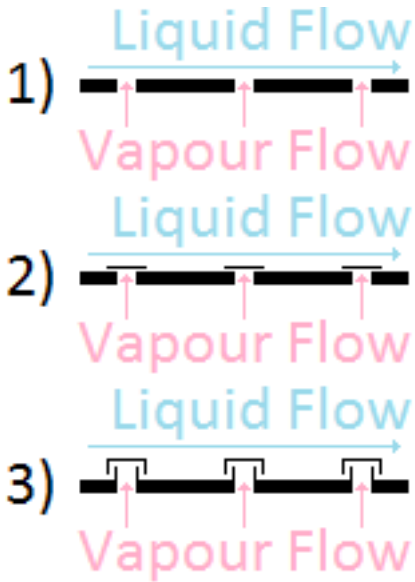

Figure 2: Tray Types: 1) Sieve; 2) Valve; 3) Bubble Cap.

A tray column uses trays, or plates, to enable contact between the vapour and liquid (Kister, 1992, p. 259). Figure 2 illustrates the three primary types of tray in operation within this column style: sieve; valve; and bubble cap. A sieve tray is a metal plate which has holes in it to allow the vapour to pass through it and relies on the kinetic energy of the vapour to keep the liquid above the tray surface (Kister, 1992, p. 260). A valve tray follows the same design as the sieve tray however the holes on the surface of the tray are covered with lift valves. This was introduced to stop occasional leaking through the openings. It also provides an increase in the range of flow rates due to the varying size of the opening when the valve lifts (Kister, 1992, p. 260). Again, the bubble cap tray is an adaption on sieve trays. However unlike valve trays which lift open from the flow, the holes are covered with a cap which the vapour flows into

and exits via small openings in the cap. The bubble cap trays were initially the most widely used however, due to the associated cost are seldom used in industry now (University of Michigan, 2010).

Regardless of their design, the trays are constructed to allow liquid hold up to cover the holes on the surface of the tray, including the valve or cap, entirely. This is achieved by the installation of a weir. This weir is located next to a conduit, referred to as a downcomer. When the liquid held up by the weir exceeds the weir height it will flow into the downcomer, through gravitation force, onto the tray plate located in the stage below. While this occurs, the vapour rises from the stage below through the holes in the tray, as shown in Figure 3, and liquid on the trays surface (Green & Perry, 2007, p. 13.4). This enables a transfer of energy between the liquid and vapour and results in some vapour condensing and liquid evaporating at each stage, aiding to the separation process (Green & Perry, 2007, p. 13.6).

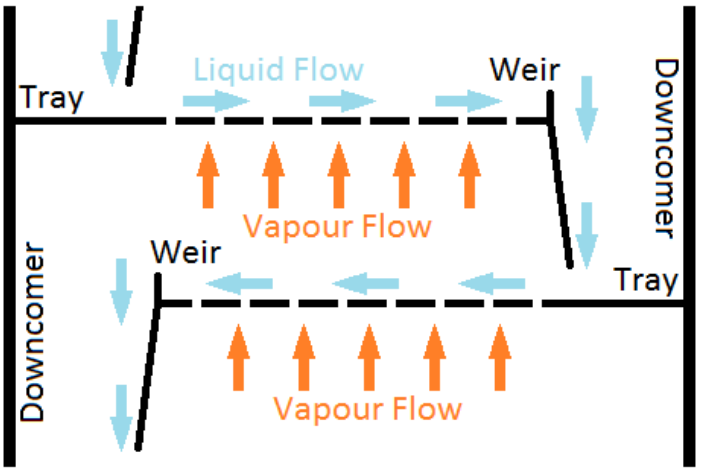

Figure 3: Flow of Liquid and Vapour within Trays.

2.2 Process Control

The objective of process control is to design and implement a controller which results in the dynamics of the process following a desired response. The effect a controller has on the output is dependent on system dynamics, but also on the type of controller used.

2.2.1 PID

Traditionally, feedback PID (Proportional K_p, Integral τ_i, Derivative τ_d) controllers are used to control most processes due to their robust design and easy implementation (Romagnoli & Palazoglu, 2005, pp. 164-165). They do not require intricate knowledge of the underlying process however do not offer optimal control of the process.

$$MV = K_p\left(1 + \frac{1}{\tau_i * s} + \tau_d * s\right)$$

1: PID Algorithm.

PID controllers have three primary parameters, see Equation 1 above, however the derivative term is scarcely used due to its unwanted sensitivity to noise (Ang, Chong, & Li, 2005, p. 561).

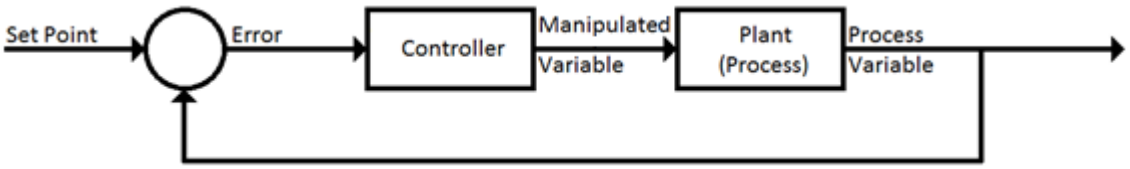

Figure 4: Feedback Block Diagram.

The controller action, M*anipulated Variable* (MV), is based on the error (ϵ) calculated at that given time. This error, the difference between the controller S*et Point* (SP) and the *Process Variable* (PV), see Equation 2, is fed back into the controller continuously to determine the action the controller should take to minimise the error (Ogunnaike & Ray, 1994, pp. 462-463).

$$\epsilon = SP - PV$$

2: Error Calculation.

By sending the PV back to the controller to calculate the error, a feedback loop is formed as shown in Figure 4. This style of control scheme is referred to as feedback control (Ogunnaike & Ray, 1994, pp. 462-463).

2.2.2 SISO v MIMO

Single-Input Single-Output (SISO) is the simplest and most common type of controller. It uses one control signal to control one output. More complicated systems however, require *Multiple-Input Multiple-Output* (MIMO) controllers due to the complex nature of their dynamics. PID is generally not MIMO control (Woolf, 2011), however recent development in advanced controllers means model based and MPC are capable of handling MIMO. It should be noted that in industrial processes, strong interaction between variables will be present and MIMO controllers will always outperform the easier to implement SISO controllers (Ogunnaike & Ray, 1994, p. 992). This interaction between control loops is referred to as coupling. It is ideal to minimise the coupling through selective pairing of MVs to PVs (Romagnoli & Palazoglu, 2005, p. 251). Furthermore, MIMO and advanced control schemes introduce the ability to overcome inherent nonlinearities and difficult process dynamics, such as: inverse response; significant time delays; and open loop instability (Ogunnaike & Ray, 1994, p. 993).

2.2.3 Generic Model Control

Generic Model Control (GMC) is a Model Based Control strategy developed by Lee and Sullivan (1988). It uses the nonlinear mathematical models of the plant to determine the controller action and desired trajectory. An advantage of GMC is its ability to completely reject *Disturbance Variables* (DV) when implemented with an accurate model. This is due to the fact the nonlinear model is directly involved in the controller action algorithm (Lee & Sullivan, 1988).

$$\left(\frac{dy}{dt}\right)_{ref} = K_1 * \epsilon + K_2 * \int \epsilon \; dt \qquad \text{3: Reference Trajectory.}$$

Equation 3 shows the reference trajectory with its two tuning parameters, K_1 and K_2, and the error term, ϵ. Equating this reference with the nonlinear model equation, it is possible to then

rearrange for the MV (Lee & Sullivan, 1988). The resulting equation is the model based controller algorithm which can be easily introduced into the control scheme like a standard PID controller. It should be noted that because GMC is dependent on the model equation its success does rely heavily on the accuracy of that model. Large deviations dramatically affect its ability to measure, estimate and predict the behaviour of the process (Lee & Sullivan, 1988).

2.2.4 Model Predictive Control

Another controller falling under the banner of Model Based Control is MPC. Ogunnaike and Ray (1994) claim MPC was born from the need to create consistent high quality product, efficient use of energy and increase constraints on plant processes to meet expanding environmental responsibilities. MPC employs a corrective controller action which predicts the plant behaviour then rectifies itself to account for any irregularities in its prediction model and direct the output as close to the SP as possible (Ogunnaike & Ray, 1994, p. 992). The key features of MPC are (Maciejowski, 2002, pp. 1-2):

- Predicts future behaviour of the process over a finite time horizon;
- Computes the future controller actions while optimising a cost objective function given equality and inequality constraints; then
- Applies the first, current time, controller action and compare the plants behaviour to that of its prediction model.

As such, MPC requires a model of the process in order to predict the plants behaviour and calculate the controller action. Determining a mathematical model of the plant can be time consuming and laborious, however, it is also possible to predict the plant behaviour given a step response (convolution) model (Maciejowski, 2002, pp. 108-115) (Romagnoli & Palazoglu, 2005, p. 321).

To correctly implement such a model, a step is applied to each MV and the open loop responses of the PV are logged. In theory, given the assumption of linearity, these models will enable the controller to predict the behaviour of the plant for any change in the MV (Maciejowski, 2002, pp. 108-109). These step response then form a step response matrix, as given in Equation 4.

$$H(t) = \begin{bmatrix} h_{11}(t) & h_{12}(t) \dots & h_{1m}(t) \\ h_{21}(t) & h_{22}(t) \dots & h_{2m}(t) \\ \dots & \dots & \dots \\ \dots & \dots & \dots \\ h_{p1}(t) & h_{p2}(t) \dots & h_{pm}(t) \end{bmatrix}$$

4: Step Response Matrix.

Where:

m is the number of MVs;

p is the number of PVs; and

$h_{pm}(t)$ is the response of PV p from a step in MV m.

This step response matrix is also referred to as the Dynamic Matrix of the plant (Maciejowski, 2002, p. 110). The convolution model is intuitive however cannot be exercised on unstable systems. An additional method, which is increasing in popularity, is *State Space* (SS). This technique does allow unstable open loop systems to be modeled though it does require significant theoretical knowledge to implement correctly (Romagnoli & Palazoglu, 2005, p. 322). It is worth noting that it is possible to convert the convolution model into SS for use with newer MPC packages (Maciejowski, 2002, pp. 113-120). Due to the ease in determining dynamic plant behaviour models, and MPC's proficiency in optimisation, this control scheme has enjoyed substantial industry success (Ogunnaike & Ray, 1994, p. 991).

2.2.4.1 Dynamic Matrix Control

MPC is an umbrella name given to the entire prediction controller family. The most well-known and used MPC in industry is DMC. It was first devised by Dr. Cutler in his Ph.D dissertation and later developed through his company DMC Corporation, before being acquired by AspenTech (Boyes, 2009, p. 623). One of the disadvantages of MPC is the large number of model coefficients needed to describe the response. As the DMC algorithm utilises the convolution model these coefficients can be obtained directly from the step response data (Romagnoli & Palazoglu, 2005, p. 323). The MPC controller parameters are (Romagnoli & Palazoglu, 2005, p. 326):

- N, model horizon;
- N_p, prediction horizon;
- N_u, control horizon; and
- W_1 and W_2, weighting matrices.

It is recommended that the model horizon be selected large enough for the open loop response to settle, $N\Delta t \geq open\ loop\ settling\ time$, in order for the controller to know the complete dynamic behaviour of the plant (Romagnoli & Palazoglu, 2005, p. 326). The control horizon decides the number of control actions to calculate in order to predict the plant output over the number of time steps specified by the prediction horizon. Romagnoli and Palazoglu (2005) recommend the prediction horizon be the summation of the model horizon and control horizon. If the prediction horizon is increased the controller will have a more conservative action. On the contrary, if the control horizon is increased it will produce excessive controller action. Lastly, the weighting matrices, W_1 and W_2, determine the amount of control action or tracking error by applying a penalty to the DMC algorithm. Typically the weighting matrices are applied using a ratio, see Equation 7 (Romagnoli & Palazoglu, 2005, p. 326).

$$W_1 = I, \quad W_2 = \rho I$$

5: Weighting Matrices.

Figure 5 shows the step response curve and the corresponding values for a_i, where $0 \leq i \leq N$.

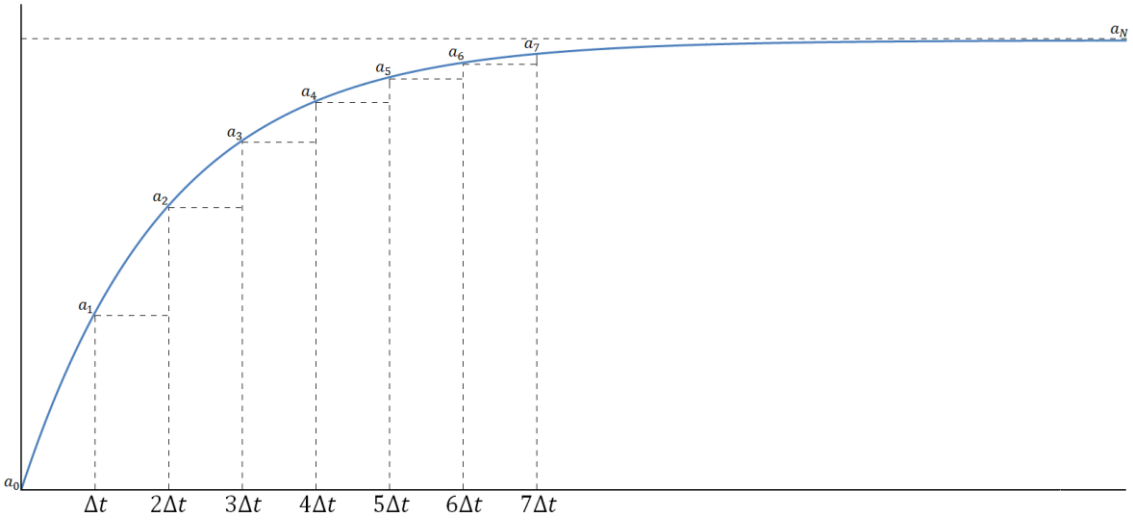

Figure 5: Convolution Model from Input Step Response.

These values are determined by sampling the step response curve at intervals of Δt, then applied to form the vector of step response coefficients, as given in Equation 8 (Romagnoli & Palazoglu, 2005, p. 323).

$$\mathrm{a} = [\mathrm{a}_1, a_2, \ldots, a_N]^T$$

6: Step Response Vector.

The coefficients of this array are then used to construct the coefficient matrix shown in Figure 9 (Romagnoli & Palazoglu, 2005, p. 324).

$$A = \begin{bmatrix} a_1 & 0 & \ldots & 0 \\ a_2 & a_1 & \ldots & 0 \\ \ldots & \ldots & & \ldots \\ a_{N_{p-1}} & a_{N_{p-2}} & \ldots & 0 \\ a_{N_p} & a_{N_{p-1}} & \ldots & a_1 \end{bmatrix}$$

7: Coefficient Matrix.

Given this, the controller action can now be determined, taking into account the weighting matrices and coefficient matrix. Equation 10 illustrates the DMC controller algorithm, allowing for the weighting penalties (Romagnoli & Palazoglu, 2005, p. 326).

$$\Delta \mathrm{U} = \left(\mathrm{A}^{\mathrm{T}} W_1 A + W_2\right)^{-1} A^T W_1 (Y^{SP} - Y^{PAST} - D)$$

$$\Delta U = K_{DMC}(Y^{SP} - Y^{PAST} - D)$$

8: DMC Algorithm.

2.2.5 Relative Gain Array

The *Relative Gain Array* (RGA), sometimes referred to as the Bristol Array after inventor Edgar Bristol (Bristol, 1966), is determined from the array of gains from the plant. Using the step response matrix, as shown in Equation 4, it is possible to determine the gains of the plant, using *Final Value Theorem* (FVT). This provides the gain matrix in Equation 5.

$$K = \begin{bmatrix} k_{11} & k_{12} & \dots & k_{1m} \\ k_{21} & k_{22} & \dots & k_{2m} \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ k_{p1} & k_{p2} & \dots & k_{pm} \end{bmatrix}$$

9: Gain Matrix.

Using the matrix determined in Equation 9, the RGA can be determined via Equation 6.

$$\Lambda = (K^{-1})^T .* K$$

10: RGA Algorithm.

This matrix then provides a measure for the interaction between each input and output variable (Romagnoli & Palazoglu, 2005, p. 256). Note that the multiplication is element by element, not matrix multiplication. As such, $\Lambda_{ij} = (K^{-1})^T_{ij} * K_{ij}$. This array is governed by a set of interaction rules which enable the best loop pairing to be selected (Romagnoli & Palazoglu, 2005, pp. 258-259):

- $\lambda_{ij} < 0$, unstable operation;
- $\lambda_{ij} = 0$, input j has no effect on output i;
- $0 < \lambda_{ij} < 0.5$, the interacting input(s) have a stronger effect on output j than input i;
- $\lambda_{ij} = 0.5$, the effect of input j has an equal contribution on output i as the interacting input(s);
- $0.5 < \lambda_{ij} < 1$, input j has a stronger effect on output i than the interacting input(s);

- $\lambda_{ij} = 1$, input j is the only variable which affects output i, there is no interaction; and
- $\lambda_{ij} > 1$, the effect input j has on output i is greater than the interaction and in the opposite direction.

Given these rules it can be stated that loop pairings should be selected to have RGA elements close to unity and never negative (Ogunnaike & Ray, 1994, pp. 735-740). Once the interaction and loop pairings are selected, only then is it possible to implement MPC.

2.2.6 Performance Criterion

All control schemes are judged against their ability to track the SP and reject DV, and although they are established on the same premise not all schemes are equal (Ang, Chong, & Li, 2005, pp. 562-563). In order to effectively assess the controller schemes, ITAE performance criterion will be minimised (Levine, 1996, p. 170). This was selected because ITAE penalises errors which persist over time heavier than those at the beginning of the response. This can lead to a sluggish initial response, essential to avoid oscillations, however does result in quicker settling times (Levine, 1996, p. 170). Equation 11 shows the algorithm for calculating the error associated with ITAE.

$$\text{ITAE} = \int t|\epsilon|\, dt \qquad \text{11: ITAE Algorithm.}$$

2.3 Software in industry

Computer simulations have been widely adopted through industry to model complex practical settings; through the use of mathematical modelling (Kheir, 1995, pp. vii-viii). These range from, but are not limited to: agriculture; industrial process; risk forecast; stock market; poison

flow; and liquid flow (Robinson, 1993; Kheir, 1995, pp. vii). The popularity of process simulation stems from its ability to support industry through: cost estimation; control and management of operational plants; and troubleshooting or testing of new and upgrading plants (Robinson, 1993). These all have the added benefit of reducing capital cost and detecting poorly designed maintenance strategies or project plants. Software packages not only provide support to industry they also form a useful tool for educating personnel and students. The ability to gain an understanding of plant dynamics and control schemes as well as obtain optimised plant performance is invaluable. It is important however, as more of industry becomes reliant on simulation software that the mathematical and simulation models and results are validated to ensure reliability and consistency of results (Ali & Petersen, 2012).

2.3.1 Historians

Computer packages such as OSIsoft PI and DeltaV are historians which provide real time data logging through a process information server. This server operates underneath a visual interface which is accessible to engineers, operators and managers for analysis and visualisation of operational performance (OSIsoft, 2015). These systems enable plant data, from multiple locations, to be stored on a central server and accessed by all plant personnel and strategic management. In addition they also provide real time control over plant instruments by operators located either on site or remote (Emerson, 2015).

Increasingly these distributed control systems are offering the implementation of advanced control, like DeltaV's MPCPro block. This upgrade in software can optimise plant performance easily through selection of MIMO variables including up to 80 process outputs and 40 inputs (Emerson, 2013). However without training or prior experience in advanced control systems it is near impossible to implement such schemes effectively.

2.3.2 SimSci PRO/II

SimSci PRO/II is a simulation software package from Schneider Electric which allows the optimisation of plant design and operational analysis. It includes a substantial chemical database which, coupled with the unit operators (such as advanced units heat exchangers, distillation columns, reactors), enables the simulation of chemical, petroleum, polymer and pharmaceutical plants. The package uses a *Process Flow Diagram* (PFD) to graphically display the unit operators and plant design and can be used to determine the effect plant upgrades have on process outputs (Schneider Electric, 2015). However, PRO/II is only a steady state simulator. Although it allows optimisation of plant specifics, such as the feed tray of the distillation column, it does not possess the capabilities to analyse dynamic plant operation. It allows the implementation of feedback control loops, using PID control schemes. However, it is not possible to implement advanced control schemes, or analyse transient plant behaviour (Schneider Electric, 2015).

2.3.3 Aspen Plus

Despite PRO/II's simulation proficiency, the inability to perform dynamic analysis leaves it well behind the industry leader. AspenTech's Aspen Plus and APD are the most widely used computer package in the chemical industry (Ma, 2013, p. 15). Aspen Plus is a steady state simulator, similar to PRO/II in operation and design, which allows complex processes to be built without the need for tedious calculations or arduous mathematics (AspenTech, 2000a). Aspen Plus utilises a database of chemical properties and applications in order to enable minimal user interaction with plant and chemical specifics (AspenTech, 2000a). Typically the end user only needs to specify the plant components and their values in order to successfully operate Aspen Plus. APD is the dynamic equivalent of Aspen Plus, allowing analysis of the dynamic behaviour of the plant and implementation of PID control schemes (Peers, 2013).

2.3.4 Aspen HYSYS

HYSYS was originally created by Hyprotech before being acquired by AspenTech and Honeywell; rereleased as Aspen HYSYS and UniSim Design respectively. HYSYS is used in the energy industry and is the leading software package for oil and gas simulation and process optimisation in design and operations (AspenTech, 2015b). It has the capacity to simulate advanced systems such as: pipelines; hydraulics; fractionation LNG; dehydration; and compression; and offers a complete package for modelling an entire refinery (AspenTech, 2015b). As stated previously in 1.0 Introduction, HYSYS will not be explored as the operation and integration of software is very similar to APD. However, for the interested reader, Professor Hanyak's Chemical Process Simulation and the Aspen HYSYS Software (2012) provides a very detailed user manual for HYSYS and the implementation of its complex unit operations. Similar to APD, HYSYS can be operated in steady state or dynamic mode through the use of its dynamic counterpart AHD. As with APD, AHD has the choice to only implement conventional PID control schemes (AspenTech, 2015c).

2.3.5 Aspen Capital Cost Estimator

An additional reason the AspenTech product range enjoys significant dominance in industry is its cost estimation software, *Aspen Capital Cost Estimator* (ACCE). The AspenTech software range can increase profits through rigorous modelling of refineries and plants, while ACCE reduces decision making processes and delivers estimates within 5-10% of the actual cost (AspenTech, 2015d). This advantage enables the end user to make better informed decisions regarding plant construction and operation.

2.3.6 MATLAB

MathWorks' *Matrix Laboratory* (MATLAB) is a high-level language and interactive environment. It is used by engineers, scientists and economists worldwide for tasks such as: numeric computation; data analysis and visualisation; programming and algorithm development; and application development (MathWorks, 2015a). These capabilities can be upgraded through the use of additional add-on products which build on MATLAB's foundation software. These cover a range of applications: optimisation; signal processing; control systems; and finance (MathWorks, 2015b). Furthermore, Simulink is a block diagram environment for model based design which runs on top of MATLAB. This software can be used to easily build and simulate models as well as connect to hardware while running in real time (Mathworks, 2015c). The computational power of MATLAB and its additional toolboxes and add-ons make it a key software package across many industries. The utilisation of these features will be explored.

2.3.7 LabVIEW

Laboratory Virtual Instrument Engineering Workbench (LabVIEW), from *National Instruments* (NI), is a development environment which uses visual programming. It is frequently used for: data acquisition; instrument control; and industrial automation; and was designed to accelerate the productivity of engineers and scientists (National Instruments, 2015a). The biggest draw of LabVIEW is its unprecedented integration with all measurement hardware and software. NI have gone to extreme lengths to ensure LabVIEW is easy to use for the end user and adaptable to most industrial needs (National Instruments, 2015a). Moreover, LabVIEW is used on University and College campuses all over the world to deliver hands-on learning and enhance research (National Instruments, 2015b). This positions LabVIEW as the ideal software

package to enable students and personnel to gain and develop skills in implementing advanced control schemes on complex plants once integration with APD is established.

2.3.8 Microsoft Excel

Lastly, Microsoft's *Microsoft Excel* (Excel) has become the industry leader for spreadsheet and data analysis. Its capabilities include functions, graphical interfaces, programming and communication (Cook, 2015; Microsoft, 2015a). The macro programming is implemented in Microsoft's *Visual Basic for Applications* (VBA), a variety of *Visual Basic* (VB), and can be used to create user functions, customising Excel or automation processes. VBA is not limited to Excel but also applies across the entire Microsoft Office software packages (Microsoft, 2013).

2.4 ActiveX

In order to establish a server connection between one software package to another, Microsoft's proprietary technology *Object Linking and Embedding* (OLE) and inter-process communication OLE Automation, also referred to as ActiveX Technology, will be used (Microsoft, 2015b). This mechanism originated for use in VB however has been expanded to all scripting languages on Windows in response to the problem of cross application macro programming. OLE Automation provides the infrastructure for applications to establish connections and manipulate shared objects between a client-server model. The coding and use of this communication follows the C and VB programing languages (Microsoft, 2015b). Figure 6 displays a central client server with a four client distribution network.

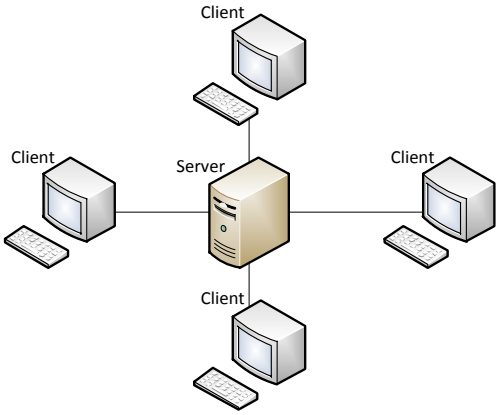

Figure 6: Distributed Server-Client Structure.

2.4.1 Client Server

The server provides a service or resource to clients. The client will make requests of the server and wait for the response of the server and any information it has requested (Microsoft, 2015b). The in-process server is implemented through a *Dynamic Linked Library* (DLL) by using OLE Automations capabilities to create a *Component Object Model* (COM) server. An advantage of Microsoft Windows is the majority of software packages are already integrated with ActiveX and servers can easily be created by using their existing programmatic identifier (Microsoft, 2015b). Examples of these identifiers are Microsoft Excel 'Excel.Application', Microsoft Outlook 'Outlook.Application' and MATLAB 'Matlab.Application'. Using the DLL 'Aspen Customer Modeler 30.0' it is possible to connect to a range of AspenTech's software through the following handles: Aspen Plus Dynamics 'AD Application'; Aspen Customer Modeler 'ACM Application'; Aspen Adsim 'ADS Application'; Aspen Chromatography 'ACH Application'; and Aspen HYSYS 'HYSYS.Application' (AspenTech, 2000b).

2.4.2 Properties and Methods

Setting up the COM server is not the only issue which needs to be explored. Once the communication is established the client needs to send and request information from the server (Microsoft, 2015b). The properties allow the client to open and close specific files, run programs and send and receive data from the server. The methods are functions which can be called from the server (Microsoft, 2015b).

3.0 Modelling

To implement model based controllers, such as GMC, an accurate mathematical model must be developed. Given that the literature on modelling multicomponent distillation columns is lacking, Sharmila and Mangaiyarkarasi's (2014) paper on binary column modelling was used as a reference to develop a binary model. In addition, using the mathematical models from Lee and Dudukovic (1998), a multicomponent model was attempted but was not successful. In order to progress with the project, either a binary column had to be adopted or the model based controllers removed. The success of GMC relies greatly on the precision of the model, in addition to the system having a relative degree of one. This means the MV must appear in the mathematical model for the controlled PV (Lee P. L., 1993). As a result, GMC has not enjoyed wide scale success in industry. Thus it was decided that instead of adopting a binary column for the project, and including model based controllers, the multicomponent column would be used. In the multicomponent distillation column the behaviour was determined using data-driven models. These models can only be used for stable open loop systems and can come in the form of either: SS; TF; zero-pole gain; and linear models. In order to create MPC controllers in MATLAB and LabVIEW these data-driven models were developed following the introduction of input steps to the plant model in APD. APD handles the complex mathematics, chemical, and thermodynamics of the distillation column and allow the user to utilise the data-driven models without the need for mathematical models.

4.0 High-Fidelity Distillation Column

The previous chapters provided an overview and background into the capabilities and theory behind distillation columns, control schemes, modelling and industrial software packages. Using this knowledge a distillation columns will be implement in Aspen Plus. The continuous distillation column uses 29 ideal stages to separate a mixture of benzene, toluene, and p-xylenes. The feed stream contains 30% benzene, 40% toluene and 30% p-xylene and flows at a rate of 500 kmol/hr. It is desired that the distillation column will recover 95% of benzene in the distillate stream. For distillation column sizing and hydraulics refer to 10.1 Appendix 1.

4.1 Aspen Plus

4.1.1 Setup

It is a common practice for any plant or refinery to be designed in AspenTech's steady state simulation software before being exported to dynamics (Peers, 2013, p. 2). To do so, the 'radfrac' block was used in Aspen Plus as the type of distillation column. A review of the documentation (South Dakota School of Mines and Technology, 2000; AspenTech, 2000a) provided the steps for using and setting up the 'radfrac' block in the steady state simulator. As this documentation is out dated, a refreshed version has been prepared and available in 10.3 Appendix 3. This supplementary document will enable future work to be done in Aspen Plus without the need to sort through multiple documents and provides the end user with a single point of reference. To setup the model, the steady state conditions, which can be found in 10.1 Appendix 1, were input following the guidelines provided in 10.3 Appendix 3. Once completed, the simulation can be run to ensure convergence occurs. Figure 7 shows the distillation column after set up in Aspen Plus.

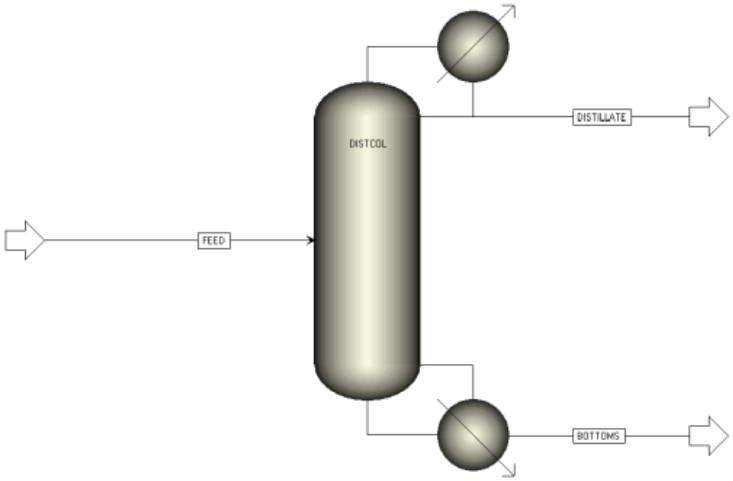

Figure 7: Distillation Column Setup in Aspen Plus.

With the model set up the system was tested under steady state conditions to ensure the plant was set up correctly. Unfortunately, this run yielded no results due to an error. Investigating the cause of this resulted in discovering the databanks in Aspen Plus were not available, as shown in Figure 8. After the installation of Aspen Plus on a new device the software will run an installation to save the databases from the server onto the device. This installation had failed and thus there were only two generic databases available within Aspen Plus. The AspenTech software packages was removed from the device and reinstalled by a Murdoch University technician and, when the software was launched,

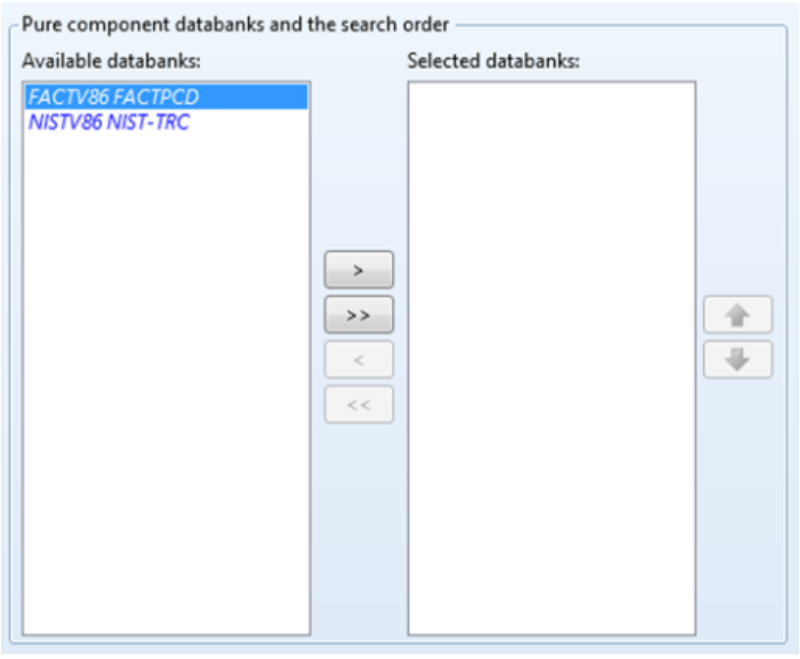

Figure 8: Databases Available in Aspen Plus.

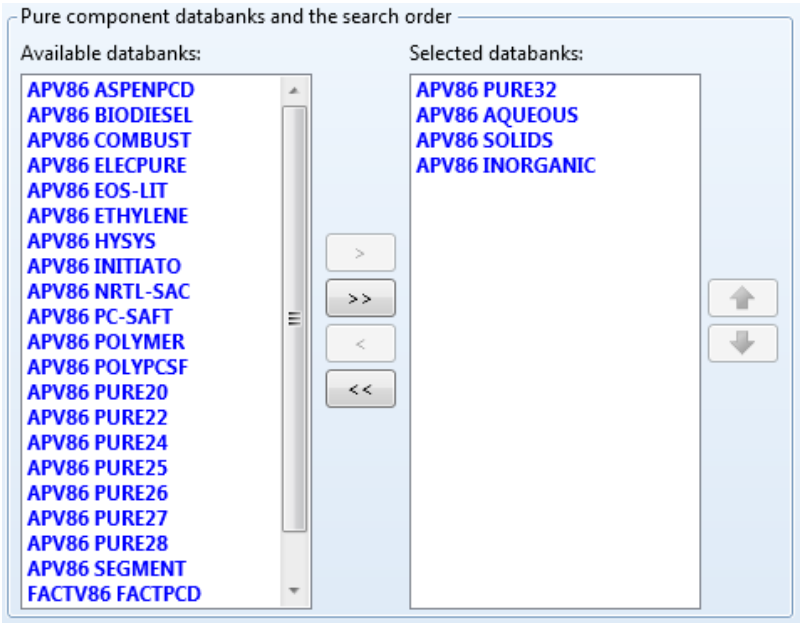

Figure 9: Correctly Installed Database Availability.

the databases successfully installed. Figure 9 displays the correctly installed library of databases, where Aspen Plus automatically selected some foundation databases based on the selection of the template at start-up.

With the databases installed correctly, a steady state run was undertaken and achieved convergence within three iterations, as shown in Figure 10. A check of the system results, given in Table 1, indicates the column was set up correctly and the model ready to export to APD. These databases are essential as without them the chemical properties of the materials are unknown.

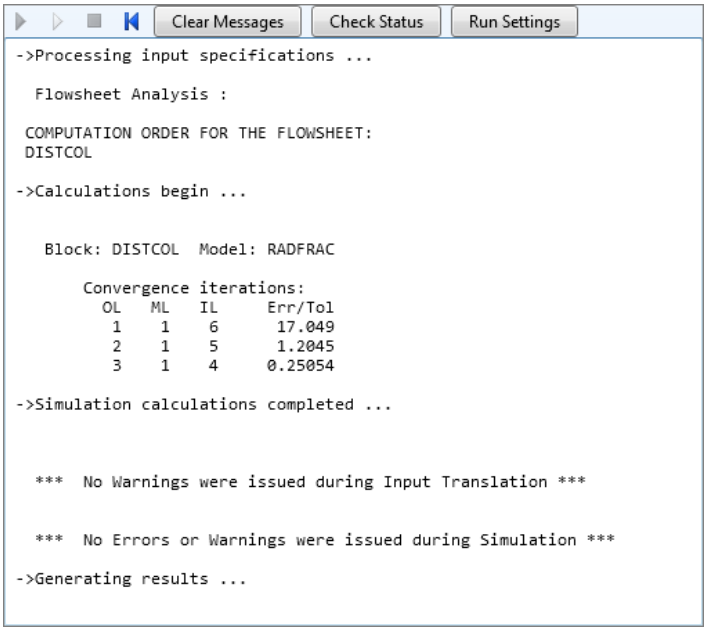

Figure 10: Simulation Output in Aspen Plus.

Table 1: Simulation Results from Aspen Plus.

	FEED	DISTIL	BOTTOMS
Mole Flow kmol/hr			
BENZENE	150.000	136.358	13.643
TOLUENE	200.000	13.642	186.358
P-XYLENE	150.000	0.000	150.000
Mole Frac			
BENZENE	0.300	0.909	0.039
TOLUENE	0.400	0.091	0.532
P-XYLENE	0.300	0.000	0.429
Total Flow kmol/hr	500.000	150.000	350.000
Temperature °C	100.000	81.977	136.498
Pressure bar	1.520	1.013	1.621

4.2.2 Model Export

Before exporting a model to APD a few steps must be undertaken (Peers, 2013, pp. 2-5):

- Isolate the unit operators to export;
- Active 'Dynamic Mode' in the 'Dynamics' tab on Aspen Plus' ribbon;
- Decide on which analysis to perform – pressure or flow driven;
- Enter unit operators dynamic specifications – such as heat-transfer and which variables to control;
- Run the simulation to ensure convergence; and
- Click on the analysis to perform – exporting the model.

10.4 Appendix 4 provides a summary of this procedure however a few key items should be highlighted. The controller MV and PV can be overridden in APD and the choice of these does not impact on the exporting of the file. These options are given to the end user to allow the simulation to set up the model in APD automatically, therefore saving time for the user. Moreover, if the user wants to analyse the pressure gradient then the pressure driven analysis

must be selected. By selecting this type of analysis the user will need to insert additional pumps or values to achieve this pressure gradient in the model (Peers, 2013, p. 3). However, if no pressure gradient is required then flow driven analysis is always the selection to be made. Most commonly this will be the choice and it does not require any addition operators to be added to the PFD (Peers, 2013, p. 3). Additionally, once the analysis mode is selected it will prompt the end user to save the file and will do so under the APD file extension.

	Stage1	Stage2	Diameter	Spacing	Weir height	Lw/D	% Active area	Overall efficiency	% Hole area	Hole dia	% Downcomer escape area	Foaming factor
			meter	in	cm					meter		
▶	2	28	1.95	18	5	0.72666	90	1	10	0.0254	10	1

Figure 11: Hydraulics Tab for the 'Radfrac' Distillation Column in Aspen Plus.

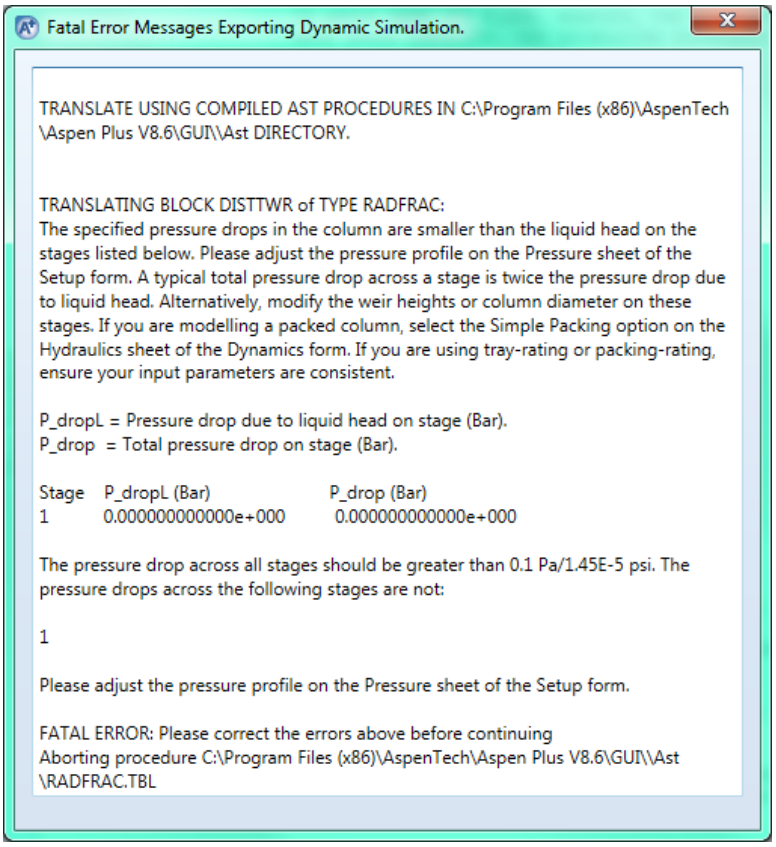

Figure 12: Fatal Error When Completing Run in Aspen Plus.

