**Teesside University**

**School of Science and Engineering**

Chemical Engineering

April 2016

**LEVEL CONTROL RESEARCH PROJECT**

Alexander David Leech

Dr David Peel

Abstract

Write LAST.

Contents

[1 Introduction 1](#_Toc447289197)

[2 Literature review 1](#_Toc447289198)

[3 Experimental Equipment 2](#_Toc447289199)

[3.1 Hardware 2](#_Toc447289200)

[3.2 Software 2](#_Toc447289201)

[4 Program Foundation Development 3](#_Toc447289202)

[4.1 Modbus Communications 3](#_Toc447289203)

[4.2 Data Logging 3](#_Toc447289204)

[4.3 Graphical Plotting 4](#_Toc447289205)

[4.4 Test Model 4](#_Toc447289206)

[4.5 Multiple OS support 5](#_Toc447289207)

[5 Preliminary Testing 5](#_Toc447289208)

[5.1 MODBUS Connection Verification 5](#_Toc447289209)

[5.2 Data Logging Tool 6](#_Toc447289210)

[5.3 Process Reaction-Curve Method 6](#_Toc447289211)

[5.4 Linearity Analysis 6](#_Toc447289212)

[5.5 Continuity analysis 7](#_Toc447289213)

[6 On/Off Control 8](#_Toc447289214)

[7 PID Control 8](#_Toc447289215)

[7.1 Background 8](#_Toc447289216)

[7.2 Program 8](#_Toc447289217)

[7.3 Tuning 8](#_Toc447289218)

[7.4 Testing 9](#_Toc447289219)

[7.5 Development 9](#_Toc447289220)

[8 System Identification 9](#_Toc447289221)

[9 Minimum Variance Control 9](#_Toc447289222)

[10 Discussion 9](#_Toc447289223)

[11 Conclusion 10](#_Toc447289224)

[References 11](#_Toc447289225)

[12 Appendices 12](#_Toc447289226)

[12.1 Appendix A 12](#_Toc447289227)

[12.2 Appendix B 13](#_Toc447289228)

Tables & Figures

**Tables**

Table 3.1 – Honeywell UDC3300 COM Settings 2

Table 4.1 – Honeywell UDC3300 Loop Value Integer Register Map (Honeywell, 2014) 3

Table 4.2 – Test Model Parameters 5

Table 7.1 – PID controller tuning parameters calculated using the Cohen-Coon tuning rules 8

**Figures**

[Figure 1 – Graphical depiction of process non-linearity 7](#_Toc447375680)

[Figure 2 – Verification of the replicability of results 7](#_Toc447375681)

[Figure 3 - Standard PID controller 40% to 50% set point change response on simulation system 11](#_Toc447375682)

[Figure 4 – Experimental Process and Instrumentation Diagram 14](#_Toc447375683)

[Figure 5 – Step test performed utilising a +2.5% step starting at 35.0% 15](#_Toc447375684)

[Figure 6 – Step test performed utilising a +2.5% step starting at 37.5% 15](#_Toc447375685)

[Figure 7 – Step test performed utilising a +2.5% step starting at 40.0% 15](#_Toc447375686)

Nomenclature

CRC - Cyclic Redundancy Check

I/P - Current to Pressure Converter

IDE - Integrated Development Environment

Kp - Proportional Gain

Ki - Integral Gain

Kd - Derivative Gain

OOP - Object Orientated Programming

OP - Operating Point

OS - Operating System

PID - Proportional Integral Derivative

PV - Process Variable

R/W - Read/Write

RTU - Remote Terminal Mode

SISO - Single-Input Single-Output

SP - Set Point

YAML - Yet Another Mark-up Language

# Introduction

Process control is at the heart of optimisation and automation throughout the industrial sector.

*This project involves the advanced process control of a liquid level system. The project will require development of a MODBUS RTU interface and the coding of advanced control algorithms.*

There is in excess of 500 different notable programming languages available to use, each having their own benefits and drawbacks. Out of these, Java, C/C++ and Python were considered in more depth. Python was determined to be the best option for the project due to its open­-source licence making it freely usable and distributable (Python Software Foundation, 2001). A large database of third party libraries are freely available to implement giving the project more freedom to focus on the engineering aspects rather than the computer science. Object orientated programming is also possible with python resulting in highly efficient programs as code can be reused multiple times to create objects with different parameters. Furthermore Python runs on a whole host of different platforms from Windows to Advanced RISC Machine (ARM) giving the possibility for experimentation and development across a wide variety of different devices.

