Comparison of Model Predictive and Advanced PID Control

Case Study

Submitted by

Nagaraj Prakash - 11139349

Prathamesh Kolekar - 11139708

Srinidhi Subbanna - 11138941

Under the guidance of Prof. Dr. Rainer Scheuring

Technology
Arts Sciences
TH Köln

Master Automation & IT

March 13, 2021

Contents

1 Introduction

Level control is most important and critical in successful operation of most of the chemical plants, refineries and Petrochemical Industries. As it can be managed through proper control of level and flow which in turns help in desired production rate and inventory management system in a production plant. Different control strategies can be implemented to achieve certain level of the tank. One of the classical approaches would be implementing advance process control with the help of cascade structure of flow and level loop in conjunction with override control to maintain certain level of tank. On the other hand, similar control objective can be achieved through Model Predictive Control or hereon abbreviated as MPC for tank level control.

In this case study, we will focus on different aspects to achieve level control using classical control strategy using Override control and model predictive control. To make it more realistic, we will implement override control strategy in plant simulation software called UniSim and for another approach, we will design MPC in python using suitable library called DO-MPC and communicate it using OPC-DA communication protocol with UniSim plant.

During course of this case study, we have found result which will be summarized in subsequent sections, also challenges and future work shall also be discussed later part of this report.

2 Plant/Process Introduction

The best way to learn something new is to start with something simple. Although the concept of MPC and Override control was not new, implementing them seemed challenging for the team when the case study was started. We opted a simple process of tank level control so that the project objectives are met within the stipulated amount of time.

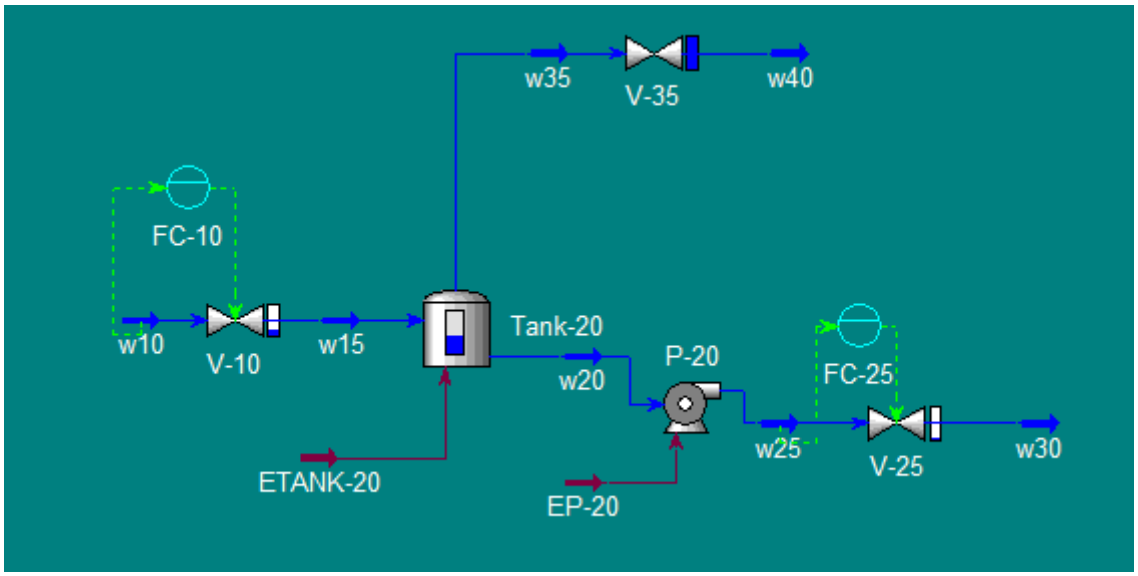


Figure 1: Unisim Model of the Plant

The model* has a tank whose level should be controlled between 20 and 80%.We use an outlet valve (Linear Valve) as Final Control Element to control level. The system consists of a pump between the tank and the valve to nullify the effect of potential energy of the water column. The pump renders uniform pressure on the outlet valve. This makes the system easier to model. The System has an input disturbance, which can be controlled by the input valve. The control system we design should adjust itself dynamically to cope up with the disturbances. The various elements of our model is described below. These specifications can be used to reproduce the same results later. Alternatively, the project can be accessed from the GitHub link shared in the references.

2.1 Elements Description

2.1.1 Pump

A pump is used to move fluid from one process to another process. Centrifugal pumps are utilized to transport fluids by converting the rotational kinetic energy to the hydrodynamic energy of the fluid flow. The elevation is at the ground level and the default efficiency of 75 percent is used.

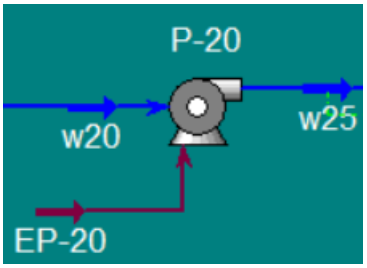


Figure 2: Pump

* The Unisim model for Override control application is provided by Prof. Dr. Rainer Scheuring, along with the override controller implementation.

PUMP	P-20
Inlet	W20 (Tank-20)
Outlet	W25 (V-25)
Energy	EP-20 (P-20)

Table 1: Pump Inlets and Outlets

Specifications	P-200
Operating Mode	Centrifugal
Efficiency [%]	75.00
Speed [rpm]	0-60
Base elevation relative to ground level [m]	0.000

Table 2: Pump Parameters

2.1.2 Valve

A valve is used to regulate, directs and controls the flow of a fluid by opening, closing it and also relieving and regulating pressure in fluid.In our project we have used valves in three different types like valve for regulating the flow , valves for draining the fluid and valves for regulating the air pressure coming from the tanks.

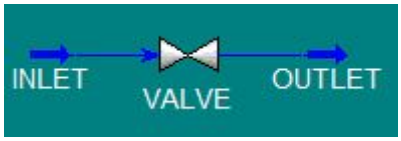


Figure 3: Valve

Valves	V-10	V-25	V-35
Inlet	W10	W25 (P-20)	W35 (Tank-20)
Outlet	W15 (TANK-20)	W30	W40

Table 3: Control valves Inlets and Outlets

Specifications	V-10	V-25	V-35
Valve Characteristic	Linear	Linear	Quick Opening
Actuator dynamics	First Order (time constant=3s)	First Order (time constant=3s)	Instantaneous
Conductance Cv	25	33	20
K Value Damp Factor	0.95	0.95	0.95
Base Elevation relative to Ground level	5	0	6

Table 4: Control Valves Specifications

2.1.3 Tank

The tank used for filling the fluid. In our project the tank we used is a flat cylinder and it has a inlet, outlet and an energy stream. The tank we used is of 12.57 m^3 volume with 4 m height and 2 m diameter.

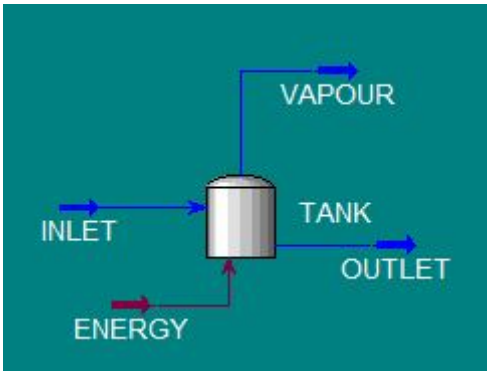


Figure 4: TANK

Tank	**Tank-20**
Inlet	W15 (V-10)
Outlet	W20 (P-20)
Energy	ETANK-20

Table 5: Tanks Inlets and Outlets

2.1.4 Controller

In industry, controllers are used to regulate flow, temperature, pressure, level etc. In our scenario, we are using controllers for regulating the flow through the valves. The controllers for flow regulating are FC-10 and FC-25, the corresponding specifications of these valves are:

Figure 5: CONTROLLER

Controller	**FC-10**	**FC-25**
Inlet	W10	W25
Outlet	V-10	V-25

Table 6: Controllers Inlets and Outlets

Element	**PV Min** $[m^3/h]$	**PV Max** $[m^3/h]$	Proportional Constant (Kc)	Integral time constant (Ti)
FC-10	0.0	50.00	0.7	0.1s
FC-25	0.0	50.00	0.7	0.07s

Table 7: Flow Control elements for Analog Valves Parameters

2.1.5 Stream

Input stream: This stream is set with a composition of 1 H2O and 0 air, therefore the stream consists of water with a temperature of 25 ℃.

Output stream: This is also same as input stream in composition and temperature.

Figure 6: STREAM

2.2 Control strategy on Final Control Element

It is well known fact that a level control process is very sluggish and thus, a simple control loop measuring the Level and adjusting the Final Control Element, the outlet valve, would perform bad in presence of disturbance. A well known Industrial strategy in such scenarios is to use a cascade control.