In Aspen Plus the left-hand side of the window displays the model explorer. This model explorer contains all the information required to simulate the plant model within different navigation panes. The reflux drum and sump geometry were entered in the distillation column dynamics navigation pane, as defined in 10.1 Appendix 1. Then the

simple tray hydraulics was selected and entered as shown in Figure 11. A detailed procedure for preparing the PFD for export to APD can be found in 10.4 Appendix 4. Once all the information was entered, flow driven analysis was selected which initiates the plant exporting process. While attempting to export Aspen Plus encountered a fatal error which stated that a pressure drop through stage 1 did not exist, see Figure 12. By comparing the column implemented to the examples found in Jump Start: Pressure Relief Scenario in Aspen Plus Dynamics V8 (2013) it was determined that the dynamic pressure profile for the distillation column stages was incorrectly entered and thus Aspen Plus had no basis for the pressure drops throughout the column. To correct this issue the pressure profile was changed from fixing the pressure in the first stage to providing an estimate of the first and last stage pressures. Once this was corrected Aspen Plus exported the model without difficulty.

4.2 Aspen Plus Dynamics

4.2.1 Start Up

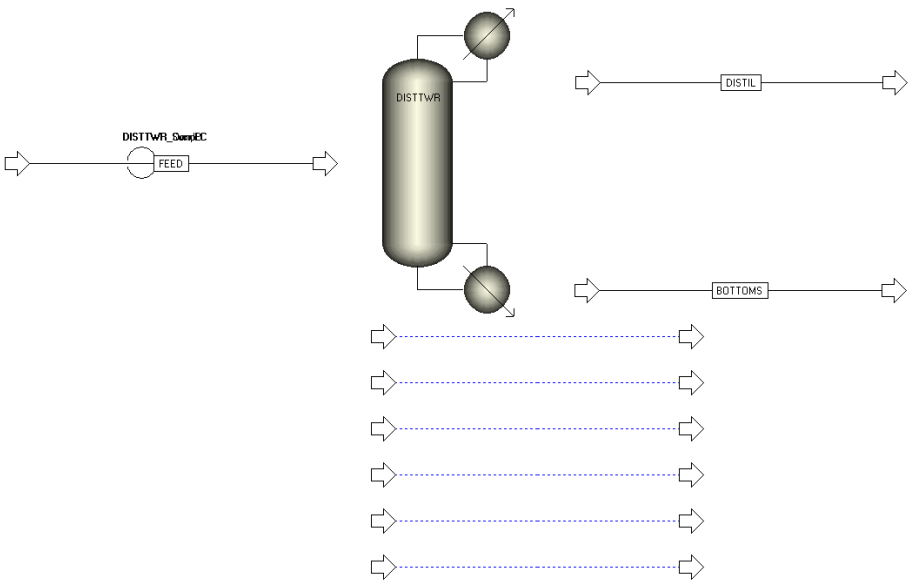

Figure 13: Disconnected Streams in Aspen Plus Dynamics.

With the model exported it was opened in APD. However, when the file was opened in APD an error occurred causing the entire product and communication streams to disconnect, as shown in Figure 13. Reconnecting the streams resulted in APD crashing. In an attempt to isolate this issue, the file was opened on an additional device however, this produced the same outcome. The Aspen Plus model was developed from scratch again to determine if an error had occurred in the creation of the model but this was not successful in correcting the issue. Finally, APD examples were opened, from AspenTech's Example Library, and the same error ensued. This suggested the issue was not in the models but the software itself. After consultation with a Murdoch University technician it was advised that the software version of Aspen Plus and APD could be the root cause of the fault. Following this advice led to the discovery that AspenTech had done only a part release of their software package and Aspen Plus was a different version to APD. To counteract this AspenTech included a special software version selector which allows the older AspenTech products to distinguish the version the original model is being imported from. Figure 14 shows this in the Start menu programs list. Once this was updated to anticipate V8.6 instead of V8.4 the models opened in APD without error. Figure 15 shows the Aspen version selector tool.

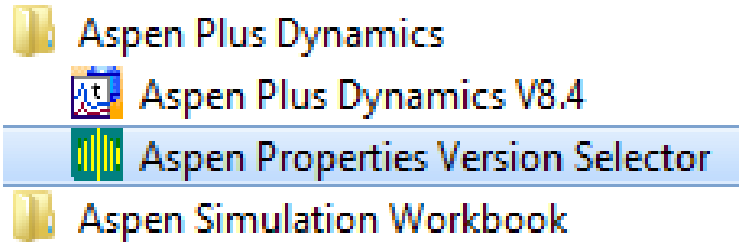

Figure 14: Aspen Properties Version Selector in the Programs List.

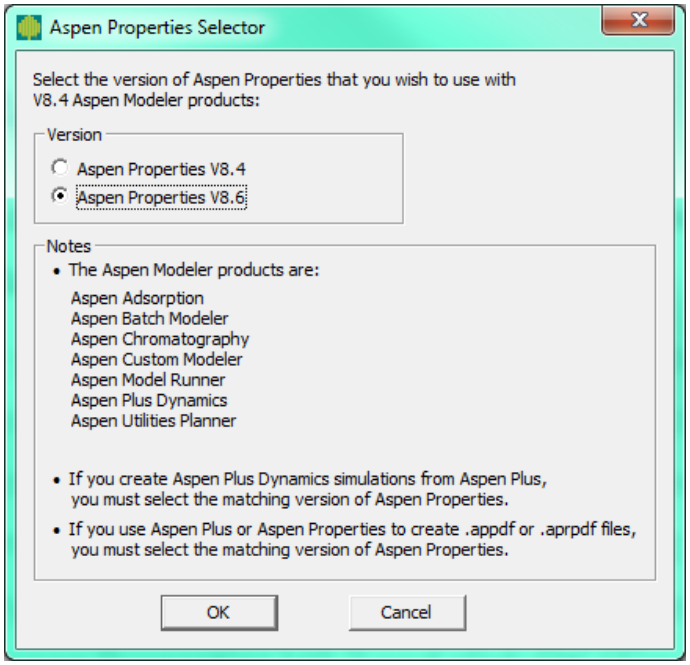

Figure 15: Aspen Version Selector.

4.2.2 Operation

With the model now in APD it is important to understand how the software works and its capabilities. Once the model was opened in APD, the PID feedback loops using the recommended control loops from Aspen Plus are generated. This is shown below in Figure 16.

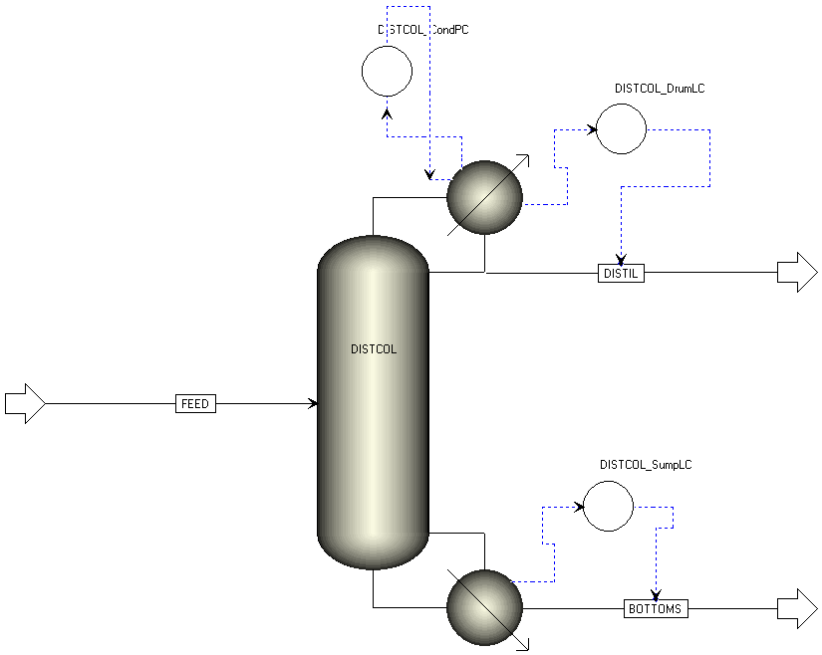

Figure 16: Dynamic PFD in APD with PID Control.

APD offers an easy to use visual interface. The simulation can be run from the ribbon using the controls found in Figure 17. Furthermore, it is important to specify the run mode, communication intervals and units in the Run Options. Figure 18 shows these options which are accessed by pressing F9 whilst in APD. The time units have been updated to seconds and the interval 1 second. Furthermore, it is possible to manipulate variables and events by writing automation tasks. The creation of these tasks is documented in 10.6 Appendix 6.

Figure 17: Simulation Methods in APD.

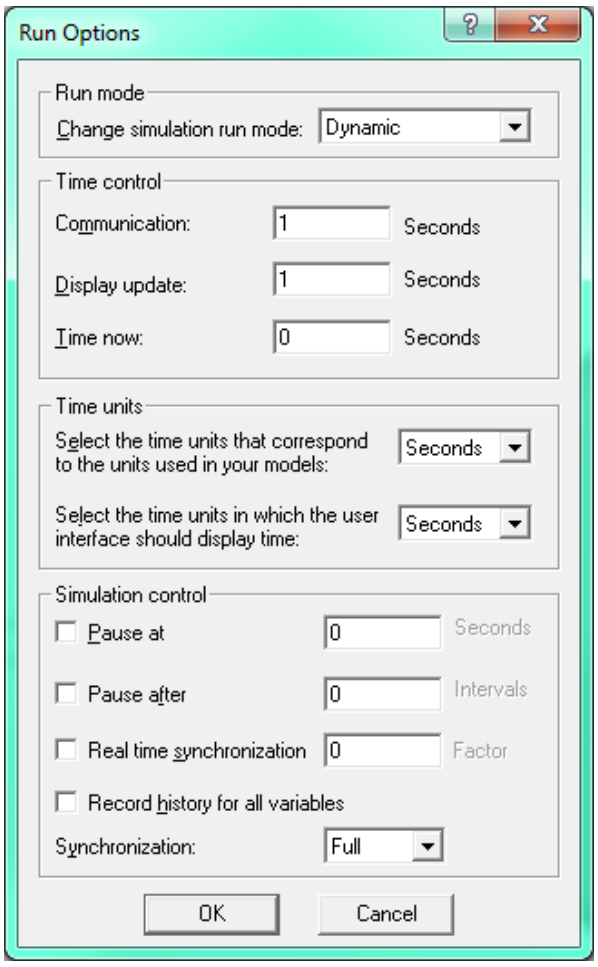

Figure 18: Run Options in APD.

By investigating PID in the PFD it is noted that APD has the capacity to tune the controllers based on two tuning rules: relay tuning; and approximate first order model based on input stepping (Ogunnaike & Ray, 1994). This enables conventional PID to be tuned quickly regardless of the complexity of the plant. 10.5 Appendix 5 details the steps involved with specifying controller parameters, tuning and plotting in greater depth. This does not however include any advanced control schemes. As this project will be implementing advanced controllers through the integrated software packages the control loops have been removed from the PFD in APD.

4.3 Aspen Advanced Process Control

AspenTech has a product which does not form part of the Aspen Engineering Suite (AspenTech, 2015e). *Advanced Process Control* (APC) Model Builder is part of Aspen Manufacturing Suite and includes a range of software packages to model, build, simulate and deploy model based controllers (AspenTech, 2011, p. 8). Although the full list of APC Model Builder's capabilities can be found in 10.7 Appendix 7, DMCplus is of particular interest.

After AspenTech acquired DMC Corporation they developed DMCplus, which has gone on to be the industry leading multivariable MPC (AspenTech, 2011, p. 1). This integrated suite offers the ability to:

- Use multiple model identification algorithms;
- Configure controllers through a validation wizard; and
- Evaluate the performance against a model to determine its suitability to noise and inaccuracies.

Furthermore, this package also includes an implementation package referred to as DMCplus Online (AspenTech, 2011, p. 2). This provides the infrastructure to implement the DMCplus controller and connect to the process instruments in the field.

4.4 Issues

During the setup of the high-fidelity distillation column in Aspen Plus a number of issues were encountered that warrant discussion. First, it is important to always ensure the software has access to the databases. If the databases are not available then ensure access to the licensing server is available. If no issue exists with the licensing server then ensure the software has completed its install correctly. The databases will be downloaded onto the device the first time

the software is launched. However, sometimes this does not occur and a few attempts need to be made. If the databases are not installed the software will fail to operate or give undesired and unexpected results. Second, the end user should be careful when establishing the dynamics of the unit operators. If they are incorrect the export will either fail or, if successful, the operation of the dynamic simulation will be inaccurate.

Additionally, make sure the software versions of the AspenTech products match. As AspenTech released updates on only half the products a version selector was made available. If the version is not selected correctly, the file being opened will either open blank or incorrectly. As shown in Figure 13, located on page 29, all the streams disconnected when opened in APD. This occurred because APD was expecting a model from V8.4 not V8.6. Note that even if the end user attempts to reconnect the streams the software will freeze and crash if the versions are not congruent.

Finally, Aspen APC does not form part of the AspenTech Engineering Suite. It is from the Manufacturing Suite and might require further download or occur additional cost to the end user. A check of the licensing server will display how many licenses remain for each AspenTech product and will advise if APC can be utilised on individual devices.

4.5 Conclusion

This chapter detailed the procedure involved in setting up and using a distillation column in Aspen Plus and APD. Furthermore, it outlined clearly the steps involved in exporting from steady state to dynamic simulations as well as the control options available in AspenTech's product range. Section 4.4 Issues emphasised the issues faced while completing this documentation in order to enable future work to run smoother. With a distillation column

established and operational in APD the integration of APD with MATLAB and LabVIEW will be investigated.

5.0 ActiveX

5.1 Overview

With the distillation column successfully implemented in APD the integration to additional software packages was investigated. Typically Windows based software packages include documentation on the OLE and ActiveX controls. However, where documentation is not available, it is possible to use Microsoft's OLE Viewer to examine the OLE typelib information and trace the methods and properties (Schwartz, Olson, & Christiansen, 1997). With the aid of AspenTech's documentation on ActiveX (AspenTech, 2000b) a flowchart was established outlining a general overview for how ActiveX will be exercised to apply control schemes in client programs and communicate data to and from AspenTech, the server. Figure 19 outlines this structure.

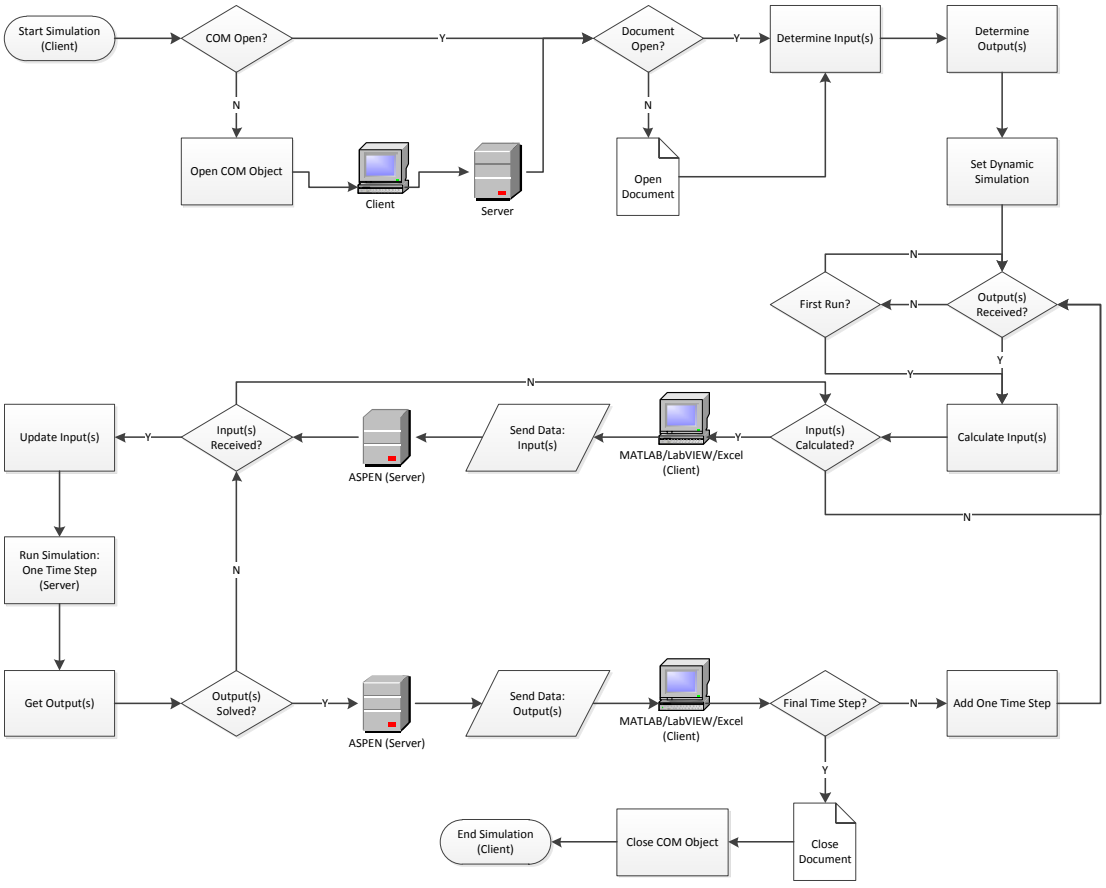

Figure 19: ActiveX COM Flowchart.

Initially the script will check to see if the COM is already open, if it is it will move on without attempting to open the COM again. If the COM is not open the COM Object will be opened. This check is in place as the software will not act as required if multiple COM servers are activated. Following this, the document being manipulated will be checked to see if open. Again, if it is open then it will not attempt to open it as this will result in a run error being established, however if it is not open then the process will launch the file. Moving on, the process will set up the server to match the specified client system settings then proceed to send and receive data. Once the program has finished the simulation the file and COM will be closed. Not closing the COM can cause the program to crash or stop the file from being manipulated. As with all software packages, it is important the results obtained through the use of ActiveX are validated. To do so a co-simulation will be undertaken to establish the validity of the integration before implementing advanced control schemes.

5.2 Active in MATLAB

Following the architecture presented in Figure 19, communication between MATLAB and APD was attempted. MATLAB has an inbuilt function 'actxserver' which takes advantage of OLE Automations capabilities to create a COM server based on the predefined DLL (MathWorks, 2015d). Using 'AD Application'(AspenTech, 2000b) an attempt was made to establish a connection from the client to the server. However, this server creation failed with the error given in Figure 20.

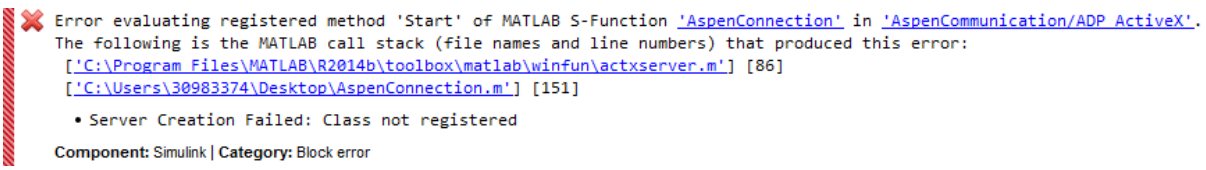

Figure 20: Failed Server Creation in MATLAB.

An issue occurred using the ‘actxserver’ function; as a result MATLAB was not able to establish the connection to APD. After consulting the MathWorks forum it was discovered that “MATLAB cannot be 64-bit for actxserver to load a 32-bit dll” (MathWorks, 2009). The current software versions installed on the device were 64-bit MATLAB and 32-bit APD. This raised two methods to continue:

- Install APD 64-bit; or
- Install MATLAB 32-bit.

The 64-bit version of APD was installed on the device and the COM server attempted again using the two 64-bit software packages. This was unsuccessful and resulted in the same server creation failed error. MATLAB 32-bit was placed onto the device and the script run again. This time, using the 32-bit software variations, the server was established without issue. It was discovered the ‘Aspen Customer Modeler 30.0’ DLL is 32-bit and this is the factor which governs which software version MATLAB must be. With the server established the next task is to communicate and manipulate data objects.

As stated in 2.4.2 Properties and Methods, in order to enable data communication and automation, the properties and methods must be raised through the COM. Using the ‘invoke’ function MATLAB is able to invoke different methods on COM objects (MathWorks, 2015e). There are several methods which can be employed which range from:

- Manipulating variables;
- Retrieving variables; and
- Automating the simulation.

Once the directory of the file being examined, as well as the number of inputs and outputs set, it was possible to view all variables available on the server (AspenTech, 2000b). The available variables include: flows; temperatures; pressures; levels; and system settings in APD. These unit operator variables are called using the following naming convention: `OBJECT("NAME").Description|Units`

10.8 Appendix 8 details how this communication from MATLAB to APD was implemented, providing an in-depth description of each step.

5.3 ActiveX in LabVIEW

In order to create a real-time simulation of APD, LabVIEW will be integrated with APD. This software package was selected as it has an existing utilisation across colleges and universities around the world, in addition to wide scale industrial use (National Instruments, 2015b). Due to this wide use the software integration could form a useful tool for educational purposes in higher education and industry. Again, this communication will be achieved through the use of ActiveX.

Connecting to MATLAB from within the LabVIEW interface is simple through the use of NI's MATLAB script node. This node connects directly to MATLAB using ActiveX technology and invokes the MATLAB server to execute scripts. The scripts written inside the node follow the MATLAB programming language. By default it is only possible to execute inbuilt MATLAB functions. However, by using MATLAB's inbuilt function 'pwd' to return the current folder it is possible to save user defined functions within this directory (MathWorks, 2015f). When using the MATLAB software all user defined

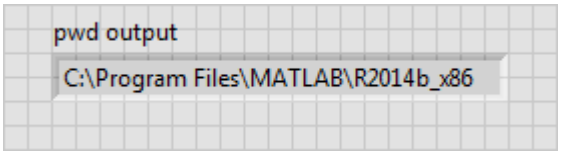

Figure 21: Current Directory Returned by MATLAB.

functions must be within the current directory folder to be executed. Figure 21 shows the current, default directory. As it is not possible to move user functions into the default file on administrator locked devices an additional method can be applied. Using MATLAB's function 'cd' it is possible to change the current directory to a different folder (MathWorks, 2015g). In doing so it is possible to ensure LabVIEW and MATLAB will always search the same directory for functions, thus removing any errors at start up. With the node set up with proper validation it is possible to call user defined functions. This will be examined in 6.2 LabVIEW Control Design Toolkit when implementing DMC in LabVIEW. The limitation of this method is the user is restricted to the standard functionality of MATLAB. This does not provide any additional benefit except the ease in implementing existing scripts into the LabVIEW visual interface. In order to gain access to true automation the ActiveX blocks were investigated.

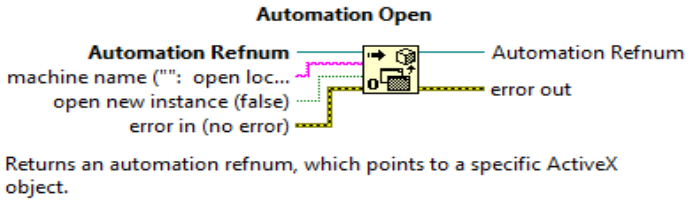

Figure 22: ActiveX Automation Open Block in LabVIEW.

Using the Automation Open block in LabVIEW, as shown in Figure 22, it was possible to specify which ActiveX object to open communication with (National Instruments, 2007a). The COM interface, referred to as Automation Refnum in LabVIEW, is daisy chained to the COM properties and methods from the Automation Open block. As previously mentioned in 2.4 ActiveX, Aspen Customer Modeler DLL is used to establish connection to APD. This is selected as the ActiveX class and connected to the Automation Open block as shown in Figure 23.

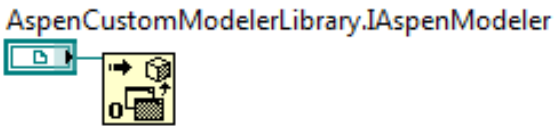

Figure 23: Aspen Customer Modeller ActiveX COM Setup.

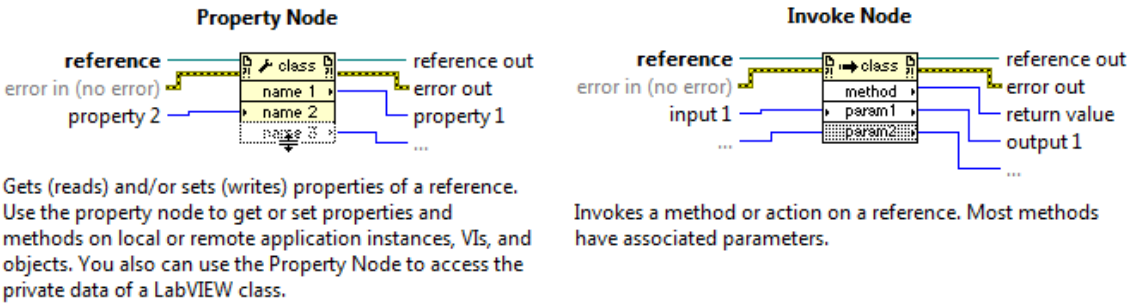

Figure 24: ActiveX Property and Invoke Nodes in LabVIEW.

Once the COM server was created the interface reference was fed to the property and invoke nodes, which are described in Figure 24. These nodes were used to make APD visible, open the document and retrieve the temperature of the feed stream. Figure 25 below displays the arrangement of the ActiveX blocks in LabVIEW. This initial configuration was used to check if the COM was established correctly. The error indicator displays any error code and description if ActiveX fails to complete a task.

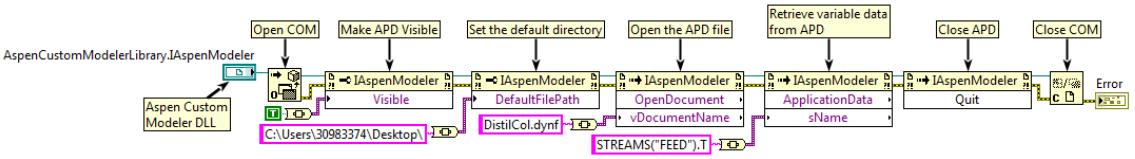

Figure 25: Graphical ActiveX Programming in LabVIEW to Create APD COM.

It can be observed in Figure 25 that all the variables must be converted into the variant data type before being usable in the ActiveX blocks. The variant data type was added to LabVIEW for the sole purpose to handling the complicated data required by ActiveX objects (Johnson & Jennings, 2006, p. 126). When this VI was run, APD launched without issue and the document, 'DistilCol.dynf', opened

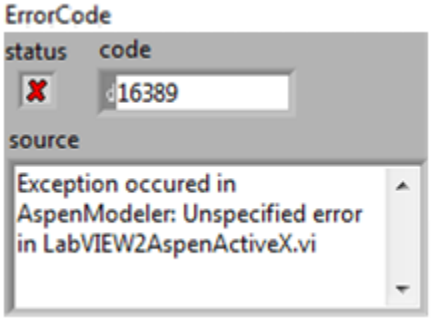

Figure 26: Error When Retrieving Data over COM in LabVIEW.

successfully. However, when retrieving the data from the server an unspecified error occurred, displayed in Figure 26. A check through the ActiveX documentation did not resolve this issue (AspenTech, 2000b; AspenTech, 2005; National Instruments, 2007). It is believed the issue originates from a mismatch in the naming syntax for the variables in APD, or from the server not expecting data to be requested. Further research into this issue is required. As a result it was not possible to continue using this integration method and an alternative was explored.

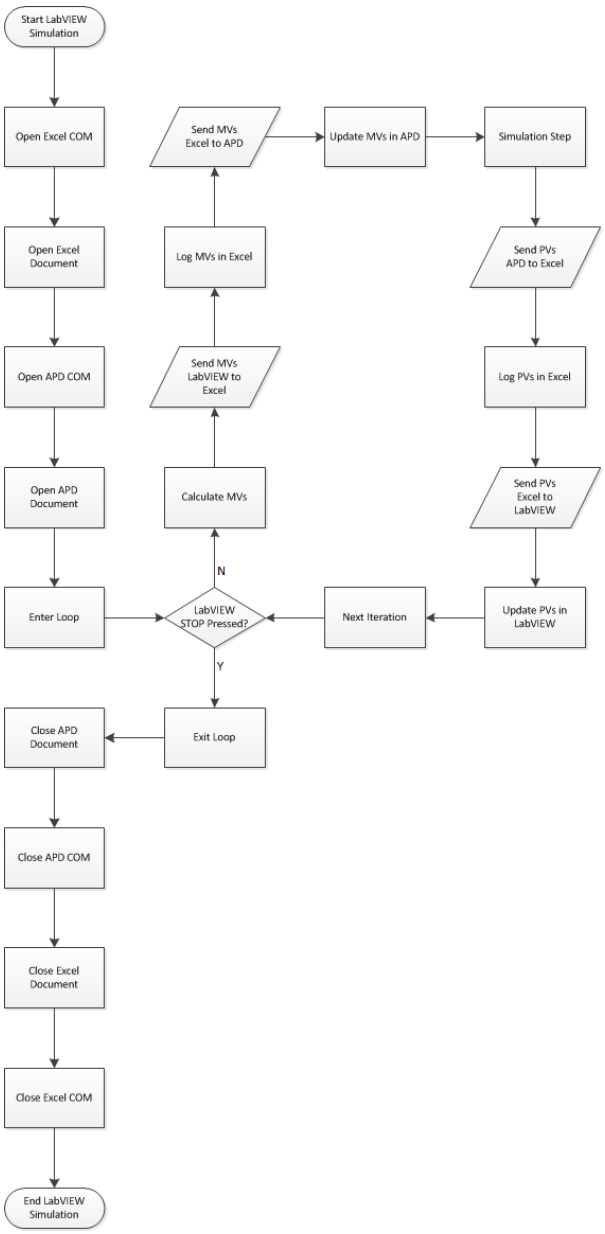

Figure 27: Structure for LabVIEW-Excel-APD Communication.

In order to achieve the integration of LabVIEW and APD, Excel could be used as an intermediary. First LabVIEW would need to establish a connection to Excel, before Excel then creates a connection to APD. Following the structure presented in Figure 27 the three software packages could work together to communicate and log data. To achieve automation through Excel, macros must be written in VBA. Before these macros are designed in Excel, to create a COM server to APD, the LabVIEW ActiveX blocks will be used to establish a server with Excel. 10.9 Appendix 9 details the steps taken to enable communication between LabVIEW and Excel. This process is relatively simple, using VBA macros to perform the

majority of the communication process depicted in Figure 27. As such, only the execution of the macros is needed to provide automation from LabVIEWs perspective.

5.4 ActiveX in Excel

In order to retrieve information from LabVIEW the variables must first be specified. When connecting to LabVIEW from Excel using ActiveX, the functions GetControlValue and SetControlValue are used. These functions will call variables based on their label in LabVIEW. However these variables must be connected to a port on the VBA node within LabVIEW for the communication to be successful. An empty VBA node is shown in Figure 28. Once the MVs, DVs and PVs have been assigned to the VBA node, the VBA macro can be written to manipulate these. 10.10 Appendix 10 outlines the creation of the macros, while 10.11 – 10.13 Appendices 11 – 13 include the VBA scripts. Once the LabVIEW COM server is created in Excel it remains open until explicitly closed. Given this the COM server is created initially, then the variables taken from LabVIEW and logged in Excel. After the simulation in APD is stepped, the PVs are retrieved and logged in Excel before being sent back to LabVIEW to override the PVs using SetControlValue.

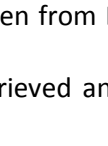

Figure 28: VBA Node in LabVIEW.

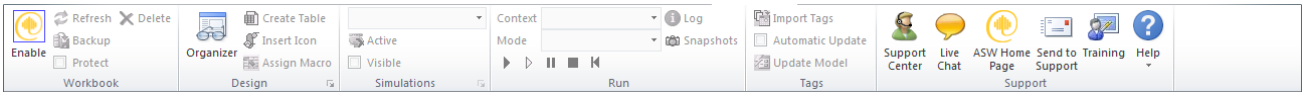

Figure 29: ASW Toolbar in Excel.

To communicate to APD from Excel, AspenTech recommends utilising their Excel add-on *Aspen Simulation Workbook* (ASW). ASW, once linked to an Aspen simulation, can manipulate and retrieve variables from the model variables (Tremblay & Mantrala, 2014, p. 2). ASW also provides the user with the Run controls from within APD, as shown in Figure 29. This method

of integration is best used when providing a user interface for complex models in Excel. It enables the capabilities of AspenTech's products to run in the background while the user does not explicitly need intimate knowledge of the software (Tremblay & Mantrala, 2014, p. 1). ASW does not however provide the automation required to achieve integration with LabVIEW. The ability to automatically step the simulation can be achieved through ActiveX and VBA. In order to achieve unanimity between the three programs VBA also provides a logical solution as the communication between LabVIEW and Excel is performed using VBA.

This configuration follows the methods described in Figure 27. Once the MVs are calculated in LabVIEW the macro 'StepAspen' is run. Once this macro is run LabVIEW will wait for a response, effectively pausing all processes within. When Excel receives the MVs from LabVIEW they are advanced to APD by manipulating the variables in the tree structure in APDs model explorer (AspenTech, 2005). With these variables updated to the current DVs and MVs, the simulation is stepped one time interval. The PVs are now extracted from the model explorer and sent back to Excel before being passed back to LabVIEW. Once they have been updated in LabVIEW the macro ends and control is passed back to LabVIEW. With control returned, LabVIEW will move to the next loop iteration and start the entire process again. This will continue until the end user presses the STOP button inside LabVIEW. Once pressed, the shutdown process is initiated and the 'CloseAspen' macro executed. This process will shut the servers between Excel and APD as well as LabVIEW and Excel. An in-depth description of this process is outlined in 10.10 Appendix 10.

5.5 Co-Simulation

With the client server models established across the software packages, the communication must be validated to ensure correct operation. To do so, PI feedback control loops were established across all three software packages. The control loops used were:

- Level in the reflux drum controlled by the distillate stream flow rate;
- Level in the sump controlled by the bottoms stream flow rate; and
- Pressure in stage one controlled by the reflux stream flow rate.

Through this co-simulation it was possible to test the implementation across the software packages and in turn verify the performance of the communication servers and client control schemes. The PI controllers are configured with the following parameters:

- Reflux Drum Level, $Kc = -36, \tau_i = 1$;
- Sump Level, $Kc = -36, \tau_i = 1$; and
- Condenser Pressure, $Kc = -72, \tau_i = 0.2$.

These parameters are not tuned for the system and are used for the sole purpose of validating the operation of the software packages.

Figure 30 displays the $20°C$ step in the feed stream temperature. The overall performance of the control schemes is not of interest in this case, however the behaviour of the system is. It can be seen in Figure 31 that the controllers implemented in APD, LabVIEW and Simulink have all produced identical MVs. This endorses the operation of the integrated software packages. Moreover, Figure 32 confirms the PVs across all software packages are indistinguishable. This endorsement proposes MPC can be investigated and implemented in LabVIEW and MATLAB, and the performance would be equal to a scheme which is implemented directly in APD. The remaining plots can be found in 10.14 Appendix 14.

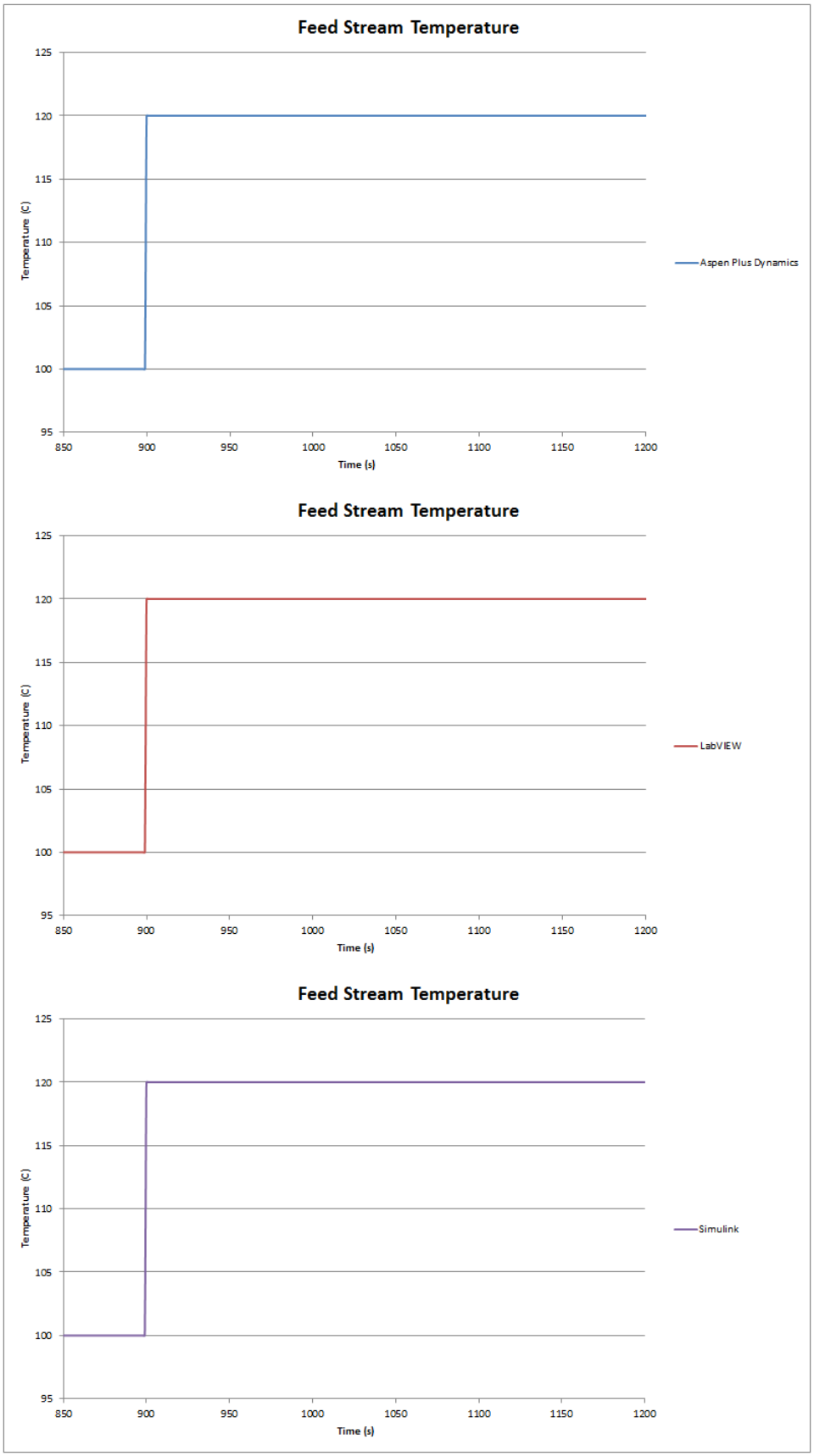

Figure 30: Plot of the DV Step Across all Software Packages.

Figure 31: Plot of the MV Across all Software Packages.