A secondary language that will also be used alongside of Python is that of YAML (Yet Another Mark-up Language). YAML is a human-readable data serialization language and therefore provides strong support for creating configuration files that are simple to read yet still easy to implement within a program. Utilisation of such will avoid the need for users to delve into main files to change operating parameters, increasing user operability.

*System Non Linearity prediction*

*Derivation of equations to use*

# Literature review

*Cohen-coon tuning*

MODBUS is an industry standard communications protocol developed in 1979 by Modicon. Serial communications are used to exchange data between a master unit and up to 247 slaves. Information on the slave is stored within four different tables, the first two of which store on/off data known as coils and the second two, numerical values known as registers. Each type has a read-only and a read-write table which the master can either request information from or write to. The data stored within said tables is identified by means of a MODBUS map, normally published by the manufacturer. MODBUS commands are sent in the form of a data packet. This packet contains a unique address to identify which slave on the network must respond, a function code and relevant data to instruct the slave what task to perform, and a cyclic redundancy check to ensure the data received has not been corrupted in any way. This 8 bit message is then framed with a start and stop bit for identification purposes as per serial communications protocol.

# Experimental Equipment

## Hardware

The equipment presented consists of a single-input single-output (SISO) 84L gravity drained tank. A Honeywell UDC3300 acts as the control interface with tank level being received via 4-20mA from an ultrasonic level sensor. Control output is fed by 4-20mA into current to pressure (I/P) converter and the resulting pneumatic signal to a control valve on the inlet flow line.

Instrument air is supplied by means of a bench scale compressor and water recirculation by means of a centrifugal pump. These two devices are not part of the control loop. A process and Instrumentation diagram of the process is provided in [Appendix A - Figure 3].

Connection to the Honeywell UDC is to be established utilising MODBUS RTU via a RS-232 interface [Table 3.1]. It should be noted that the Honeywell UDC will not be used as a controller for the duration of this project. Its primary use will be to act as a hardware interface.

Table 3.1 – Honeywell UDC3300 COM Settings

|  |  |
| --- | --- |
| Parameter | Value |
| Method | ‘rtu’ |
| Port | ‘COM3’ (Windows)  ‘/dev/ttyS0’ (Linux) |
| Stopbits | 1 |
| Bytesize | 8 |
| Parity | ‘N’ |
| Baudrate | 9600 |
| Timeout | 1 |

## Software

As previously described, Python 2.7 is the chosen language. Due to the nature of the project, the Spyder IDE (Scientific PYthon Development EnviRonment) will be utilised for all code development and debugging. It is natively supported on Linux and runs via the anaconda environment on Windows. The project source code is hosted within an open source GitHub repository that is available at (Leech, 2016) and licensed under the MIT license.

# Program Foundation Development

To enable swift development a set of base classes have been built to provide the tools required for each and every program. As python is an OOP language, each of these classes can be imported and used without having to duplicate the code. This yields extremely efficient programs that take very little space thereby allowing them to be used on minor architectures such as the ARM architecture. The importable classes that have been setup are described in the following section and references to all assumptions stated herein.

## Modbus Communications

To simplify the process of generating the packets, the library *‘pymodbus’* has be used as a core foundation to build upon (Collins, 2015). It handles the core operations such as opening a port, compiling the data into a MODBUS standard packet, sending the data and verifying the response. Upon inspection of the MODBUS map only the R/W holding registers need to be utilised. [Table 4.1] contains an excerpt from the controller MODBUS map (Honeywell, 2014) and provides the core addresses required to access all required data for the purpose of this study. The methods therefore utilised from the *pymodbus* library are *‘read\_holding\_registers’* and *‘write\_register’*. The derived class is available within the code repository at ‘*…/src/dataLoggingTool/Modbus.py’.*

Table 4.1 – Honeywell UDC3300 Loop Value Integer Register Map (Honeywell, 2014)

|  |  |  |
| --- | --- | --- |
| **Controller Holding Registers** | | |
| Address | Access | Parameter Name |
| 0 | R | Process Variable (PV) |
| 1 | R | Remote Set Point (RV) |
| 2 | R/W | Set Point (SP) |
| 3 | R/W | Operating Point (OP) |

## Data Logging

A crucial element to almost all advanced plant controllers is the element of historical data logging. The data is key to calculating controller parameters, determining controller performance and analysing efficiency. In a distributed control system (DCS), the process of data logging and management is completed using a historian server. Due to the scale of said project, *‘openpyxl’* has been used to log in Microsoft Excel format instead. Said library contains a variety methods that open R/W access to a workbook.