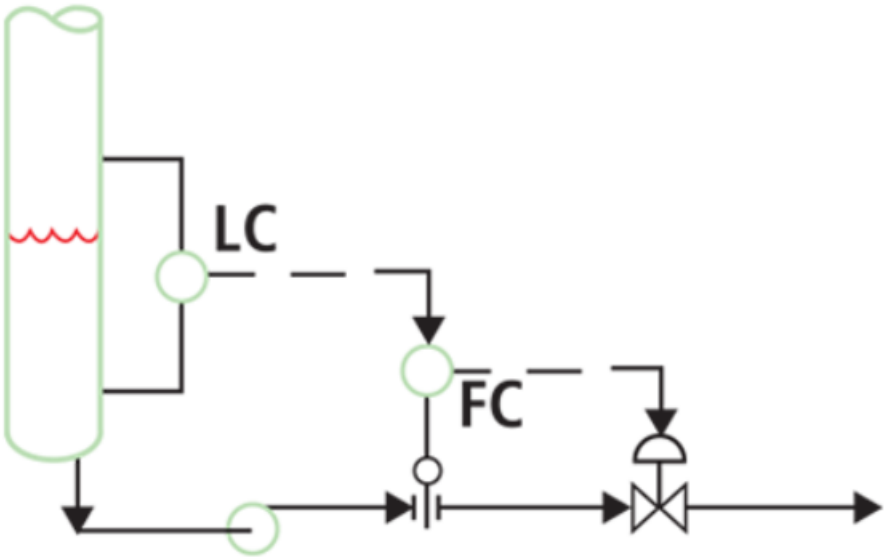


Figure 7: Cascade Control of Level and Flow

The fast response of the secondary loop will act as a early inhibition of the effect of disturbance. While designing a Cascade control, it is to be made sure that the control loop (primary and secondary) should be directly correlated and the secondary loop should be considerably faster than the primary loop. In our scenario, the slower Level control loop is augmented with the faster flow control loop to have a better control over the disturbances. When introducing a Cascade loop, there is no need of a new control element. However, a new PID controller and a new sensor is necessary. The primary PI/PID controller generates output from the error in Level. This output is scaled in the range of the Flow min and max values and fed as setpoint to the seconadry loop (Flow control loop). The secondary loop controls the final control element directly based on a setpoint control. The disturbances in the system generates a new setpoint for the secondary loop. The faster secondary loop copes up with the disturbances, reducing its effect and providing an early inhibition.

2.3 Control Objective

The main control objective in our scenario is to make sure that the level remains within the range of 20 to 80%. The following material balance equation explains mathematically that the level is a rate of accumulation of effective flow in and out of the system.

$$\text{accumulation of Level} = \text{Flow into the system - Flow out of the system}$$

or

$$\frac{dV}{dt} = q_{in} - q_{out}$$

where,
V is the volume or level
q_in is the in flow rate
q_out is the out flow rate

The in flow rate is an unknown to the controller or in other words, it is a disturbance to the system.

Thus, to our control system, the above equation can be written as,

$$\frac{dV}{dt} = d - q_{out}$$

Where,
d is the disturbance.

We are employing an Override Control strategy and an MPC controller to address the control problem. Both of these approaches work great on the given control problem but they have their differences. These differences are studied in the case studies and reported in the further sections.

3 Project Architecture and Environment

3.1 OPC DA server

The OPC DA is Server-Client architecture based. The OPC DA Licence of Unisim is an OPC DA client. Python offers numerous libraries for OPC UA but unfortunately we could not find a working Python OPC DA server. So, we have decided to use a third party as server. We use 'Matricon OPC server for simulation and testing' as the server to which the Python and Unisim clients can connect to and exchange information can be exchanged between the clients through the Server. We need two real/floating type variables for exchanging the current Process value (Level) and the next control value (Flow) between Unisim and Python. Thus, we do not need more than 2 variables to be hosted on the server. The Matricon OPC simulation server does not allow users to create any new variables. However they offer some built-in objects which can be used for our purpose. The naming of these objects do not match our convention however, we can assign these objects an alias name. [6]

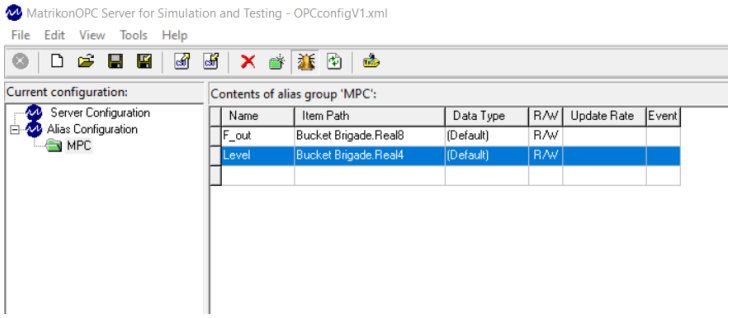


Figure 8: Matricon OPC DA server

Steps to configure the Alias:

1. Right click on the Alias configuration and add a new Alias group (MPC).

2. Click on the empty rows to add a new alias name, fill the details in the dialog box.

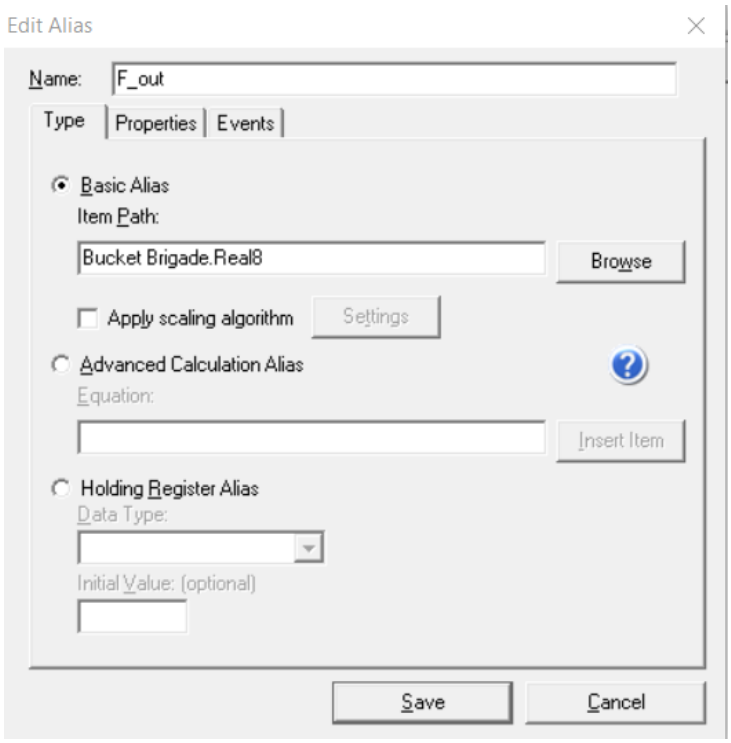


Figure 9: Configuring Alias for OPC Variables

3. As shown in the above figure, an Alias name of our convention is entered. We have found that the variables from Bucket Brigade sections suits our purpose and we have used real type variables from this section.

To test these settings we have used Matricon OPC client/explorer. The client can browse the servers on the same machine or on the network. For OPC DA connections outside the device, DCOM configurations must be done (Section 3.4). The preferred server is chosen and connect option is clicked.

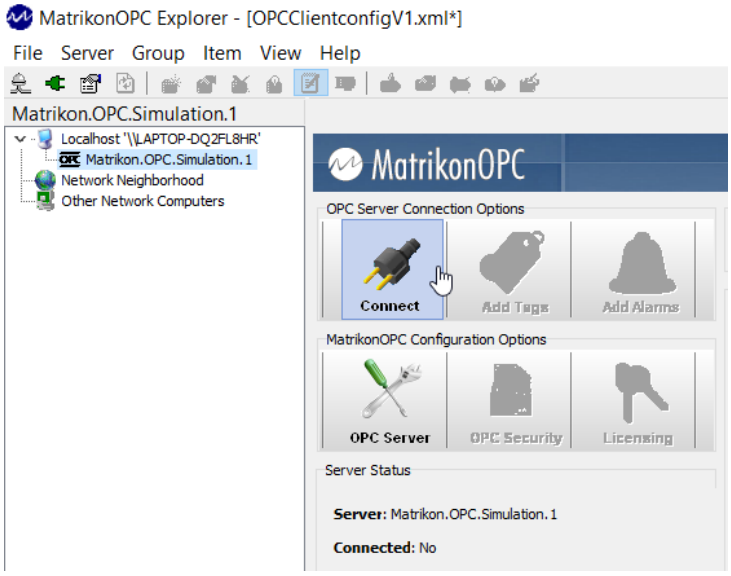


Figure 10: OPC DA client from Matricon

To see the tags configured in the server, we click on add tags button which opens up the following dialog box.

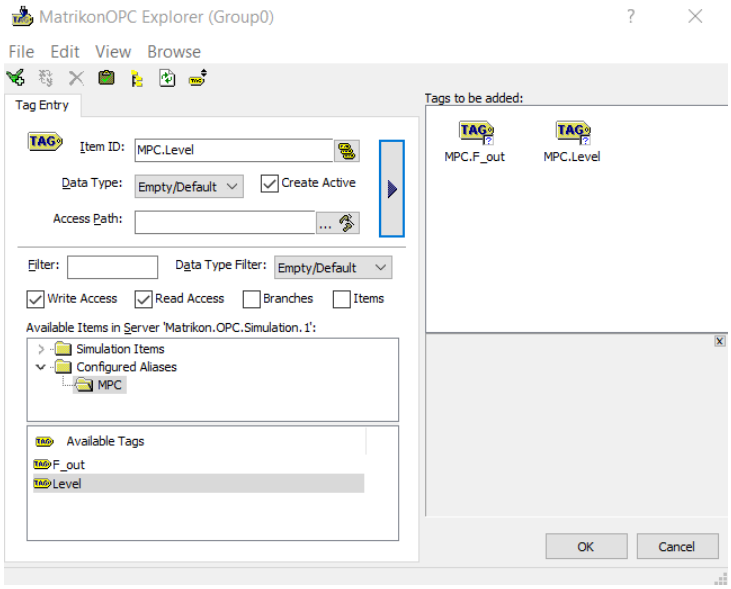


Figure 11: Browsing server objects on client

Select the tags from the 'configured aliases' section and add them to the monitor window. The monitor looks as shown below. The tags can be read and written from here.