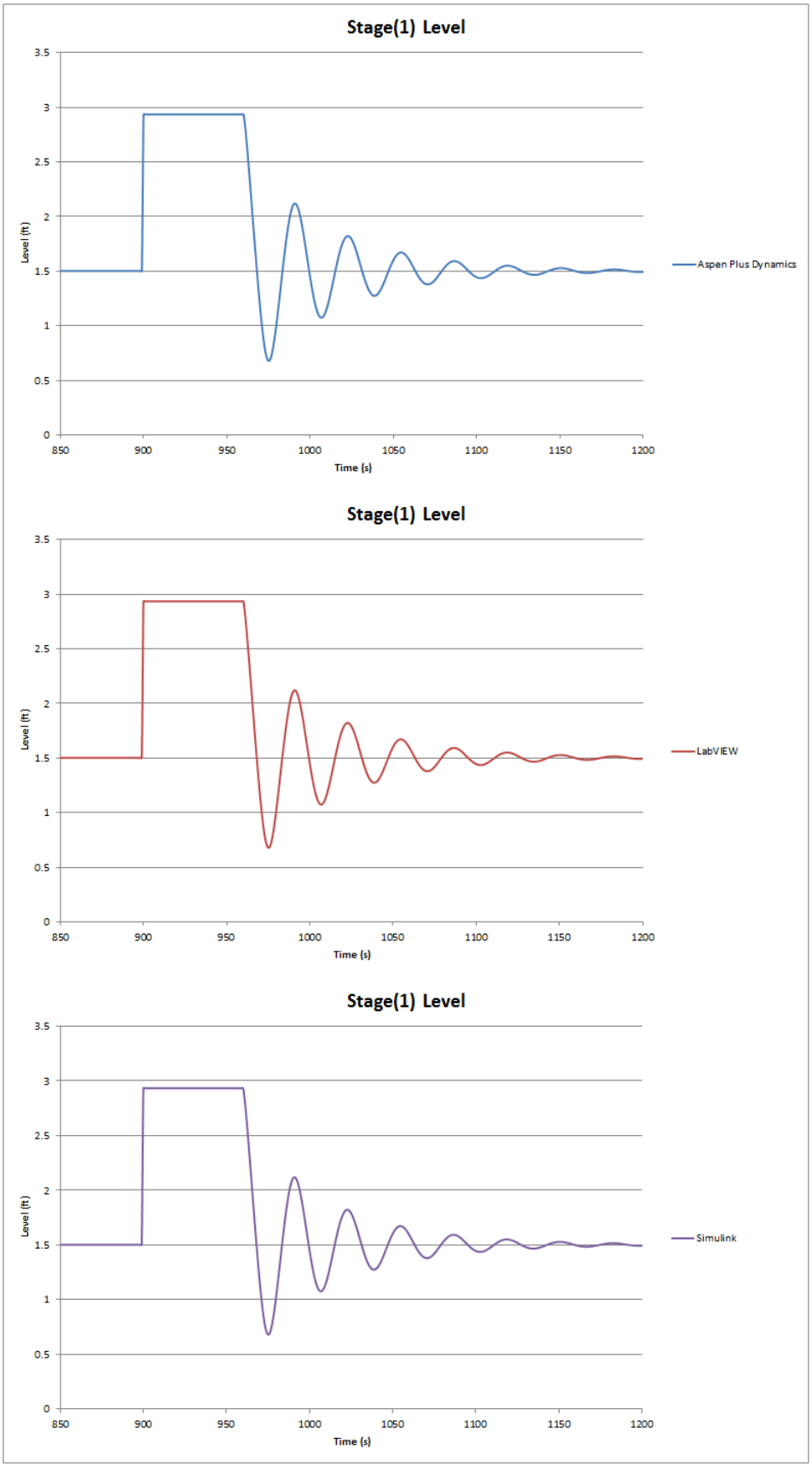

Figure 32: Plots of the PV Across all Software Packages.

5.6 Issues

During the development of the ActiveX communication servers a range of problems were faced that warrant discussion. In order to use the Aspen Custom Model DLL, 32-bit MATLAB is required. This is due to the fact the DLL is 32-bit not 64-bit. There is no fix for this except for the change in software version. It was not possible to get the Aspen Custom Modeler DLL to work in LabVIEW. When invoking methods across the COM an unspecified error would occur. Further research is needed in order to document the correct syntax and provide the ability to remove the use of a data highway, in the form of Excel. It is also important to note that the documents from MATLAB; Simulink; APD; and LabVIEW; must be located in the same directory. If they are not they will fail to launch when the current directory is searched. It is possible to change directories as the process is operating however the constant change in directory will introduce an unwanted time delay.

When troubleshooting the software packages it is easier to follow the flow of information and processes when the software is visible. This applies to Excel and APD. If the software is not visible while initial setup is conducted it can be difficult to isolate issues. On top of this, note that the client will not continue to operate while the COM server is executing a command. Consequently, the software can only complete a loop as quickly as the COM server can communicate data. If the loop time drops below $730\ ms$ a time delay will be introduced into the system every loop iteration and eventually void the real time simulation in LabVIEW.

When communicating to any ApsenTech product, if the *Units of Measurement* (UOM) are not specified it will use the English UOM by default. These are as follows:

- Foot (ft);
- Pound (lb);

- Pound per square inch (psi);
- Degrees Fahrenheit (°F); and
- British thermal unit (Btu).

5.7 Conclusion

This chapter explored the idea of Microsoft's ActiveX and outlined a structure for achieving communication between differing software packages. 5.6 Issues highlighted the main issues associated with the implementation of ActiveX in MATLAB, LabVIEW and Excel. With communication established the advanced control toolboxes will be explored in MATLAB and LabVIEW.

6.0 Model Predictive Controllers

6.1 MATLAB MPC Toolbox

With a valid connection between MATLAB and APD, the MPC Toolbox in MATLAB can be considered. This toolbox provides functions and Simulink blocks for designing, analysing and simulation MPCs (MathWorks, 2015h). These controllers follow the control algorithms and principles discussed in 2.2.4 Model Predictive Control. The MPC controller block in Simulink is displayed in Figure 33. By default this block expects the (MathWorks, 2015i):

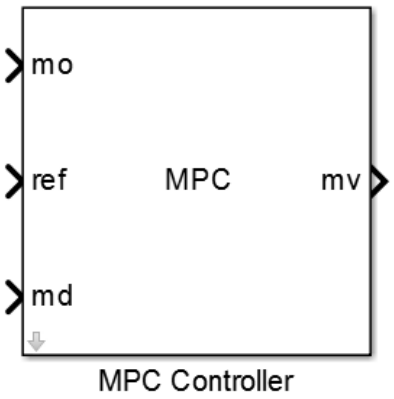

Figure 33: MPC Block in Simulink.

- Measured output, denoted as mo and previously referred to as the PV;
- SP, denoted as ref; and
- Measured disturbance or DV, denoted as md.

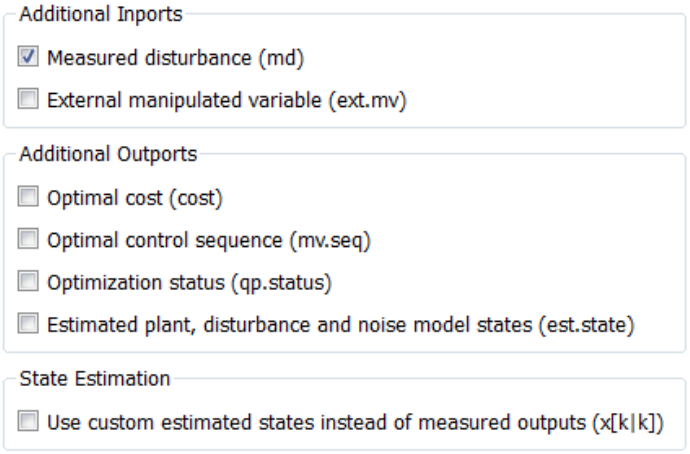

Figure 34: Additional MPC Input and Output Variables in Simulink.

On top of the default inputs it is possible for the MPC controller to expect various optimisation objective functions and plant noise, as presented in Figure 34. This affords Simulink the ability to recreate realistic plant behaviour with measured and unmeasured DV present.

In addition to these features, Figure 35 indicates the controller constraints and penalty weightings can also be defined. By enabling these it is possible to make a more adaptive control schemes. However these variables can be set within the MPC toolbox itself if adaptive control is not required.

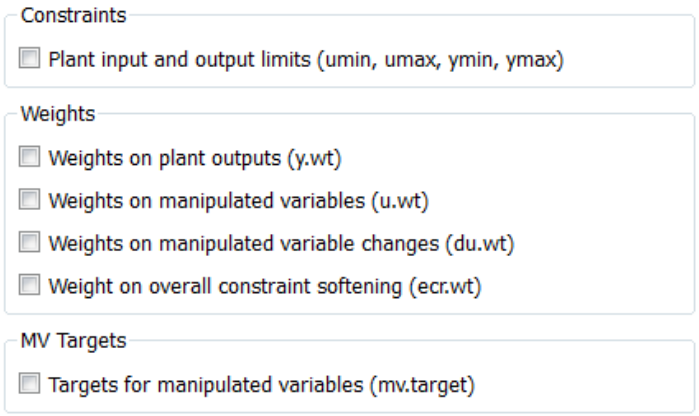

Figure 35: MPC Controller Parameters in Simulink.

Once the controller block was added to the Simulink model and connected to the plant model, the controller design was launched. When the design tool launches MATLAB will automatically linearise the existing plant in Simulink to use as a reference for the predictive nature of the controller. This however is not possible over the ActiveX server. It is noted that MATLAB performs the linearisation based on the Simulink blocks used on the model not through step tests. Thus, when the toolbox attempts to linearise the plant it produces an error, as found in Figure 36. This limitation removes the toolboxes ability to quickly generate MPC control schemes. In order to

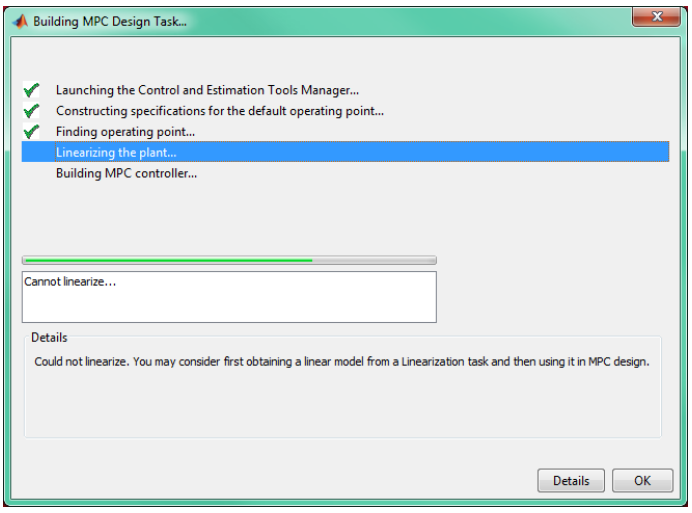

Figure 36: Design Tool Linearising Plant Model in Simulink.

overcome this, the model of the plant must be determined. MATLAB suggests in Figure 36 that a linear model should be obtained first.

Given the capabilities of MATLAB this can be achieved using the system identification toolbox, ident (MathWorks, 2015j). However an issue associated with this method is the time investment required to determine accurate models for each output against all the inputs. To overcome this, a new function and script was written to automate this process and remove the need for tedious calculations by the end user. 10.15 Appendix 15 documents the function which exercises the least squares method to determine a first order or capacitive *Transfer Function* (TF) model (Strang & Borre, 1997, pp. 174-176). This function enables quick prediction of the TF by minimising the sum of the error squared based on the input of the PV, MV and time arrays.

Given this function, the automation of Simulink was written through MATLAB script. This is detailed in 10.17 Appendix 17 and performs steps on each input in Simulink to determine the TF for each element in the MIMO matrix. Once the relationships

```
MIMOPlantModel =

  From input 1 to output...
           5
   1:  ----------
       1.99 s + 1

       0.3
   2:  ---
        s

           2
   3:  ----------
       0.98 s + 1

         0.09999
   4:  -----------
       99.99 s + 1

  From input 2 to output...
          -2
   1:  ----------
       0.98 s + 1

           15
   2:  ----------
       0.45 s + 1

       1
   3:  -
       s

          0.77
   4:  ----------
       0.98 s + 1

  From input 3 to output...
           2
   1:  -----------
       50.01 s + 1

       -0.75
   2:  -----
         s

           8
   3:  ----------
       1.49 s + 1

       0.3
   4:  ---
        s

  From input 4 to output...
       5
   1:  -
       s

           1
   2:  ----------
       7.01 s + 1

          0.173
   3:  -----------
       32.01 s + 1

          6.42
   4:  ----------
       2.99 s + 1

Continuous-time transfer function.
```

Figure 37: MIMO TF Output from the MIMO Script in MATLAB.

the input has on each output are discovered the next input is stepped and process repeated. The script will continue this process until all input variables have been stepped and analysed. Figure 37 displays a MIMO TF matrix which can be utilised in the MPC toolbox. With the MIMO TF determine this can be input into the MPC Toolbox by selecting 'Import Plant' in the MPC Toolbox shown in Figure 38. The MPC Toolbox can be launched from the MATLAB Command Window using the function 'mpctool'.

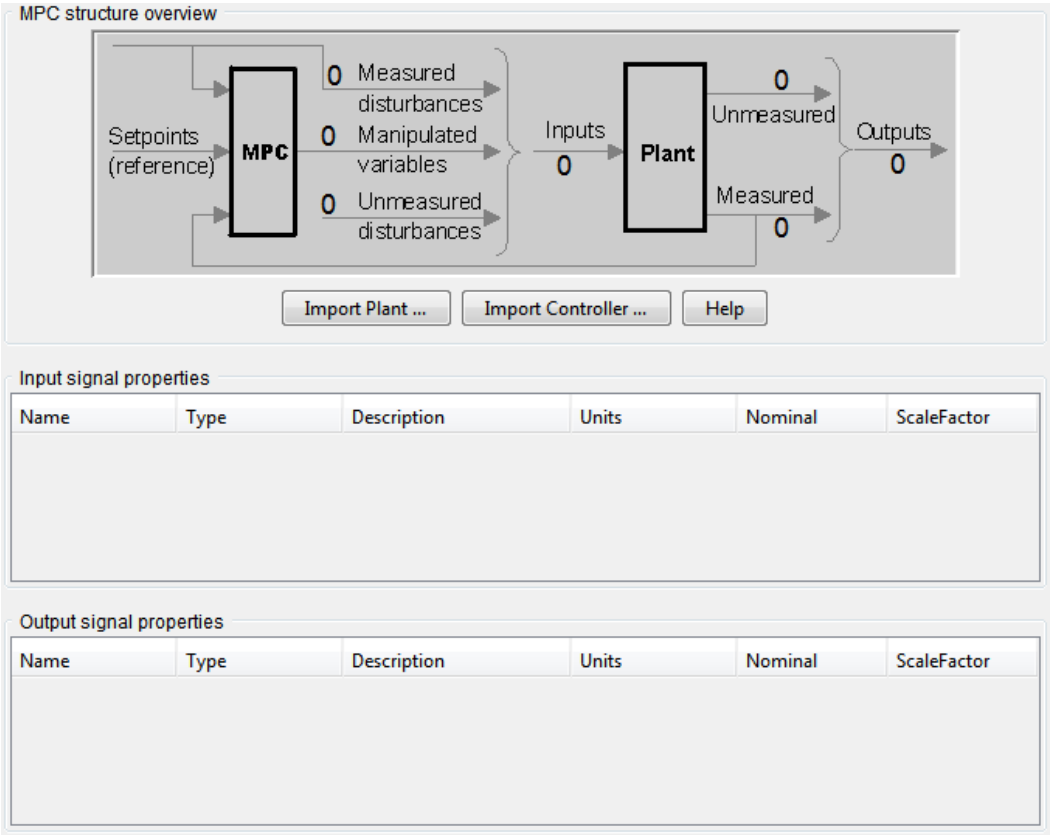

Figure 38: MPC Toolbox Design Task in Simulink.

Once the plant model has been imported, the input and output variables will be determined by the toolbox. This model will then be used as a reference for the controller design. Following this the controller parameters, constraints and controller weighting must be specified. Figure

39 shows the controller tuning window. 10.18 Appendix 18 provides further detail on the setup of MPC controllers in MATLAB and Simulink.

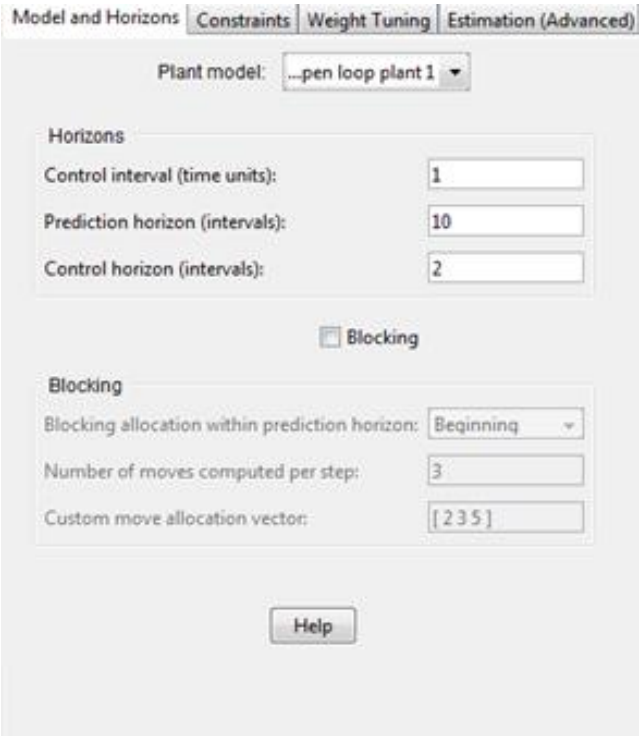

Figure 39: MPC Controller Parameters in the MPC Toolbox in Simulink.

On a side note, the MIMO TF can be used for RGA analysis by utilising the algorithm given in Equation 10, found on page 14. This will provide the best loop pairings given the variables available in the TF. This is important because the MPC Toolbox will match the first MV with the first PV and so forth. It does not determine the best pairing even though it has the information and model to do so.

Furthermore, the TF parameters can be utilised when designing DMC controllers (Ogunnaike & Ray, 1994, pp. 1000-1007). To implement DMC in Simulink, a MATLAB function was written. The DMC function, as shown in the Simulink block in Figure 40, follows the design steps in Romagnoli and Palazoglu (2005). The controller inputs are:

- Control variables:
  - PV;
  - SP; and

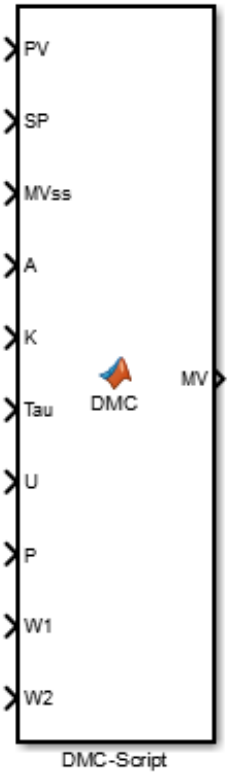

Figure 40: Function Block in Simulink.

- Bias or steady state MV.

- Approximate first order model:
  - Time delay, denoted by A to represent alpha (α);
  - Gain, denoted by K; and
  - Time constant, denoted by Tau.

- Controller parameters:
  - Control horizon, denoted as U;
  - Prediction horizon, denoted as P; and
  - Weighting matrices, denoted as W_1 and W_2.

The output of the function is the MV. This controller action is calculated based on the convolution model discussed in 2.2.4.1 Dynamic Matrix Control. It is possible to then run the MV through a saturation block in Simulink to apply any MV constraints before being sent to the plant. 10.19 Appendix 19 describes the DMC function in greater detail. This script was designed to offer an alternative to the MPC toolbox.

6.2 LabVIEW Control Design Toolkit

The MATLAB node, discussed previously in 5.3 ActiveX in LabVIEW, can be used to implement the DMC script in LabVIEW. The functions inputs are fed to the node and the function called. This then utilises its ActiveX connection to run

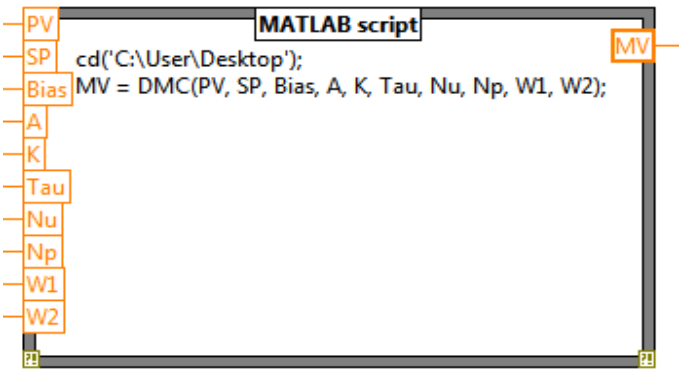

Figure 41: DMC Script in the MATLAB Node in LabVIEW.

the function and determine the DMCs MV. Its implementation can be seen in Figure 41.

Implementing MPC in LabVIEW is done so by using three blocks. Initially the controller must be created by preparing the 'Create MPC' block, shown in Figure 42. However before this block can be used, the plant model must be defined in SS representation (National Instruments, 2009, p. 18.3). The limitation on model input fits the common theme across the majority of

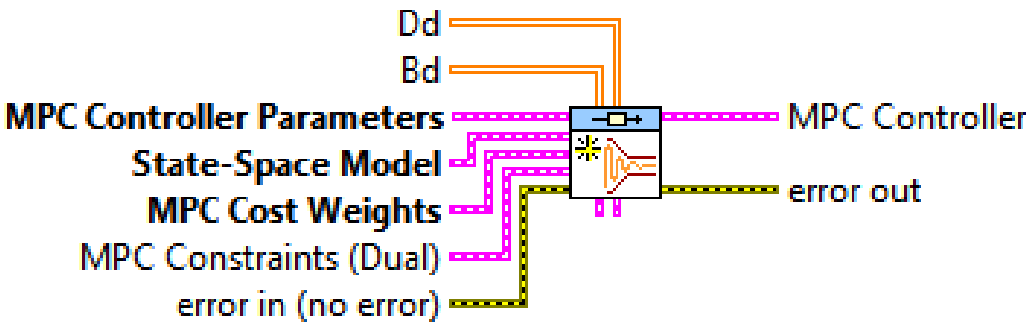

Figure 42: Create MPC VI in LabVIEW.

MPC controllers in industry. Typically only DMC uses non SS representation. It is possible however, to convert a TF model into SS before being input into the MPC block (National Instruments, 2007b). The 'Convert to SS' block will take the TF input, perform any zero-pole cancellations and output a SS representation of the model. This SS model is then fed to the 'Create MPC' block to use as reference for the controller predictions. Additionally the 'Create MPC' block expects the follow inputs:

- MPC controller horizons; and
- Weighting coefficients.

The bold inputs in Figures 42 and 43 denote required node inputs. The MPC Controller Parameters are broken down into the following:

- Prediction horizon;
- Control horizon;

- Minimum delay of the model; and
- Whether to include integral action.

The integral action is used when the plant's mathematical model does not match the plant's actual model. Furthermore, the MPC Cost Weights is broken down into the:

- Output error weighting;
- Change in controller action weighting; and
- Controller action weighting.

With the required inputs specified, the MPC Controller was fed to the 'Implement MPC' node and used as a reference to calculate the MV. However, it is also possible to define the initial conditions and constraints on the MVs and PVs.

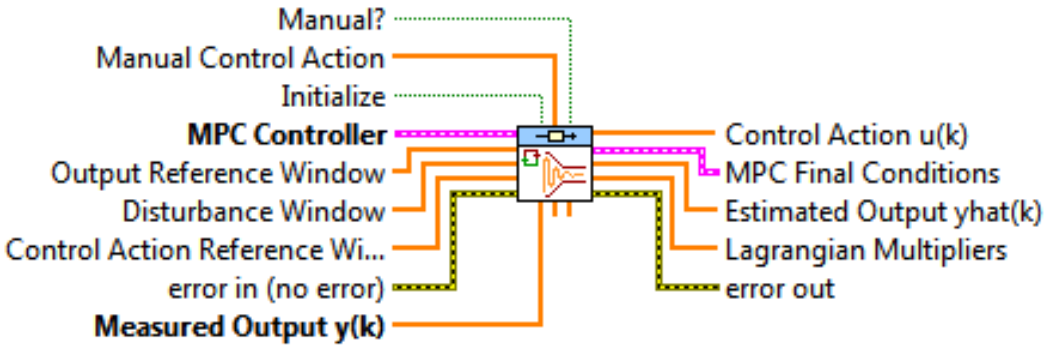

Figure 43: Implement MPC VI in LabVIEW.

With the MPC controller constructed, it is fed to the 'Implement MPC' block shown in Figure 43. This block is used to calculate the MV of the MPC generally inside a loop. It applies the MPC controller designed in 'Create MPC' to determine the next MV based on controller parameters, constraints, and weighting. The 'Implement MPC' block needs the MPC controller and PV input in order to calculate the current MV. It is possible to change the controller to manual mode, in addition to feed the pervious MV back to the controller. This is useful as it ensures the controller is aware of any discrepancies between its suggested action and the

actual MV, potentially due to actuator limitations. The estimated output can also be withdrawn from the block for reference purposes to track how well the controller is predicting the behaviour of the plant. The implementation of MPC in LabVIEW, including MIMO, can be found in 10.19 Appendix 19.

6.3 Issues

When documenting the functionality of the MPC toolboxes in MATLAB and LabVIEW, a number of concerns were raised. When using MPC, or DMC, it is important that the Δt is consistent amongst the software packages and their controllers. If they are not congruent the controller will make erratic controller predictions and result in unexpected plant behaviour. This stems from the controller predicting over a different sample time.

The MATLAB MPC Toolbox explicitly uses the blocks in the Simulink model to linearise the plant. When these blocks call a MATLAB script, or external information through a server, the linearisation method will fail. In order to move beyond this drawback a model must be developed and imported into the controller. This can be either a TF or SS in MATLAB, or SS in LabVIEW. Nevertheless, the system identification toolbox in MATLAB only handles SISO identification. In order to build a MIMO TF matrix the toolbox must be utilised repeatedly. This process is tedious and thus a MIMO system identification script was written. This script can be used for any system which is developed following the guidelines specified in 10.17 Appendix 17. In LabVIEW however, there is no tool which offers system identification. This requires further research into how such a tool could be implemented, or how Excel and the solver add-on could be incorporated with LabVIEW to achieve this automatously.

Finally, neither of the MPC toolboxes have the capacity to perform RGA analysis. This means the RGA must be determined beforehand by the user. An advantage of the MIMO system identification script in MATLAB is that it can be used to determine the gain array and RGA. This highlights the need for a more innovative control toolbox is both software packages.

6.4 Conclusion

This chapter detailed the functionality of the advanced control toolboxes in MATLAB and LabVIEW, in addition to offering some alternatives and improvements. The major flaws in functionality and operation were detailed in 6.3 Issues. With the capabilities expanded and toolboxes documented a comparison of MPC against PI can be undertaken. This will be used to confirm the original idea and driving force behind the integrated software that implementation of MPC will reduce the amount of energy usage in distillation columns.

7.0 Control Scheme Comparison

With the MPC application in MATLAB and LabVIEW documented it is possible to implement these controllers on the high-fidelity distillation column. Before that can be done an open loop simulation must be run to determine the effect each input has on the outputs. Using the MIMO TF matrix script in 10.16 Appendix 16 it was possible to construct the MIMO model automatically. This script was run on the open loop system shown in Figure 44.

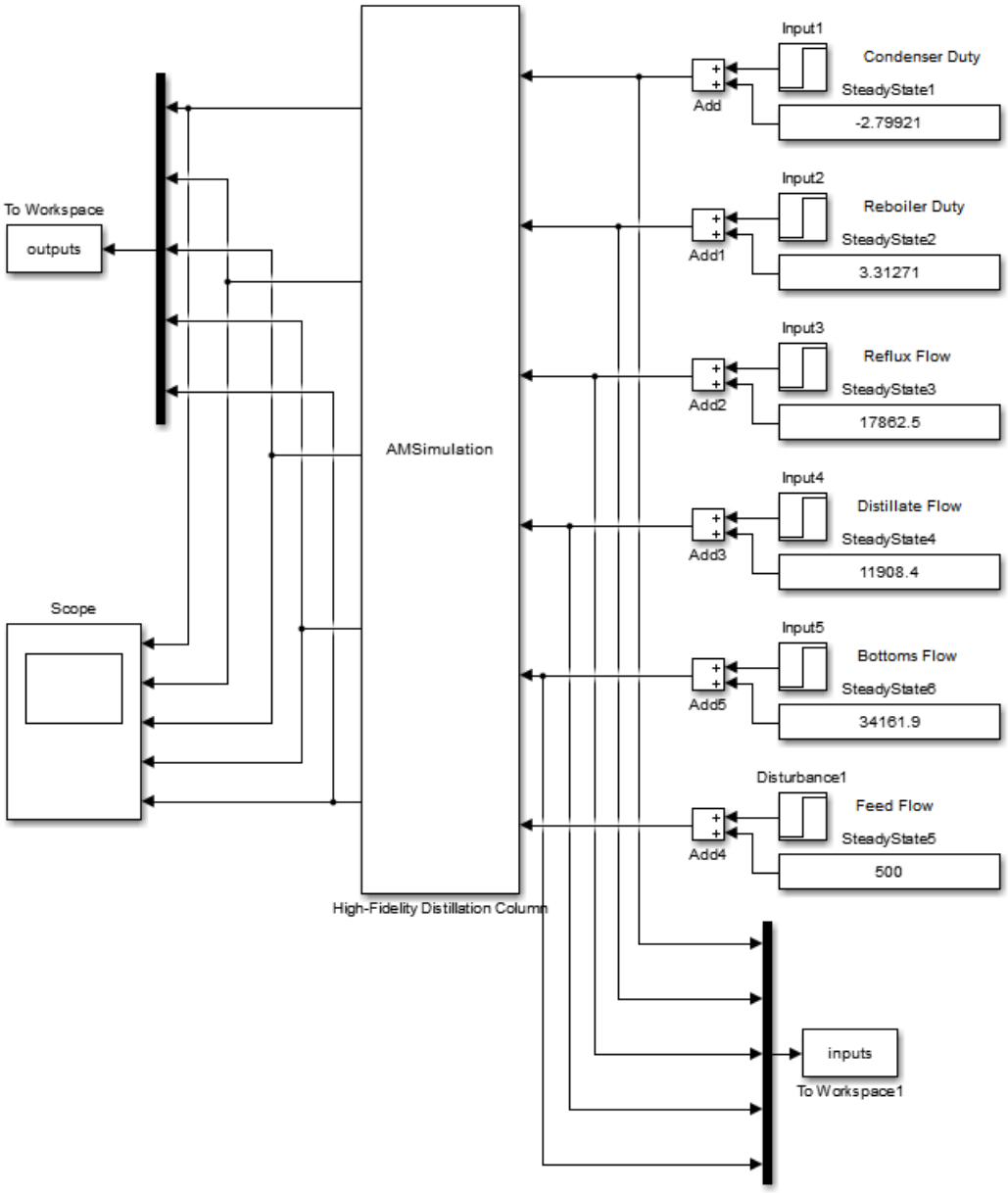

Figure 44: Open Loop Simulink Model for a High-Fidelity Distillation Column.

This script utilises the function provided in 10.15 Appendix 15 to solve for the TF of a given input and output. It repeats this process until all inputs have a corresponding TF for each output. The resulting MIMO TF matrix can be found in 10.20 Appendix 20 while the gain matrix is given in Table 2 below.

Table 2: Gain Array Matrix for a High-Fidelity Distillation Column.

K	Input 1	Input 2	Input 3	Input 4	Input 5
Output 1	1.896E+00	1.761E+00	-1.358E-05	-6.618E-07	-9.206E-07
Output 2	-3.171E+00	1.656E+00	-4.891E-04	-4.508E-04	7.359E-06
Output 3	3.316E+00	-1.633E+00	5.256E-04	-5.806E-05	-2.249E-03
Output 4	-1.981E-01	1.158E-01	-5.082E-05	8.687E-08	-5.803E-08
Output 5	1.034E-01	-4.785E-02	9.218E-06	-7.482E-08	1.524E-09

Using Equation 10 it was possible to determine the RGA. This is given in Table 3, where the best loop pairings are highlighted in blue. These pairing were selected given the rules provided in 2.2.5 Relative Gain Array.

Table 3: RGA for a High-Fidelity Distillation Column.

Λ	Input 1	Input 2	Input 3	Input 4	Input 5
Output 1	0.30	0.70	0.00	0.00	0.00
Output 2	0.00	0.00	0.00	1.00	0.00
Output 3	0.00	0.00	0.00	0.00	1.00
Output 4	-0.36	-0.26	1.61	0.00	0.00
Output 5	1.06	0.55	-0.61	0.00	0.00

The input and output variables are described in 10.2 Appendix 2, with the ideal loop pairings:

- Condenser pressure controlled by the reboiler duty;
- Condenser level controlled by the distillate flow;
- Sump level controlled by the bottoms flow;
- Toluene in the distillate controlled by the reflux flow; and
- Benzene in the Bottoms controlled by the condenser duty.

Using these pairings MPC controllers are implemented in MATLAB and LabVIEW. The MIMO TF in 10.20 Appendix 20 is applied as the model reference in both the software packages, with the TF model converted to SS in LabVIEW. Figure 45 shows the closed loop set up in MATLAB with the MPC controller established.

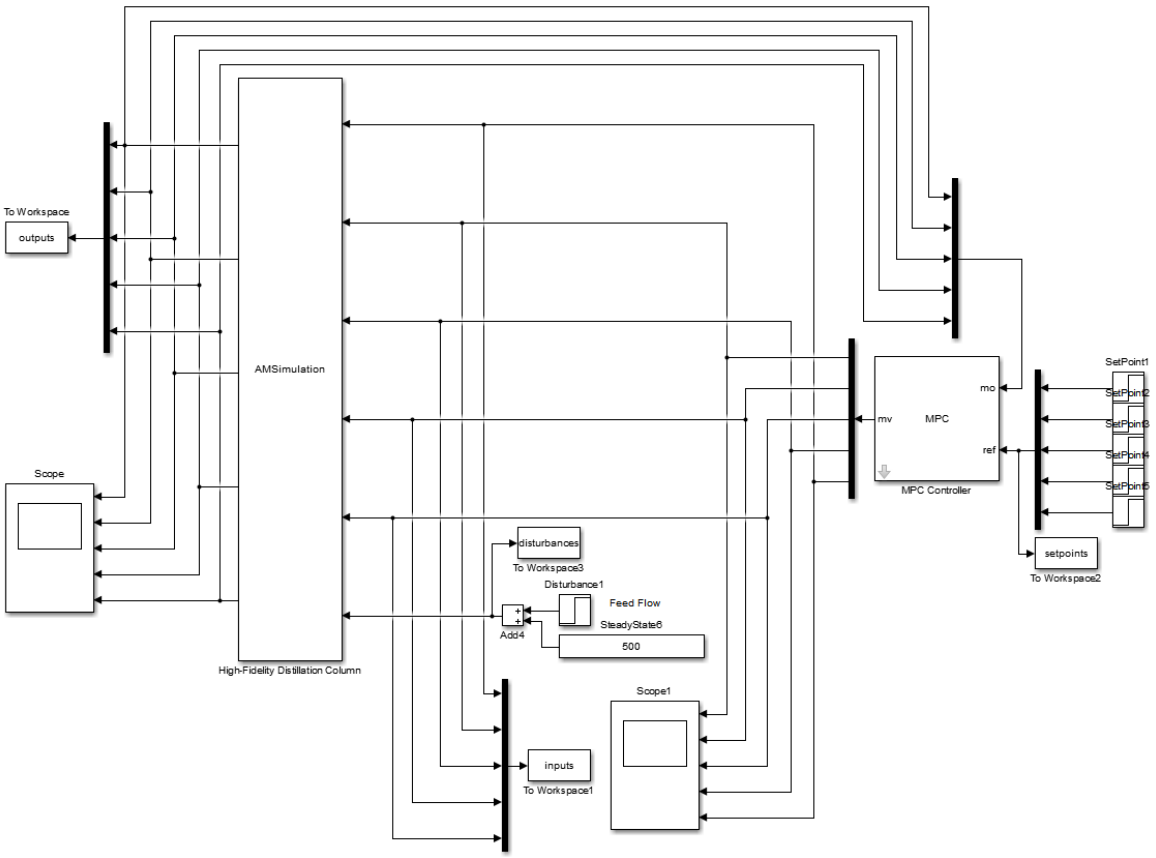

Figure 45: Closed Loop Simulink Model for a High-Fidelity Distillation Column with MPC.

The MPC controller was designed with the following parameters:

- $W_1 = 1$;
- $W_2 = 0.2$;
- $N_u = 5$; and
- $N_p = 20$.

When this control scheme was run the system did not operate as expected. The MIMO TF had to be manipulated because the MPC controllers in MATLAB and LabVIEW do not allow for the inputs to control different outputs. It simply connects input one with output one and so forth. Due to this the inputs were altered positions to allow them to match the number of the output they are controlling. Table 4 displays the updated RGA, using the new arrangement, with the input n controlling output n.

Table 4: Updated RGA for a High-Fidelity Distillation Column.

Λ	Input 1	Input 2	Input 3	Input 4	Input 5
Output 1	0.70	0.00	0.00	0.00	0.30
Output 2	0.00	1.00	0.00	0.00	0.00
Output 3	0.00	0.00	1.00	0.00	0.00
Output 4	-0.26	0.00	0.00	1.61	-0.36
Output 5	0.55	0.00	0.00	-0.61	1.06

To test the performance of this control scheme, a comparison against tuned PI control was completed. These tests were performed to confirm the suggestion MPC would result in better energy usage in the distillation column. When implementing PI control around five variables, to match the MPC control scheme, APD would fail. APD would lose convergence once the simulation completed two iterations. To overcome this, two controllers were removed from the scheme and operation was resumed. The control loops being examined are:

- Condenser duty controlling the condenser pressure;
- Distillate flow rate controlling the reflux drum level; and
- Bottoms flow rate controlling the sump level.

The PI controller parameters were determined from the Zeigler-Nichols approximate model tuning rules. These are:

- $Kc = 1.34, \tau_i = 4.00$;
- $Kc = 4.56, \tau_i = 9.11$; and

- $Kc = 5.67, \tau_i = 4.11.$

In order to test the performance of the control schemes, $\pm 20\%$ SP changes were introduced individually in Simulink. The performance of each system was then compared against one another using the ITAE performance criterion, provided in 2.2.6 Performance Criterion. Table 5 displays the ITAE values for the PI and MPC control schemes, with the variable tracking their SP highlighted in blue. This performance criterion was used for both the SP tracking and DV rejection.

A supplementary feature which was not active in the toolboxes was the addition of integral action within the MPC controller. This action can be employed to remove the mismatch caused from inaccurate plant models. Further research into the implementation of this is required to determine if there is an improved effect on the control schemes, and how significant this effect is.

Table 5: ITAE Performance Criterion for PI and MPC Control Schemes on a High-Fidelity Distillation Column.

	Pressure +20%		Pressure -20%		Drum Level +20%		Drum Level -20%		Sump Level +20%		Sump Level -20%	
	MPC	PI	MPC	PI	MPC	PI	MPC	PI	MPC	PI	MPC	PI
Pressure	23.956	8.374	23.956	8.374	0.000	1.719	0.000	1.720	0.000	4.025	0.000	4.038
Drum Level	0.123	532.336	0.123	532.336	6.026	46.724	6.027	46.724	0.000	6.463	0.000	6.465
Sump Level	0.292	681.954	0.292	681.954	0.000	3.782	0.000	3.783	9.983	140.899	9.986	140.902

It is noted from Table 5, the MPC controller produces significantly lower ITAE values across all scenarios except when tracking SP changes in the condenser pressure. Changes on this SP resulted in the PI controller outperforming MPC, as shown in Figure 46, potentially due to the system being over damped.