The class that has been written creates a new worksheet with date and time stamp, creates all column headers and formats the sheet appropriately. Support is available to log either four or six variables which, as will become evident in [Section X], covers all scenarios presented within the project scope. Said file is available within the code repository at ‘*…/src/dataLoggingTool/xlsLogging.py’.*

## Graphical Plotting

The addition of a SCADA interface is beyond this project scope and substantially increases complexity, therefore a standalone graphical plotting class has been formulated to allow visualisation of experimental progress. The basis of said class is the library *‘matplotlib’.* It presents an interface similar to that offered by MATLAB for ease of use.

To ensure continuity with the data logging tool, there is support for plotting either 4 or 6 variables. To avoid scaling issues with the axis, when plotting in 6 variable mode two plots are drawn in a single figure. Furthermore a configuration file has been created in YAML to give the user full control over parameters such as line colour and type of each variable. Said class is available within the code repository at ‘*…/src/dataLoggingTool/plotActiveGraph.py’* and its respective configuration file at‘*…/src/dataLoggingTool/plotPenConfiguration.yaml’.*

## Test Model

To avoid wasting time in the labs by making adjustments to sections of code that may not work as planned, a test environment was developed. This is in the form of a continuous time model developed from the theoretical system equation [Equation X]. As the model does not need to be identical to the online system, only similar, it has been assumed that the tank outlet flow (qout) fits Torricelli’s law with no resistance. This leads to [Equation 4.1] which can be used providing the previous tank height is known (h), flow rate in (qin) and time step between (t).

Equation 4.1

A problem that exists with the test model is that of determining qin. With respect to the real system, the valve characteristic determines how the flow rate varies with valve changes. As this information is not available at present, a back calculation approach utilising [Equation 4.1] and the step test data in [Section 5.3] has been employed. Unfortunately this is a very limited data set, however it is also the primary operating range and therefore if data outside of it is inaccurate it shouldn’t have much of an influence. It should be noted that above 42.5% the flow does not increase. Fitting the data to a third order polynomial gives an equation that relates value position (u) to qin (Leech, 2016, pp. 59-61). At this point, a variable degree of noise is also added to the flow to allow the robustness of each controller to be tested.

Table 4.2 – Test Model Parameters

|  |  |  |
| --- | --- | --- |
| Parameter Name | Value | Unit |
| Area Outlet Pipe (Av) | 4.9E-4 | m2 |
| Tank Area (At) | 6.0E-2 | m2 |
| Gravity Constant (g) | 9.81 | m/s2 |
| Noise Multiplier | 2.0 | - |
| h Initial | 30 | % |
| u Initial | 40 | % |

The class has been written in such a way that it accepts and returns data in the same format as the MODBUS class, effectively therefore acting as a like for like replacement of the online system. Within the start-up script for each controller, the option for simulation mode is available to make switching between the two seamless. Controller parameters are all in a YAML configuration file and the class itself available at *‘.../tests/testModel.py’*.

## Multiple OS support

The final support class is available at ‘*…/src/dataLoggingTool/osTools.py’* and contains a variety of methods to ensure all programs will work on multiple operating systems.

# Preliminary Testing

Prior to the testing of any forms of control, the MODBUS connection needs to be tested and a degree of basic system identification needs to take place. This identification will take place in the form of identifying the system steady states, and performing open loop step tests. Utilisation of methods such as the process reaction-curve method are then employed to calculate system gain and time constants.

## MODBUS Connection Verification

To verify the correct connection parameters and ensure the library was working as it should be, two initial testing files were utilised. The first of these opens a connection and performs a read test upon the holding registers every four seconds, printing the results to screen. These numbers were then cross checked against those shown on the controller display for verification. Although there was some initial trouble with getting the correct unit address, once found no further problems reading the registers were encountered. The second program opens a connection to the controller and at a four second interval writes the valve either open or closed. This test proved successful therefore proving the ability to write to the correct controller register. Each testing program can be found at *‘/tests/readTest.py’* and *‘…/tests/writeTest.py’* respectively. It should be noted that the aim of this verification was primarily to prove the connection and therefore the base structural classes formulated in [Section 4] were not tested during this exercise due to their heightened complexity.