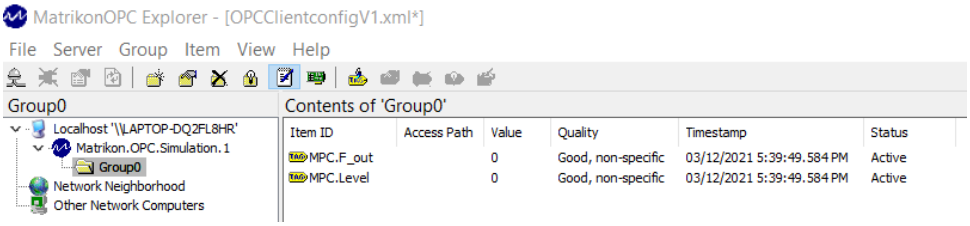


Figure 12: Tag monitoring on the OPC client

3.2 Unisim

Unisim is a process modelling software for process design, simulation, safety studies, operations monitoring and debottlenecking, process optimization and business planning.

UniSim Design Suite offers an accurate and intuitive approach for process modeling that helps engineers to construct steady-state and dynamic plant for control design, safety studies, performance analysis, troubleshooting, operational enhancement, business planning and asset management models.

UniSim Design Suite is a software family of online and off-line process design and optimization tools from Honeywell. UniSim applications help you collect and exchange process information, increase plant profitability and optimize returns on simulation technology investments, give users the power to assess process workflows, equipment sizing and rating requirements.

UniSim Design Suite offers:

- An integrated steady-state and dynamic environment throughout a project or plant asset life-cycle to quickly reuse, upgrade and transition the process models.
- A user-friendly GUI that lets engineers quickly view and envision information about the process and identify trends.
- Industry standards built-in, reducing the need to scan the literature while sizing and rating equipment.
- Integration of third-party specialized technology to provide the latest technological approach for process simulation.
- Integration of process historians, DCS and safety technologies, and other advanced applications to optimize the advantages of green-field, brown-field, and revamp projects.

Benefits:

- Improved Process Designs
- Equipment/Asset Performance Monitoring
- Reduced Engineering Costs

Features:

- Easy-to-Use Windows Environment
- Comprehensive Thermodynamics
- Comprehensive Unit Operation Library
- Flexible License Manager

[3]

3.2.1 Unisim Version

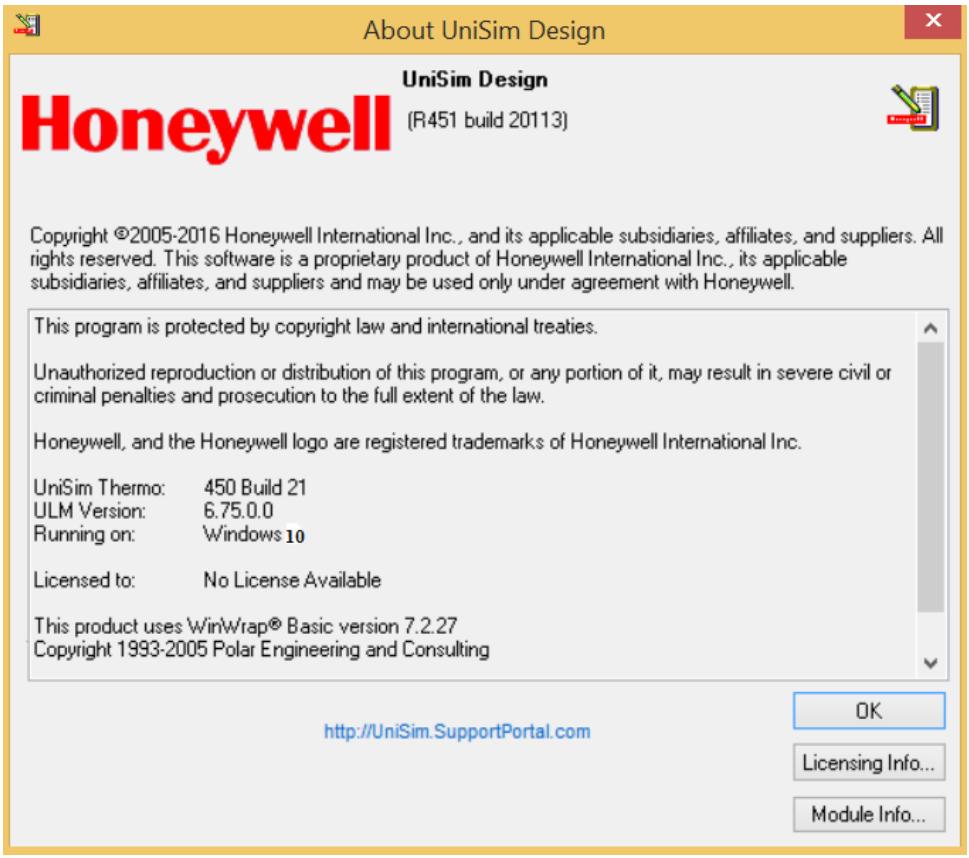


Figure 13: Unisim Version

The implementation of the industrial process is done in unisim R451 version for this case study.

3.2.2 Unisim License Manager

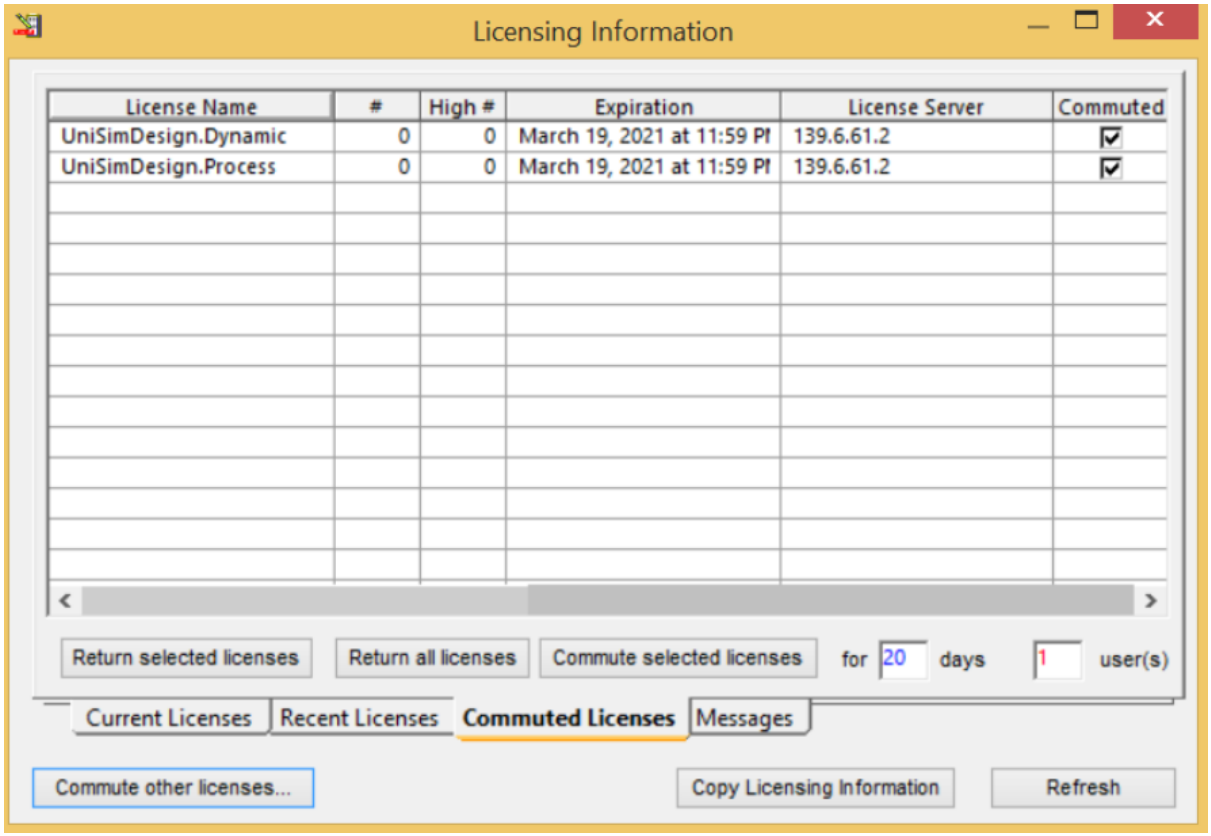


Figure 14: License Manager

We have commuted the UnisimDesign.Dynamic and UnisimDesign.Process licenses for the required number of days.

UNISIM acts as a client in our case study. The windows machine in which Unisim Simulation software is running is windows 10, there is no additional set up need to be done. Both client and server should be in the same network. For our case study, both the machines are connected to the same LAN Adapter, pinged both the machine and they are able to respond.

3.3 Python

Python is an interpreted, high-level and general-purpose programming language. Python is often used by software developers as a support language for build control and management, debugging, and a variety of other tasks. Python is one of the most versatile programming languages available because it has a simple syntax and is not too complex, allowing natural language to take center stage. Python programs can be written and implemented even quicker than most programming languages due to their ease of understanding and use.