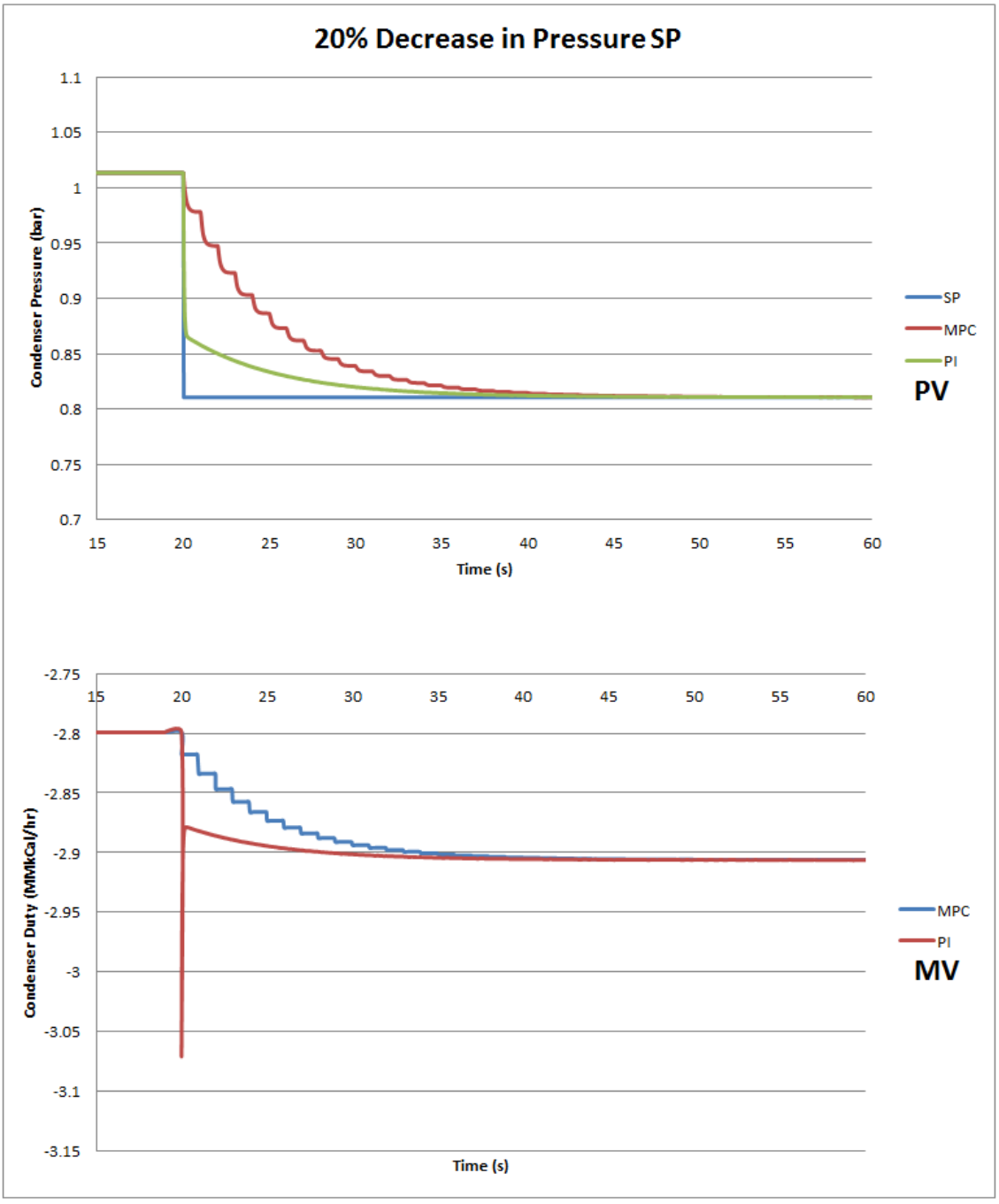

Figure 46: Plots of the PV and MV with a 20% Decrease in Condenser Pressure SP.

However the other PVs were not able to reject the DVs under these conditions. The deviation in the drum and sump levels shown in Figure 47 produced large ITAEs for the PI schemes when the pressure profile was altered, approximately 532 and 682 respectively. The MPC scheme rejects the disturbance as it occurs due to the predictive nature of the controller.

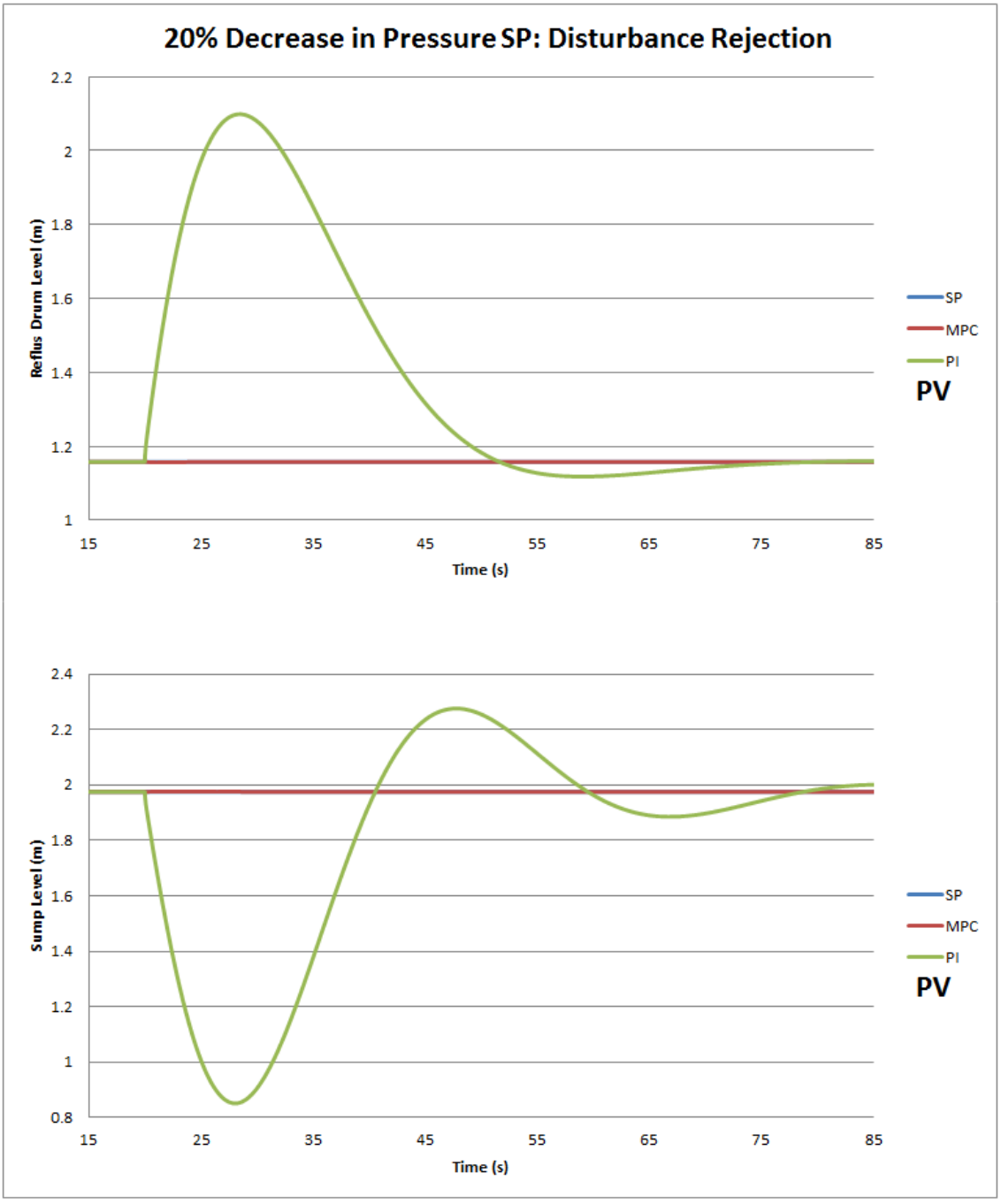

Figure 47: Plots of Drum and Sump Level Rejecting a 20% Decrease in Condenser Pressure SP.

Figure 48 displays the control schemes tracking the SP of the sump level. The SP was increased 20% with the MPC settles quickly at the new SP and avoiding overshoot. The PI eventually settled at the set point however as it is not able to predict the nature of the plant and the effect its input changes will have it overshoot by approximately 8%. This type of behaviour was consistent across the remaining steps and can be found in 10.21 Appendix 21.

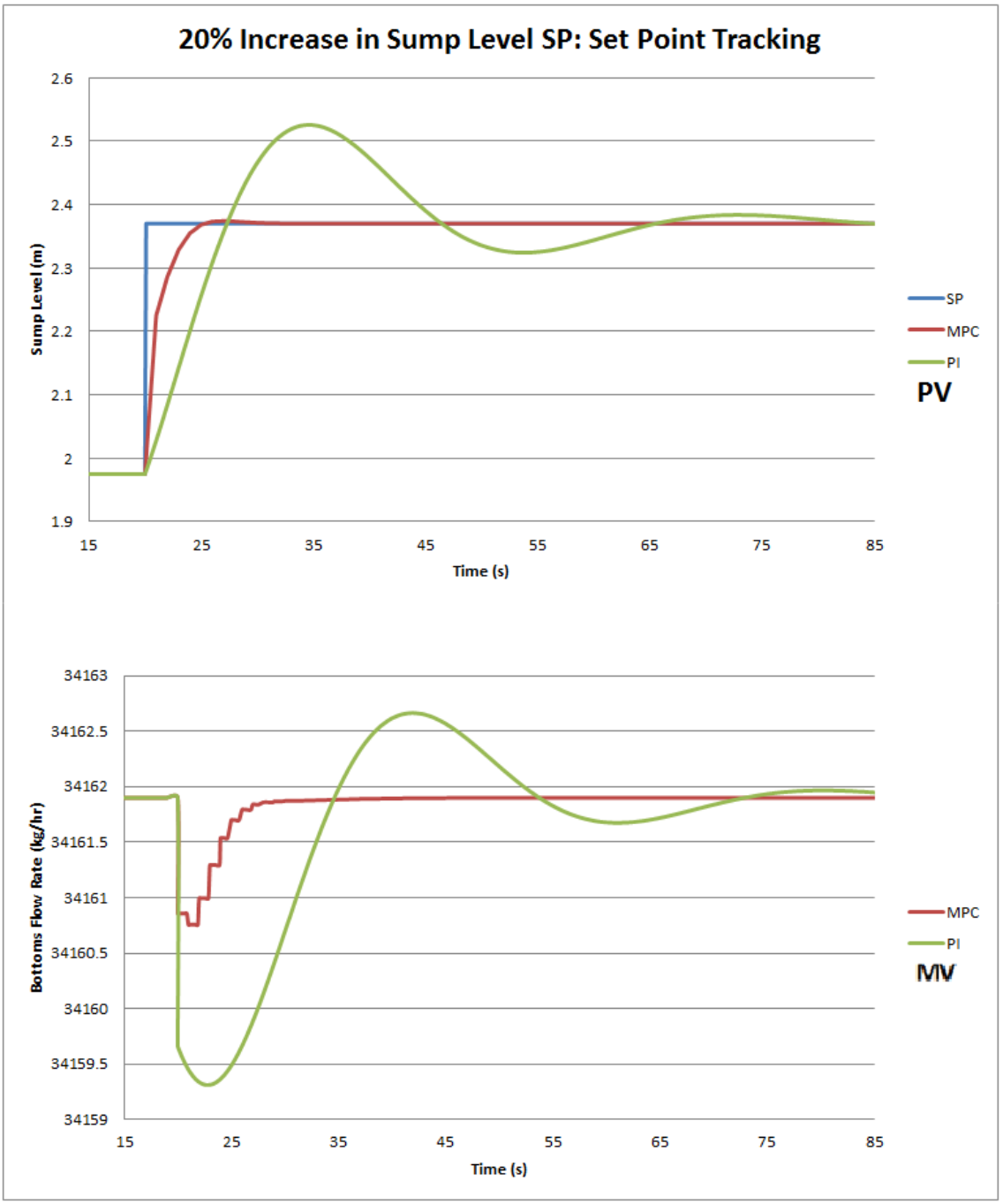

Figure 48: Plots of the PV and MV with a 20% Increase in Sump Level SP.

8.0 Conclusion

8.1 Summary

8.1.1 Software Integration

To conclude, the issue with the current software used in the chemical and petroleum industries is the lack of advanced control strategies. By integrating MATLAB with APD, and LabVIEW with APD, it was possible to implement MPC on the plant constructed in APD. This was achieved using ActiveX across all software packages. LabVIEW however was not connected to APD directly due to lack of documentation on the object tree structure. As such its current form lacks dynamic functionality. This paper documented the steps involved in creating the communication servers in the hope that this research will lead to the development of a template which can be used to train the user on advanced control schemes. This includes educating users on the implementation and tuning of predictive control and training users on the dynamics of complex systems.

8.1.2 Model Predictive Control Functionality

With communication established between these software packages the MPC toolboxes were explored. The main finding from this investigating was the limitations placed around the functionality of these toolboxes. Of these, the biggest limiting factors in MATLAB are the inability to quickly determine the MIMO plant model and perform RGA analysis. Originally it was thought the toolbox determined the model of the plant by introducing input steps into the open loop system. MATLAB could then utilise this data to determine the system model through its system identification tool. This unfortunately is not the case. The MPC toolbox linearises the system by evaluating the current Simulink blocks and parameters. It uses its existing knowledge of these blocks to conclude the SS representation of the plant. This causes

a major issue for the current set up as the plant is located on a different software package and thus MATLAB cannot determine the model. The toolbox also does not distinguish between the inputs and outputs of the system. If the user connects an input in position one and an output in position one the MPC controller will pair the two together. This is the case regardless of their interaction. As the toolbox does not determine which loop pairings are ideal this must be performed prior to connecting the streams to the controller.

As MATLAB did not possess the capabilities to perform these tasks a function and script was written in MATLAB to automate this process. Prior to this automation the user would have to import step data into the system identification tool, ident, and determine SISO TFs for each input and output combination. This is tedious and time consuming, especially when considering a 5 input, 5 output system. To determine the complete dynamics of this system the user would need to generate 25 SISO TFs before merging them into one MIMO matrix. The automation of this process not only performed steps on each input and created a data-driven MIMO matrix, it also provided the information necessary to accomplish the RGA analysis. The steady state gain of each interaction can easily be determined from the MIMO TF, through FVT, and used for constructing the RGA.

Similarly, LabVIEW did not afford the user any increase in functionality over MATLAB. The MPC toolkit was very rigid in needing the SS model in order to develop the MPC controller. Additionally, no system identification tool exists within LabVIEW. In order to develop the controller in LabVIEW the SS representation must be known. This can be developed mathematically or through input step tests. Given the script written in MATLAB it is possible to determine the MIMO TF matrix and use the LabVIEW block to convert the TFs to SS

representation. Alternatively, step tests can be performed in LabVIEW and the data logged. This logged data can then be used to identify the system model using Excel's solver add-on.

8.1.3 Control Scheme Performance

The present research also demonstrates the advantages of MPC control schemes. The control schemes were compared using the ITAE performance criterion. This comparison of PI and MPC illustrates that MPC control significantly outclasses the PI control schemes in all scenarios except condenser pressure SP tracking. Specifically, there is a large difference in the ITAE values between the control schemes. The improved performance of the MPC control scheme is likely due to its ability to deal with difficult dynamic systems, such as large time delays or inverse response. Not only is the scheme able to predict the behaviour of a MIMO system it also enforces operation within system constraints. Moreover, its ability to predict the behaviour of the plant and make sophisticated control changes means it settles quicker and avoids oscillations or overshoot. These findings support the literature, and highlight the advantages of MPC in industrial environments.

The reduction in the ITAE, along with the added benefit of operating closer to constraints, could aid in minimising energy usage in the distillation process. However the current perception in industry is MPC is too complex to implement without a specialist and consequently too costly and time consuming. In order to overcome such a hurdle better education and hands on experience is needed. To do so educational tools must be developed. This paper has provided a basis for establishing such a tool which could be used to educate personnel and students on the operation and dynamics of advanced control schemes around complex plants. Ideally, with the integration of the software packages, this can be performed in a safe, simulated environment however still provide a real visual experience through

LabVIEW. By increasing education it could be possible to remove the stigma surrounding MPC. However, additional work is required to establish a user friendly interface which complements users' knowledge and makes available the tools to assist in performing advanced control theory. Furthermore, the tool will also enable the user to gain experience operating complex plant operators without the cost or dangers involved. These are discussed in detail in 8.2 Future Works.

8.2 Future Works

One of the challenges with setting up a connection from LabVIEW to APD was the lack of documentation on the Aspen Custom Modeler DLL. Without knowing the syntax to use with the property and method nodes in LabVIEW multiple errors were encountered. Typically ActiveX errors are numbered and can be traced in the supplementary documentation. The errors encountered when exercising ActiveX to connect LabVIEW directly to APD were not documented thus this method was abandoned. Further research is needed in isolating these errors and creating documentation. Experience in computer systems and a deep understanding of object linking would be necessary during this research. If documentation was developed it would be possible to remove Excel from the communication sequence and minimise the amount of software used in achieving integration. An added side effect could be a reduction in the loop simulation time and see the communication interval be reduced from $730\ ms$ to approximately $500\ ms$.

Additionally, in order to employ LabVIEW as a training tool for implementing advanced control schemes on complex plants, software templates must be developed. The programs developed through this paper achieve the desired integration however a comprehensive knowledge of their operation is still required. This stems from a lack of adaptability within the programs

constructed. To successfully roll out such an educational product, the amount of hard coding requiring manipulation by the end user should be minimised. Currently the user must change the variable names within the Excel VBAs to ensure the correct variables are updated in APD. If LabVIEW was connected directly to APD and the Aspen Custom Modeler DLL was adequately documented the input required by the end user would be reduced. However, if this was not achieved Excel would remain essential for the data communication, and a dynamic way of sending the variable names from LabVIEW to Excel would be required. The infrastructure of the current template enables the variables names to be communicated from LabVIEW to Excel. However attempts to use this information in Excel, which is saved in a string, were not successful. If a technique to exploit this string information was determine and used to set the manipulated variables in APD the need for the end user to edit the VBA macro can be removed. When this transpires, a dynamic template in LabVIEW and Excel could be created and used in laboratories or training courses.

If a dynamic template was developed, training material would need to be created. The idea is focused around MPC on complex plants however the students could gain insight into both advanced control schemes and the behaviour of complex plants. The laboratories could cover the following ideas across LabVIEW, APD and AHD:

- Setup complex chemical or petroleum plants in APD and AHD respectively;
    - Gain an understanding of the Aspen products and their general operation.
- Introduce open loop dynamics of the complex systems;
    - Look into the development of input functions and plotting. These inputs could range from step or ramp inputs through to saw tooth and sinusoidal inputs.
- Implement PI control loops and tuning;

    - Different tuning methods could be employed and the variances in system behaviour compared. This also provides an introduction to performance criteria.
- Implement cascade and/or feedforward control schemes;
- Implement basic MPC through the LabVIEW toolkit and Aspen APC;
    - These could cover different features across a few laboratories building on their previous knowledge each week.
- Create sub VI's, functions or scripts which can achieve adaptive MPC or adaptive RGA;
    - Challenge the students to create their own adaption to the base template and attempt to apply their knowledge of advance control techniques in LabVIEW.
- Implement advanced MPC through LabVIEW toolkit and Aspen APC.
    - A final comparison of the toolboxes full capabilities. The students could be challenged to see who can create the best performing, or who can make the best adaptive, MPC controller in a final project. And also look at optimising a profit function to see which scheme resulted in the biggest margins.

The laboratories would be designed around increasing the users' knowledge of advanced control schemes and it is believed the hands on approach and real time simulation will provide a good basis for increased implementation in industry. By receiving increased exposure to MPC, the stigma surrounding MPC will slowly be removed. If this difficulty is removed, and the benefits sold, then as students expand into industry they will be open to the implementation of such control schemes, potentially even the driving force behind it.

Furthermore, an additional proposal surrounding the template is the ability to expose personnel to specific scenarios. A good plant operator understands how each section of a plant

functions and the effect small operating changes have on the system. Therefore, an operator will be trained over many years to ensure they understand the plant dynamics and how to safely operate the plant. If the plant they are learning was established in APD or AHD then the template could be used for the testing of different input changes as well as fault and training scenarios. A set of scenarios could be developed to put the plant into a volatile state. The operator would then need to take corrective action in order to avoid an event of differing magnitudes. As the LabVIEW template will operate as a real time simulator it will provide an experience similar to controlling the plant in reality with the added safety benefit.

Finally, as was produced in MATLAB, a function or set of sub VIs need to be developed to achieve system identification in LabVIEW. Currently there is no tool in LabVIEW which allows for a SISO or MIMO system to be identified. It could be possible to integrate Excel and the solver add-on with LabVIEW to perform this. This could possibly be done by:

- Introducing a step into the system;
- Wait for either steady states or a predetermined fix time period to pass;
- Analyse the data using the solver add-on and determine the TF;
- Convert the TF into SS and save it into an array;
- Step the next input and repeat the process; then
- Once all the inputs are completed the sub VI could output the MIMO SS model.

This process could then be performed before entering a loop to control the system in the real time simulator. Furthermore, this could also be used for RGA analysis once the matrix was constructed. Overall the MATLAB toolbox and LabVIEW toolkit are lacking around adaptive control and modelling. They assume the user has the SS model ready to feed into the ‘Create MPC’ node. By researching and expanding on the toolboxes it could open more avenues for

operators to switch to MPC through the diminished fear surrounding their lack of knowledge on the subject.

Overall the driving theme surrounding this, and all proposed research is providing more accessible tools and documentation for students and personnel on MPC. The increased exposure and understanding of MPC will in turn provide greater confidence in advanced control scheme. Furthermore, the experience in implementing MPC and observing the benefit on energy and cost reduction might compel more of industry to change to advanced optimised control schemes.

9.0 Bibliography

Ali, N. B., & Petersen, K. (2012). A Consolidated Process for Software Process Simulation. *38th Euromicro Conference on Software Engineering and Advanced Applications*, pp. 4-5.

Ang, K. H., Chong, G. C., & Li, Y. (2005). PID Control System Analysis, Design and Technology. *IEEE Transcations on Control Systems Technology*, 559-576.

AspenTech. (2000a). *Aspen Plus User Guide 10.2.* Cambridge: Aspen Technology.

AspenTech. (2000b). Using the Aspen Plus ActiveX Automation Server. In *Aspen Plus User Guide 10.2* (pp. 37-1;38-35). Cambridge: Aspen Technology.

AspenTech. (2005). *Aspen Custom Modeller Reference Guide.* Massachusetts: AspenTech.

AspenTech. (2011). *Advanced Process Control.* Massachusetts: AspenTech.

AspenTech. (2015a). *Design and Optimise Chemical Processes with Aspen Plus*. Retrieved September 11, 2015, from Angineering Products: http://www.aspentech.com/products/aspen-plus.aspx

AspenTech. (2015b). *Aspen HYSYS*. Retrieved September 18, 2015, from AspenTech: http://www.aspentech.com/products/aspen-hysys/

AspenTech. (2015c). *Aspen HYSYS Dynamics*. Retrieved September 25, 2015, from Products: https://www.aspentech.com/products/aspen-hysys-dynamics.aspx

AspenTech. (2015d). *Aspen Capital Cost Estimator*. Retrieved September 28, 2015, from AspenTech: http://www.aspentech.com/products/aspen-kbase.aspx

AspenTech. (2015e). *APC Model Builder*. Retrieved September 08, 2015, from AspenTech: https://www.aspentech.com/products/APC-Model-Builder/

Boyes, W. (2009). Advanced Control for the Plant Floor. In *Instrumentation Reference Book* (pp. 623-62). Oxford: Butterworth-Heinemann.

Bristol, E. H. (1966). New Measure of Interaction for Multivariable Process Control. *IEEE Transactions on Automatic Control*, 133-134.

Cheremisinoff, N. P. (2000). Distillation Equipment. In *Handbook of Chemical Processing Equipment* (pp. 162-165). Oxford: Elsevier.

Cook, S. (2015, Sept 24). *Advantages of Microsoft Excel*. Retrieved Dec 14, 2015, from Computer Software: http://hubpages.com/technology/Advantages-of-Microsoft-Excel

Douglas, J. M. (1988). Azeotropic Systems. In *Conceptual Design of Chemical Processes* (pp. 173-192). New York: McGraw-Hill.

Emerson. (2011). *MOL Reduces Energy Consumption at its Algyo Gas Plant Using Emerson's SmartProcess Distillation Optimizer*. Retrieved August 29, 2015, from Documents: http://www2.emersonprocess.com/siteadmincenter/PM%20DeltaV%20Documents/ProvenResults/OilGasRefining/RES_OG_MOL_Algyo_APC_Final_7-11.pdf

Emerson. (2013). *DeltaV Product Data Sheet.* Minnesota: EMERSON.

Emerson. (2015). *DeltaV DCS System Overview*. Retrieved October 08, 2015, from DeltaV: http://www2.emersonprocess.com/en-us/brands/deltav/differentiators/pages/systemoverview.aspx

Gorak, A., & Sorensen, E. (2014). Energy Considerations in Distillation. In *Distillation: Fundamentals and Principles* (pp. 226-267). Oxford: Elsevier.

Green, D. W., & Perry, R. H. (2007). Distillation. In *Perry's Chemical Engineers' Handbook* (pp. 13.4 - 13.9). New York: McGraw-Hill.

Hanyak, M. E. (2012). *Chemical Process Simulation and the Aspen HYSYS Software.* Pennsylvania: Bucknell University.

Johnson, G. W., & Jennings, R. (2006). LabVIEW Data Types. In *LabVIEW Graphical Programming* (p. 126). New York: McGraw-Hill.

Kheir, N. (1995). Preface. In *Systems Modeling and Computer Simulation* (pp. vii-viii). New York: Marcel Dekker.

Khoury, F. M. (2005). The Equilibrium Stage. In *Multistage Separation Processes* (pp. 61-62). Florida: CRC Press.

Kister, H. Z. (1992). Tray Design and Operation. In *Distillation Design* (pp. 259-267; 421-434). New York: McGraw-Hill.

Lee, J.-H., & Dudukovic, M. P. (1998). A Comparison of the Equilibrium and Nonequilibrium Models for a Multicomponent Reactive Distillation Column. *Computers and Chemical Engineering*, 159-172.

Lee, P. L. (1993). Generic Model Control - The Basics. In *Nonlinear Process Control* (pp. 11-13). London: Springer-Verlag.

Lee, P. L., & Sullivan, G. R. (1988). Generic Model Control. *Computers and Chemical Engineering*, 573-580.

Levine, W. S. (1996). Design Using Performance Indicies. In *The Control Handbook* (p. 170). Florida: CRC Press.

Ma, Y. (2013). Introduction to Engineering Informatics. In *Semantic Modeling and Interoperability in Product and Process Engineering* (p. 15). London: Springer-Verlag.

Maciejowski, J. M. (2002). *Predictive Control: with Constraints.* Essex: Pearson Education.

MathWorks. (2009). *Server Creation Failed: Class not registered*. Retrieved September 03, 2015, from MATLAB Central: http://au.mathworks.com/matlabcentral/newsreader/view_thread/257718

MathWorks. (2015a). *MATLAB*. Retrieved October 19, 2015, from Mathworks: http://au.mathworks.com/products/matlab/

MathWorks. (2015b). *Product and Services*. Retrieved October 19, 2015, from Mathworks: http://au.mathworks.com/products/

Mathworks. (2015c). *Simulink*. Retrieved October 19, 2015, from Mathworks: http://au.mathworks.com/products/simulink/

MathWorks. (2015d). *actxserver*. Retrieved August 04, 2015, from Documentation: http://au.mathworks.com/help/matlab/ref/actxserver.html

MathWorks. (2015e). *invoke*. Retrieved September 06, 2015, from Documentation: http://au.mathworks.com/help/matlab/ref/invoke.html

MathWorks. (2015f). *pwd*. Retrieved September 08, 2015, from Documentation: http://au.mathworks.com/help/matlab/ref/pwd.html

MathWorks. (2015g). *cd*. Retrieved September 08, 2015, from Documentation: http://au.mathworks.com/help/matlab/ref/cd.html

MathWorks. (2015h). *Model Predictive Control Toolbox*. Retrieved August 21, 2015, from Documentation: http://au.mathworks.com/help/mpc/index.html

MathWorks. (2015i). *MPC Controller*. Retrieved August 25, 2015, from Documentation: http://au.mathworks.com/help/mpc/ref/mpccontroller.html

MathWorks. (2015j). *Identify Linear Models Using System Identification App*. Retrieved October 24, 2015, from Documentation: http://au.mathworks.com/help/ident/gs/identify-linear-models-using-the-gui.html

Microsoft. (2013). *Office 2013 VBA Documentation*. Retrieved October 18, 2015, from Microsoft Download Center: http://www.microsoft.com/en-us/download/details.aspx?id=40326

Microsoft. (2015a). *Excel*. Retrieved October 15, 2015, from Microsoft Office: https://products.office.com/en-us/excel

Microsoft. (2015b). *Microsoft OLE DB*. Retrieved August 04, 2015, from Windows Data Access Components: https://msdn.microsoft.com/en-us/library/ms722784(v=vs.85).aspx

Mujtaba, I. M. (2004). *Batch Distillation: Design and Operation.* London: Imperial College Press.

National Instruments. (2007a). *ActiveX and LabVIEW.* Retrieved September 10, 2015, from National Instruments: http://www.ni.com/white-paper/2983/en/

National Instruments. (2007b). *CD Convert to State-Space Model (Control Design Toolkit)*. Retrieved October 15, 2015, from Manuals: http://zone.ni.com/reference/en-XX/help/370853D-01/lvctrldsgn/convert_state_space_model/#parent

National Instruments. (2009). Creating and Implementing a Model Predictive Controller. In *Control Design User Manual* (pp. 18.1 - 18.19). Texas: National Instruments.

National Instruments. (2015a). *LabVIEW System Design Software*. Retrieved October 20, 2015, from National Instruments: http://www.ni.com/labview/

National Instruments. (2015b). *LabVIEW for Higher Education*. Retrieved October 20, 2015, from National Instruments: http://www.ni.com/labview/applications/academic/

Neelis, M., Worrell, E., & Masanet, E. (2008). *Energy Efficiency Improvement and Cost Saving Opportunities for the Petrochemical Industry.* Berkeley: University of California.

Ogunnaike, B. A., & Ray, W. H. (1994). Process Control. In *Process Dynamics, Modeling and Control* (pp. 461-675; 723-807; 992-1022). Oxford : Oxford University Press.

Olujić, Ž., Sun, L., Gadalla, M., de Rijke, A., & Jansens, P. J. (2008). Enhancing Thermodynamic Efficiency of Energy Intensive Distillation Columns. *Chemical and Biochemical Engineering Quarterly*, 383-392.

OSIsoft. (2015). *What Is PI*. Retrieved October 09, 2015, from OSIsoft: http://www.osisoft.com/software-support/what-is-pi/What_Is_PI.aspx

Peers, Z. (2013). *Jump Start: Pressure Relief Scenario in Aspen Plus Dynamics V8.* Cambridge: Aspen Technology.

Robinson, S. (1993). The Application of Computer Simulation in Manufacturing. *Integrated Manufacturing Systems*, 18.

Romagnoli, J. A., & Palazoglu, A. (2005). Model Predictive Control (MPC). In *Introduction to Process Controll* (pp. 319-335; 251-265). Florida: CRC Press.

Schneider Electric. (2015). *PRO/II Process Simulation Datasheet.* California: Schneider Electric.

Schwartz, R. L., Olson, E., & Christiansen, T. (1997). *Introduction to OLE Automation*. Retrieved October 04, 2015, from Learning Perl on Win32 Systems: http://docstore.mik.ua/orelly/perl/learn32/ch19_01.htm

Sharmila, M., & Mangaiyarkarasi, V. (2014). Modeling and Control of Binary Distillation Column. *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, 105-111.

South Dakota School of Mines and Technology. (2000). *RadFrac for Dummies: A How to Guide on Aspen Plus.* South Dakota: SDSM&T.

Strang, G., & Borre, K. (1997). Least-Squares Approximation. In *Linear Algebra, Geodesy, and GPS* (pp. 174-176). Cambrdige: Wellesley-Cambrdige Press.

Tremblay, D., & Mantrala, V. (2014). *Jump Start: Aspen Simulation Workbook in Apsen HYSYS V8.* Massachusetts: AspenTech.

U.S. Energy Information Administration. (1991). *1991 MECS Survey Data*. Retrieved September 01, 2015, from Manufacturing Energy Consumption Survey: http://www.eia.gov/consumption/manufacturing/data/1991/

U.S. Energy Information Administration. (1994). *1994 MECS Survey Data*. Retrieved September 01, 2015, from Manufacturing Energy Consumption Survey: http://www.eia.gov/consumption/manufacturing/data/1994/

U.S. Energy Information Administration. (1998). *1998 MECS Survey Data*. Retrieved September 01, 2015, from Manufacturing Energy Consumption Survey: http://www.eia.gov/consumption/manufacturing/data/1998/

U.S. Energy Information Administration. (2002). *2002 MECS Survey Data*. Retrieved September 01, 2015, from Manufacturing Energy Consumption Survey: http://www.eia.gov/consumption/manufacturing/data/2002/

U.S. Energy Information Administration. (2006). *2006 MECS Survey Data*. Retrieved September 01, 2015, from Manufacturing Energy Consumption Survey: http://www.eia.gov/consumption/manufacturing/data/2006/

U.S. Energy Information Administration. (2010). *2010 MECS Survey Data*. Retrieved September 01, 2015, from Manufacturing Energy Consumption Survey: http://www.eia.gov/consumption/manufacturing/data/2010/

University of Michigan. (2010). *Encyclopedia of Chemical Engineering Equipment*. Retrieved September 05, 2015, from Bubble Cap: http://encyclopedia.che.engin.umich.edu/Pages/SeparationsChemical/DistillationColumns/Hotspot/BubbleCap.html

Woolf, P. (2011). *University of Michigan.* Retrieved September 17, 2015, from Michigan Chemical Process Dynamics and Controls: https://controls.engin.umich.edu/wiki/images/e/e9/

10.0 Appendices

10.1 Appendix 1

Initial Conditions

Feed

- Benzene: 30%
- Toluene: 40%
- Xylenes: 30%
- Steady State Flow: $500\ kmol/hr$
- Steady State Temperature: $100\ °C$
- Steady State Pressure: $1.5\ atm$
- Input Stage: 15

Tray

- Stages: 29
- Type: $Seive$
- Spacing: $18\ in$
- Diameter: $1.95\ m$
- Weir Height: $5\ cm$

Reflux Drum

- Nominal Liquid Depth: $0.67\ m$
- Length: $1.34\ m$
- Diameter: $1.95\ m$

Sump

- Nominal Liquid Depth: $1.4875\ m$

- Height: $2.975\ m$
- Diameter: $1.95\ m$

Hydraulics

- Simple Tray: $Stages\ 2 - 28$
- Lw/D: 0.72666

10.2 Appendix 2

Control Specifications

Plant Inputs

- Condenser duty
- Reboiler duty
- Reflux mass flow rate
- Distillate mass flow rate
- Bottoms mass flow rate
- Feed molar flow rate

Plant Outputs

- Condenser pressure
- Reflux drum liquid level
- Sump liquid level
- Mass fraction toluene in the distillate
- Mass fraction benzene in the bottoms

Objectives

- Maintain tower pressure
- Maintain 5% toluene in distillate, or 95% benzene in distillate

- Maintain 1.7% benzene in bottoms
- Maintain liquid levels in reflux drum and sum at nominal levels found in 10.1 Appendix 1.

10.3 Appendix 3

Aspen Plus 'Radfrac' Setup

To launch Aspen Plus go to the Start Menu – All Programs – AspenTech – Process Modeling V8.6 – Aspen Plus - Aspen Plus V8.6. When launching, if prompted to register the license, select Register Later as shown in Figure 49.

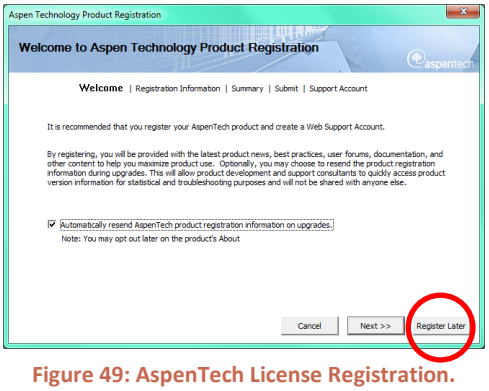

Figure 49: AspenTech License Registration.

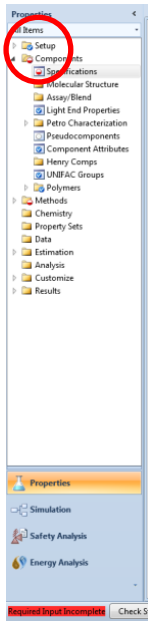

Figure 50: Properties Explore in Aspen Plus.

Once the software begins select New – Blank and Recent – Blank Simulation – Create. This will launch a blank template and display the Components – Specifications. Select Setup at the top of the list, shown in Figure 50, enter the title of the project "High-Fidelity Distillation Tower" and set the global unit set to "MET" for metric. Click Next and it will return to the Components – Specifications page.

Aspen Plus has a database of components which it can auto fill in the details for (type, component name, alias). You must list all the components which will be included in the project here. Start by entering Benzene into the component ID and press enter. If Aspen Plus does not

auto fill the data there is a problem with the available databases. Figure 51 highlights the Enterprise Database displaying an error to the user.

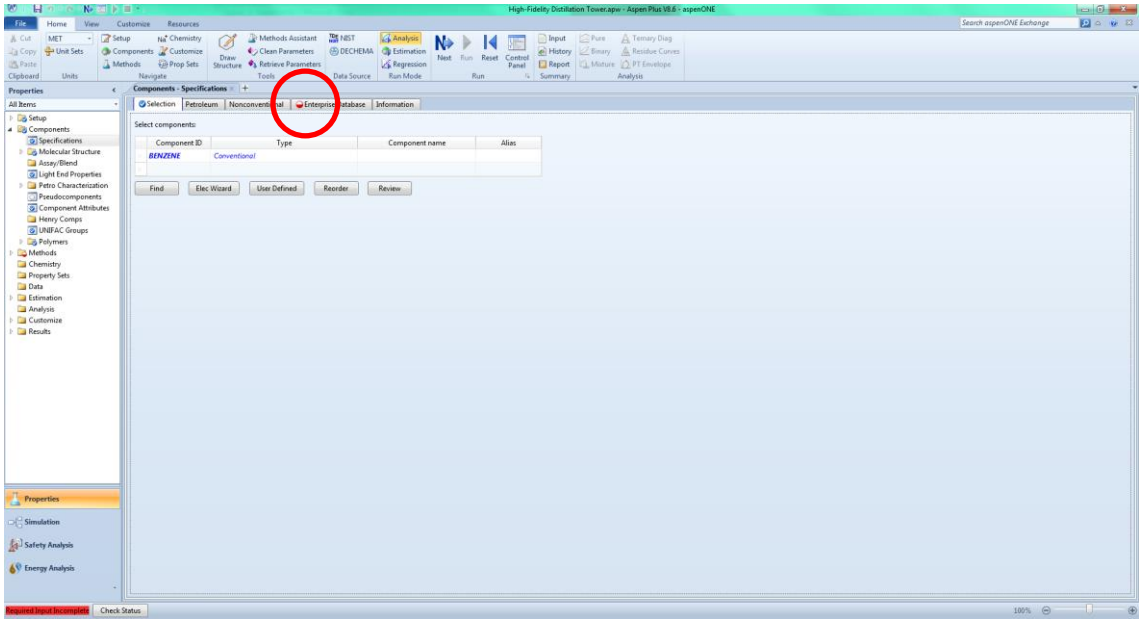

Figure 51: Error Notification in Aspen Plus for the Enterprise Databases.

Select this tab and determine if the databanks are available. Aspen Plus typically has a large selection of databases available. If these are not available then the databases are not correctly installed on the device and a reinstall should be performed before continuing.

Once the databases have been successfully installed the appropriate data will automatically complete, as seen in Figure 52. Enter the remaining components, Toluene and p-Xylene, then select Next.

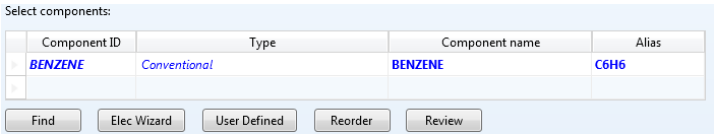

Figure 52: Component Entry in Aspen Plus.

This will display the Method Specifications page. The base method use will be NRLT which applies Ideal gas and Henry's law. Select next.

This will prompt the user to run a property analysis to determine if the setup is completed before moving to the PFD. Figure 53 displays the prompt from Aspen Plus. If this completed without error the PFD will be displayed. If not then review the previous step of this documentation to ensure all the information has been entered correctly. The results page will inform the user of the issues encountered when performing the analysis.

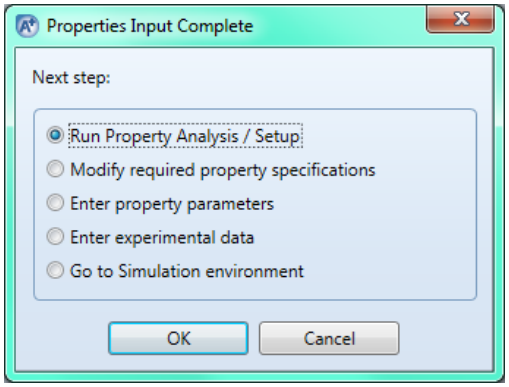

Figure 53: Property Analysis in Aspen Plus.

The PFD will currently be empty. Select Columns from the Model Palette and drag the 'RadFrac' column onto the flow sheet as shown in Figure 54. The object will be named B1 by default. Right click on the 'RadFrac' column and select 'Rename Object', name the object DISTCOL.