## Data Logging Tool

Utilising the base classes as described in [Section 4], a data logging tool has been devised. This not only allows for the step tests progress to be recorded and tracked, but also acts as the verification that the base structural classes are functioning as intended. Although any number of variables could be logged, the program has been limited to support for 4 or 6. The reason for such is that a minimum of the four variables depicted in [Table 4.1] are required for a basic data set, however during the later stages of control development two extra variables, α & β, are introduced by the controller model [Section 8] increasing the number required to six.

Initial tests revealed a multitude of features that needed to be added such as exception handlers and a safe exit method to avoid data loss. These features were successfully implemented in revision two of the program. Furthermore testing on Windows and Linux based systems also proved to be successful. The data logging tool main file can be found at *‘…/src/dataLoggingTool/dataLoggingTool.py’* with its appropriate start up script located in *‘…/script/<OS>/’.*

## Process Reaction-Curve Method

The process reaction-curve method, or otherwise known as the Ziegler-Nichols Open-Loop tuning method is a simple way of determining the gain (K) and time (tau) constants of a linear system. This information can then be used in the tuning of a PID controller or initial model parameter identification. The method involves making a step change in the system OP and logging the respective change in PV. From the graphical plot produced, gain is calculated as change in PV over change in OP [Equation 5.1], and tau from the time difference between intersection at the end of dead time, and the PV reaching 63% of its total change [Equation 5.2] (Smuts, 2011).

Equation 5.1

Equation 5.2

Steady states were initially determined to exist between 40% and 50% valve OP, however due to a degree of complication with oil contamination in the instrument air supply, the valve characteristics altered shifting steady states to between 35% and 42.5%. Tests were performed using a 2.5% OP step within the stated range yielding the results depicted in [Appendix B, Figures 3, 4 & 5]. Comparison of K and tau at each point gives [Figure 1].

## Linearity Analysis

As the gain and time constants have been calculated at a variety of different operating points, the linearity of the system around said region can be defined. In a linear system the parameters do not vary across the operating range and produce a line with zero gradient for both K and Tau whereas in a non-linear system the parameters vary creating a non-zero gradient. As predicted in [Section X], [Figure 1] clearly depicts a large variation over a relatively small operating range, primarily due to the tanks gravity drained nature. In terms of controller tuning and performance it introduces a number of problems that have to be considered. These will each be addressed in their relevant section.

Figure 1 – Graphical depiction of process non-linearity

## Continuity analysis

To verify to the replicability of the results obtained, a continuity analysis was performed. This was performed by inputting an identical step into the system on two separate days, and comparing the results by overlaying one graphical plot upon the other. The results as displayed in [Figure 2] show good relation deeming tests completed on separate days are indeed comparable.

Figure 2 – Verification of the replicability of results

# On/Off Control

On/Off control is the most basic form of control available and comes with a great degree of limitations and disadvantages. Despite this it should not be overlooked as in extremely slow moving processes or those that are not often used it can provide the best solution

# PID Control

One of the most widely used control algorithms across the process industry is that of the PID controller. Due to its widespread usage, it is a well-established and documented in great levels of detail. Within this section, a PID controller will be formulated, implemented and tested to verify its robustness.

## Theory

The continuous time PID control equation is given by [Equation 7.1]. It consists of three parts, a proportional term, integral term, and a derivative term. Utilising various different combinations of these terms yield different results therefore presenting the user with added flexibility. The commonly used combinations are that of PI and PID, however P and PD are also valid for specific applications.

Equation 7.1

The proportional term is the driving force of the controller equation and depends only upon the error value. The proportional gain (Kp) determines the ratio of the output response to the error signal (National Instruments, 2011). Due to this term merely being a ratio, if used by itself, usually a steady state error greater than zero will exist.