3.3.1 Python Version

The implementation of the MPC controller is done in python version — for this case study. Python acts as a client in our case study. Matrikon OPC is the server and Unisim, Python are OPC clients.

3.3.2 Python Libraries

Importing various packages and libraries required to initialize the setup. The short description of each package is below:

- **Numpy**:Numerical python based computational library.

- **CasADi**: CasADi is an open-source tool for nonlinear optimization and algorithmic differentiation.
- **Matplotlib**: Library to plot Graphs.
- **pdb**:PDB stands for "Python Debugger", and is a built-in interactive source code debugger with a wide range of features, like pausing a program, viewing variable values at specific instances, changing those values, etc.
- **sys**: The python **sys** module provides functions and variables which are used to manipulate different parts of the Python Runtime Environment.
- **do_mpc:** do-mpc is a comprehensive open-source toolbox for robust model predictive control (MPC) and moving horizon estimation (MHE).
- **os**: The OS module in Python provides functions for interacting with the operating system.
- **time**: time() The time() function returns the number of seconds passed since epoch.
- **pywintypes**: pywintypes is part of the Python for Windows extensions, otherwise known as pywin32.

3.3.3 DO-MPC

do-mpc is an open-source toolbox for robust Model Predictive Control (MPC) and Moving Horizon Estimation (MHE). It includes methods for dealing with complexity and time discretization, as well as the efficient formulation and solution of control and prediction problems for nonlinear systems. Do-mpc has a modular structure with simulation, estimation, and control modules that can be conveniently generalized and combined to meet a variety of applications. [2]

In summary, do-mpc offers the following features:

- Nonlinear and economic model predictive control
- Robust multi-stage model predictive control
- Moving horizon state and parameter estimation
- Modular design that can be easily extended
- Support for differential algebraic equations (DAE)
- Time discretization with orthogonal collocation on finite elements

3.3.4 CasADi

CasADi is an open-source tool for algorithmic differentiation and nonlinear optimization. It allows for the quick and efficient implementation of various numerical optimal control methods, both in an offline setting and for nonlinear model predictive control (NMPC). CasADi helps you save time while prototyping concepts, overcoming complicated technical challenges, and creating competent optimization software. It has a wide range of scholarly and industrial applications.

- **Algorithmic Differentiation (AD)** : CasADi's backbone is a symbolic framework implementing forward and reverse mode of AD on expression graphs to construct gradients, large-and-sparse Jacobians and Hessians. These expression graphs, encapsulated in Function objects, can be evaluated in a virtual machine or be exported to stand-alone C code.
- **Dynamic systems** : Initial value problems in ordinary or differential-algebraic equations (ODE/DAE) can be calculated using explicit or implicit Runge-Kutta methods or interfaces to IDAS/CVODES from the SUNDIALS suite. Derivatives are calculated using sensitivity equations, up to arbitrary order.
- **Nonlinear and quadratic programming** : Nonlinear programs (NLPs), possibly with integer

variables (MINLP), can be solved using block structure or general sparsity exploiting sequential quadratic programming (SQP) or interfaces to IPOPT/BONMIN, BlockSQP, WORHP, KNITRO and SNOPT. Solution sensitivities, up to arbitrary order, can be calculated analytically. Quadratic programs (QPs), possibly with integer variables (MIQP), can be solved using a primal-dual active-set method or interfaces to CPLEX, GUROBI, HPMPC, OOQP or qpOASES.

- **Composition of the above** : CasADi offers a rich set of differentiable operations for its matrix-valued expression graphs, including common matrix-valued operations, serial or parallel function calls, implicit functions, integrators, spline-based lookup tables, and external codes. These building blocks allow the user to code a wide variety of optimal control problem (OCP) formulations. For example, a single shooting code can be created by embedding a call to an integrator in an NLP declaration. [4]

3.4 DCOM settings

3.4.1 Install OPC Core Components

The first important step is to install the OPC core components from the OPC Foundation to enable the protocol to be fully used by the PCs. It must be installed in all the computers that will be used to communicate trough OPC DA. This core components can be found at `https://opcfoundation.org/developer-tools/samples-and-tools-classic/core-components/`

[21]

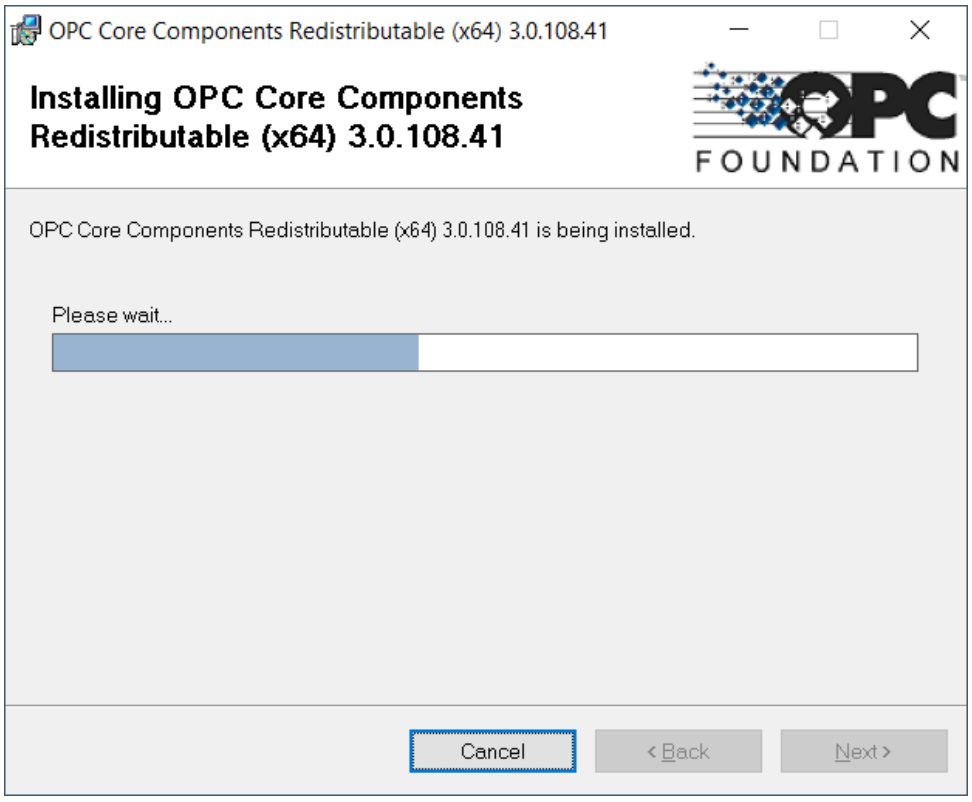


Figure 15: Install OPC Core Components

3.4.2 Ping command

In order to be able to establish a successful DCOM communication, the first requirement is that a ping between the devices is able to respond correctly. This ensures the basic Windows communication has been establish in between them and then it is possible to intend to create a more complex protocol setup. The successful reply from the Ping command is shown in Fig. 16.

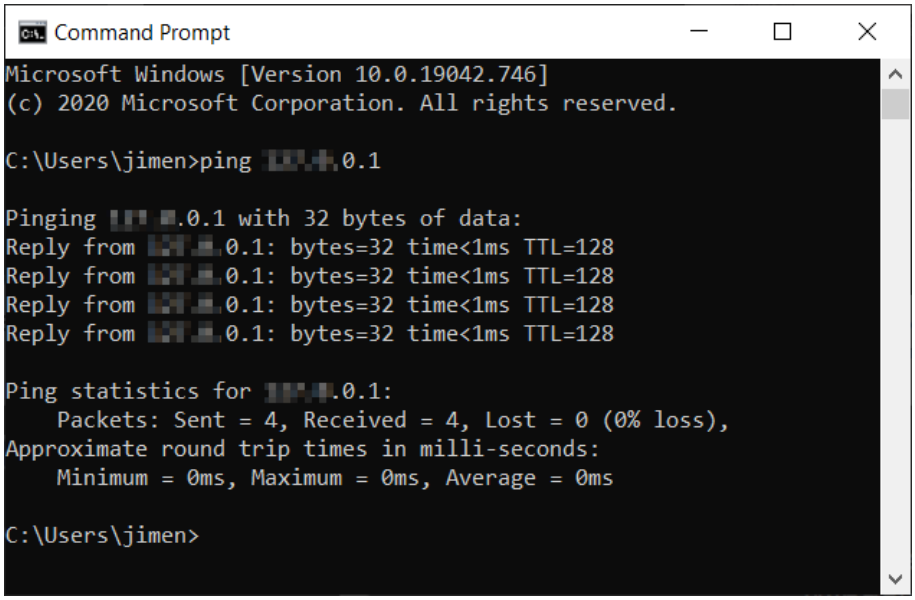


Figure 16: Ping between computers

3.4.3 User creation

After the basic communication test. The computers need to have a known user and password from the other computer to be able to communicate. To create the user, open the Computer Management tool from windows. It can be accessed with a right mouse click on the Windows start button and select the Computer Management option.