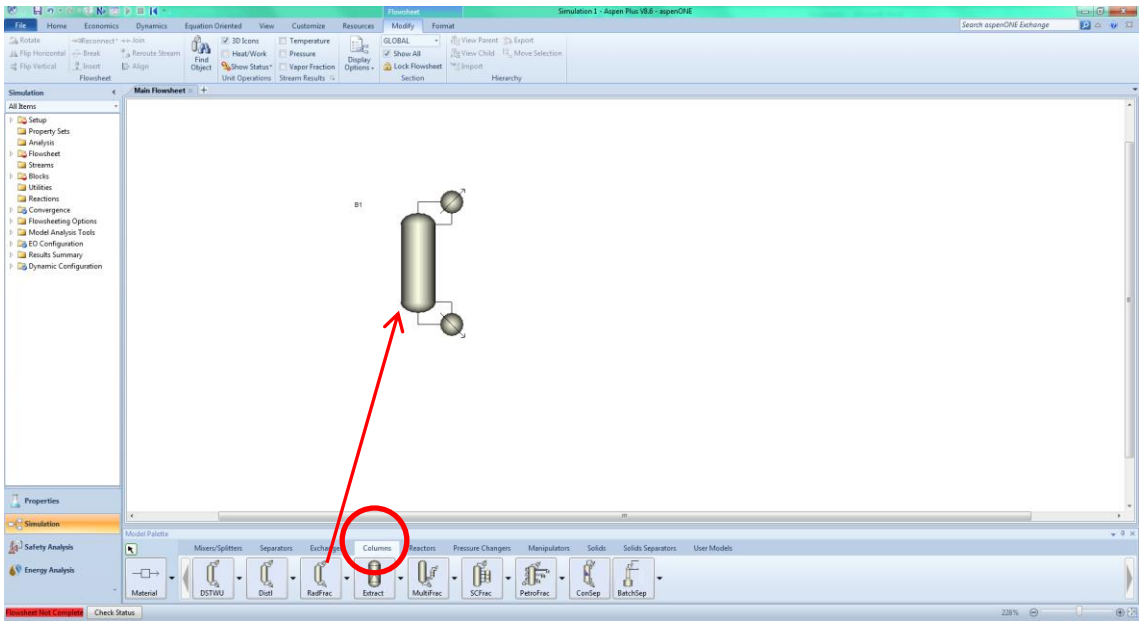

Figure 54: 'Radfrac' Column in Aspen Plus.

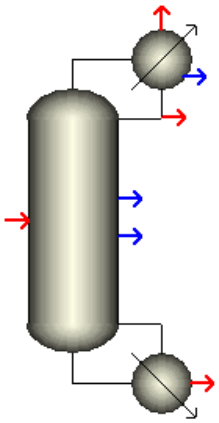

Figure 55: 'Radfrac' Column Expecting Material Streams in Aspen Plus.

The column now needs the feed, distillate and bottoms streams created. Select the Material stream. This will highlight the input and output ports for the given material stream. The red ports are essential while blue are optional streams. Once all three streams are connected they can be renamed by right-clicking on the stream and selecting Rename Stream. These streams are named:

- Feed;
- Distillate; and
- Bottoms.

Figure 56 shows all the necessary streams and objects on the PFD. It is now possible to enter the numerical specifications for operation.

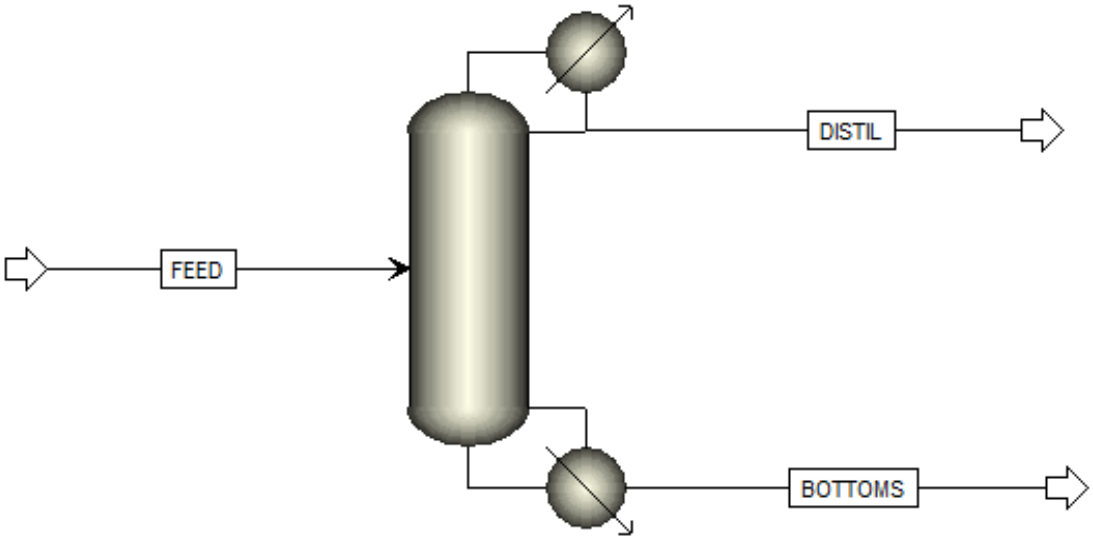

Figure 56: PFD in Aspen Plus.

By examining the Simulation Explorer located on the left-hand side of the PFD, any object without a blue tick requires attention. Select Streams – Feed – Input.

There are three state variables which can be selected; however you may only choose two of the three. For this example we will be specifying temperature and pressure. Change temperature into Celsius and pressure to atmosphere. Then enter 100 $°C$ and 1.5 atm. As stated in 10.1 Appendix 1 the Feed is 500 $kmol/hr$. Leave the total flow basis as Mole and enter 500. Now the composition must be entered: the feed stream contains 30% of benzene, 40% of toluene and 30% of xylenes. Change the drop down to Mole-Frac and enter the variables. It should Total 1 or 100%. Figure 57 displays the completed Feed specifications.

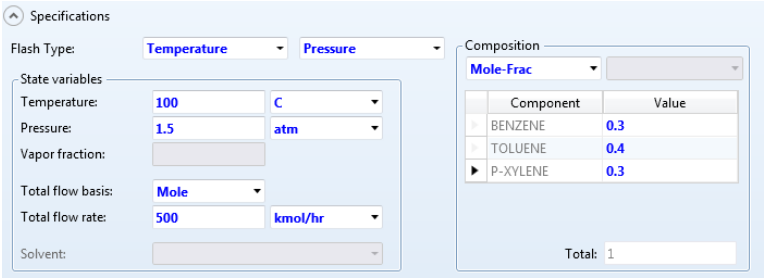

Figure 57: Completed Column Feed Specifications in Aspen Plus.

Select Next and the distillation columns specification setup will be displayed. There are 29 stages in the column and the condenser is of type total, specified in 10.1 Appendix 1. Set the distillate rate at 150 $kmol/hr$, as you want to recover 95% of the benzene, with 5% toluene, and the reflux ratio at 1.5 mol as shown in Figure 58.

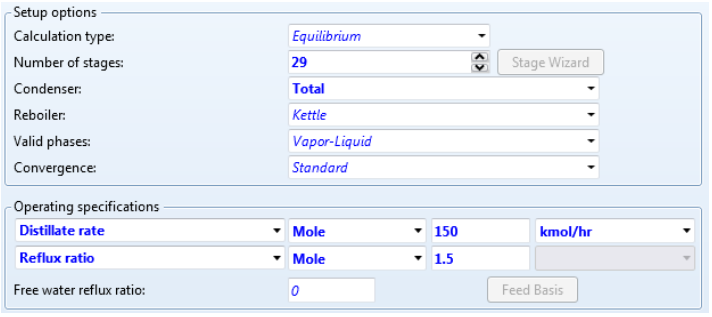

Figure 58: Distillation Column Specifications in Aspen Plus.

Now the feed stage is specified as stage 15 and the distillate and bottoms product streams stage 1 and 29 respectively. As the process is ideal no efficiencies will be changed. Select Next and the completion prompt in Figure 59 will appear. Do not run the simulation yet as the tray sizing has not been specified.

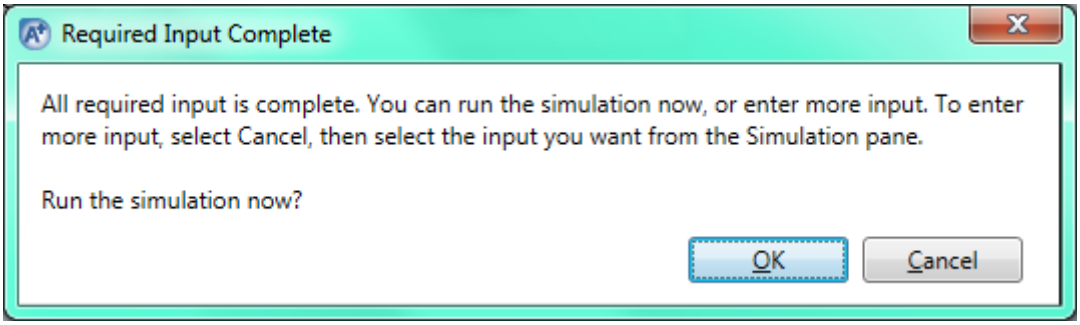

Figure 59: Simulation Prompt in Aspen Plus.

Select Cancel then the Sizing and Rating tab in the Simulation Explorer, shown in Figure 60. Select New and enter the tray specifications from 10.1 Appendix 1. As the condenser and reboiler count as stages the starting tray is specified as 2 and ending stage as maximum trays minus one, or 28. The tray type is 'Sieve' and spacing is 18 in.

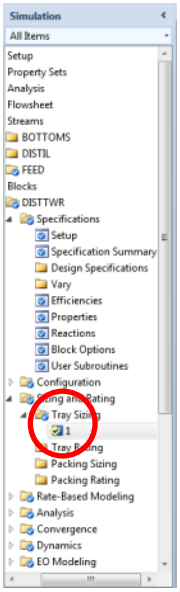

Figure 60: Tray Sizing tab inside the Simulation Explorer in Aspen Plus.

Now select tray rating below and select stages 2 and 28 for the starting and ending trays, as well as the Sieve type. Now enter the tray geometry: diameter and tray spacing from 10.1 Appendix 1. Enter the diameter as 1.95 m, tray spacing 18 in and weir height of panel A as 5 cm.

Finally, select the next tab Design/Pdrop. Press Update section pressure profile and define the pressure at the top of the column as 1 atm and bottom stage as 1.6 atm. Aspen Plus will determine the pressure drop through the column based on these estimates. All the inputs are

now complete and the prompt shown in Figure 59 will be displayed again. This time the simulation can be run and the convergence results will be displayed as shown in Figure 61. This will inform the user if any items require attention.

```
          4    1    4      9.4411...

 Flowsheet Analysis :

COMPUTATION ORDER FOR THE FLOWSHEET:
DISTTWR

->Calculations begin ...


  Block: DISTTWR  Model: RADFRAC

      Convergence iterations:
        OL   ML   IL      Err/Tol
         1    1    4       457.46
         2    1    4       43.924
         3    1    5       23.302
         5    1    4      0.73359

->Simulation calculations completed ...


  ***  No Warnings were issued during Input Translation ***


  ***  No Errors or Warnings were issued during Simulation ***

->Generating results ...

     INFORMATION WHILE GENERATING REPORT FOR UNIT OPERATIONS BLOCK: "DISTTWR"
     (MODEL: "RADFRAC")
     TPSAR MESSAGE:   117.48%  JET FLOOD IN COLUMN DISTTWR , SECTION   1
                      EXCEEDS 80%.

     INFORMATION WHILE GENERATING REPORT FOR UNIT OPERATIONS BLOCK: "DISTTWR"
     (MODEL: "RADFRAC")
     TPSAR MESSAGE:   SECTION   1 IN COLUMN DISTTWR
                      IS ABOVE 100% JET FLOOD. HOWEVER, THE DOWNCOMERS
                      HAVE SPARE CAPACITY. TRY DECREASING DOWNCOMER WIDTHS.
```

Figure 61: Convergence Iterations in Aspen Plus.

Furthermore the results can be analysed in the Stream Results tab found in Figure 62.

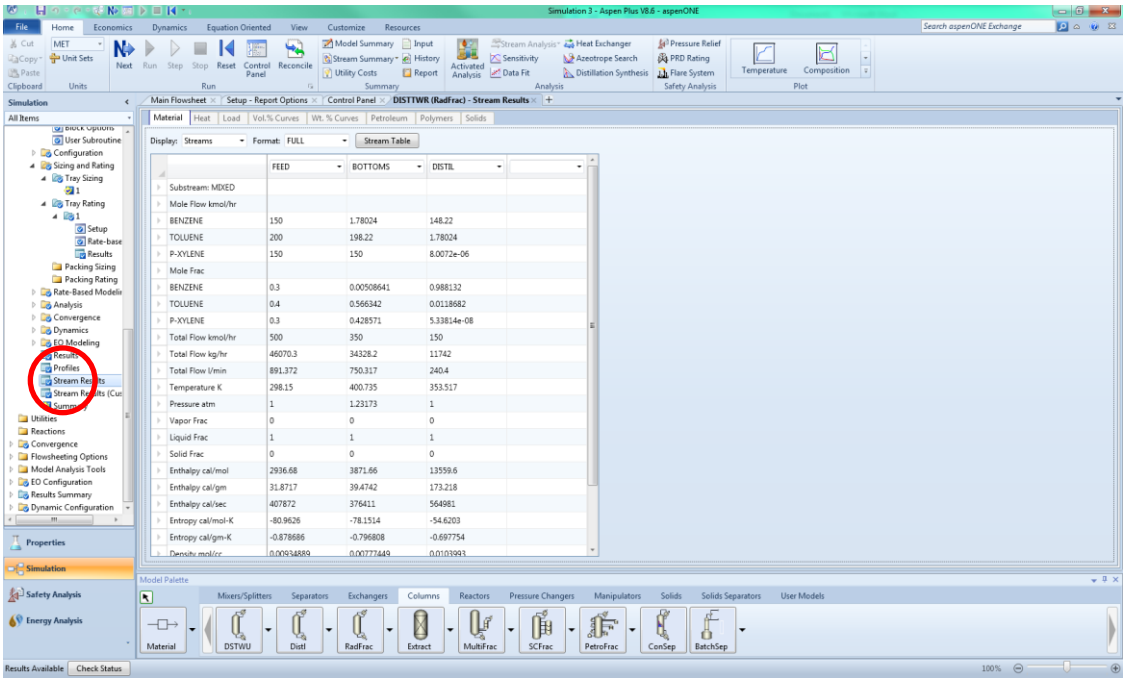

Figure 62: Stream Results in Aspen Plus.

10.4 Appendix 4

Aspen Plus Steady-State to Dynamic

APD allows the end user to simulate and optimise continuous and batch processes. This appendix provides the necessary information to transition a steady state model from Aspen Plus to APD. Additional information is available for the interested reader through AspenTech's support, found at support.aspentech.com. A general rule of thumb is to design the PFD inside the steady state simulation package as it is easier to maintain the properties and avoids duplication of processes.

Once the steady state model is developed in Aspen Plus the unit operators must be isolated for exportation to APD. The first step is selected Dynamic Mode from the Dynamics tab in the ribbon, as shown in Figure 63. With Dynamic Mode selected the simulation is run.

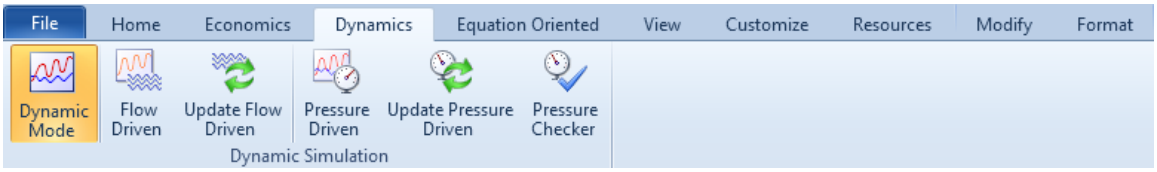

Figure 63: Dynamics Tab in the Aspen Plus Ribbon.

Note that if you want analyse the entire PFD then you do not need to isolate the unit operations. However if you wanted to only look at certain operations then you can isolate that unit by right-clicking on the unit's feed streams and selecting 'Reconcile'. This will launch a popup which allows the user to select the variables to reconcile. Once all feeds have been reconciled the remaining unit operators can be removed from the PFD.

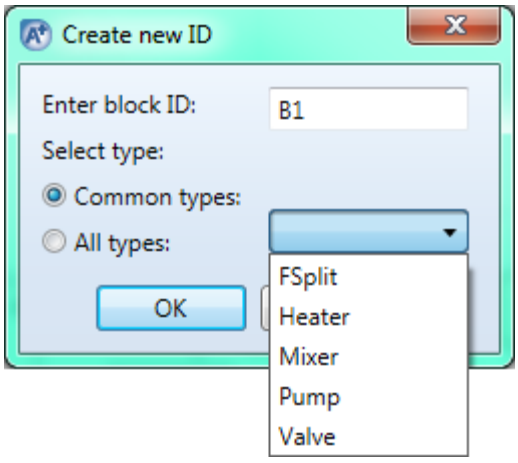

Figure 64: Block Options to Insert to an Existing Stream.

Now the type of analysis must be selected: flow driven; or pressure driven. These are selected in the Dynamics tab as shown in Figure 63. Most analysis will only require flow driven however if the user needs to analyse the pressure gradient or relief then a pressure driven analysis must be performed. A pressure driven analysis requires pumps and valves be added to the PFD. To quickly add an object to an existing stream right-click on the stream and select 'Insert Block'. This will launch a popup, as shown in Figure 64, which allows the user to select either a:

- Stream Splitter (FSplit);
- Heater;
- Mixer;
- Pump; or
- Value.

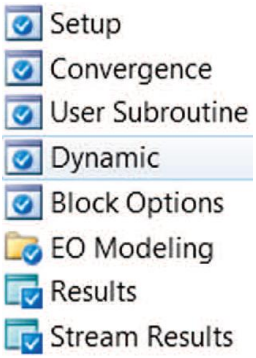

Figure 65: Navigation Pane Options for a Unit

Once the values or pumps are added and the outlet pressure defined the dynamics of the unit operators can be entered. The primary difference between steady state and dynamic operation is the sizing, efficiency and heat transfer of equipment. The valves do not require dynamic specifications but the unit operator dynamics can be specified in the navigation pane as shown in Figure 65. By opening this tab it is possible to specify the sizing and heat transfer of the unit operation, as shown in Figure 66.

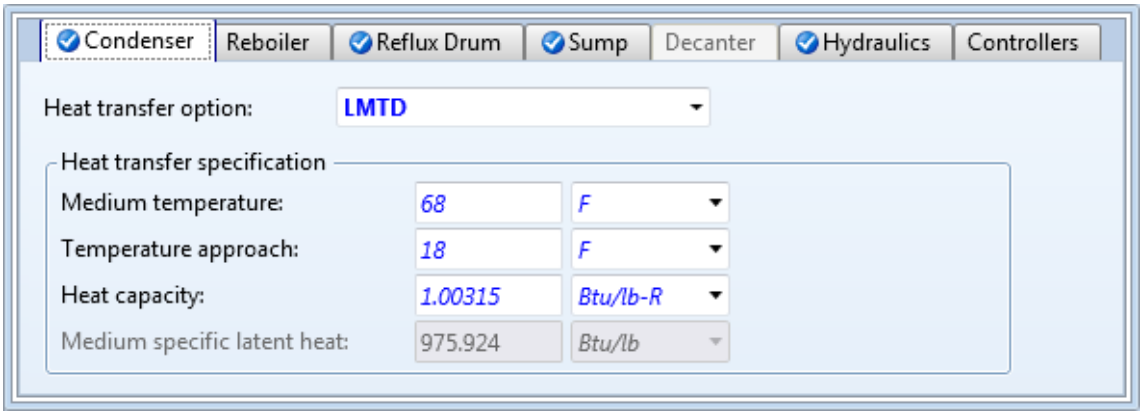

Figure 66: Dynamic Options for a 'Radfrac' Distillation Column in Aspen Plus.

Furthermore, the controllers can be pre-selected before exporting to APD by selecting the Controllers tab. APD will automatically add controllers based on default selections when the model is exported to APD. By changing the setting in the Controllers tab it is possible to manipulate which control loops are created in APD. Figure 67 displays the controller options for the distillation column, this tab will differ depending on the unit operation being controlled, which will override the defaults if selected.

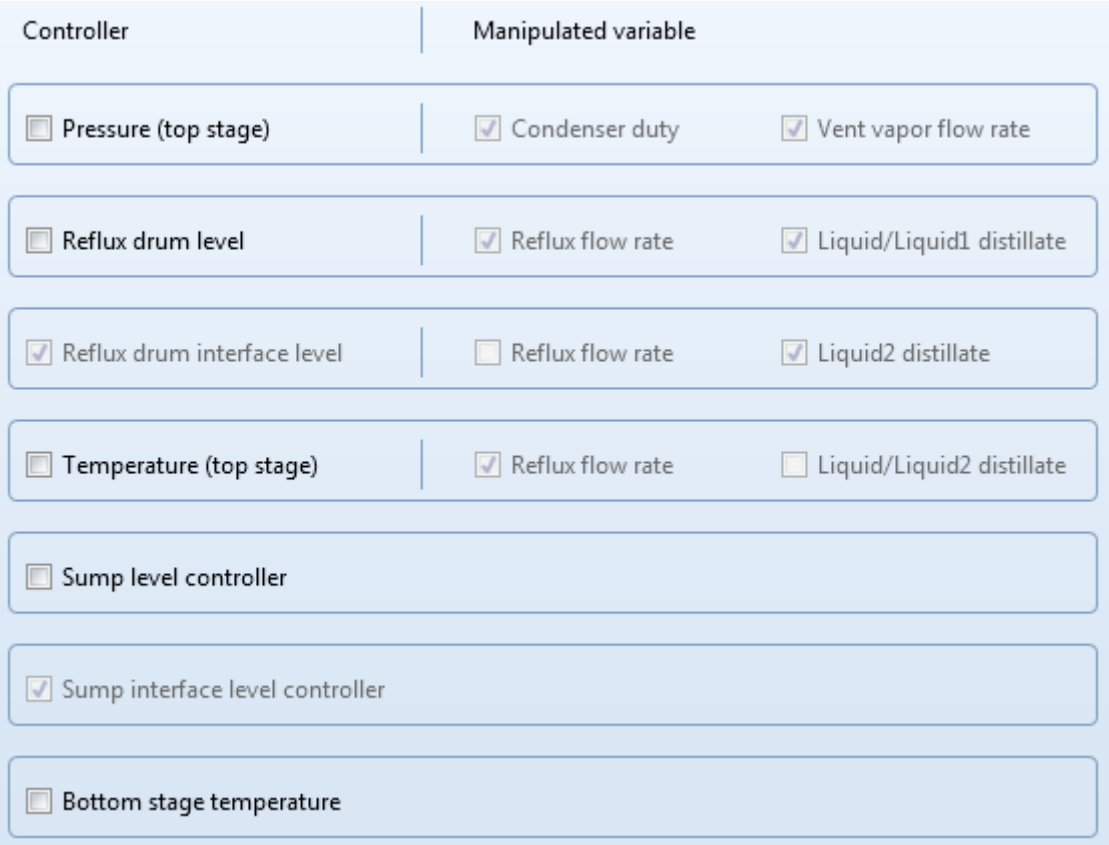

Figure 67: Controller Selection Pane in Aspen Plus.

With all the dynamic specifications completed the simulation should be run again to confirm convergence then can be exported by specifying and selecting the analysis to be completed. This will launch an exporter which will ask for a new name of the file to be specified. Once it has saved the model in the APD extension it is possible to open in APD.

10.5 Appendix 5

Aspen Plus Dynamics Controller Setup

To add a controller to the PFD in APD you add an object from the 'Controls' or 'Controls 2' tabs in the Dynamics library. Figure 68 below shows these libraries; the 'Controls' library has conventional controller operators while 'Controls 2' has advanced control units.

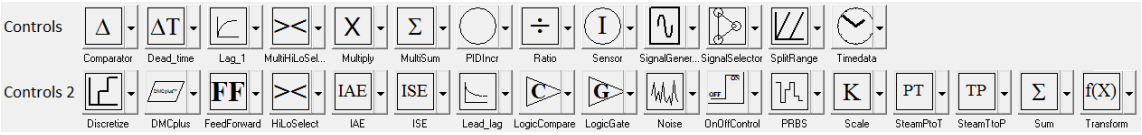

Figure 68: The Controls and Controls 2 Libraries in APD.

To add PID control to the model, the PIDIncr from the 'Controls' library is added to the PFD near the unit operation it will control. Then by selecting the control signal from the list of streams, see Figure 69, the available control input and outputs will be available on the PFD, as shown in Figure 70.

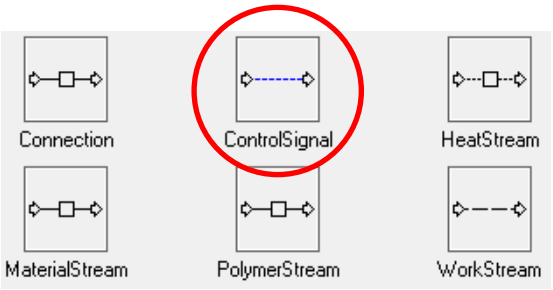

Figure 69: Different Streams Available in APD.

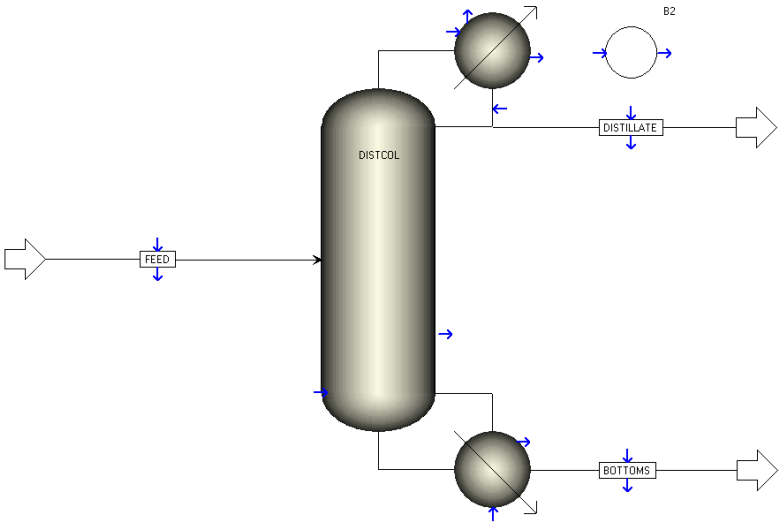

Figure 70: Available Input and Output Ports for the Control Signal.

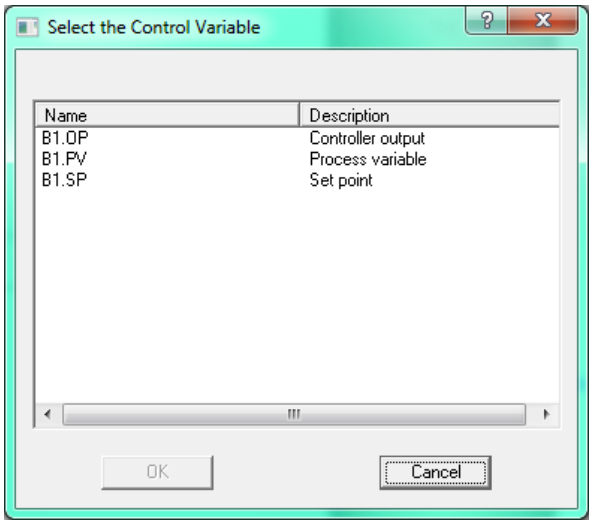

Figure 71: Output Port Variable Selection of Control B1.

The naming convention for controller units is Bn, where n is increased in increments of 1 as more controllers are added to the PFD. When you connect the control signal to the output of the controller it will prompt the user to select which variable is the MV. This can be seen in Figure 71 where the option of the MV, PV and SP are available. AspenTech uses the naming convention OP for the controller output variable instead of the MV as in this paper; these are interchangeable and will be referred to as MV throughout this paper.

Once the MV is selected on the controller is can be connected to an input port of which variable will be the controlled. Note that if you select a stream or unit operator which has only one variable it will automatically select this as the MV. If you connect to a unit operation which has multiple variables it will prompt the user to select which variable will be the MV. Figure 72 portrays this with the long list of available variables in the 'Radfrac' unit operator.

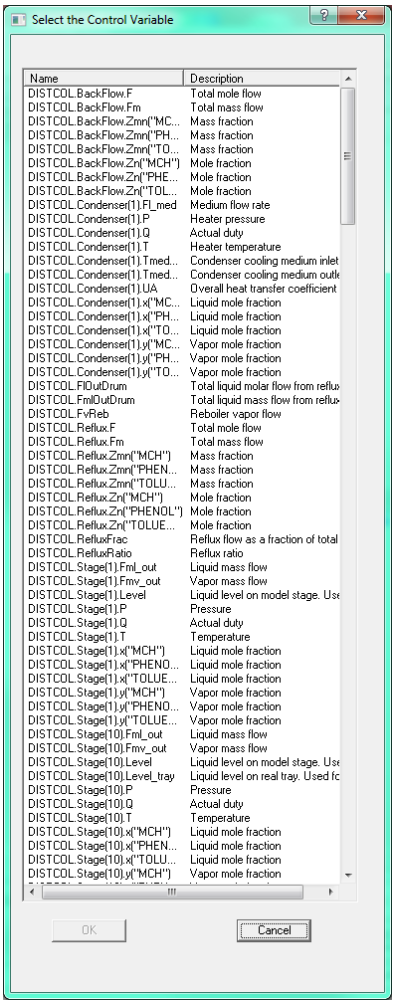

Figure 72: Available MV for the 'Radfrac' Distillation Column in APD.

Once this is completed, the input port to the controller must be connected to the PV in a similar fashion. Note that if you have already connected the MV to the control unit it will not provide that option when connecting to the

available port. This procedure is identical when connecting to the DMCplus unit operator from the 'Controls 2' library. However the DMCplus model must be loaded into the model from Aspen APC.

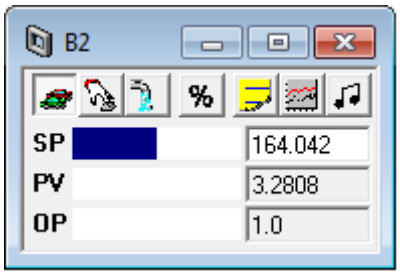

Figure 74: Controller Overview in APD.

Once all controllers are set up they must be tuned. If you double click on a controller it will launch the controller overview window, as in Figure 74. The first page will display the standard controller information: SP; PV; and MV. The first two buttons, from the left-hand side change the controller between automatic and manual modes. The third button is used if cascade control is being utilised in the model while the fourth simple changes the display from units to percentages. The last three buttons however are used to configure, plot and tune the controller respectively.

The configuration panel, see Figure 73, allows the actual PID parameters to be entered. This panel also includes operating limitations on the PV and MV, filtering and which controller algorithm to use. APD has 6 PID algorithms available; these are shown in Figure 75.

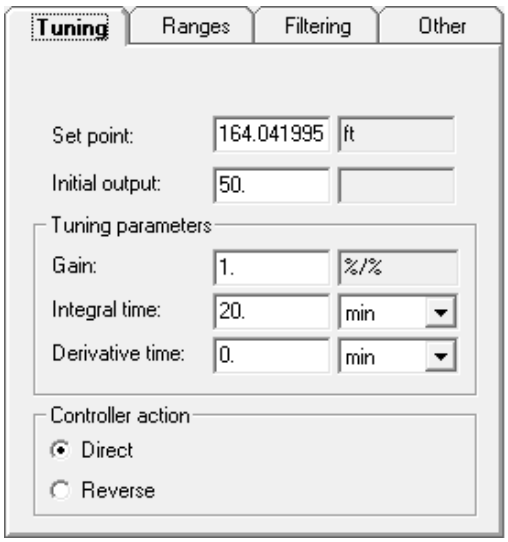

Figure 73: Controller Configuration Panel in APD.

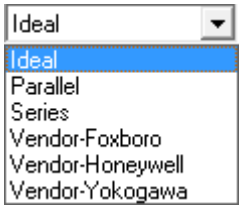

Figure 75: PID Algorithms.

Finally the Tune button allows two tuning methods to be employed:

- Open Loop Approximate Model Tuning; or
- Closed Loop Relay Tuning, or *Auto-Tune Variation* (ATV).

To do so the system is run with no changes implemented from steady-state. Then inside the tuning panel a test started. Figure 76 shows the two types of methods which can be employed and the step amplitude or relay amplitude depending on the method selected. Once the test has been operating for an extended period of time it can be finished by pressing 'Finish test'. This will cause the calculated loop characteristics to be displayed and the tuning parameters to be determined.

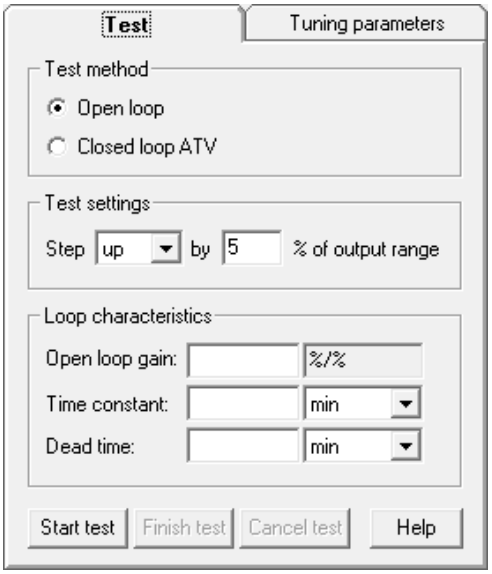

Figure 76: Controller Tuning Panel in APD.

Depending on the tuning method used will depend on the options available for the tuning parameters. Approximate model tuning rules available for PI and PID are:

- Ziegler-Nichols;
- Cohen-Coon;
- IMC;
- IAE;
- ISE; or
- ITAE.

The stability margin tuning rules available for PI only with closed loop relay tuning are:

- Ziegler-Nichols; or
- Tyreus-Luyben.

Once the controllers have been tuned and the user is satisfied it is possible to now run the dynamic simulation. Note that it is good practice to test the simulation after controllers have been made to ensure the system is still operating at its steady state conditions.

10.6 Appendix 6

Aspen Plus Dynamics Automation Tasks

In order to create scenarios within APD the user can either manual change variables or create automatic tasks. These tasks are created by selecting the 'New Task' button located in the ribbon, see Figure 77 below.

Figure 77: Creation Tabs in APD Ribbon.

This prompts the user to name the task, Figure 78 shows the popup, before showing the task creation script. These scripts are written in *Formula Translation* (Fortran).

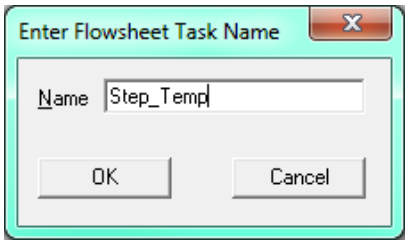

Figure 78: New Task in APD.

```
Task Step_Temp // <Trigger>
// event driven tasks, <Trigger> can be one of:
//   Runs At <time>           e.g. Runs At 2.5 or
//   Runs When <condition>      e.g. Runs When b1.y >= 0.6 or
//   Runs Once When <condition> e.g. Runs Once When b1.y >= 0.6
// Ramp (<variable>, <final value>, <duration>, <type>);
// SRamp(<variable>, <final value>, <duration>, <type>);
// Wait For <condition> e.g. when b1.y < 0.6;
// (Use Wait For to stop the task firing again once trigger condition has been met)
End
```

Using this it is possible to manipulate any variables in the PFD and create multiple tasks for disturbance or set point changes. An example of stepping the temperature in the feed stream

by 15°F at time 450 is provided below. Note the units are not specified in this script and it will use the default units of the current simulation.

```
Task Step_Temp Runs When Time == 450
Streams("FEED").T: 225; // Changes the temperature of feed to 225 F
End
```

10.7 Appendix 7

Advanced Process Control

AspenTech's APC provides two options for advanced control of processes and plants:

- Aspen DMCplus; and
- Aspen Control Platform.

Aspen DMCplus is the leading multivariable MPC in industry. It has been applied in industry to refining, chemicals and petrochemical processing. It utilises a set of desktop tools for controller design and simulation and an online component for controller implementation. Within the desktop suite there are three components.

- DMCplus Model is used to allow system identification, utilising multiple identification algorithms;
- DMCplus Build makes the control configuration through the use of a configuration wizard; and
- DMCplus Simulate allows evaluation and testing of controller performance against model inaccuracies and noise.

This software package is beneficial as it provides a useful and simple tool for constructing and testing MPC. Furthermore, DMCplus Online suite is a package which enables the controllers to

be connected online and includes input validation, steady state calculations and dynamic move calculations. To connect to a field instrument the control is set up through DMCplus Connect.

The second main feature of APC is Aspen Control Platform. This environment provides a single location for the user to build, test and deploy controllers. Included in this is the servers, applications as well as data collection and historian. It enables the entire APC application to be managed through one program including tracking of controller and plant data in real time. Furthermore, as with all MPC, the ability to predict and optimise controllers is performed within the software and the direct implementation removes the need for controllers to be placed in field. The controllers built in Aspen Control Platform are capable of handling nonlinear processes and allows any of AspenTech's system identification algorithms to be used. These are:

- Finite Impulse Response;
- Linear MIMO State Space; and
- Nonlinear MIMO State Space.

Within these there are a few more additional extras. Sequence Control integrates with the real time database to deliver information to process control systems and operators. Sequence Control Manager enables the end user to create complex transition strategies based on sophisticated logic and rules. On top of this standard Key Perfromance Indicators (KPIs) allow detection of aging equipment and changing economic conditions. These KPIs are determined online in real time. Furthermore, the package allows the controllers to run online simulation scenarios which can be used to determine why an event occurred or the best corrective action to take. This means the controllers have the ability to access historical data and adapt the model based on the plants regression over time.

10.8 Appendix 8

MATLAB ActiveX Automation

In order to create the COM automation server in MATLAB the inbuilt function 'actxserver' is used. The programmatic identifier for the COM server is entered and the output of the function is server's default interface. To access the entire AspenTech software family the follow identifiers are used:

- Aspen HYSYS, 'HYSYS.Application';
- Aspen Plus, 'APWN.Document'; or
- Aspen Customer Modeler, 'AMSimulation.Control'.

Note that within the Aspen Customer Modeler DLL the following programs are called (AspenTech, 2000b):

- Aspen Plus Dynamics 'AD Application';
- Aspen Customer Modeler 'ACM Application';
- Aspin Adsim 'ADS Application'; or
- Aspen Chromatography 'ACH Application'.

Once the server is created the properties and methods can be determined from the COM object using the 'get' and 'invoke' functions. Figure 79 displays a capture of the MATLAB Command Window displaying the result of the 'invoke' command on the HYSYS COM.

```
>> invoke(HYS)
    OpenStorage = Variant OpenStorage(handle, int32)
    GetUserVariable = handle GetUserVariable(handle, string)
    CreateUserVariable = handle CreateUserVariable(handle, string, string, UserVarType_enum, UnitConversionType_enum, int32)
    Help = void Help(handle, Variant(Optional))
    Quit = void Quit(handle)
    BindToUniqueID = handle BindToUniqueID(handle, string)
    Trace = void Trace(handle, string, bool)
    CommandProtectionFor = handle CommandProtectionFor(handle, string, bool)
    PlayScriptRelativeTo = void PlayScriptRelativeTo(handle, handle, string)
    PlayScript = void PlayScript(handle, string)
    LoadPreferences = void LoadPreferences(handle, string)
    UseShortPhaseNames = void UseShortPhaseNames(handle, bool)
    DoEvents = void DoEvents(handle)
    SetHYSYSOTSPasscode = void SetHYSYSOTSPasscode(handle, string)
    ChangePreferencesToMinimizePopupWindows = void ChangePreferencesToMinimizePopupWindows(handle, bool)
    GetDefaultCasesPath = string GetDefaultCasesPath(handle)
    PlayScriptRelativeToWithPassword = void PlayScriptRelativeToWithPassword(handle, handle, string, string)
    PlayScriptWithPassword = void PlayScriptWithPassword(handle, string, string)
    put_VisibleWithPassword = void put_VisibleWithPassword(handle, string, bool)
```

Figure 79: Aspen HYSYS COM Server Creation.

Figure 79 provides the essential methods for automating HYSYS, this in turn makes the automation process more intuitive. As the Aspen Custom Modeler DLL provides access to four software packages the use of it is slightly different to typical COM automation servers. The server is created using actxserver, like all other applications, however it is not possible to extract the properties and methods automatically from the COM object. This is due to the fact there are four software packages available within this library.