The integral term acts as a summation of error over time, driving the steady-state error to be zero. The effect of such term is governed by the integral gain parameter (Ki). In most situations this is extremely useful, however a problem known as integral windup can occur when the summation of error leads to controller saturation without driving the error to zero (National Instruments, 2011). A variety of different methods exist, all of which solve the problem. These range from changing the set point slowly in a large ramp, to disabling the integral function altogether until a controllable region is entered. The chosen method to implement is that of ‘clamping’ upon saturation. Doing this prevents the error summation from trying to drive the controller output outside of its limits, for example, a valve greater than 100%.

The derivative term acts as a brake to slow down the output effect if the process variable is shooting off in a given direction. It helps avoid overshoot and when tuned correctly can stabilise oscillations in the controller. Its effect is governed by the derivative gain term (Kd). Notably the derivative term is usually the smallest as too much of an effect can easily provide the opposite effects to those described above.

The three parameters Kp, Ki and Kd are calculated by means of a tuning method. Numerous different algorithms exist for completing this process, each and every one unique in their own way and suited to a specific range of processes. The most commonly used is the Ziegler-Nichols method published in 1942. Using values from the process reaction curve described in [Section 5.3] and applying them to (Shahrokhi & Zomorrodi, 2012, pp. 3, Table 5) allows evaluation of tuned values of Kp, Ki and Kd for the process in question. The rules aim to tune the process to give a quarter amplitude damping response. Cohen and Coon in 1953 built upon these rules, producing a similar table (Michigan Techonological University, 17, pp. 2, Table 1), but suited to a wider variety of processes. Other commonly utilised methods are the C-H-R method, Fertik method, Ciancone-Marline method and Minimum error criteria. However, due to the popularity of the Cohen-Coon rules and they’re applicability to a wide range of processes they have been selected to be utilised hereafter. Comparison of the various tuning methods is beyond the scope of this report.

## Program

The PID controller program is made up of two additional files on top of the base classes. The first of these is the PID class that contains [Equation 7.1] in python format. The PID algorithm code has been based upon that given by (mstarlabs, 1994) due to its comprehensive yet simple implementation. Anti-windup has been implemented in the form of clamping upon saturation. It should also be noted that a transition method is implemented that calculates a value for the integral error then the controller is first switched on. This reduces controller ‘bump’ solving the problem of the controller having to build up its own error valve from scratch and therefore leading to a seamless transition when first switched on. The file can be found at *‘…/src/PIDControl/PIDController.py’*.

The second file contains the main method which handles importing all the appropriate classes to read data from the system, pass it to the controller and write the result back to the controller. This is all performed in a time based main controller loop with definable sample time. This main file can be found at *‘…/src/PIDControl/researchProjectPID.py’* with its appropriate start up script located in *‘…/script/<OS>/’.*

## Tuning

Before testing the controller on a live system, it has to be tuned to establish a set of parameters. The method selected to perform this was the Cohen-Coon method as stated in [Section 7.1]. At this point the first problem from system non-linearity arises. Parameters are calculated using gain and time values at various points across a reaction curve. As noted in [Section 5.4], the parameters change quite substantially over the operating range meaning the tuning parameters also vary. The controller will therefore only be tuned efficiently within a very small range. This does not have a huge influence on processes that only operate around a single set point as the controller can be tuned specifically for that range. However non-linear processes that commonly change set points will encounter inefficient tuning at all but one state. The Cohen-Coon rules are available in (Michigan Techonological University, 17, pp. 2, Table 1) and have been used in conjunction with the data in [Appendix B, Figure 5] to generate the results shown in [Table 7.1]. The 37.5%-40% step was chosen as it represents the mid operating range of the tank and contains the 40% SP mark that is used for all tests herein.

Table 7.1 – PID controller tuning parameters calculated using the Cohen-Coon tuning rules

|  |  |  |  |
| --- | --- | --- | --- |
|  | Kc | τInt | τDer |
| P | 6.24 |  |  |
| PI | 5.58 | 31.37 |  |
| PID | 8.29 | 24.11 | 3.58 |

## Results

Tests were completed successfully on both the simulated and live system producing the results shown in [Figure 3] and [figure x] respectfully. It should be noted that an excess of noise was applied to the test system so as to verify the controller robustness.

Figure 3 – PID controller response to varying step changes whilst running in simulation

## Discussion

<Need Live System Data For This>

One of the main problems that was picked up on during lab testing was the presence of overshoot resulting in the process taking far longer than necessary to stabilise. This is primarily due to the tuning technique being a quarter decay method. In an industrial setting this will increase settling time between steady states, an effect of which is highly undesirable. An innovative solution to the problem is evaluated and tested within [Section 8].