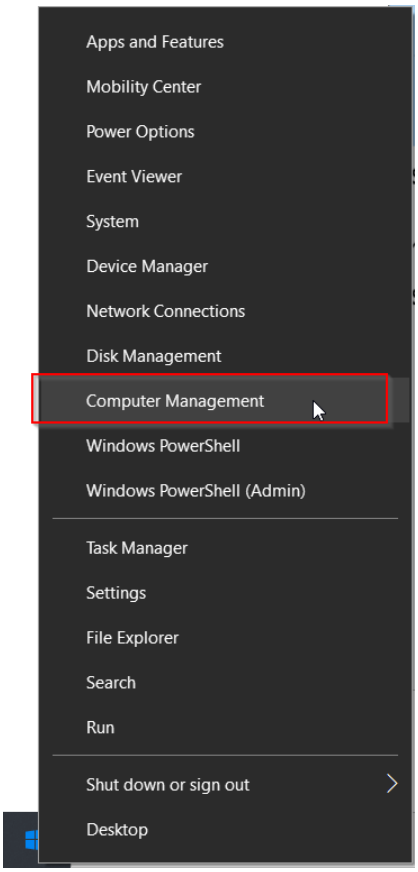


Figure 17: Open Computer Management

Once the Computer Management is open, create a new user in the folder "System Tools / Local Users and Groups / Users". This new user will be used for the communication and must match in both PCs.

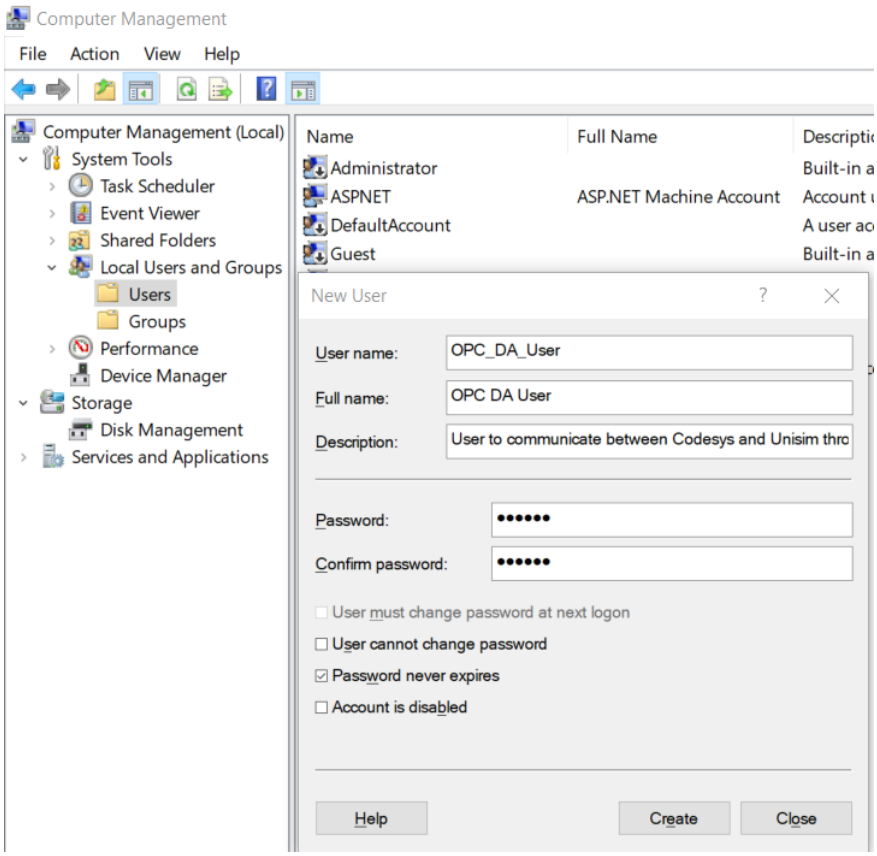


Figure 18: New Windows User creation

This user must be logged in at the OPC Client to open the program that will be used to read and write the tags. From the OPC Server side, the user can or can not be used to run the application.

The next step is to include the new user into the "Distributed COM Users". This is configured also at the Computer Management tool in the "System Tools / Local Users and Groups / Groups" section as shown in Fig. 19.

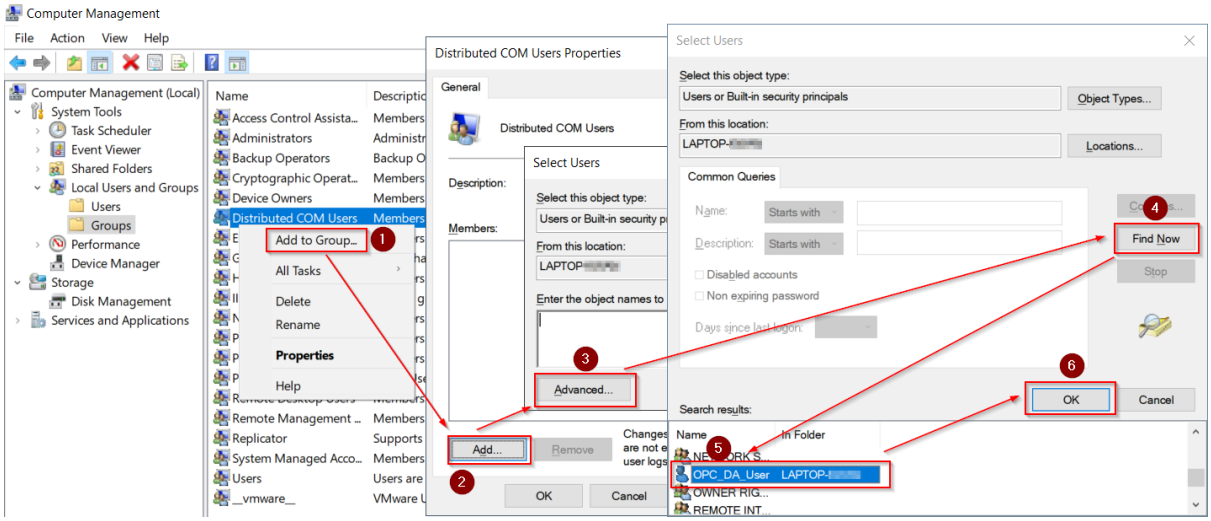


Figure 19: Insert User into Windows DCOM Group

3.4.4 Windows Firewall, port 135

The next process to be done in order to have a successfully communication is to setup correctly the DCOM elements into the Windows Firewall as the default Windows configuration will keep blocked all inbound communication from third party devices. First, open the Windows Firewall windows with the use of the "wf.msc" command at the Windows Run tool; this tool can be executed pressing the "Windows+R" keys at the same time. [1]

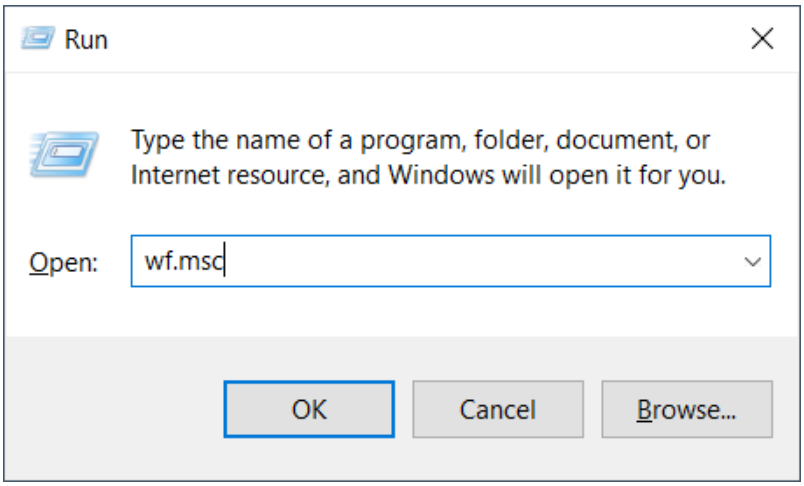


Figure 20: Open Windows Firewall

The port used for the DCOM communication is the number 135. At the Windows Firewall window, right mouse click on the Inbound Rules option, select "New rule", select "Port" and click on "Next".

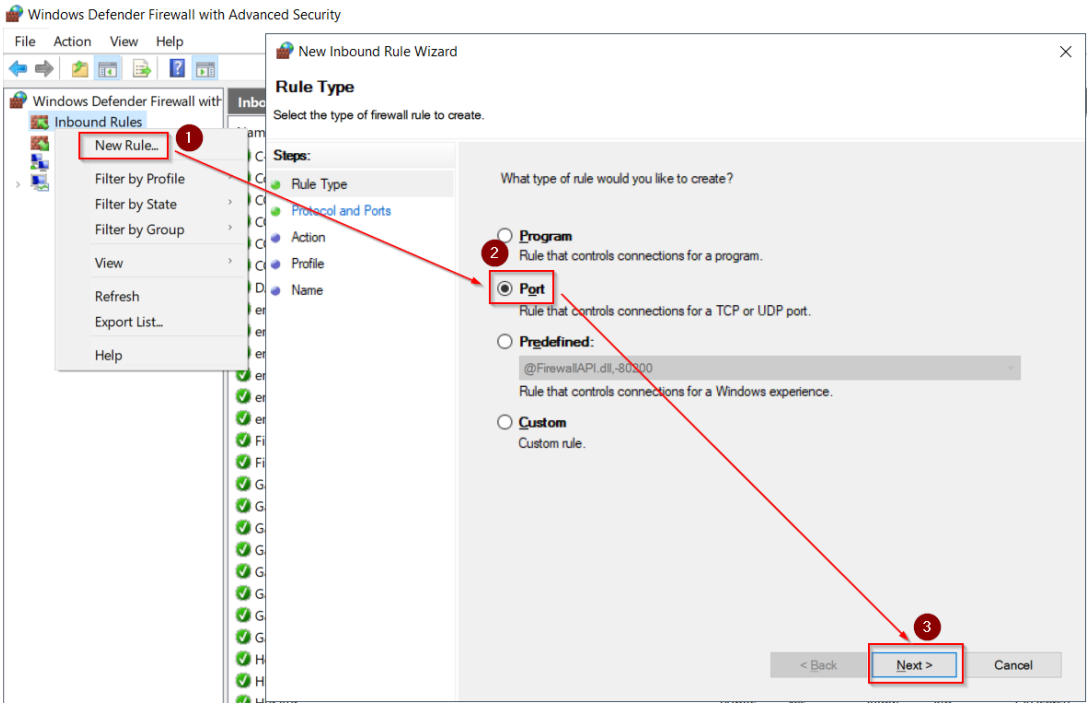


Figure 21: Create Port Inbound Rule

Select the TCP Rule, specify the port 135 and click on "Next".