Once the COM server is established the user must define which program within the DLL it wishes to automate. To do so 'StartRun' must be invoked and the specific program, file name and variables must be defined. If the server was created as such:

```
ACMApp = actxserver('AMSimulation.Control')
```

Then the file will be opened by using the following template.

```
invoke(ACMApp, 'StartRun', pwd, 'C:\...\File.dynf', 'Application',
NoInputs, 'InputNames', NoOutputs, 'OutputNames', Visible);
```

Where:

- ACMApp is the COM Server;
- 'StartRun' is the method being invoked;
- pwd is an inbuilt MATLAB function and returns the current working directory;
- 'C:\...\File.dynf' is the directory of the file;
- 'Application' is the specific application to launch, in this case 'AD Application';
- NoInputs is the number of inputs to be manipulated;
- 'InputNames' is the name of all the inputs separated by a backslash;
- NoOutputs is the number of outputs to be controlled;
- 'OutputNames' is the name of all outputs separated by a backslash; and
- Visible is either true or false to make the application visible on the device.

Once this line is executed the specific file will have been opened within the application and prepared for simulation.

To update the manipulated variables the method 'UpdateInputs' is used to prepare the file to accept variable changes. Then the 'SetInputValue' method overrides the input variables specified in 'InputNames'. Now following the flow chart in 5.0 ActiveX the system is run by invoking the 'Run' method. This is input as follows:

```
invoke(ACMApp, 'Run', EndTime, 'C:\...\File.dynf')
```

This will run the simulation from its current time to the time specified in EndTime. Once the simulation reaches EndTime it will stop and return control to MATLAB to continue through the script.

Once the simulation has run the output variables can be received from APD using 'GetOutputValue'. This process can be repeated until the simulation is cancelled by the user or the simulation reaches the final time period. When this occurs it is important the file is terminated by invoking the method 'Terminate' or using `Quit(ACMApp)`. Following termination the COM server must be closed by using `ACMApp.delete`.

When creating COM automation servers the initial step is to attempt to retrieve the properties and methods from the server. If this is unsuccessful then a look into further documentation is required. The Aspen Custom Modeler DLL is a special case when dealing with COM automation. As it does not provide the properties and methods through the COM server it can be difficult to execute. To overcome this the Aspen Customer Modeler Reference Guide (2005) was utilised. This guide outlines the available properties, methods and syntax to successfully establish automation through the Aspen Custom Modeler DLL.

10.9 Appendix 9

LabVIEW ActiveX Automation

To communicate to a software package from LabVIEW, using ActiveX, the ActiveX class must be selected. This is the library which will be referenced to invoke methods and properties over the COM. Once the Automation Open block is placed on the LabVIEW Block Diagram the input refnum can be specified by right-clicking and browsing the available DLL as shown in Figure 80. Once selected, this will launch the selection tool displayed in Figure 81.

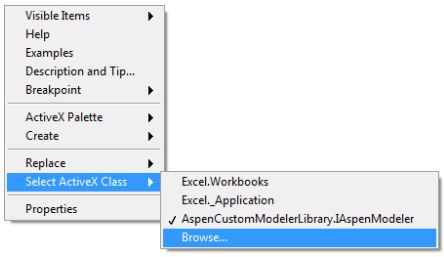

Figure 80: LabVIEW ActiveX Selection.

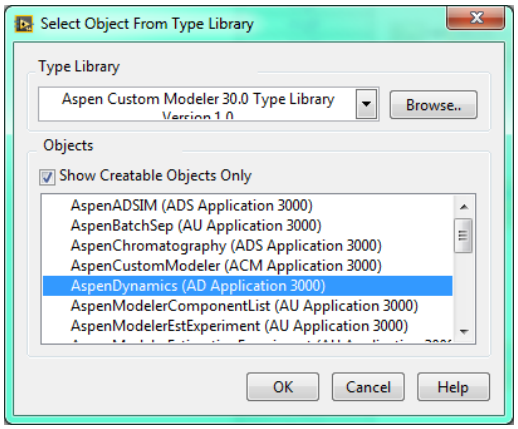

Figure 81: ActiveX Class Selection Tool in LabVIEW.

Initially a connection to APD was attempted. Aspen Custom Modeler 30.0 DLL was elected from the list of available libraries then 'AD Application' was nominated from the objects. With the COM now open the reference stream can be connected to the property and invoke nodes. As detailed in BODY this method was not successful so connection to Excel, through 'Excel.Application', was used.

Figure 82 displays the block connection to open a COM server from LabVIEW to Excel and provides the option to make the Excel application visible or invisible during operation. During troubleshooting and testing it is best to make the application visible as it can be easier to

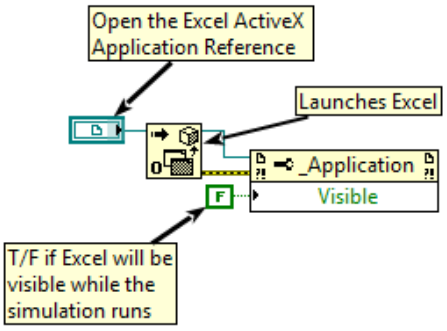

Figure 82: Opening Excel COM Object in LabVIEW.

confirm operation of the server. Once the server is established correctly it can be made invisible so the end user only deals with one software package.

In order to ensure the correct sequence of data transfer, simulation and data logging between LabVIEW, Excel and APD, Excel macros will be utilised. With the COM active it is possible to open the workbook and execute the macros within. Figure 83 below shows the infrastructure required to achieve this. The specific Excel document to open is specified from the Front Panel VI and concatenated with the default directory and macro-enabled workbook extension. Once the workbook is open the initial macro was be executed to create a server to APD from Excel. This entire sequence is completed before entering a while loop as the COM servers do not need to be created multiple times and opening and closing the server every loop introduces a significant delay in operation. The macro to open communication between Excel and APD, called 'OpenAspen', is found in 10.11 Appendix 11.

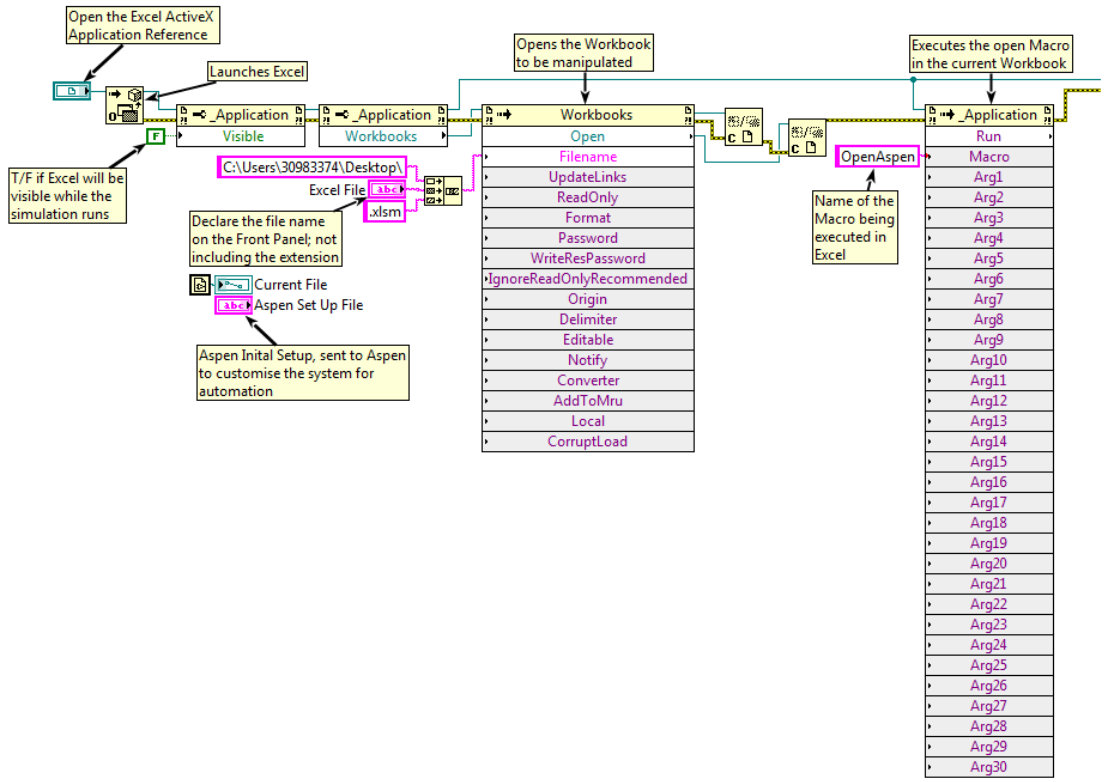

Figure 83: Initial Setup of Excel COM in LabVIEW.

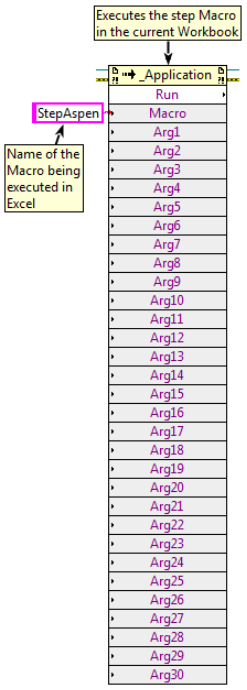

Figure 84: Executing a VBA Macro in LabVIEW.

With communication between LabVIEW to Excel and Excel to APD established the controller action will be calculated inside a while loop and sent to APD every iteration. The controller algorithms are not limited by the communication and can be coded in a number of ways. These control schemes are discussed in detail in 2.2 Process Control. Each loop the control error is calculated using Equation 2 on page 8 and fed to the control algorithms. This MV is sent to Excel by executing the macro 'StepAspen' which can be found in 10.12 Appendix 12. This process will extract the MVs from LabVIEW for all the control loops, log them in Excel and update APD. Once the variables have been updated in APD the simulation is stepped one time unit, which mirrors the loop time in LabVIEW. Then the PVs retrieved and logged in Excel before being updated in LabVIEW. Excel undertakes the bulk of this operation. LabVIEW only executes the macro remotely and waits until the PVs are updated for the given MVs.

This process will continue until the user stops the simulation. This is achieved by pressing the STOP button located on the Front Panel and triggers the closure of the COM servers. When the while loop is departed the 'CloseAspen' macro, found in 10.13 Appendix 13 is executed. This macro will close APD then the COM from Excel to APD. Once this is completed, functionality will return to LabVIEW and it will close Excel then the Excel application COM. As stated previously in 5.0 ActiveX, it is important to close the COM servers as unexpected software behaviour can result.

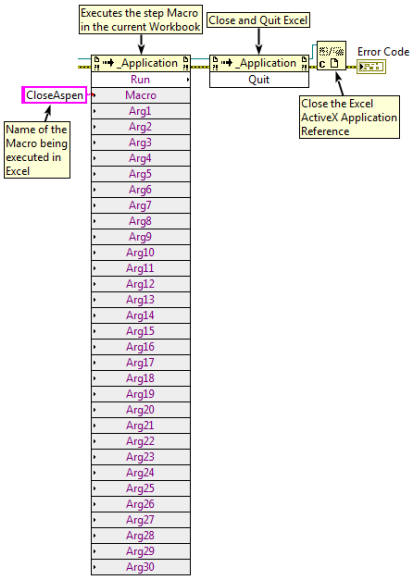

Figure 85: Closing the COM Using ActiveX Blocks in LabVIEW.

10.10 Appendix 10

Excel ActiveX Automation

Before any data can be communicated between LabVIEW and Excel or APD and Excel the COM servers must be established. First an ActiveX object is created for LabVIEW using the handle 'LabVIEW.Application'. Once this is completed the specific LabVIEW document can be set as the reference.

```
' Establish Communication to LabVIEW
Set LabVIEWApp = CreateObject("LabVIEW.Application")
Set LabVIEWDocument = LabVIEWApp.GetVIReference("C:\Users\30983374\Desktop\NIMacro.vi")
```

Figure 86: Macro Extract to Open COM with LabVIEW and set the Reference Document.

As the LabVIEW document is already open there is no need to reopen this file, however the COM does need to know what file to reference. Once this is complete it is possible to extract information from LabVIEW using GetControlValue and manipulate variables with SetControlValue. This specific method of communication requires the control values to be specified in LabVIEW. In the top right corner of LabVIEW is the tool which connects to each variable which can be either manipulated or read. This is configured as shown in Figure 87 with the MV, DV, PV and system information assigned to a free port. It does not matter if the variable is an indicator or control in LabVIEW, it is assigned to a port the same way. With the variables connected it is possible to call them over the COM using the syntax shown in Figure 88.

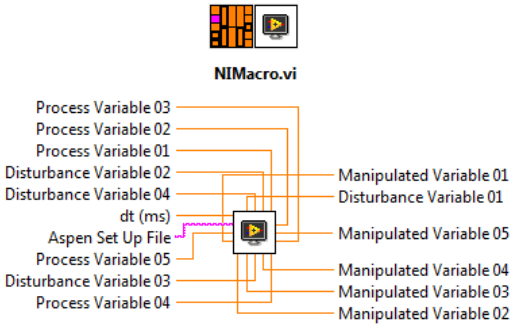

Figure 87: Control Values Available in LabVIEW.

```
LabVIEWDocument.GetControlValue("Name of Control Value")
LabVIEWDocument.SetControlValue("Name of Control Value", Value)
```

Figure 88: Syntax to Send and Receive Data from LabVIEW over ActiveX.

```
' Determine the Set up information
' Seperate the string at every ' and create an array
SetUp = Split(LabVIEWDocument.GetControlValue(ParamNames(1)), ",")
' Remove the spaces and string quotations
For i = LBound(SetUp) To UBound(SetUp)
    If i = 1 Then
        ' Needed to remove the additional apostrophes in
        ProgramID = Split(SetUp(i), "'")
        SetUp(i) = ProgramID(1)
    Else
        SetUp(i) = Replace(SetUp(i), " ", "")
    End If
Next i
```

Figure 89: Code to Retrieve the Aspen System Setup Variables in VBA.

Now the ActiveX object for APD must be created and the file specified. This uses a similar convention as shown in Figure 86 however the handle used is 'AD Application'. Once established the APD file must be opened and system settings customised. As the idea was to create LabVIEW as the main program for the end user the settings from APD are defined in LabVIEW and sent to Excel during the first macro. This string is called "Aspen Set Up File" and is split using the code found in Figure 89. This information includes:

- The file name to open;
- Which AspenTech product to launch;
- The input variables;
- The output variables;
- If the Aspen software should be visible; and
- If Excel should clear the data from the workbook.

Given this information it is possible to open the ActiveX object and set the simulation options of the Aspen software package. Figure 90 displays the code required to: open the COM; make the server visible; change the run mode to dynamic; set the time interval; and sent the time units.

```
' Set up the ASPEN product for simulation and open the connection
' Create COM to ASPEN Product
Set ACMApp = CreateObject(SetUp(1))
' Make the product visible
If SetUp(8) = "True" Then
    ACMApp.Visible = True
Else
    ACMApp.Visible = False
End If
' Open specific file
Set ACMDocument = ACMApp.opendocument(SetUp(0))
Set ACMSimulation = ACMApp.simulation
' Change run mode to Dynamic
ACMSimulation.runmode = "Dynamic"
' Extract the time from LabVIEW then change the step time in ASPEN to match
ACMSimulation.communicationinterval = LabVIEWDocument.GetControlValue(ParamNames(0)) / 1000
' Check the units in ASPEN and change to seconds if needed
If ACMSimulation.Options.TimeSettings.CommunicationUnits <> "Seconds" Then
    ACMSimulation.Options.TimeSettings.CommunicationUnits = "Seconds"
End If
```

Figure 90: Code to Open a COM with APD in VBA.

10.11 Appendix 11 shows the macro associated with the opening and initial setup of APD. With the setup completed the next macro covers the communication of data. This macro, called 'StepAspen', will be executed every loop iteration in LabVIEW. Initially it will retrieve the MVs and DVs using GetControlValue. These variables will then be logged into their respective columns in the Excel workbook. Once they are logged Excel will send them to Aspen using the following convention:

```
ACMApp.Simulation.Flowsheet.STREAMS("NAME").TYPE.Value("UNITS")
```

This is used for both updating and extracting variables in APD thorough ActiveX as shown below in Figure 91.

```
' Sends the value from Cell(n, i) to APD as the Flow rate of the Distillate in lb/hr units
ACMApp.Simulation.Flowsheet.STREAMS("DISTIL").FMR.Value("lb/hr") = Cells(CurrentRow, i).Value
' Receives the level of stage 1 in the distillation column in ft and save the value in Cell(n, i + 1)
Cells(CurrentRow, i + 1).Value = ACMApp.Simulation.Flowsheet.Blocks("DISTILCOL").Stage(1).Level.Value("ft")
```

Figure 91: VBA Code for Sending and Receiving Data in Excel to APD.

There are two operators which can be called using this syntax, streams and blocks. Then the name of the operator as displayed in APD must be entered. The variables being manipulated can be found through the Aspen Model Explorer, as shown in Figure 92. By comparing the

naming convention specified in Figure 91 to the tree structure in Figure 92 it can be concluded that the ActiveX connection directly follows the model explorer tree structure to manipulate the variable specified. Each period separates a new level on the tree.

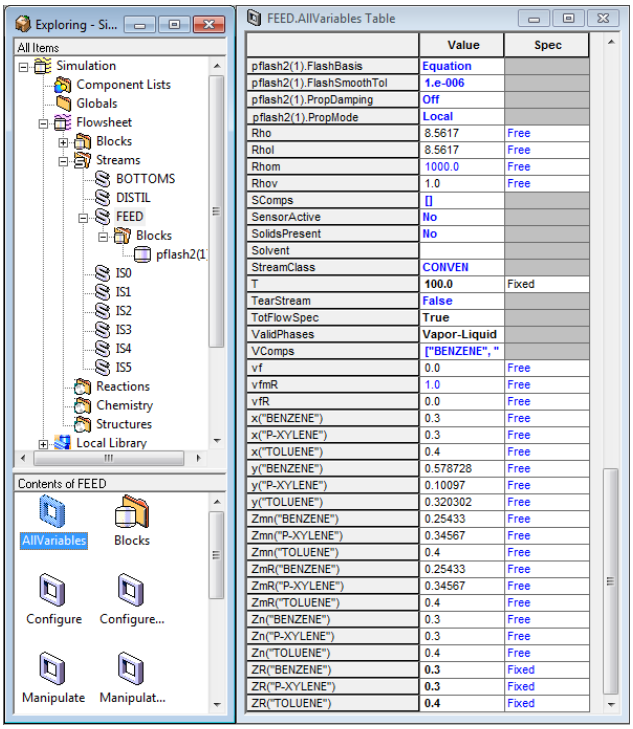

Figure 92: Model Explorer in APD.

Once the MVs are updated in APD the simulation is stepped by one time period, using ACMApp.Simulation.Step, before retrieving the PVs from APD and saving the workbook. This macro can be found in 10.12 Appendix 12. This process will continue each iteration until the STOP button is pressed in LabVIEW and the 'CloseAspen' macro is called. This final macro closes the APD document followed by closing the COM server and saving the workbook. Once this macro has completed it returns control to LabVIEW which closes the workbook and Excel COM server. This macro is documented in 10.13 Appendix 13. The final product in Excel from this process is provided in Figure 93.

TIME	INPUTS			DISTURBANCES	OUTPUTS		
ACMSimulation.Time	Streams("DISTILLATE"). FMR.Value	Blocks("DISTCOL"). Condenser(1).FI_med.Value	Streams("BOTTOMS"). FMR.Value	Streams("FEED"). T.Value	Blocks("DISTILCOL"). Stage(1).Level.Value	Blocks("DISTILCOL"). Stage(1).P.Value	Blocks("DISTILCOL"). Stage(22).Level.Value
Seconds	lb/hr	lb/hr	lb/hr	F	ft	psi	ft
0	19605.0029	180249.7066	131396.3800	100.0000	1.5017	16.0000	3.2494

Figure 93: Logged Data in Excel from Automation with LabVIEW and APD.

10.11 Appendix 11

Excel Open Aspen Macro

The following macro provides the basis for creating a COM object to LabVIEW and APD. This is used to establish the connection and setup the parameters within APD.

```
' Global Variables
Dim ACMApp As Object
Dim ACMDocument As Object
Dim ACMSimulation As Object
Dim LabVIEWApp As Object
Dim LabVIEWDocument As Object
Dim ParamNames(15) As String
Dim ParamValues(15) As Variant
Dim SetUp() As String

Sub OpenAspen()
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' Written by Joshua M Eggins, Murdoch University 28/09/2015
    ' This sub will start one of the ASPEN products and format the workbook to ensure it is ready
for the
    ' Declared variables
    Dim Inputs() As String
    Dim Disturbances() As String
    Dim Outputs() As String
    Dim CurrentRow As Long
    Dim CurrentColumn As Long
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' Extract the data from LabVIEW to initiate the software
    ' SetUp(0): File and directory
    ' SetUp(1): Application to launch
    ' SetUp(2): Number of inputs
    ' SetUp(4): Number of disturbances
    ' SetUp(6): Number of outputs
    ' SetUp(8): Visible? Y/N
    ' SetUp(9): Erase all data? Y/N
    ' Inputs(0): Input 1
    ' Inputs(1): Input 2
    ' ...
    ' Disturbance(0): Disturbance 1
    ' Disturbance(1): Disturbance 2
    ' ...
    ' Outputs(0): Input 1
    ' Outputs(1): Input 2
    ' ...
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

```
' Turn off Screen Updates Until all Actions Completed
Application.ScreenUpdating = False
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Setup the Parameter Names
ParamNames(0) = "dt (ms)"
ParamNames(1) = "Aspen Set Up File"
ParamNames(2) = "Manipulated Variable 01"
ParamNames(3) = "Manipulated Variable 02"
ParamNames(4) = "Manipulated Variable 03"
ParamNames(5) = "Manipulated Variable 04"
ParamNames(6) = "Manipulated Variable 05"
ParamNames(7) = "Disturbance Variable 01"
ParamNames(8) = "Disturbance Variable 02"
ParamNames(9) = "Disturbance Variable 03"
ParamNames(10) = "Disturbance Variable 04"
ParamNames(11) = "Process Variable 01"
ParamNames(12) = "Process Variable 02"
ParamNames(13) = "Process Variable 03"
ParamNames(14) = "Process Variable 04"
ParamNames(15) = "Process Variable 05"
' Establish Communication to LabVIEW and send and pull information from
Set LabVIEWApp = CreateObject("LabVIEW.Application")
Set LabVIEWDocument =
LabVIEWApp.GetVIReference("C:\Users\30983374\Desktop\NIMacro.vi")
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Determine the Set up information
' Seperate the string
SetUp = Split(LabVIEWDocument.GetControlValue(ParamNames(1)), ",")
' Remove the spaces and string quotations
For i = LBound(SetUp) To UBound(SetUp)
   If i <> 1 Then
      SetUp(i) = Replace(SetUp(i), " ", "")
   Else
      ProgramID = Split(SetUp(i), """")
      SetUp(i) = ProgramID(1)
   End If
Next i
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Store the input, disturbances and outputs in arrays then seperate them to be displayed
' in the workbook with the desired units below the name of the variable
' Erase all data in workbook
If SetUp(9) = "True" Then
   Cells.Delete Shift:=xlUp
End If
' Time Variables
Cells(1, 1).Value = "TIME"
Cells(2, 1).Value = "ACMSimulation.Time"
Cells(3, 1).Value = "Seconds"
' Input Variables
Inputs = Split(SetUp(3), "\")
```

```
Cells(1, 2).Value = "INPUTS"
For k = LBound(Inputs) To UBound(Inputs)
   tempstr = Split(Inputs(k), "|")
   Cells(2, k + 2).Value = tempstr(0)
   Cells(3, k + 2).Value = tempstr(1)
Next k
' Disturbance Variables
Disturbances = Split(SetUp(5), "\")
If SetUp(4) > 0 Then
   Cells(1, k + 2).Value = "DISTURBANCES"
   For l = LBound(Disturbances) To UBound(Disturbances)
      tempstr = Split(Disturbances(l), "|")
      Cells(2, k + l + 2).Value = tempstr(0)
      Cells(3, k + l + 2).Value = tempstr(1)
   Next l
Else
   l = 0
End If
' Output Variables
Outputs = Split(SetUp(7), "\")
Cells(1, k + l + 2).Value = "OUTPUTS"
For j = LBound(Outputs) To UBound(Outputs)
   tempstr = Split(Outputs(j), "|")
   Cells(2, k + l + j + 2).Value = tempstr(0)
   Cells(3, k + l + j + 2).Value = tempstr(1)
Next j
' Change the column sizing to fit the entire directory
Columns("A:ZZ").ColumnWidth = 35
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Set up the ASPEN product for simulation and open the connection
' Create COM to ASPEN Product
Set ACMApp = CreateObject(SetUp(1))
' Make the product visible
If SetUp(8) = "True" Then
   ACMApp.Visible = True
Else
   ACMApp.Visible = False
End If
' Open specific file
Set ACMDocument = ACMApp.opendocument(SetUp(0))
Set ACMSimulation = ACMApp.simulation
' Change run mode to Dynamic
ACMSimulation.runmode = "Dynamic"
' Extract the time from LabVIEW then change the step time in ASPEN to match
ACMSimulation.communicationinterval =
LabVIEWDocument.GetControlValue(ParamNames(0)) / 1000
' Check the units in ASPEN and change to seconds if needed
If ACMSimulation.Options.TimeSettings.CommunicationUnits <> "Seconds" Then
   ACMSimulation.Options.TimeSettings.CommunicationUnits = "Seconds"
End If
```

```
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' Place a thin solid line below the last row of data, this can be used to distinquish between
data sets
    CurrentRow = Cells(Rows.Count, 1).End(xlUp).Row
    CurrentColumn = Cells(CurrentRow, Columns.Count).End(xlToLeft).Column
    With Range(Cells(CurrentRow, 1), Cells(CurrentRow,
CurrentColumn)).Borders(xlEdgeBottom)
        .LineStyle = xlContinuous
        .ColorIndex = 0
        .TintAndShade = 0
        .Weight = xlThin
    End With
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' Freeze the label panes in the first three rows to enable better analysis of data
    ActiveWindow.FreezePanes = False
    Range("A4").Select
    ActiveWindow.FreezePanes = True
    Range("A1").Select
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' Simulation is now set up and the workbook formatted
    ' Save the Workbook for next set of updates
    ThisWorkbook.Save
    ' Turn on Screen Updates
    Application.ScreenUpdating = True
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
End Sub
```

10.12 Appendix 12

Excel Step Aspen Macro

With APD set up it is possible to update the inputs, step the simulation one time period and retrieve the outputs. The following macro completes these steps and communicates with LabVIEW to retrieve the new MVs and update the PVs.

```
Sub StepAspen()
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' Written by Joshua M Eggins, Murdoch University 28/09/2015
    ' This sub will step the open ASPEN product at the given step intervals and save the
workbook
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' Determine the current row to log data to
    Dim CurrentRow As Long
    CurrentRow = ActiveSheet.Range("A" & Rows.Count).End(xlUp).Row + 1
```

```
    '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' Turn off Screen Updates Until all Actions Completed
    Application.ScreenUpdating = False
    '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' Setup the Parameter Names
    ParamNames(0) = "dt (ms)"
    ParamNames(1) = "Aspen Set Up File"
    ParamNames(2) = "Manipulated Variable 01"
    ParamNames(3) = "Manipulated Variable 02"
    ParamNames(4) = "Manipulated Variable 03"
    ParamNames(5) = "Manipulated Variable 04"
    ParamNames(6) = "Manipulated Variable 05"
    ParamNames(7) = "Disturbance Variable 01"
    ParamNames(8) = "Disturbance Variable 02"
    ParamNames(9) = "Disturbance Variable 03"
    ParamNames(10) = "Disturbance Variable 04"
    ParamNames(11) = "Process Variable 01"
    ParamNames(12) = "Process Variable 02"
    ParamNames(13) = "Process Variable 03"
    ParamNames(14) = "Process Variable 04"
    ParamNames(15) = "Process Variable 05"
    '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' Extract the time from LabVIEW then change the step time in ASPEN to match
    If ACMSimulation.communicationinterval <>
LabVIEWDocument.GetControlValue(ParamNames(0)) / 1000 Then
        ACMSimulation.communicationinterval =
LabVIEWDocument.GetControlValue(ParamNames(0)) / 1000
    End If
    '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' Inputs
    ' SetUp(2): Number of inputs
    For i = 1 To 4
        Cells(CurrentRow, i + 1).Value = LabVIEWDocument.GetControlValue(ParamNames(1 + i))
        If i = 1 Then
            ACMSimulation.Flowsheet.STREAMS("DISTILLATE").FMR.Value("lb/hr") =
Cells(CurrentRow, i + 1).Value
        ElseIf i = 2 Then
            ACMSimulation.Flowsheet.Blocks("DISTCOL").Condenser(1).Fl_med.Value("lb/hr") =
Cells(CurrentRow, i + 1).Value
        ElseIf i = 3 Then
            ACMSimulation.Flowsheet.STREAMS("BOTTOMS").FMR.Value("lb/hr") =
Cells(CurrentRow, i + 1).Value
        ElseIf i = 4 Then
            ACMSimulation.Flowsheet.STREAMS("FEED").FMR.Value("lb/hr") = Cells(CurrentRow, i +
1).Value
        ElseIf i = 5 Then
        '   Additional Input
        End If
    Next i
    i = i - 1
```

```
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' Disturbances
    ' SetUp(4): Number of disturbances
    For j = 1 To SetUp(4)
       Cells(CurrentRow, i + j + 1).Value = LabVIEWDocument.GetControlValue(ParamNames(6 +
j))
       If j = 1 Then
          ACMSimulation.Flowsheet.STREAMS("FEED").T.Value("F") = Cells(CurrentRow, i + j +
1).Value
       ElseIf i = 2 Then
       '    Additional Disturbance
       ElseIf i = 3 Then
       '    Additional Disturbance
       ElseIf i = 4 Then
       '    Additional Disturbance
       End If
    Next j
    j = j - 1
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' Update the inputs and disturbances
    ACMSimulation.Step (True)
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' Outputs
    ' SetUp(6): Number of outputs
    For k = 1 To SetUp(6)
       If k = 1 Then
          Cells(CurrentRow, i + j + k + 1).Value =
ACMSimulation.Flowsheet.Blocks("DISTCOL").Stage(1).Level.Value("ft")
       ElseIf k = 2 Then
          Cells(CurrentRow, i + j + k + 1).Value = ACMSimulation.Flowsheet.Blocks("DISTCOL
").Stage(1).P.Value("psi")
       ElseIf k = 3 Then
          Cells(CurrentRow, i + j + k + 1).Value = ACMSimulation.Flowsheet.Blocks("DISTCOL
").Stage(22).Level.Value("ft")
       ElseIf k = 4 Then
          Cells(CurrentRow, i + j + k + 1).Value =
ACMSimulation.Flowsheet.Streams("DISTILLATE").Zmn("TOLUENE").Value("kg/kg")
       ElseIf k = 5 Then
       Cells(CurrentRow, i + j + k + 1).Value =
ACMSimulation.Flowsheet.Streams("BOTTOMS").Zmn("BENZENE").Value("kg/kg")
       End If
       PVControlValueVI = LabVIEWDocument.SetControlValue(ParamNames(10 + k),
Cells(CurrentRow, i + j + k + 1).Value)
    Next k
    Cells(CurrentRow, 1).Value = ACMSimulation.Time
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' Save the Workbook for next set of updates
    ThisWorkbook.Save
    ' Turn on Screen Updates
    Application.ScreenUpdating = True
```

```
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
End Sub
```

10.13 Appendix 13

Excel Close Aspen Macro

This macro closes the open APD document and then the ActiveX object.

```
Sub CloseAspen()
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' Written by Joshua M Eggins, Murdoch University 28/09/2015
    ' This sub will close one of the ASPEN products and save the workbook
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
    ' Close the ActiveX COM
    ACMApp.Quit
    ' Save the Workbook
    ThisWorkbook.Save
    ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
End Sub
```

10.14 Appendix 14

ActiveX Co-Simulation

The following plots are the remaining ActiveX validation plots. These are simply a reference to display the correct implementation of the communication between MATLAB and APD in addition to LabVIEW, Excel and APD. 5.5 Co-Simulation displays the step of the DV in Figure 30. Figure 94 displays the level in the sump due to this disturbance change and Figure 95 the controller action to track the set point. Similarly, Figure 96 is the pressure in the condenser while being controlled by the reflux flow rate shown in Figure 97.

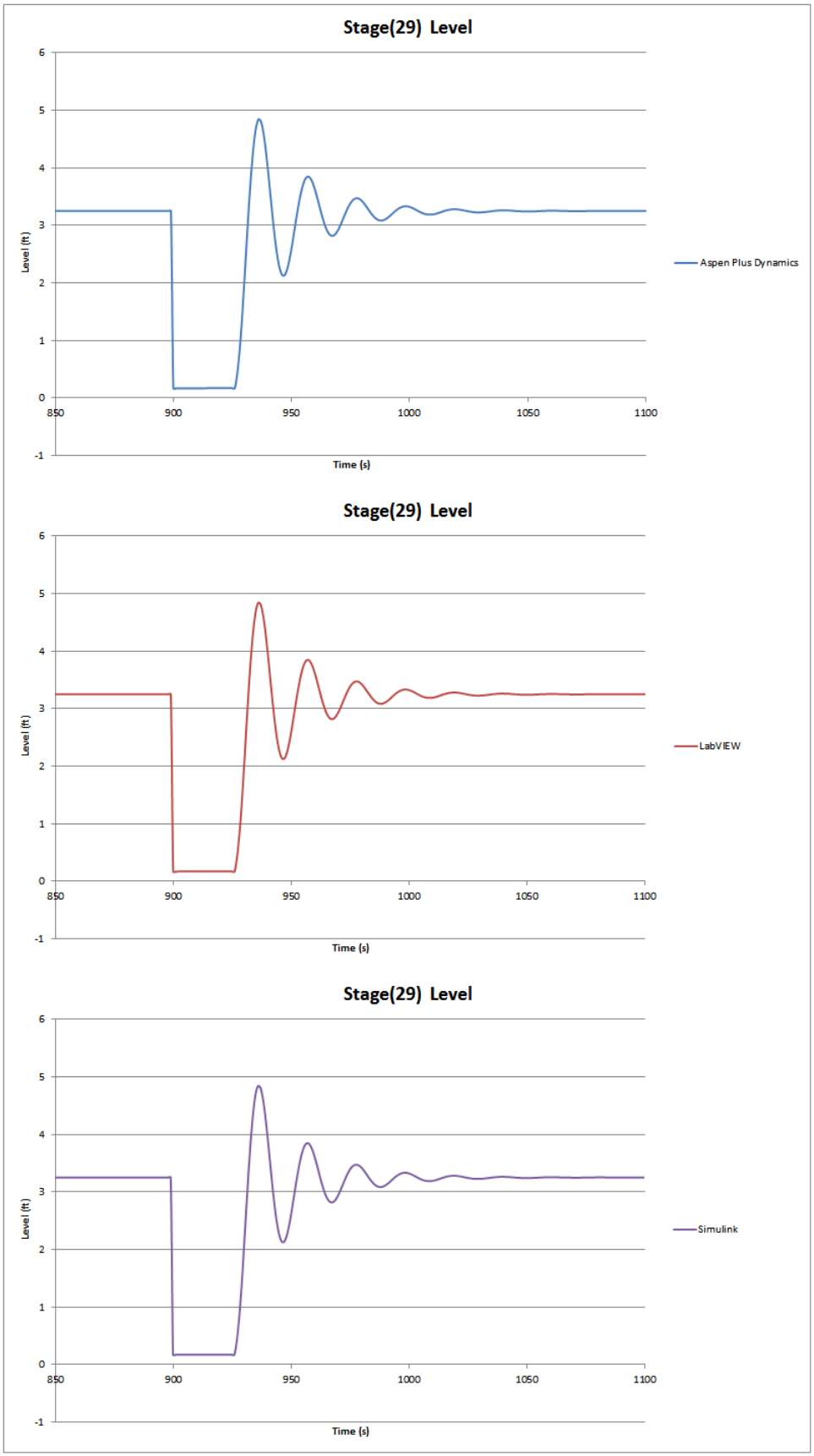

Figure 94: Plot of the Level in the Sump Controlled After a Disturbance Change.

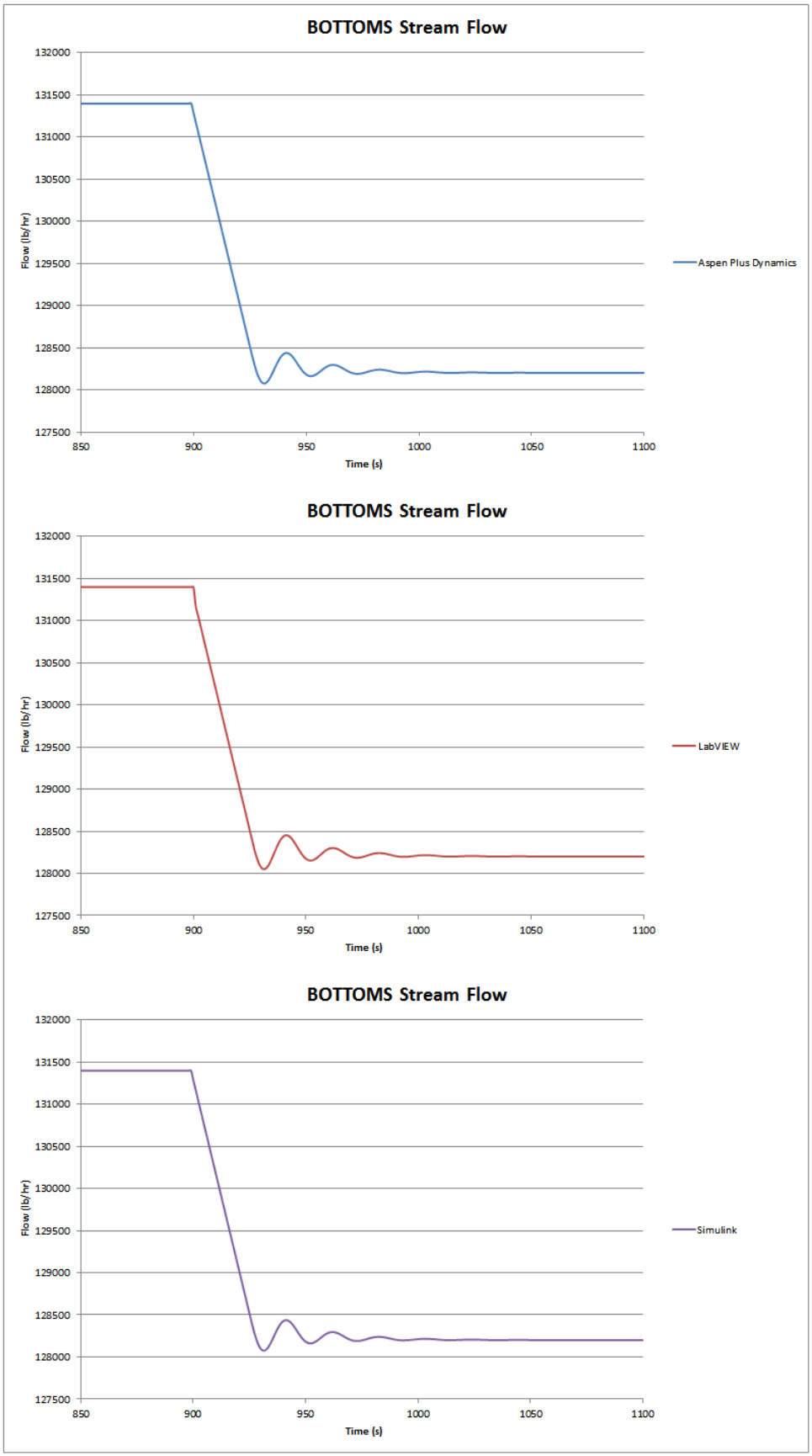

Figure 95: Plot of the Manipulated Flow Rate of the Bottoms Stream to control the Level in the Sump.

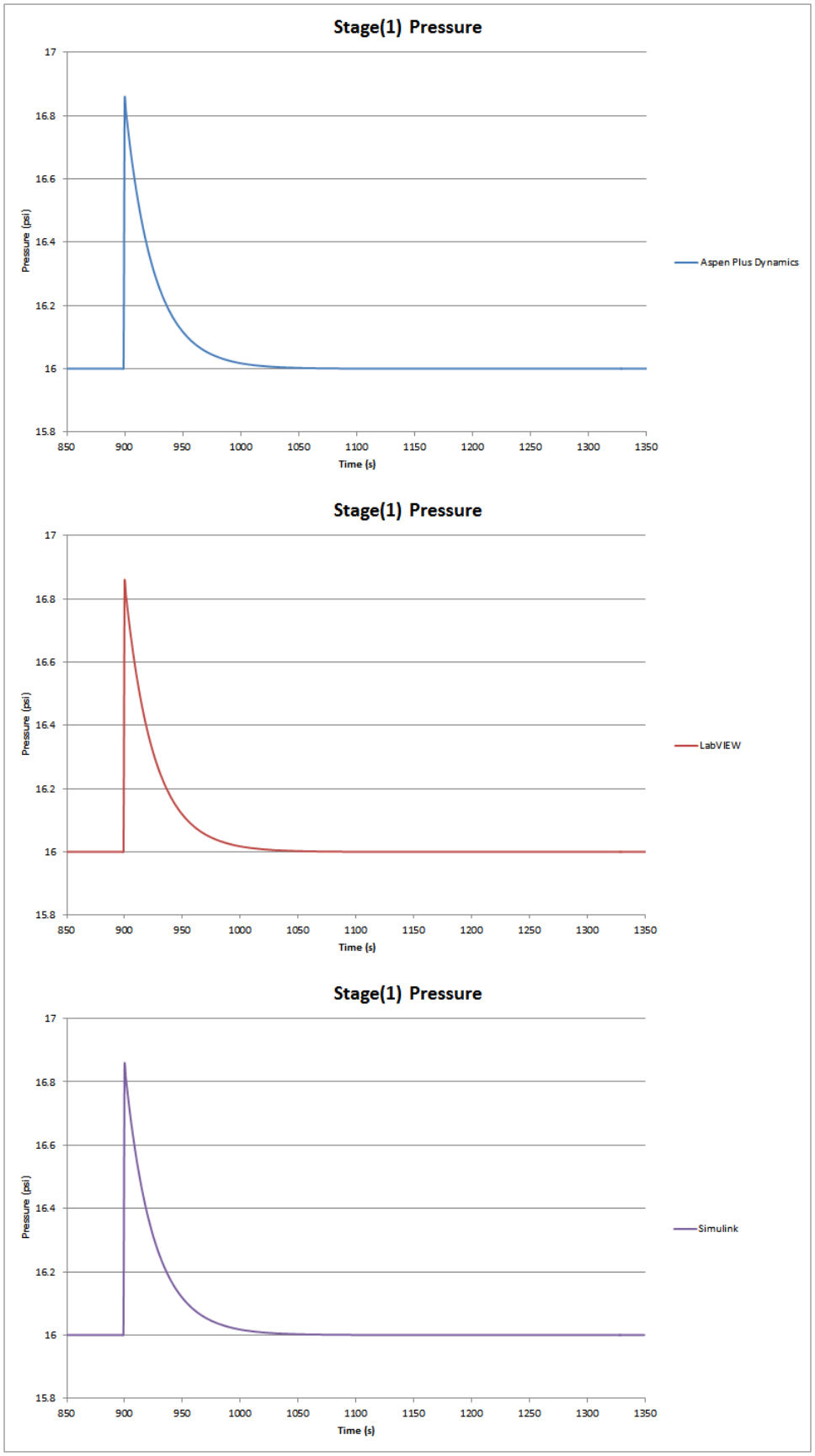

Figure 96: Plot of the Pressure in the Condenser Controlled After a Disturbance Change.

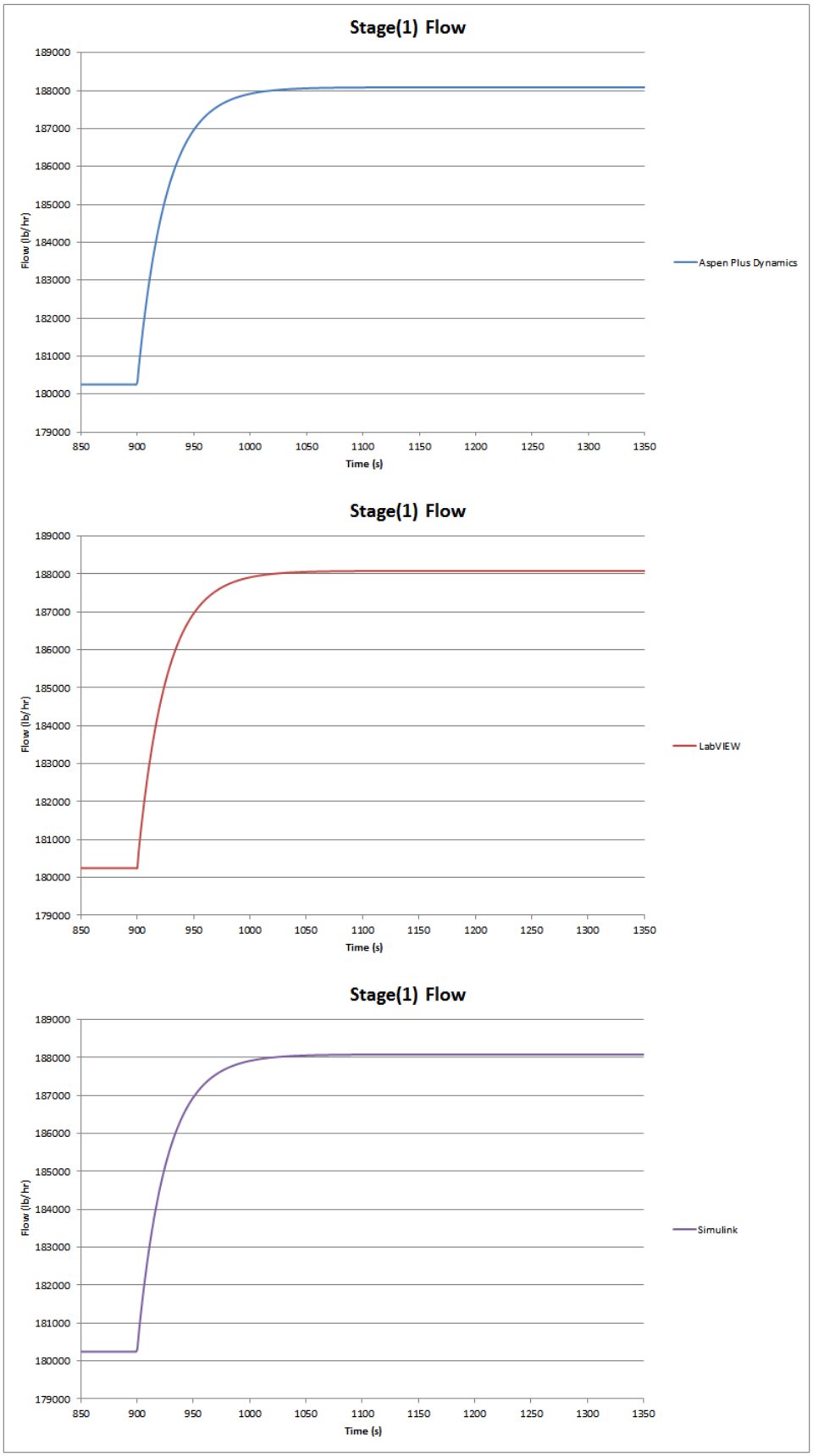

Figure 97: Plot of the Manipulated Reflux Flow Rate to Control the Condenser Pressure.

10.15 Appendix 15

MATLAB System Identification Function

In order to easily determine a system's transfer function the least squares method was employed within a user defined function. This enabled the MIMO transfer function script the capabilities to quickly determine a system's model without using too much computer memory or user time. In order to achieve this identification, the data is fed into the function. This is then separated into the input and output streams and the time of the step established. An additional feature was added which allows the user the ability to view the plot and select if any data needed to be removed from the end of the plot due to unwanted dynamics. This issue arises from the limitations being reached in capacity systems when the simulation logged data for an excessive period of time. If there is information on the plot which is not needed for the system identification the user can select it on the plot with a simple mouse click. The function will then remove all the data from that time period onwards from the modelling algorithm.

Once the data has been trimmed, it can be used to establish the transfer function. This process is only capable of SISO identification. However the MIMO transfer function modeller found in 10.17 Appendix 17 utilises this function to determine the MIMO matrix. There are two models which the function examines:

- First Order; and
- Capacity.

The first order model tries to minimise the sum of the error squared (Strang & Borre, 1997, pp. 174-176), also known as the method of least squares, to approximate the model parameters. As the model parameters are changed the sum of the error squared is calculated. If it is greater than the previous it is possible the function is heading in the wrong direction so corrections are

made. Once the function has converged on the minimum it will exit the loop. Now the capacity model is calculated using the standard linear model and the error is calculated.

Before outputting the system's transfer function, a comparison of the two models is undertaken and the model with the smallest sum of error squared is selected as the solution. This function was validated against Microsoft Excel's solver add-on and found to have no variations. The following script is the function script from MATLAB for user's reference.