# PID Controller Enhancement

It was noted in [Section 7.5] that the tuning method utilised results in a degree of overshoot before the process settles. The following section builds upon the foundation established in [Section 7] to remove the effect of overshoot altogether without sacrificing process response.

## Theory

This is primarily due to the quarter amplitude damping response aimed for by such a method. Tweaking the parameters obtained can decreased this overshoot at the cost of response time. For many industrial processes, this trade-off has to be made or alternate methods utilised. The overshoot is caused by accumulation of the integral error term. A trial enhancement to the PID program has therefore been made in which the integral error term is ‘bumped’ to a suitable value upon the PV error sign inverting. To utilise this method requires a secondary form of tuning to take place in which the integral error term is evaluated over a range of set points. This data is then used to form a correlation using standard curve fitting techniques.

## Program

A separate controller class available at *‘…/src/EnhancedPIDControl/PIDControllerEnhanced.py’* has been devised to account for the changes that introducing this method makes. Further upstream developments have also been included such as the inclusion of a configuration file for PID parameters. The class makes ‘*True’* a Boolean flag (SPC) upon the set point changing. Whilst this flag is ‘*True’*, upon every loop the sign of the error is checked for an inversion, i.e. the PV crossing the SP. Upon inversion, the integral error value is ‘bumped’ by recalculation using the correlated equation and the SPC flag made ‘*False’* to avoid repetition.

## Results

The controller proved to work as expected completely eliminating the overshoot and bringing the process directly to a steady state. A comparison of the effect is shown by [Figure 3] and [Figure 4] where *‘spErr’* represents integral error term. Within [Figure 4], the effect of the ‘bump’ is clear where at time 650s a sudden drop in both spErr and OP takes place. This has the desired effect with the PV staying within 0.5% thereafter. The standard PID controller takes an extra 75s to bring it back within this limit which accounts to the whole process of settling being 23% longer. Although on this scale of process it does not have a significant effect, on large tanks that take hours to fill this time scale could be in the order of hours.

Figure 4 - Standard PID controller 40% to 50% set point change response on simulated system

Figure 5 – Enhanced PID controller 40% to 50% set point change response on simulated system

## Discussion

The primary problem that exists with utilising this method is that if the process changes in some way, the correlation originally calibrated will be incorrect and need reconfiguring. Although this is a disadvantage, it exists for PID controllers in general as in any case of process change, the main PID parameters would also need reconfiguring. It is therefore assumed both processes could take place at the same time. A secondary problem is that in a slow moving process, going between different set points to gather such data may take large amounts of time or not be possible altogether. Further developments could be made by including the ability to ‘bump’ the integral error upon large disturbances.

# System Identification

Many different advanced control algorithms utilise system models to predict the future behaviour of the system providing a greater level of control. These models utilise parameters specific to a system that can either be calculated in a variety of ways. The following section covers a variety of these techniques and they’re effectiveness when applied to the system in question.

## Theory

Theory

Numerically, [Equation X] can be solved by applying Euler’s method.

Equation 9.1

Equation 9.1

Equation 9.2

### Numerical Solution

Simultaneous

### Batch Least Squares

BLS

### Recursive Least Squares

RLS

## Program

# Minimum Variance Control

…

# Discussion

Why it happened?

Equiptment limitations

# Conclusion

Sum it all up and add future work

1. Improve datalogging class for any no of variables

# References

Collins, G., 2015. *Github - Pymodbus.* [Online]   
Available at: https://github.com/bashwork/pymodbus  
[Accessed 2015 10 01].

Honeywell, 2014. *Modbus RTU Serial Communications User Manual.* [Online]   
Available at: https://www.honeywellprocess.com/library/support/Public/Documents/51-52-25-66.pdf  
[Accessed 01 10 2015].

Leech, A. D., 2016. *Level Control Research Project Repo.* [Online]   
Available at: https://github.com/FlaminMad/researchProject  
[Accessed 2016].

Leech, A. D., 2016. *LogBook.* 1st ed. s.l.:s.n.

Michigan Techonological University, 17. *Standard PID Tuning Methods.* [Online]   
Available at: www.chem.mtu.edu/~tbco/cm416/tuning\_methods.pdf  
[Accessed 14 10 2015].

mstarlabs, 1994. *Building PID Controls in Software.* [Online]   
Available at: http://www.mstarlabs.com/apeng/techniques/pidsoftw.html  
[Accessed 10 14 2015].