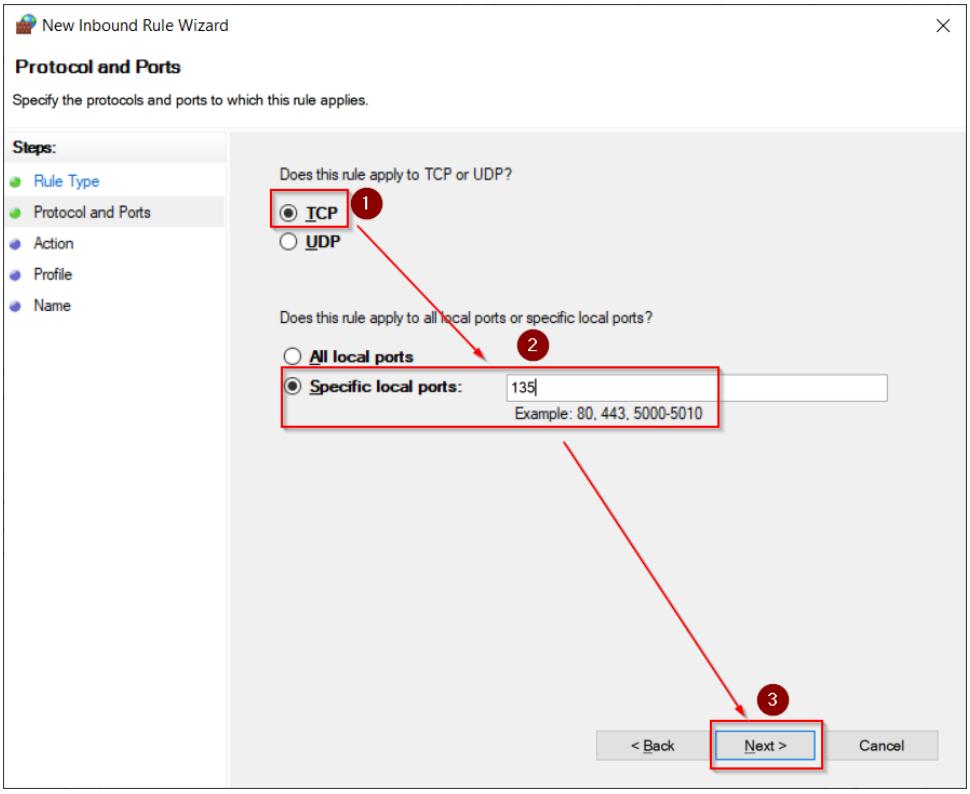


Figure 22: Select Port 135

Select "Allow the connection" and click on "Next". This will configure the proper inbound access through the port 135.

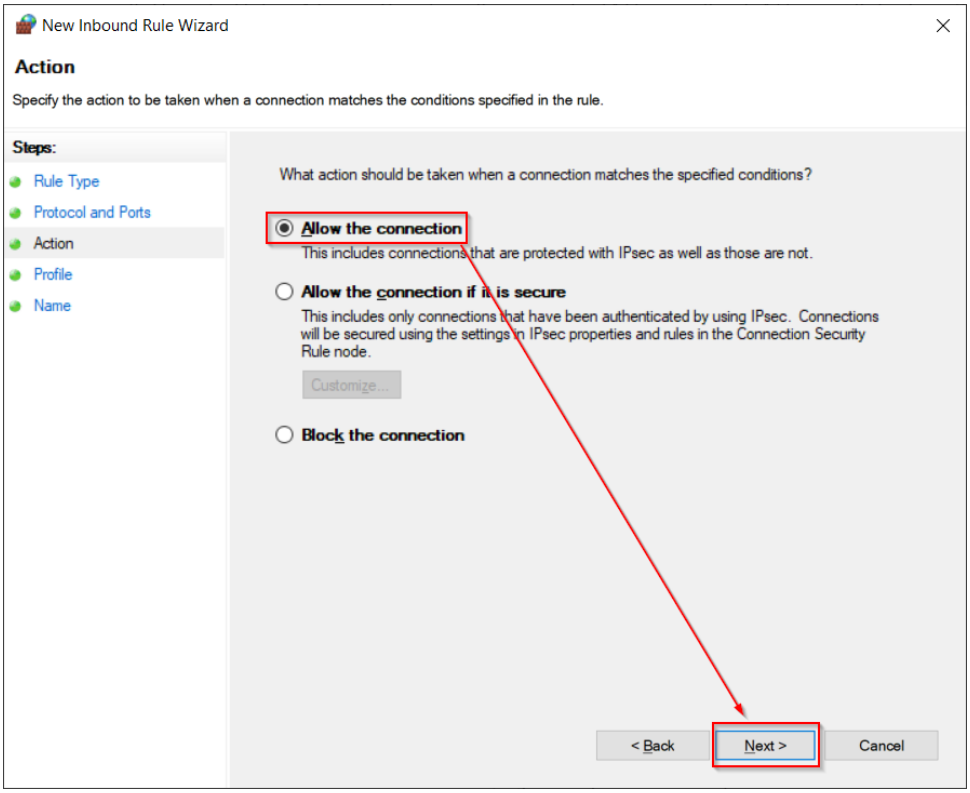


Figure 23: Allow the connection

The final step is to select the proper Profiles where the rule should apply. In order to increase security, select only the needed profile. If it is not known which network profile you will use, select the three of them.

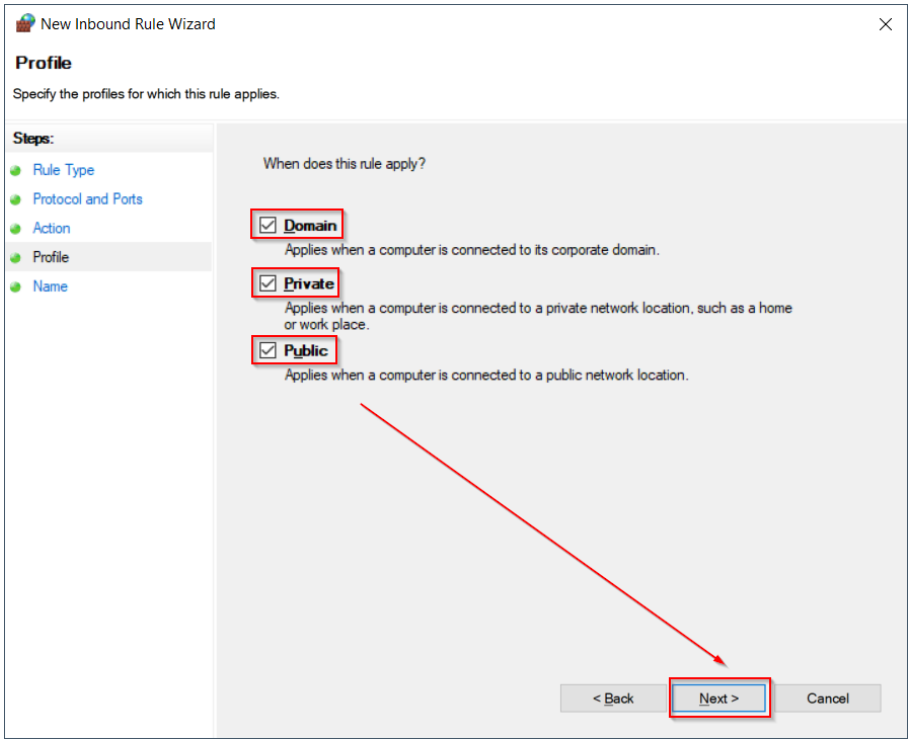


Figure 24: Select required Profiles

The final step is to assign a name for the created rule. Optionally it is possible to add a description to extend the explanation for future reference of the created rule.