```
function [mod, dt] = solverlstsq(stepdata, tout)
%% Step Data Model Builder
% Joshua M Eggins, Murdoch University, 14/09/2015
% Take the output of Simulink and determine the first order model or
linear
% capacity system of the system to be imported into any model
% based controller. Such as Dynamic Matrix Control (DMC).

%% Time Step
% Check if the model data is available
if length(stepdata) < 2
    error('No model data found. Re-simulate the model.')
end
% Set the sample time of the data
dt = tout(2) - tout(1);

%% Extract Data
% Extract the data into arrays for processing
PV = stepdata(:,1);
MV = stepdata(:,2);
Time = tout(:,1);
% Determine the steady state values and deviate the variables
PVsteady = PV(1);
MVsteady = MV(1);
PVdev = PV - PVsteady;
MVdev = MV - MVsteady;

%% Input Step Time
% Check for when the input changes
for signalposition = 1:length(MVdev) - 1
    if MVdev(signalposition + 1) ~= MVdev(signalposition)
        MVStepTime = signalposition;
    end
end
% Remove excess data pre input step
MVdevSnip = MVdev(MVStepTime+1:length(MVdev));
PVdevSnip = PVdev(MVStepTime+1:length(MVdev));
TimedevSnip = Time(MVStepTime:length(MVdev)-1)-Time(MVStepTime);
```

```
% Allow user to check for any unwanted behaviour at the end of the
plot
% that affect dynamics
% ie. Maximum or minimum reached
disp('If there is any unwanted data in the figure, click on the plot
to define the position to remove. Any information after this time will
be removed for the model prediction algorithm. If no data should be
removed pressed Enter.')
figure(10)
plot(TimedevSnip, PVdevSnip)
[t,y] = ginput(1);
close(figure(10))
% Make sure there was a selection made and if so then remove excess
from
% plot before calculating models
if t > 0
    position = find(TimedevSnip==round(t));
    MVdevSnip = MVdevSnip(1:position);
    PVdevSnip = PVdevSnip(1:position);
    TimedevSnip = TimedevSnip(1:position);
end

%% Model Prediction - First Order Model
% Determine the input step and system gain.
% Additional checks could involve checking to ensure not more than one
step
% happens over the checking of the data as this would void the error
% calculations and model.
A = MVdevSnip(length(MVdevSnip));
K = PVdevSnip(length(PVdevSnip)) / A;
% Arbitrary large error to initiate the array
SumError = 10000000;
SumError2 = 100000;
tau = 5;
direct = 1;
Tau = 0;
same = 0;
% Solve for the minimum sum of the error squared
% When the minimum is found, save tau for use once all options are
% completed.
while same < 10
    SumError3 = SumError;
    SumError = 0;
    for ii = 1:length(PVdevSnip)
        % Extract actual data and calculate the predicted
        time = (ii-1)*dt;
        ModelPredicted = A*K*(1-exp(-(time/tau)));
        ModelActual = PVdevSnip(ii);
        % Calculate the error
        Error = ModelPredicted - ModelActual;
        SumError = SumError + (Error)^2;
    end
    % If the sum of the error squared is a new minimum save the error
and
    % tau for use in next iterations
    if SumError > SumError3
        direct = direct*-1;
    end
```

```
    tau = tau + direct*0.01;
    Tau2 = Tau;
    if SumError < SumError2
        SumError2 = SumError;
        Err = SumError;
        Tau = tau;
    end
    if Tau == Tau2
        same = same + 1;
    else
        same = 0;
    end
end

%% Model Prediction - Capacity Model
% Determine the slope of the line.
finaly = PVdevSnip(length(PVdevSnip));
finalx = TimedevSnip(length(TimedevSnip));
SumErrorLin = 0;
KLin = finaly / finalx / A;
% Calculate the sum of the error squared
for ii = 1:length(PVdevSnip)
    % Extract actual data and calculate the predicted
    time = (ii-1)*dt;
    ModelPredicted = A*KLin*time;
    ModelActual = PVdevSnip(ii);
    % Calculate the error
    Error = ModelPredicted - ModelActual;
    SumErrorLin = SumErrorLin + (Error)^2;
End

%% Result
% Plot the result to confirm the operation of the model predictor in
% addition to display the results. This can be used as a reference
against
% user predicted models.
% Determine which model gave a better response and output data
if SumErrorLin < SumError
    fprintf('Linear Model Found:  A = %0.3f  K = %0.3f  dt =
%0.3f\nThe sum of the squared error is %0.4f\n', A, KLin, dt,
SumErrorLin)
    %plot(TimedevSnip, PVdevSnip, 'b', TimedevSnip,
A*KLin*TimedevSnip, 'r')
    mod = tf(KLin,[1 0]);
else
    fprintf('First Order Model Found:  A = %0.3f  K = %0.3f  Tau =
%0.3f  dt = %0.3f\nThe sum of the squared error is %0.4f\n', A, K,
Tau, dt, Err)
    %plot(TimedevSnip, PVdevSnip, 'b', TimedevSnip, A*K*(1-exp(-
(TimedevSnip/Tau))), 'r')
    mod = tf(K,[Tau 1]);
end
end
```

10.16 Appendix 16

MATLAB Automatic MIMO TF Modeller Script

In order to use the MPC Toolbox in MATLAB and Simulink a model must be obtained. The issue associated with integrating MATLAB to APD is the MPC Toolbox is not able to automatically linearise the plant and determine the plant model. To overcome this issue a script has been written which performs unit steps on the inputs, in the open loop Simulink file, one by one and determines the transfer function for each output to the inputs. These transfer functions are then saved into a MIMO transfer function and input into the MPC Toolbox.

The user can define the Simulink file to open, the final time in the simulation and the time of the input step. It will then do a single iteration simulation of the Simulink file to determine how many inputs and outputs are connected to the sinks in addition to the sampling time of the simulation. Once this is completed it will reset all the inputs back to zero and all step times back to zero before stepping the first input by the amplitude and at the time specified at the start of the program. Once the simulation is completed for the first input, the input and the first output is sent to the model predictor. This is performed within a loop in order to find a model for all the outputs against that input.

Once this is completed, the input is reset to zero and the second input is stepped. This process will repeat until the final input has been stepped and the entire MIMO transfer function matrix constructed. It is possible to then import the variable 'MIMOPlantModel' into the MPC Toolbox and design the advanced controller against this model. The full script, with comments, is given below. The function to identify the system model is found in 10.15 Appendix 15.

```
%% Solve for the MIMO transfer function models of a plant
```

```
% Joshua M Eggins, Murdoch University, 13/10/2015
% This dynamic predictor can be used in conjunction with Simulink to
% determine the MIMO transfer function matrix for the plant.
% This script is dynamic and will work for any plant provided the
inputs
% are connected to a 'to workspace' sink labelled 'inputs' and the
outputs
% connected to a separate sink labels 'outputs'.
clear
clc

%% Set the name of the Simulink file to open
simulinkFile = 'FileNameOpenLoop';
finalTime = 3000;
stepTime = 500;
stepAmplitude = 1;

%% Initiate the Simulation
% Run the simulation as a test to determine the data to be examined
and how
% many of each variable exists
set_param(simulinkFile, 'StopTime', '1');
simOutput = sim(simulinkFile, 'ReturnWorkspaceOutputs', 'on');
set_param(simulinkFile, 'StopTime', num2str(finalTime));
inputs = simOutput.get('inputs');
noinputs = size(inputs, 2);
outputs = simOutput.get('outputs');
nooutputs = size(outputs, 2);
tout = simOutput.get('tout');
timestep = tout(2) - tout(1);
tout = [0:timestep:finalTime]';

%% Initiate the Input Array
% Empty array to input the sample time for each transfer function
dt = zeros(nooutputs, noinputs);
inputname = cell(noinputs, 1);
for position = 1:noinputs
    % Determines and store the name of each input step block to enable
    % dynamic analysis
    inputname{position} = strcat(simulinkFile, strcat('/Input',
num2str(position)));
    % Reset all the step data to 0, step time to 10
    set_param(inputname{position},'Before','0');
    set_param(inputname{position},'After','0');
    set_param(inputname{position},'Time',num2str(stepTime));
end

%% Model Predictions
% Step each input separately and determine the model of the response.
% Call the function solverlstsq to determine the transfer function
model
% These models will be stored within a MIMO transfer function matrix.
for input = 1:1%noinputs
    % Step the step amplitude
    set_param(inputname{input},'After',num2str(stepAmplitude));
    % Rerun the simulation and extracts the outputs
    simOutput = sim(simulinkFile, 'ReturnWorkspaceOutputs', 'on');
    inputs = simOutput.get('inputs');
```

```
    outputs = simOutput.get('outputs');
    for output = 1:nooutputs
        % Put the raw data into an array and send to the function
        rawdata=[outputs(:, output), inputs(:, input)];
        [MIMOPlantModel(output, input), dt(output, input)] =
solverlstsq(rawdata, tout);
    end
    set_param(inputname{input},'After','0');
end
% Step each input seperatly and
MIMOPlantModel
```

10.17 Appendix 17

MATLAB MPC Toolbox

A major issue with MIMO control schemes is loop coupling. Traditional PI feedback loops do not have the capabilities to deal with this and thus advanced control implemented. MathWorks' MPC Toolbox optimises the controller action for MIMO systems subject to defined variable constraints. This documentation will outline the necessary steps for implementing MPC control in Simulink.

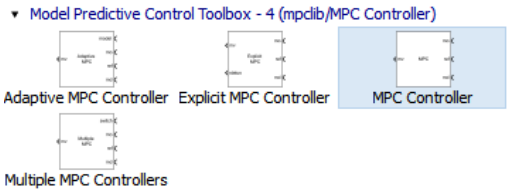

Figure 98: MPC Toolbox Controller Blocks in Simulink.

Initially the MPC Controller found in Figure 98 is added to the Simulink model. The PV, SP and DVs are connected to the controller block. Although the DVs are not essential it is recommended to connect them into the controller even if they will be unmeasured. Within the estimate window it is possible to allocate what type of variable these DVs are, measured or unmeasured. Once the inputs are connected the output, MV, should be connected to the plant.

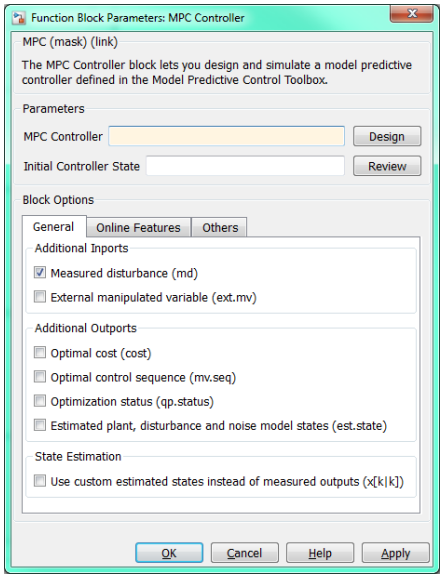

Figure 99: MPC Controller Mask in Simulink.

By double clicking on the MPC block Simulink launches the MPC controller mask shown in Figure 99. Within this window it is possible to allocate ports for additional inputs and outputs as well as controller parameters, such as:

- Constraints;
- Weightings; and
- Sample time.

By pressing Design the controller will ask the user to specify the number of MVs and PVs to be controlled in addition to the sample time of the data acquisition. Once these have been entered, as shown in Figure 100, the toolbox will analyse the plant to determine the operating points and linearise around those points. Figure 101 is the window prompt when performing these tasks. Once it has linearised the plant the MPC toolbox will provide the model in the following representations:

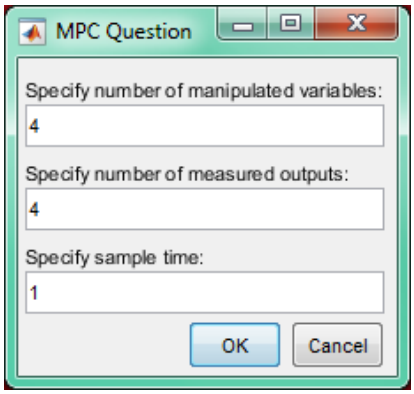

Figure 100: Variable Specifications in Simulink.

- SS;
- TF; and
- Zero-pole gain.

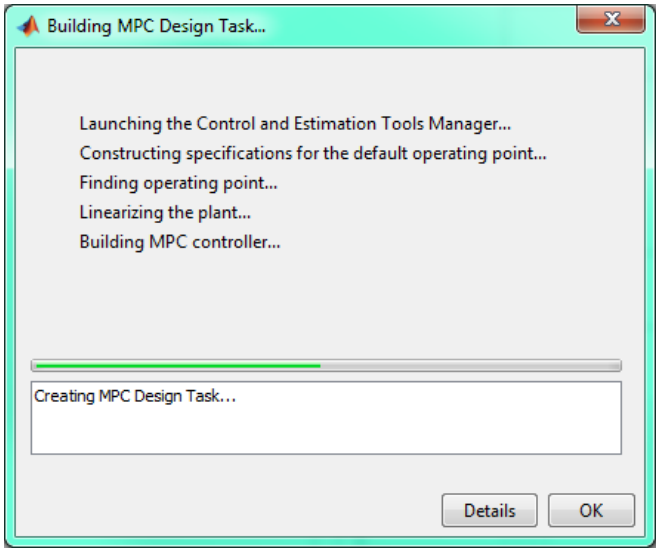

Figure 101: MPC Toolbox Design Task Linearisation in Simulink.

When using this method for the integrated software the linearisation will fail. This is occurs because the MPC Toolbox uses the Simulink blocks to linearise the plant, as the plant is actually a script it cannot determine the variables or model. In order to overcome this a MATLAB script, which can be found in 10.16 Appendix 16, was written to perform step tests on each input and determine the MIMO TF matrix. Once this matrix is created the toolbox can be launched from the MATLAB Command Window using 'mpctool'. When the Control and Estimation Tools Manager has launched it is possible to import the plant model using the Import Plant button shown in Figure 102.

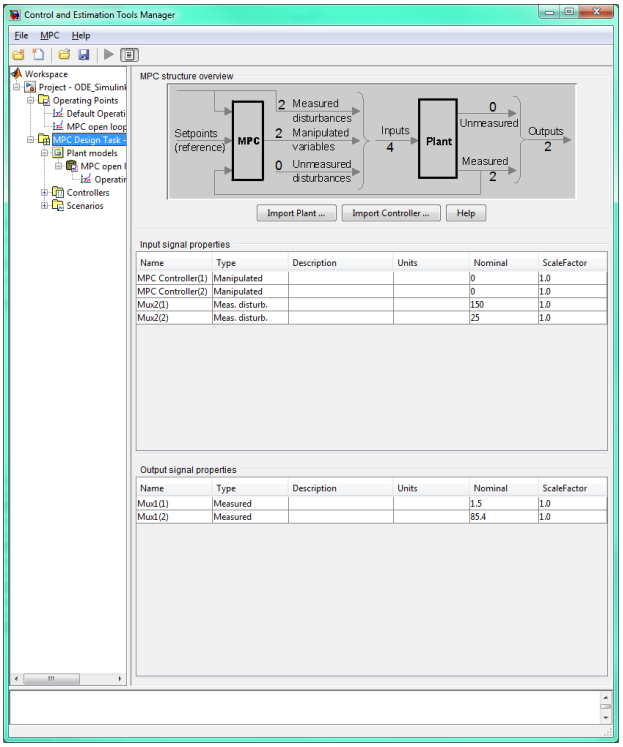

Figure 102: Control and Estimation Tools Manager in MATLAB.

With the plant imported the variable names can be changed in addition to the nominal values set. The description and units fields shown in Figure 102 are used only for reference and will not affect the performance of the controller. Note that if the controller linearised the plant automatically it will determine the nominal values as well. If the plant model was imported these values must be entered by the user.

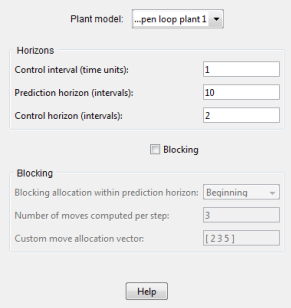

Figure 103: MPC Parameters in MATLAB.

Now the controller parameters can be entered by selecting the Controllers tab. If a previous controller was designed it can be imported here otherwise a new MPC can be designed. Figure 103 displays the initial parameters which must be specified for the MPC controller. The specified plant model will be used as a reference to perform the prediction inside the MPC algorithm. Similarly, the horizons are entered and sample time, inherit from the initial specifications window shown previously in Figure 100. Note the control horizon should be less than the prediction horizon, which typically should be long enough to capture the major dynamic behaviour of the plant. Once these variables are selected the next tab allows input and output constraints to be quantified. As stated previously these can be input via the Simulink model or inside the MPC Manager. The variables which can be specified are, as given in Figure 104:

Minimum	Maximum	Max Down Rate	Max Up Rate

Figure 104: Variable Constraints in MATLAB.

- Absolute minimum;
- Absolute maximum;
- Maximum rate of change downwards; and
- Maximum rate of change upwards.

If there are no constraints it is possible to enter ‘inf’ as the limitation.

The Weighting Tuning tab allows different weights to be applied to the inputs and outputs. These can be specified in the Simulink model or through this tab. If they are specified on the model then those values override those set inside the controller.

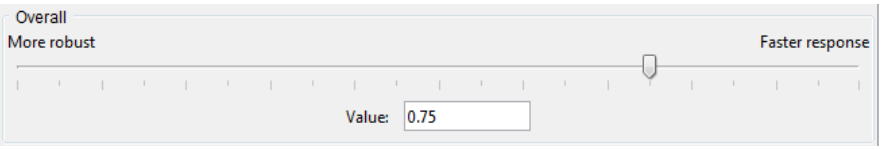

Figure 105: Scroll Bar to Select the Response Type in MATLAB.

The overall performance of the controller can also be manipulated by using the scroll bar shown in Figure 105. These variables can be changed later. Once satisfied with the weighting the controller tuning is completed and the MPC controller can be tested against the plant model.

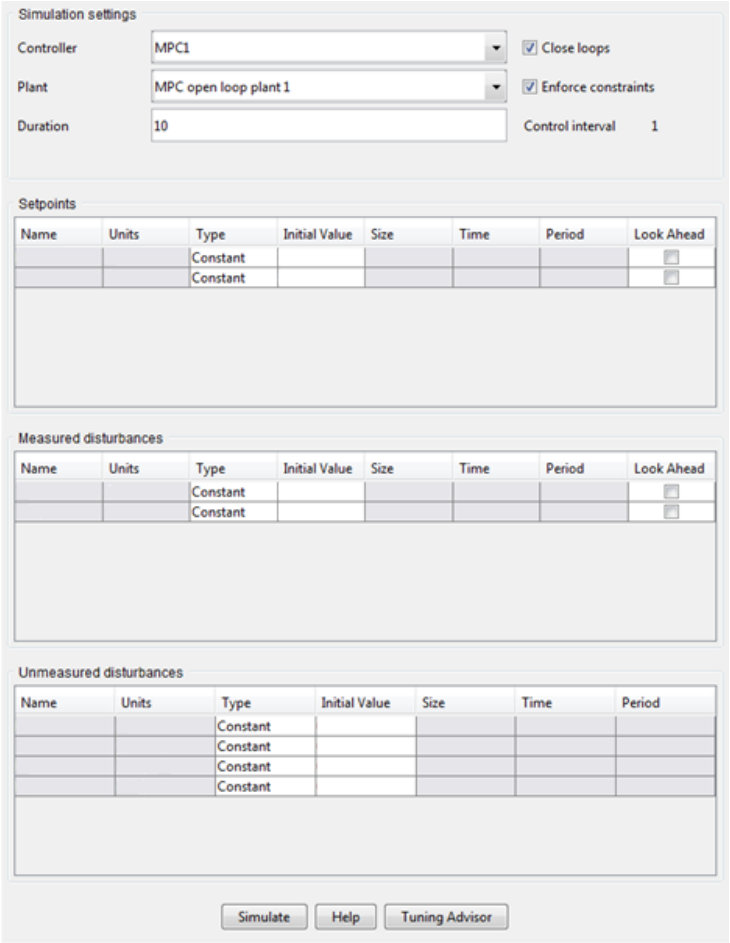

Figure 106: MPC Controller Simulation in MATLAB.

Figure 106 shows the MPC simulation window. On this window it is possible to test the controller developed against the plant model for SP tracking and DV rejection. The SPs or DVs can be changed to one of the following, as shown in Figure 107:

- Constant
- Step;
- Ramp;
- Sine;
- Pulse; or
- Gaussian.

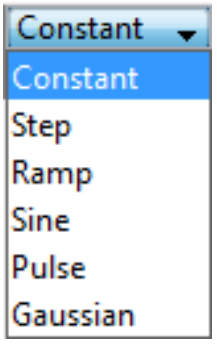

Figure 107: Different Signal Types in MATLAB.

Once the type of scenario and variables are selected the simulation can be run by selecting the Simulate button located at the bottom of the window. Once the simulation has completed two plots will appear. The first will display the PVs against their SP. The other shows the DVs. If the user is not happy with the performance of the controller the parameters can be altered in the Controllers tab and the scenarios run again. Once the MPC controller is performing as wanted it can be exported to MATLAB by returning to the Controllers tab and selecting Export. This will launch the Controller Exporter, shown in Figure 108, which will ask for which controller to export and the name.

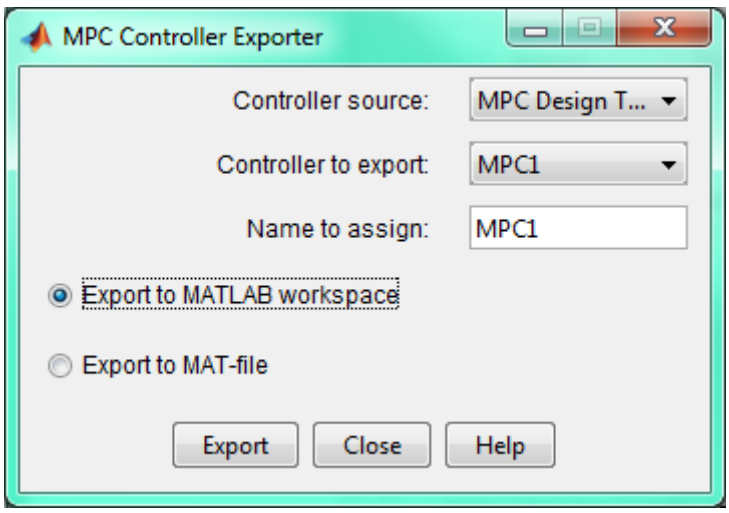

Figure 108: MPC Controller Exporter in MATLAB.

With the controller exported to the MATLAB workspace it can now be used in the MPC

Controller block in Simulink. Return to the Simulink model and double click on the MPC block. This will launch the MPC controller mask shown in Figure 99 again. However this time instead of selecting design, the name of the controller can be entered in MPC Controller field.

10.18 Appendix 18

MATLAB DMC Function

As an alternative to the MPC Toolbox in MATLAB and Simulink, a DMC script was created. This script takes advantage of the SISO system identification function found in 10.15 Appendix 15 to determine the transfer function for the given input and output pairing. This is then used as the convolution model to design the reference for the DMC. Following the theory found in 2.2.4.1 Dynamic Matrix Control, the controller was implemented as a function which requires three sets of information to be specified. First are the controller specifications: PV; SP; and MV Bias. Then the first order step response model parameters: step amplitude; model gain; and model time constant. Finally the DMC algorithm parameters: control horizon; prediction horizon; and the weighting matrices.

This function will then perform the DMC algorithm for the current plant variables and determine the action it should take to achieve the SP. Using the first control action and bias it will output the MV for this current time period. The function script is given below.

```
function MV = DMC(PV, SP, Bias, A, K, Tau, v, u, w1, w2)
% DMC.m, Joshua M Eggins, Murdoch University, 18/08/2015
% Function to computer the controller action of DMC. The user must
% input the transfer function model parameters determined from the
% SISO system identification function solverlstsq as well as the
% DMC parameters.
% v is the control horizon
% u is the prediction horizon
% w1 and w2 are the weighting matrices
```

```
%% Time
% Initial time array and the sample time, the user should change
% this is using a different time.
dt = 0.5;
Tdmc=dt:dt:v+dt;
Tdmc=Tdmc';
% Reference First Order Model
CP=A*K*(1+exp(-Tdmc/Tau));

%% Definitions
% Define A matrix
A=zeros(v,u);
for j = 1:1:u
    for i = j:1:v
        A(i,j) = CP(i-j+1,1);
    end
end
% Define diagonal weighting matrices
W1 = w1*eye(v);
W2 = w2*eye(u);
% Define H matrix
st = max(size(CP))-1;
fh = zeros(st+v,1);
h = zeros(v,st);
for i = 1 :1 :st
    fh(i,1) = CP(i+1,1) - CP(i,1);
end
for j = 1 :1 :st
    for i = 1 :1 :v
        h(i,j) = fh(i+j-1,1);
    end
end
% Define Controller Gain
Kdmc=inv(A'*W1*A + W2)*A'*W1;
%% DMC Calculations
% Initialise Arrays
E = SP - PV;
P = zeros(v,1);
dmv = zeros(st,1);
cv = zeros(st,1);
% Calculate change in MV and the predicted outputs
dMV = Kdmc * E;
dmv(1,1) = dMV(1,1);
CV = A * dMV;
cv(1,1) = CV(1,1);
s = h * dmv;
P(1,1) = s(1,1);
for j = 2:1:v
    P(j,1) = P((j-1),1) + s(j,1);
end
%% Controller Action
% Calculate the error
ER = E - P - cv(1,1);
% Extract current MV and add bias term
MV = Kdmc * ER;
MV = Bias + MV(1,1);
end
```

10.19 Appendix 19

LabVIEW MPC Toolkit

Within the LabVIEW Control Design and Simulation Module is the Control Design Toolkit. The Predictive Control palette contains various Vis which are used to develop MPC control. Figure 109 shows the Predictive Control Palette in LabVIEW. If the user wishes to set up dynamically informed SPs and DVs the real-time target RT FIFO VIs are used. These blocks queue the information each iteration before removing it from memory. It is typically used when there are large amounts of information being accessed every loop.

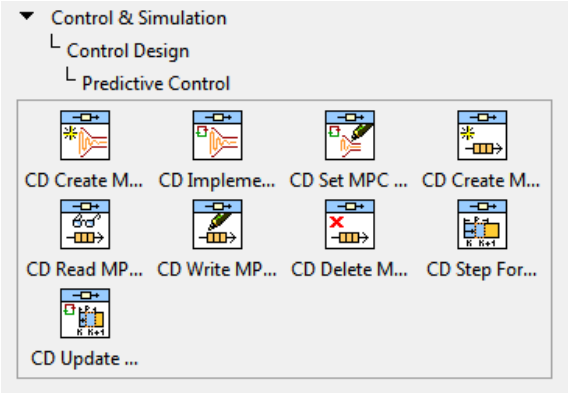

Figure 109: The Predictive Control Palette in the Control Design Toolkit in LabVIEW.

For smaller systems the CD Create MPC block is used to create the initial controller instance. Figure 42 displays this block, shown previously in 6.2 LabVIEW Control Design Toolkit. It expects the SS model of the plant to be input in addition to the controller and weighting parameters. The MPC controller parameters are:

- Prediction horizon;
- Control horizon; and
- Integral action.

The integral action is used when the SS model provided has large variations on the actual performance of the plant. If the behaviour of plant is represented accurately by the discrete SS the performance of the MPC controller will be improved.

The error weightings follow the idea that no penalty is applied when at unity. However if the weighting coefficient it less than one it will decrease the weight of that item, similarly, if the coefficient is greater than one it will increase the weighting of that item. Also note that the weight cannot be less than zero. The weighting matrices for the MPC controller are:

- PVs error;
- MVs rate of change; and
- MVs error.

When creating the MPC controller the constraints must be set via the MPC Constraints input. These constraints cover minimum and maximum:

- MVs;
- MVs rate of change; and
- PVs.

If any of the variables are not specified LabVIEW will assume $\pm$Inf. Furthermore, it is possible to specify optimisation stopping criteria within this category. These conditions cover the total time elapsed, iterations and rate of change.

Finally, the initial conditions of the controller can be specified. If these are not identified then LabVIEW will assume they are all zero. This can result in unexpected behaviour from the controller on startup, such as offset. The following conditions can be fed into the MPC Initial Conditions:

- MVs;
- PVs;
- DVs; and
- MVs rate of change.

With all these variables setup the controller is ready to fed into a while loop and the CD Implement MPC block. This block, shown in Figure 110, received the created MPC controller and PVs to determine the MVs.

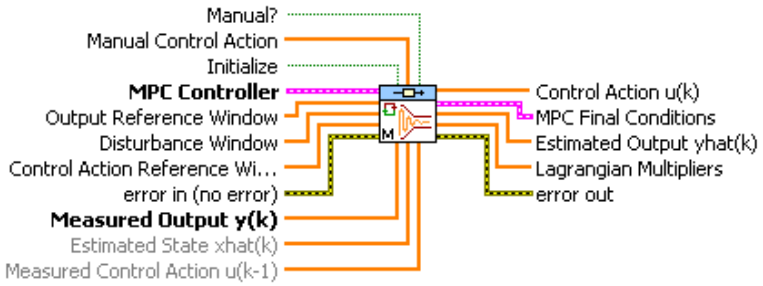

Figure 110: CD Implement MPC VI in LabVIEW.

It is possible to change between manual and automatic control using the top input ports. If TRUE is quantified the controller will not complete its algorithm and simply pass through the value input to the Manual Control Action port. However if FALSE then that input is ignored and the controller will calculate the MVs based on the PVs and SPs. The PVs are input through the Measured Output input while the SPs must be specified previously in the CD Step Forward MPC Window VI.

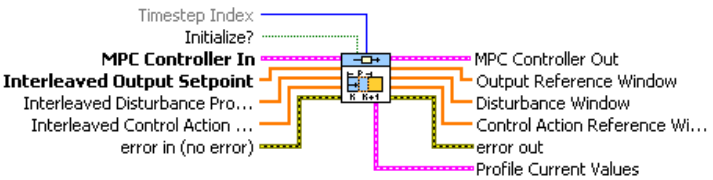

Figure 111: CD Step Forward MPC Window VI in LabVIEW.

Figure 111 shows the Step Forward Window VI. This block transforms the SP values into the Output Reference Window which is sent to the Implement MPC VI. The MPC controller reference is passed from the Create MPC VI to the Step Forward MPC VI then to the Implement MPC VI.

It is recommended the previous measured output is fed into the controller as this allows the block to know if an actuator is not able to execute its suggested MV. It is possible saturation

within the system might not have been input and this check allows the controller to easily detect any issues within the systems behaviour and update itself accordingly. Moreover, the predicted PVs can be withdrawn for the system through the Estimated Output port. These can provide insight into the validity of the plant model. If the controller has large errors between the predicted and actual PVs then it is possible the model being used is not viable for MPC reference.

Figure 112 displays the setup of MPC control within LabVIEW. The controller is created given the specifications input on the Front Panel. However these variables can be constants set on the Block Diagram as they will not be updated once the simulation is underway and loop entered. The MPC controller is sent to the Step Forward VI and the SPs set. Finally the Implement MPC VI receives the SPs, PVs, previous MVs and the MPC controller reference. Using these it calculates the MVs to reach the predicted PVs.

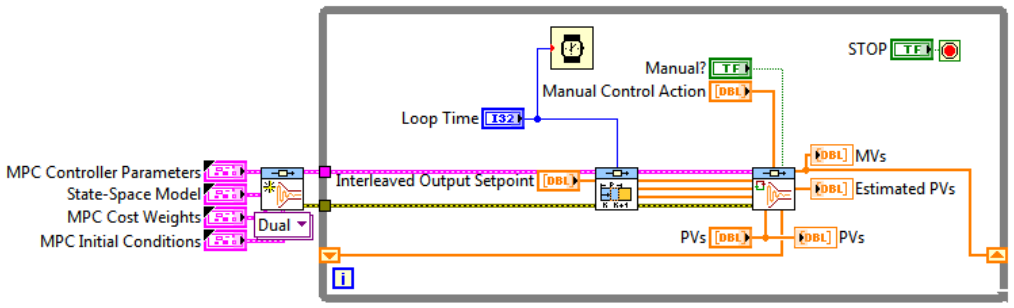

Figure 112: MPC Controller Design in LabVIEW.

10.20 Appendix 20

MIMO TF Model

The MIMO TF was determined from open loop Simulink model provided in Figure 44 in 7.0 Control Scheme Comparison. By stepping each input independently the system identification function, found in 10.15 Appendix 15, determined the following:

```
MIMOPlantModel =


  From input 1 to output...