National Instruments, 2011. *PID Theory Explained.* [Online]   
Available at: http://www.ni.com/white-paper/3782/en/  
[Accessed 01 04 2016].

Python Software Foundation, 2001. *Python.* [Online]   
Available at: https://www.python.org/about/  
[Accessed 30 03 2016].

Shahrokhi, M. & Zomorrodi, A., 2012. *Comparison of PID Controller Tuning Methods.* [Online]   
Available at: http://www.ie.itcr.ac.cr/einteriano/control/clase/Zomorrodi\_Shahrokhi\_PID\_Tunning\_Comparison.pdf  
[Accessed 01 04 2016].

Smuts, J., 2011. *Ziegler-Nichols Open-Loop Tuning Rules.* [Online]   
Available at: http://blog.opticontrols.com/archives/477  
[Accessed 20 10 2015].

# Appendices

## Appendix A

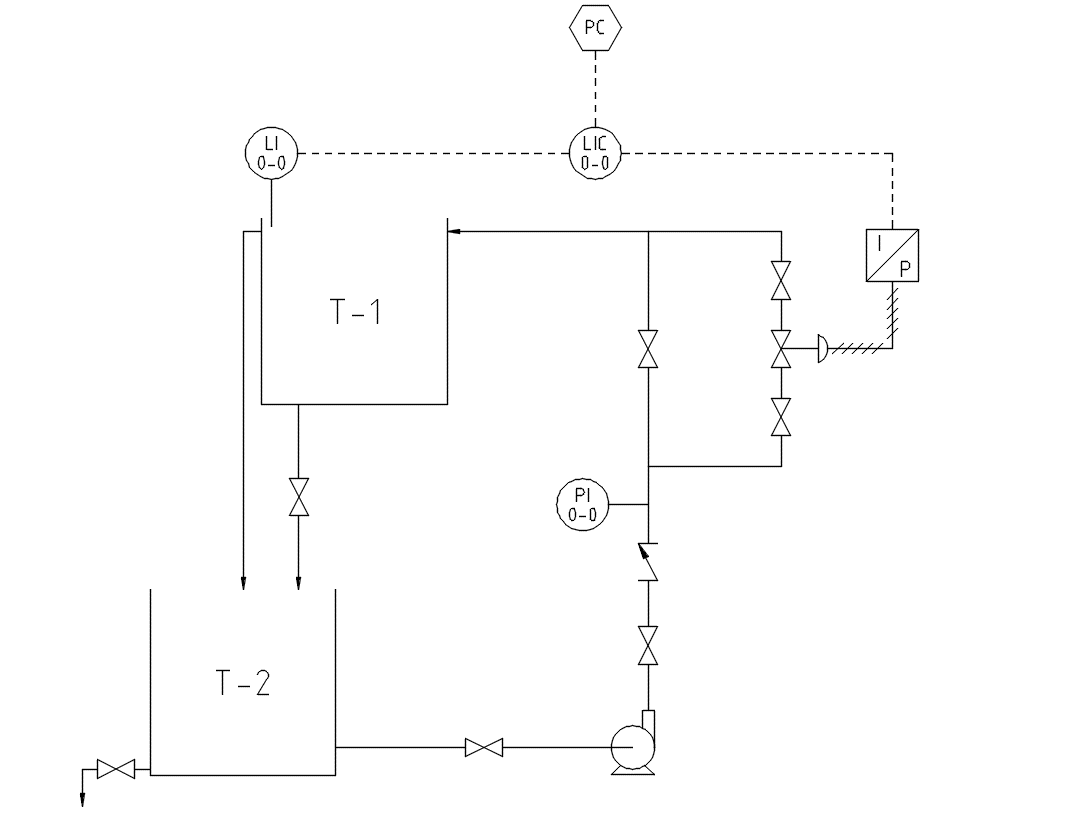


Figure 6 – Experimental Process and Instrumentation Diagram

## Appendix B

Figure 7 – Step test performed utilising a +2.5% step starting at 35.0%

Figure 8 – Step test performed utilising a +2.5% step starting at 37.5%

Figure 9 – Step test performed utilising a +2.5% step starting at 40.0%