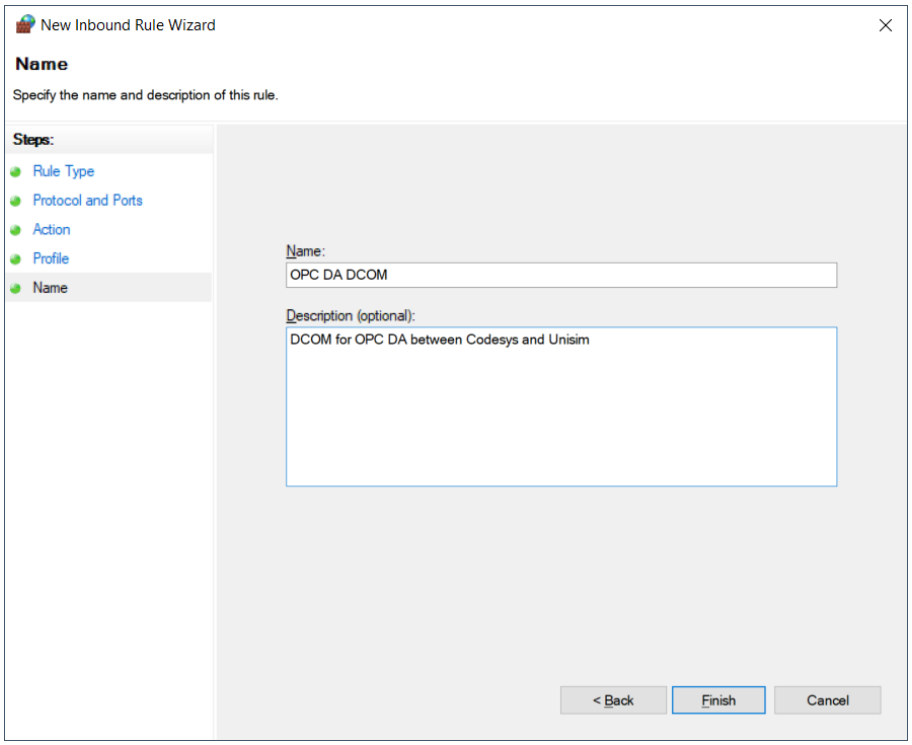


Figure 25: Assign name to new Inbound Rule

3.4.5 Optional - Additional Inbound Rules configuration

The last changes may be sufficient for most PCs to communicate correctly. But it can be the case that this changes still do not allow the PCs to communicate through the DCOM protocol. In this case, at the newly created Inbound Rule properties, select the configuration at the tab "Protocols and Ports" as shown in Fig.

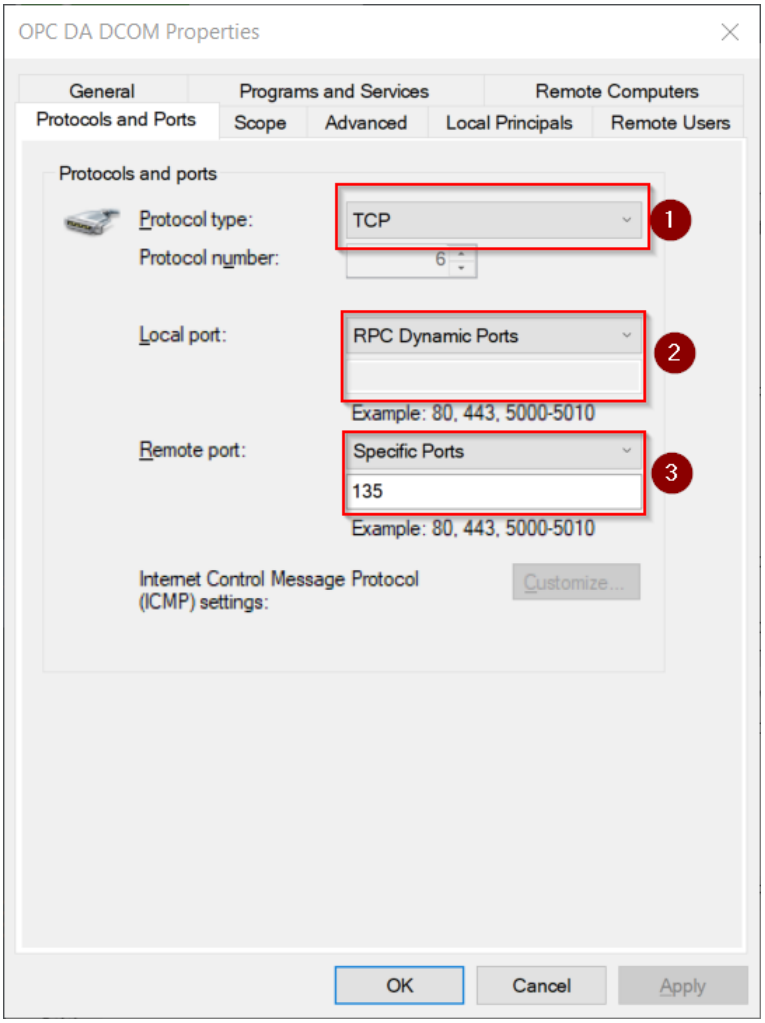


Figure 26: Changes in DCOM Inbound Rule Properties

3.4.6 Windows Firewall, OPC ENUM

Apart from the Inbound Rule for port 135, the PC also requires that the program that will be used as the OPC Server has an Inbound Rule to ensure the communication is enabled. In this context, the program used is called OPC ENUM. [1]

The first step is to create also an Inbound Rule. The rule type needs to be "Program".

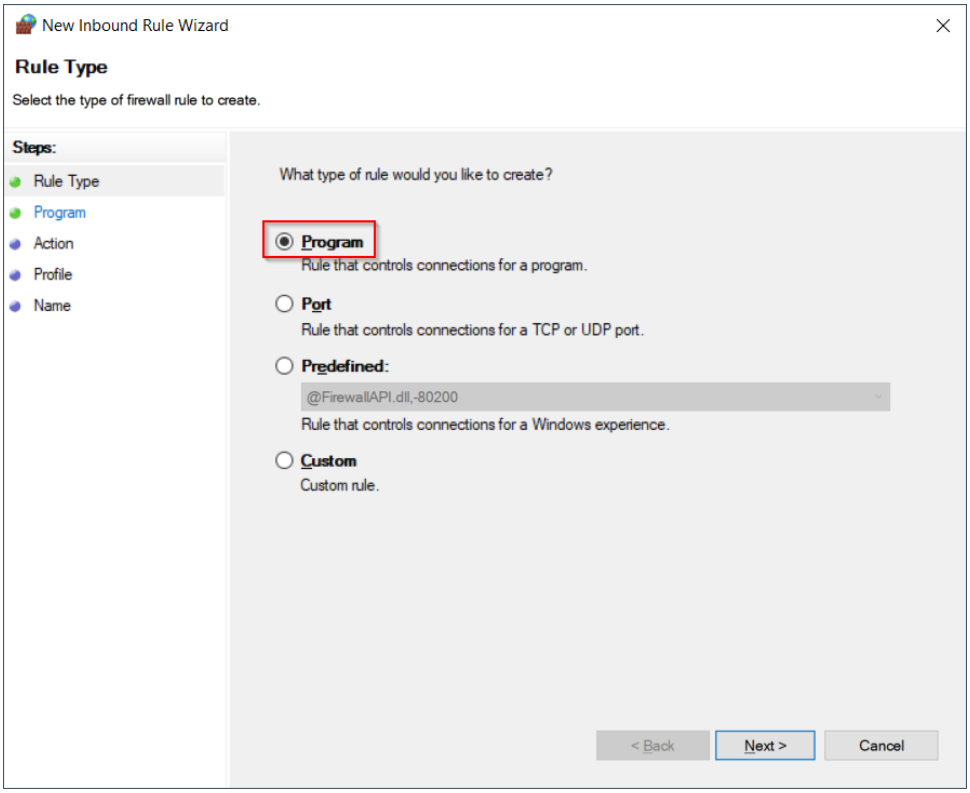


Figure 27: Create Program Inbound Rule

The program to be included in the rule is located in the C:\Windows\SysWOW64 folder.

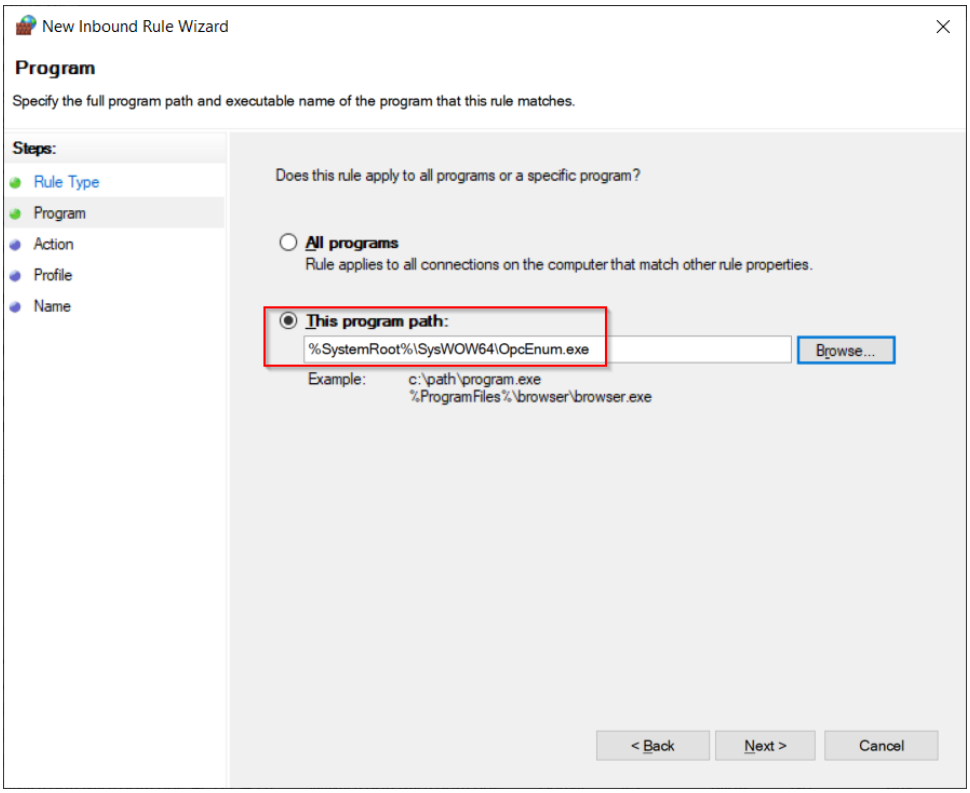


Figure 28: Select OPC ENUM path

Select "Allow the connection" and click on "Next" button.