         1.896
   1:  ----------
       0.14 s + 1

       -3.171
   2:  ------
         s

       3.316
   3:  -----
         s

        -0.1981
   4:  ----------
       0.16 s + 1

         0.1034
   5:  ----------
       0.14 s + 1


  From input 2 to output...

         1.761
   1:  ----------
       0.15 s + 1

       1.565
   2:  -----
         s

       -1.633
   3:  ------
         s

         0.1158
   4:  ----------
       0.17 s + 1

        -0.04785
   5:  ----------
       0.02 s + 1


  From input 3 to output...

       -1.358e-05
   1:  ----------
       0.14 s + 1

       -0.0004891
   2:  ----------
           s
```

```
        0.0005256
  3:  ---------
          s

        -5.082e-05
  4:  ----------
        0.15 s + 1

        9.218e-06
  5:  ----------
        0.13 s + 1


From input 4 to output...

        -6.618e-07
  1:  ----------
        0.13 s + 1

        -0.0004508
  2:  ----------
          s

        -5.806e-05
  3:  ----------
        1.34 s + 1

        8.687e-08
  4:  ----------
        0.14 s + 1

        -7.482e-08
  5:  ----------
        0.12 s + 1


From input 5 to output...

        -9.206e-07
  1:  ----------
        0.14 s + 1

        7.359e-06
  2:  ----------
        0.01 s + 1

        -0.002249
  3:  ----------
        1.83 s + 1

        -5.803e-08
  4:  ----------
        0.17 s + 1

        1.524e-09
  5:  ----------
        0.02 s + 1

Continuous-time transfer function.
```

10.21 Appendix 21

Comparison Plots

The following figures have been provided as a reference to the scenarios run. These compliment the ITAE values found in Table 5 in 7.0 Control Scheme Comparison.

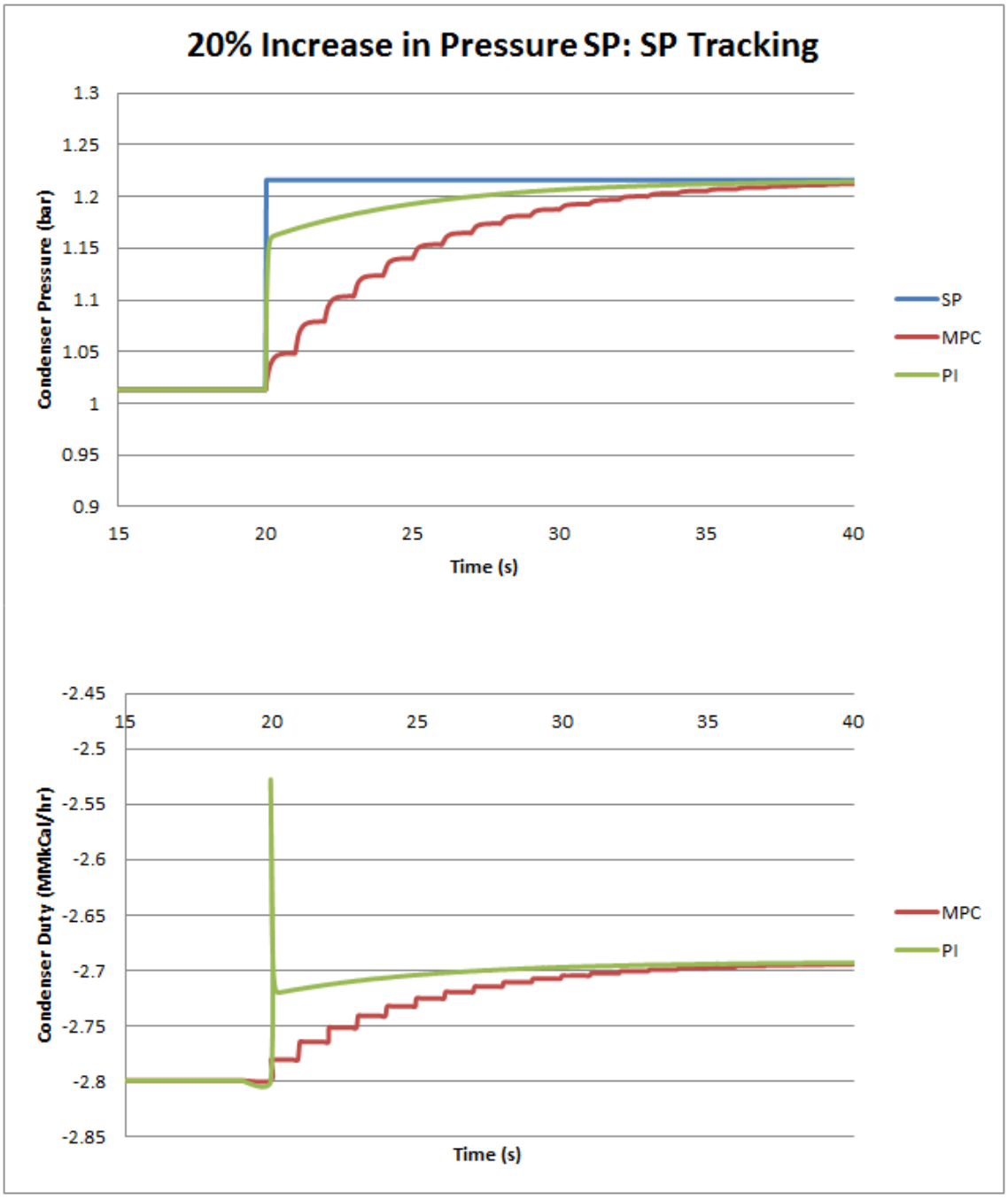

Figure 113: Plot of the Condenser Pressure Tracking the SP as Pressure Increase 20%.

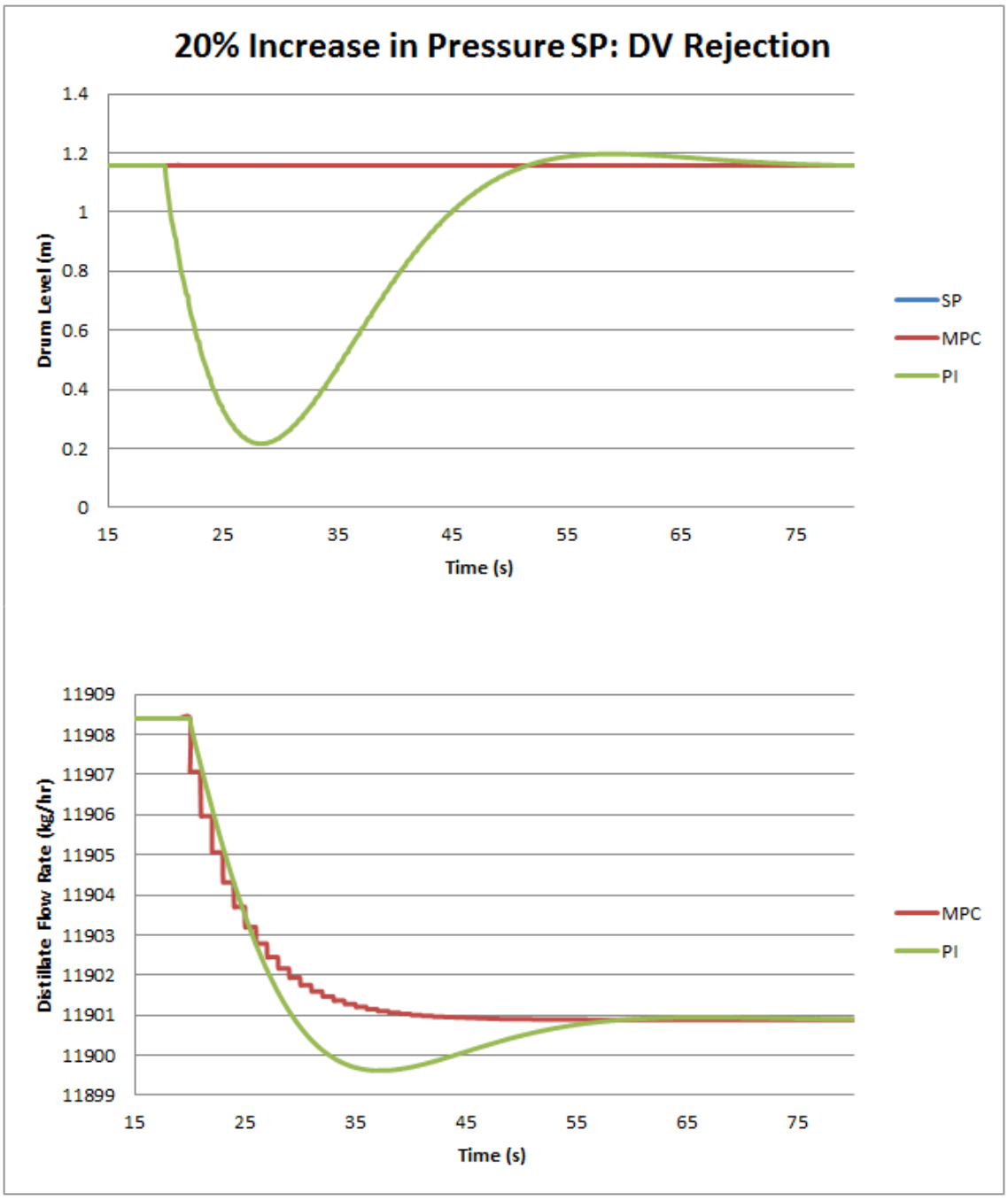

Figure 114: Plot of the Drum Level Rejecting Disturbance as the Pressure Increases 20%.

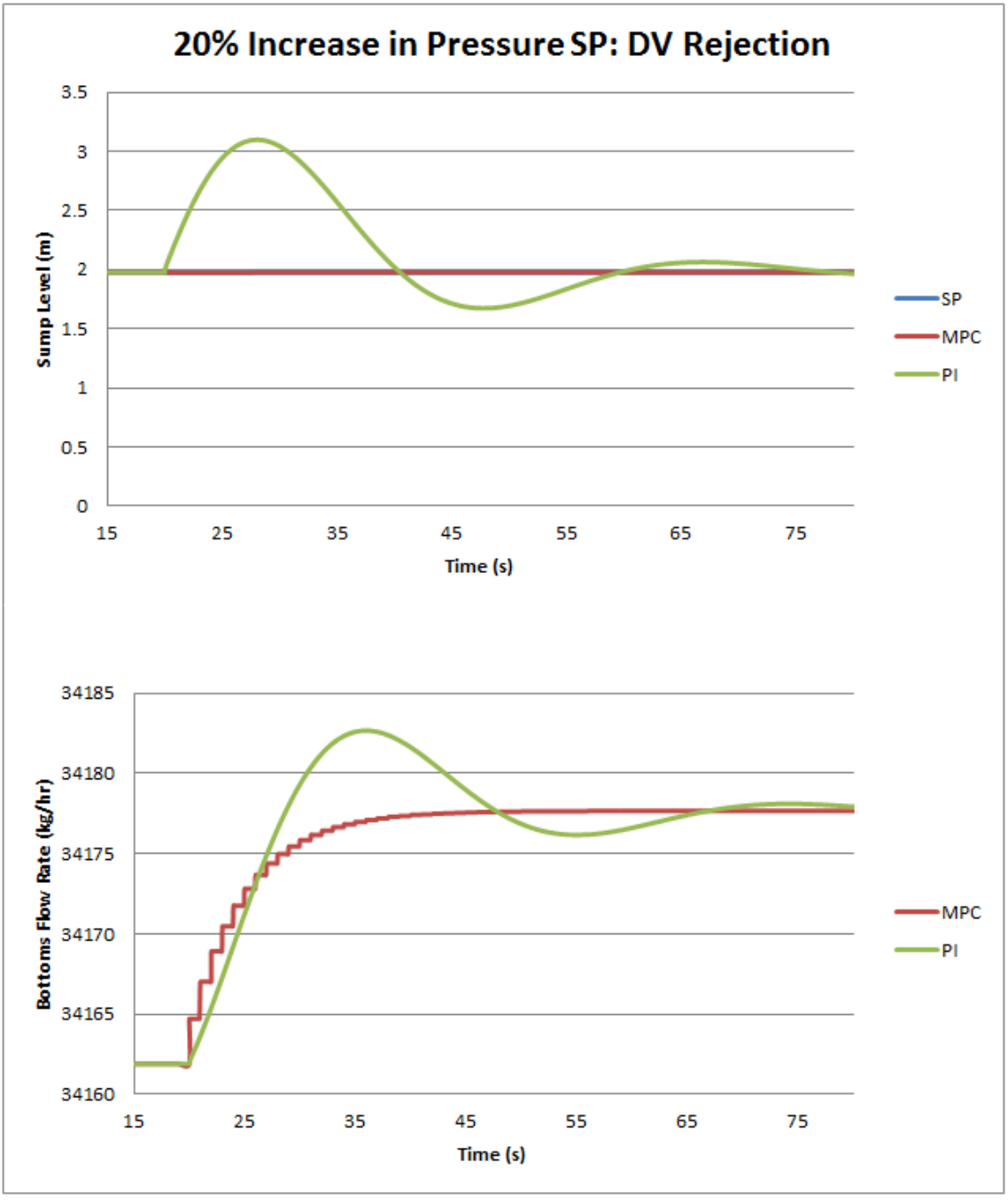

Figure 115: Plot of the Sump Level Rejecting Disturbance as the Pressure Increases 20%.

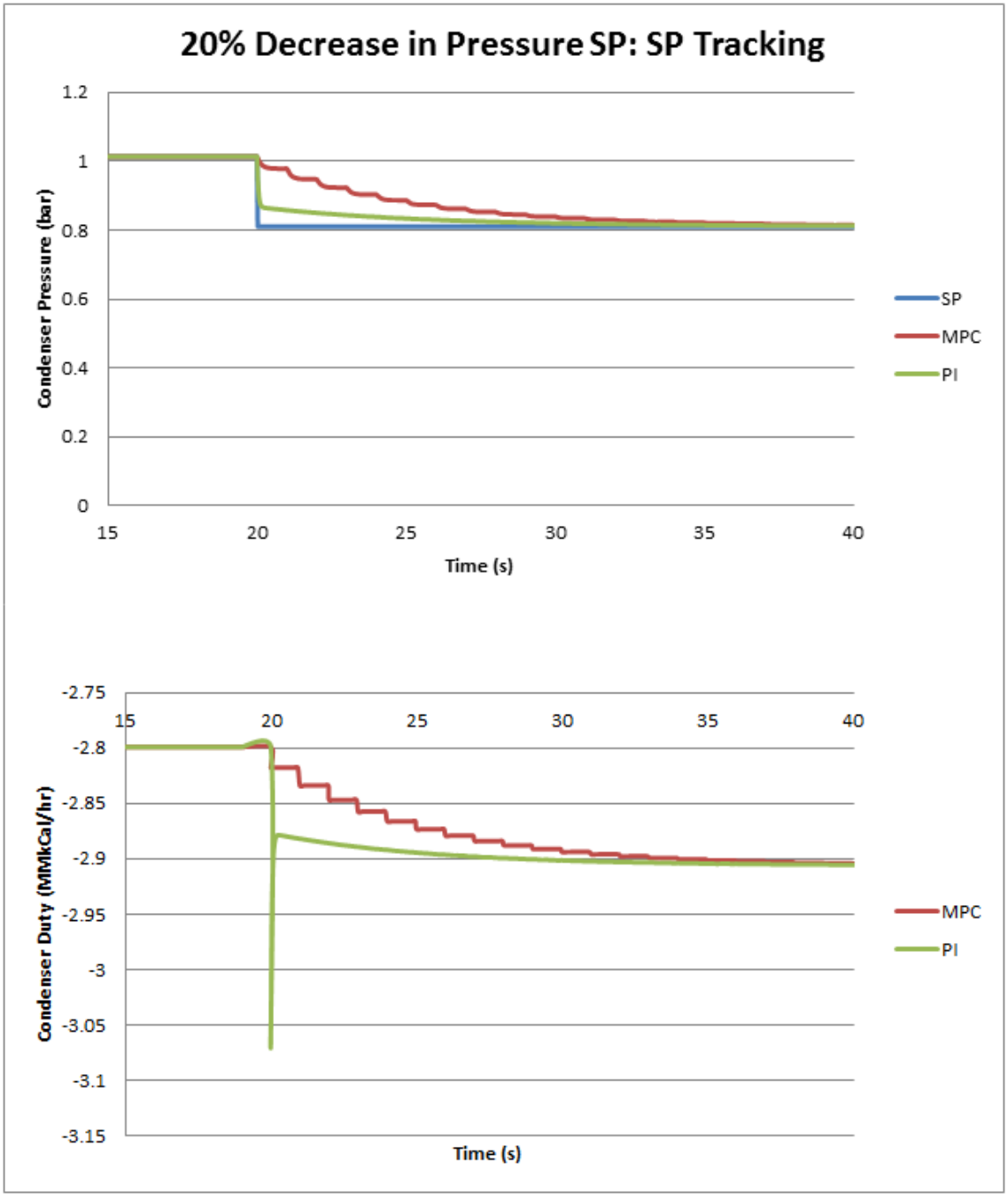

Figure 116: Plot of the Condenser Pressure Tracking the SP as Pressure Decreases 20%.

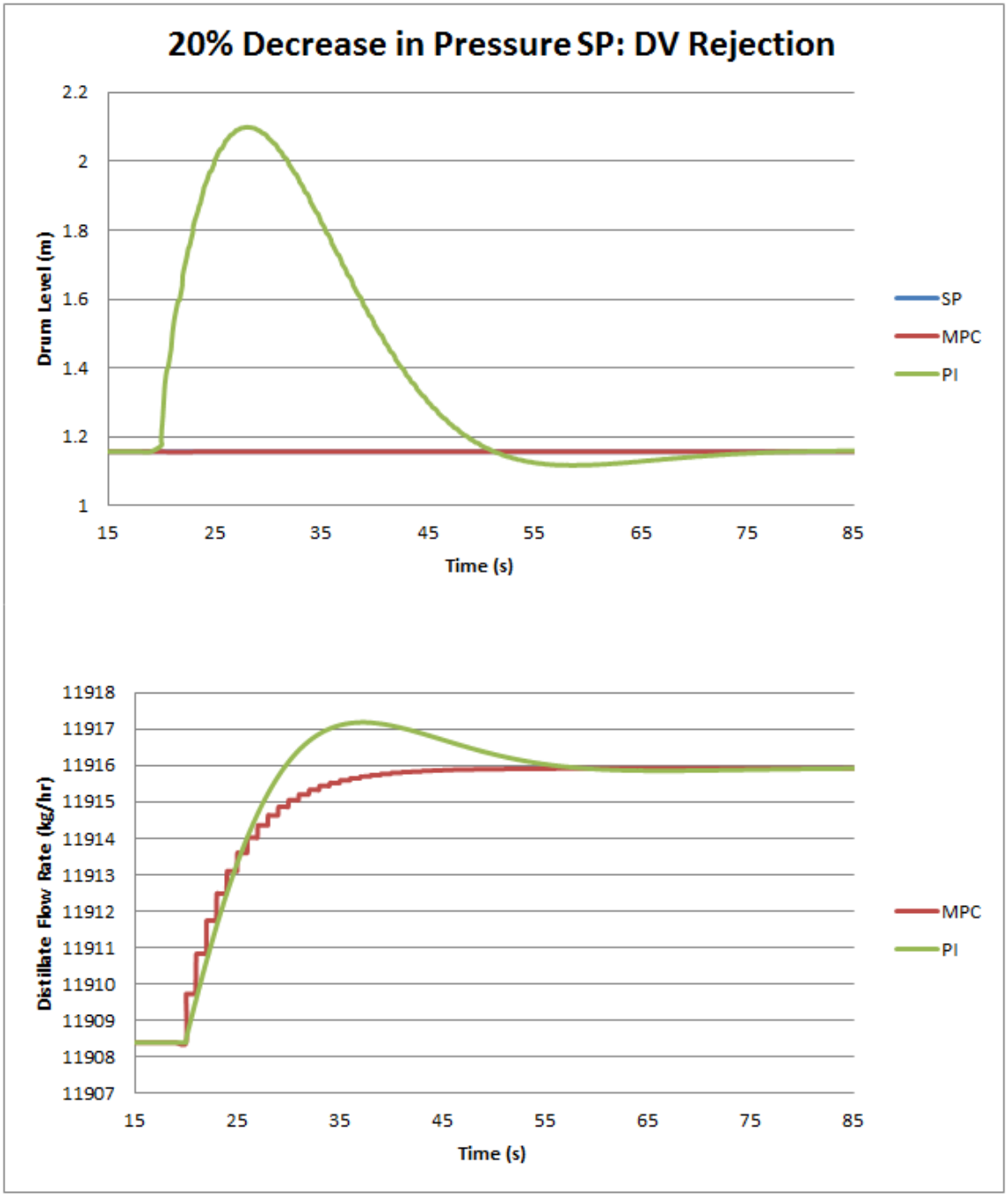

Figure 117: Plot of the Drum Level Rejecting Disturbance as the Pressure Increases 20%.

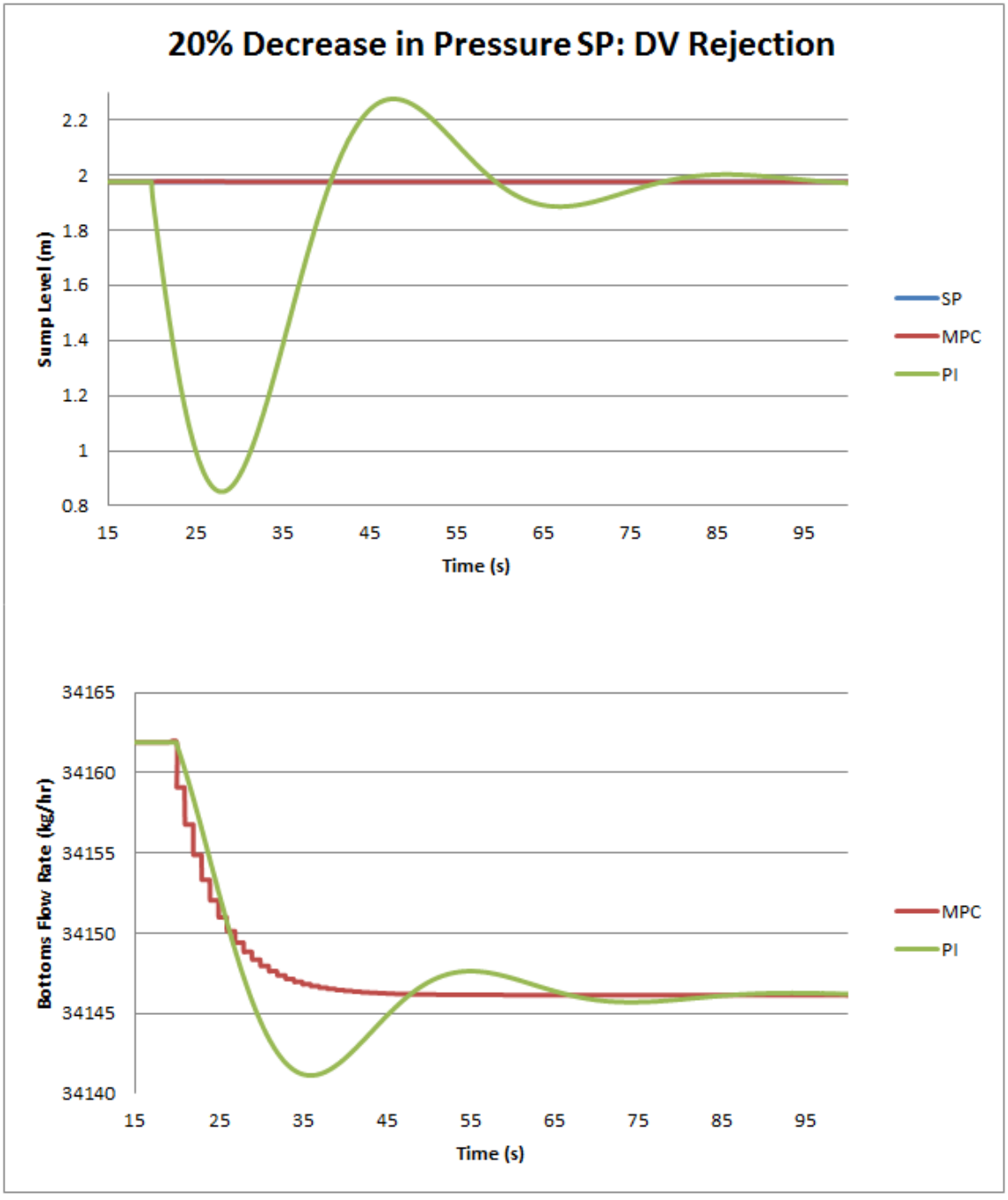

Figure 118: Plot of the Sump Level Rejecting Disturbance as the Pressure Increases 20%.

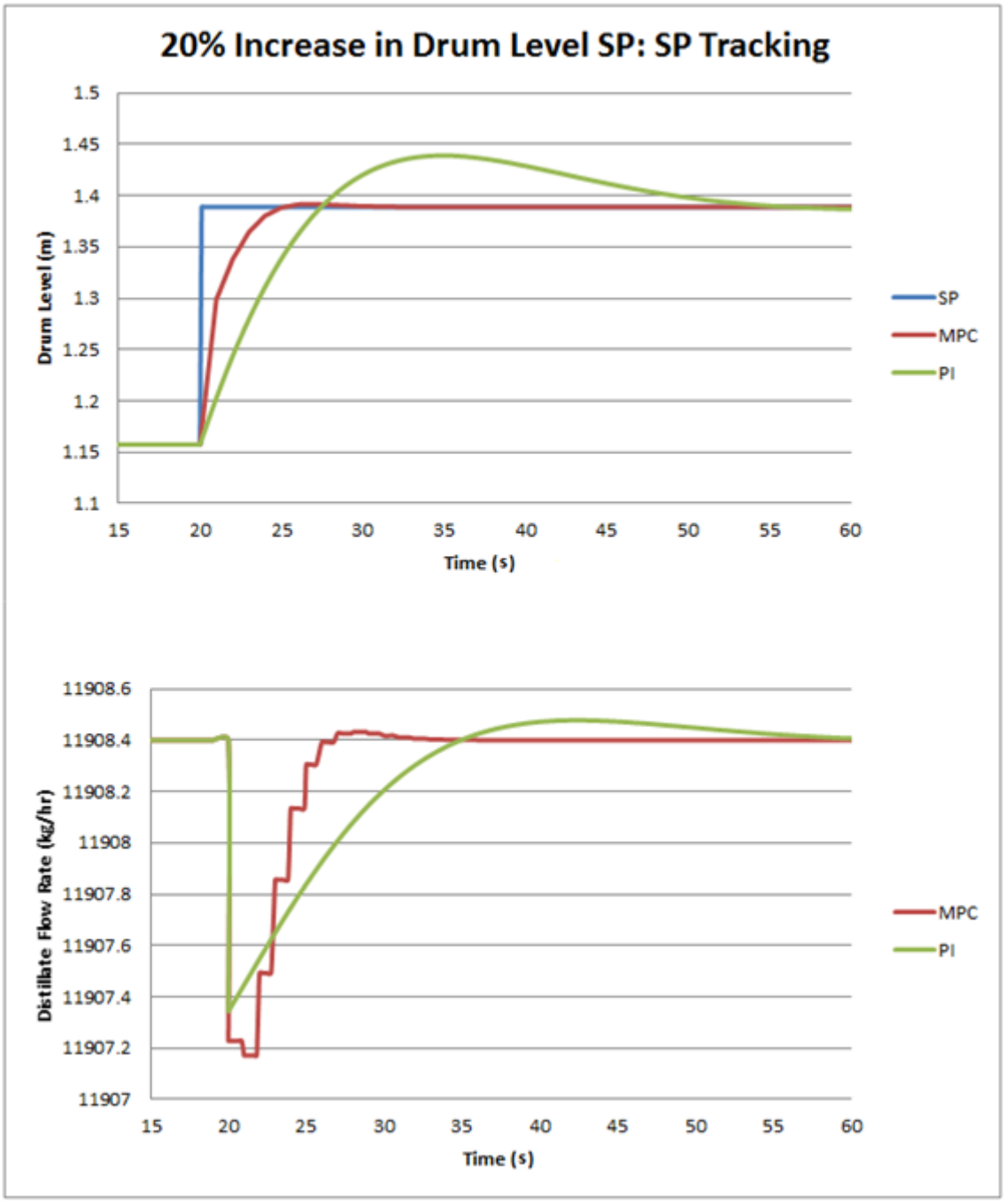

Figure 119: Plot of the Drum Level Tracking the SP as Level Increases 20%.

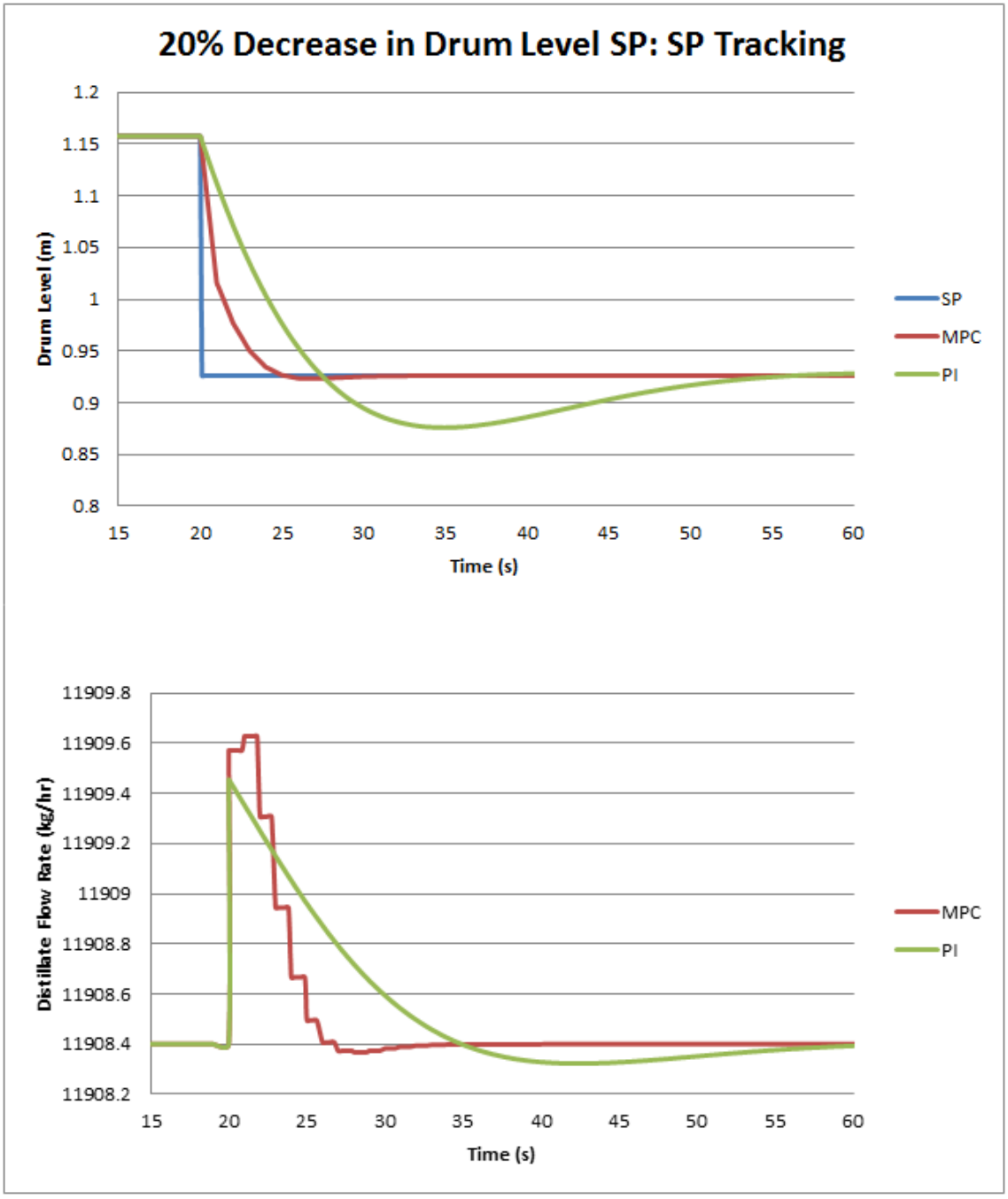

Figure 120: Plot of the Drum Level Tracking the SP as Level Decreases 20%.

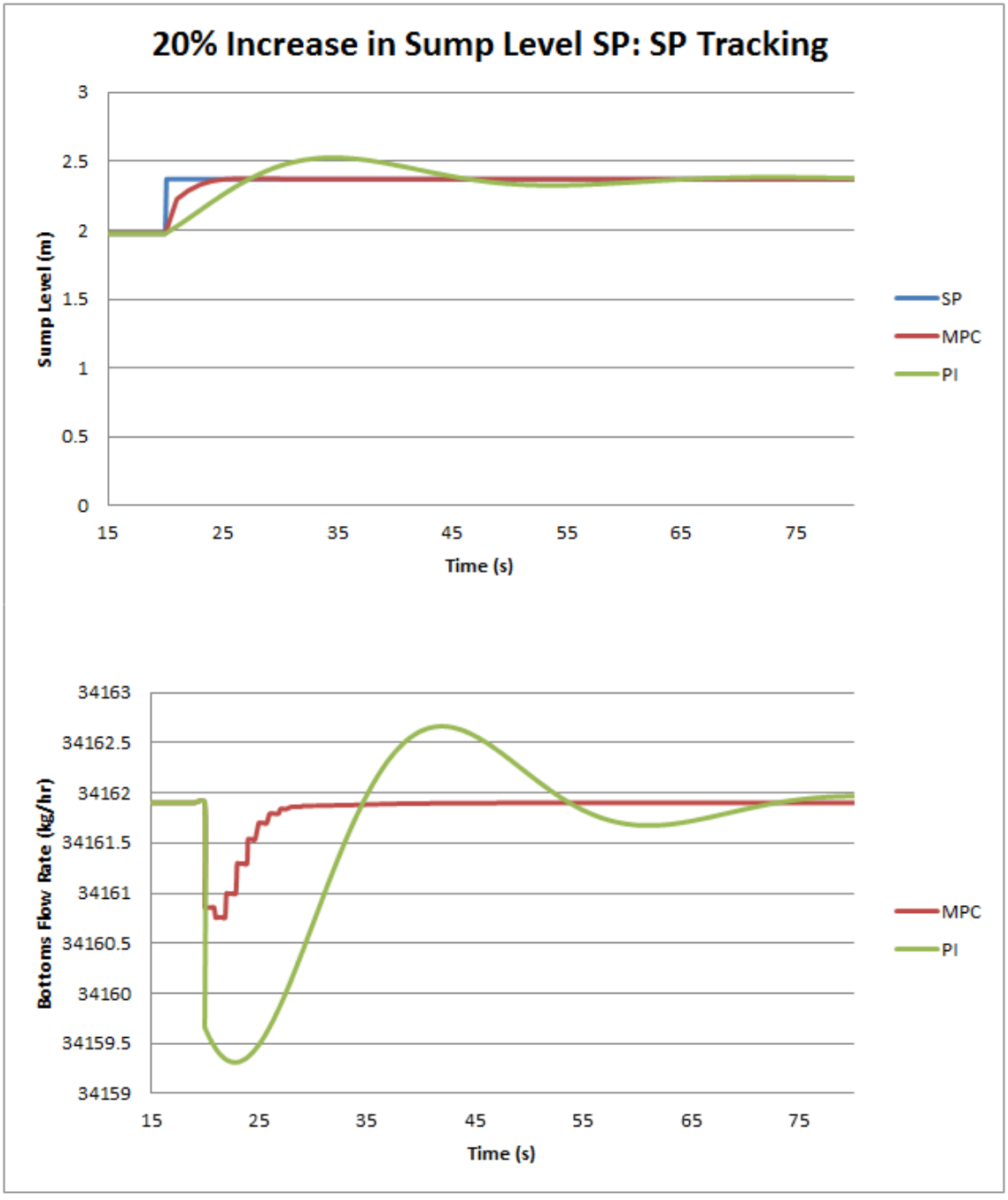

Figure 121: Plot of the Sump Level Tracking the SP as Level Increases 20%.

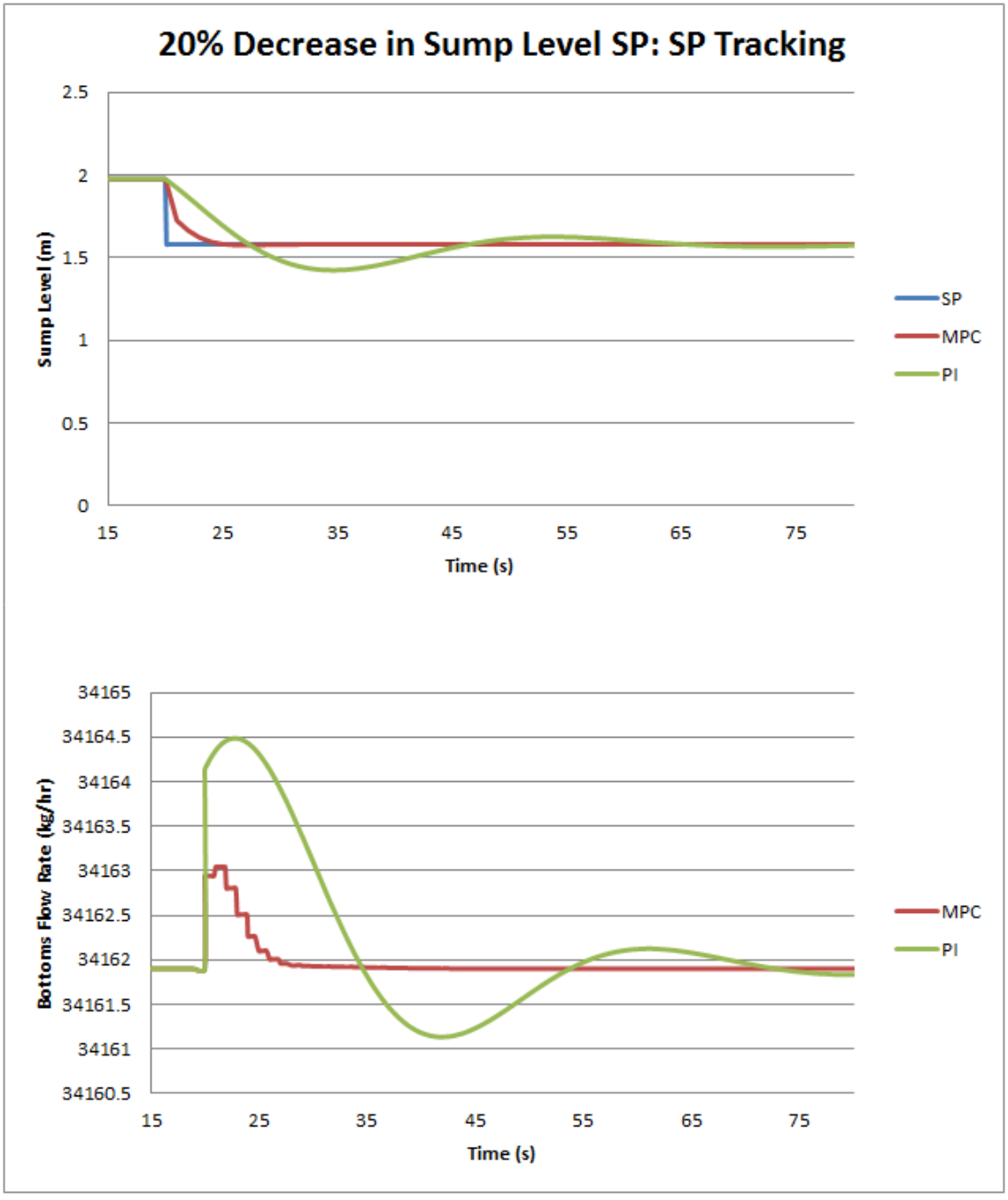

Figure 122: Plot of the Sump Level Tracking the SP as Level Decreases 20%.