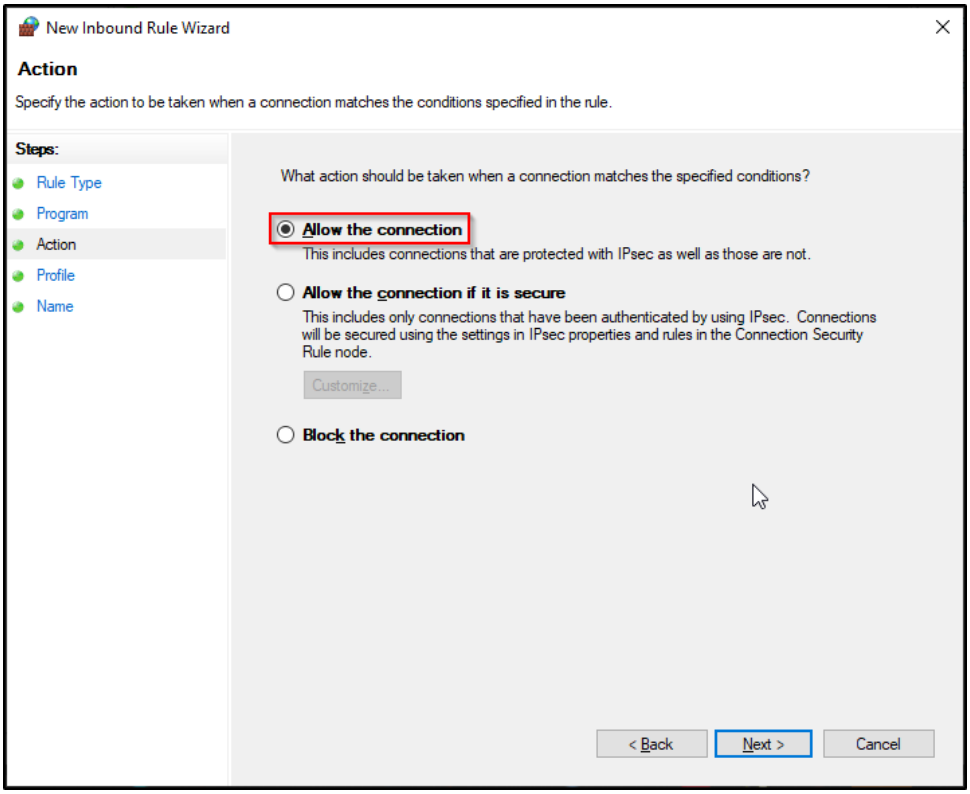


Figure 29: Allow the Connection

The next step is to select the proper Profiles where the rule should apply.

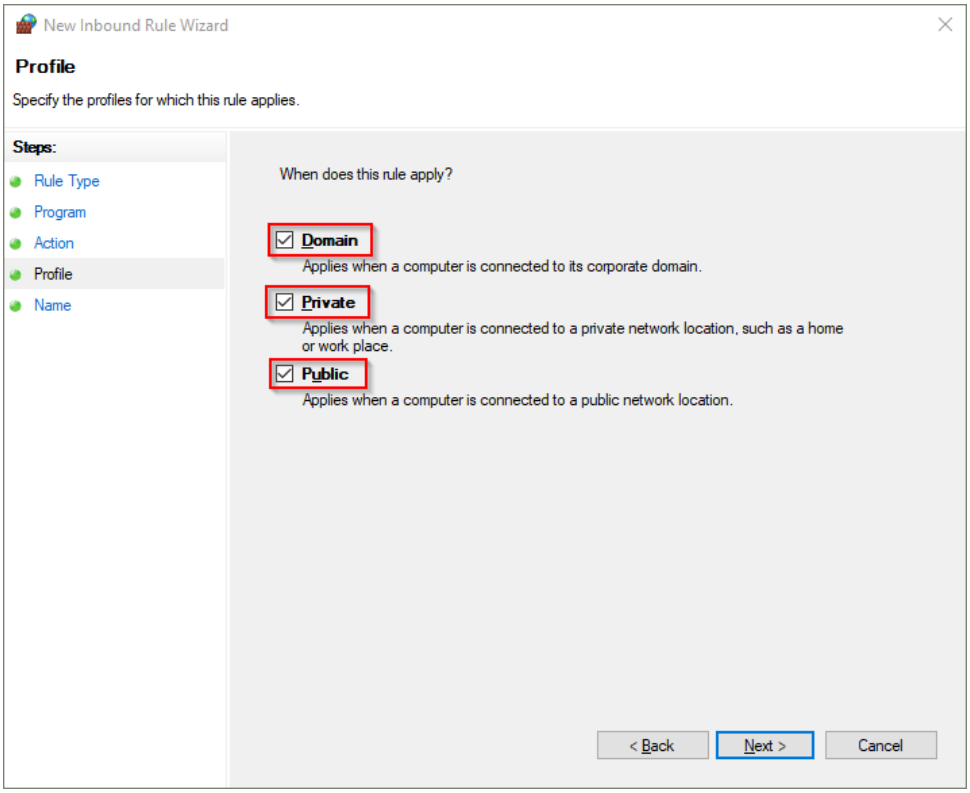


Figure 30: Select required Profiles

The final step is to assign a name for the created program rule.

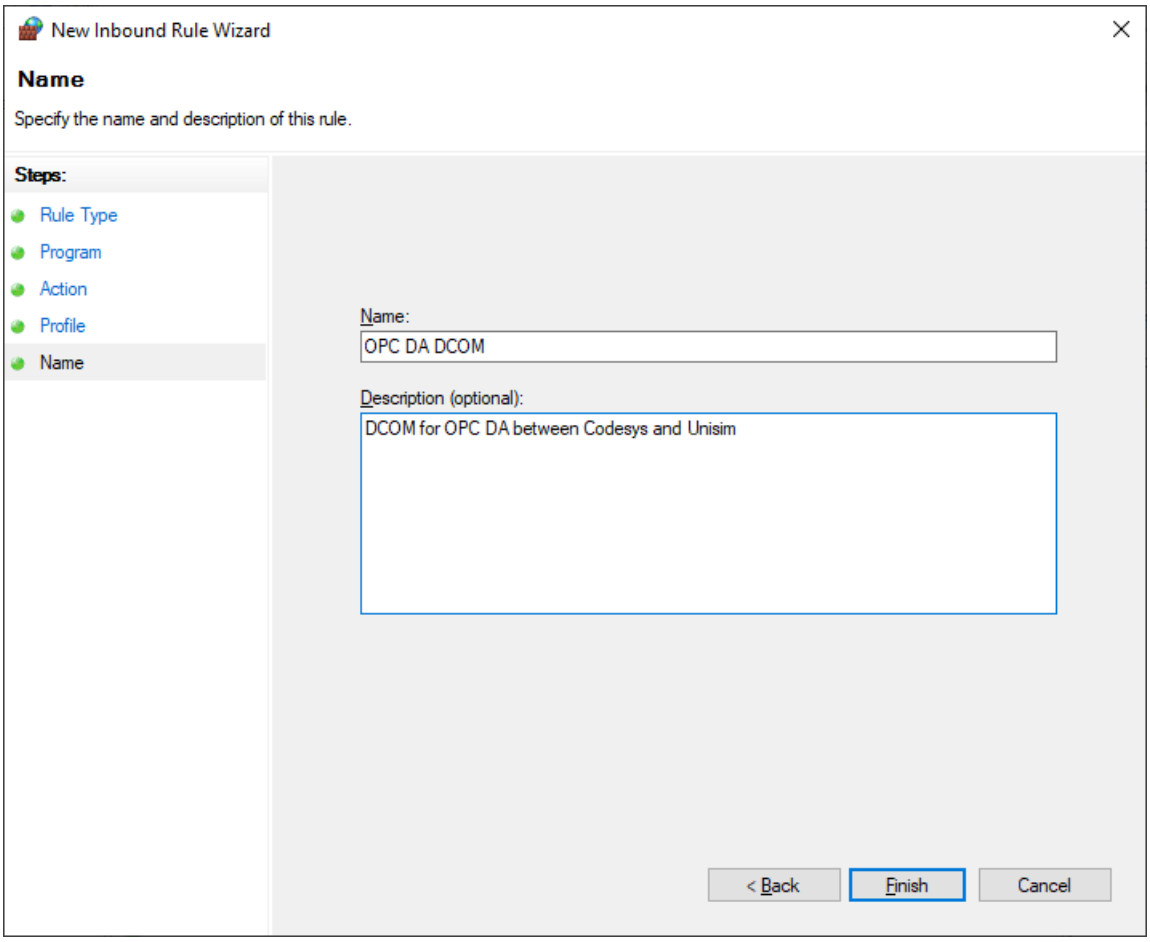


Figure 31: Assign name to new Inbound Rule

3.4.7 DCOM Configuration Settings

For OPC servers to run correctly, you should specify the DCOM network and security properties.First, run the "dcomcnfg" program trough the Windows Run tool (Windows+R keys). [5]

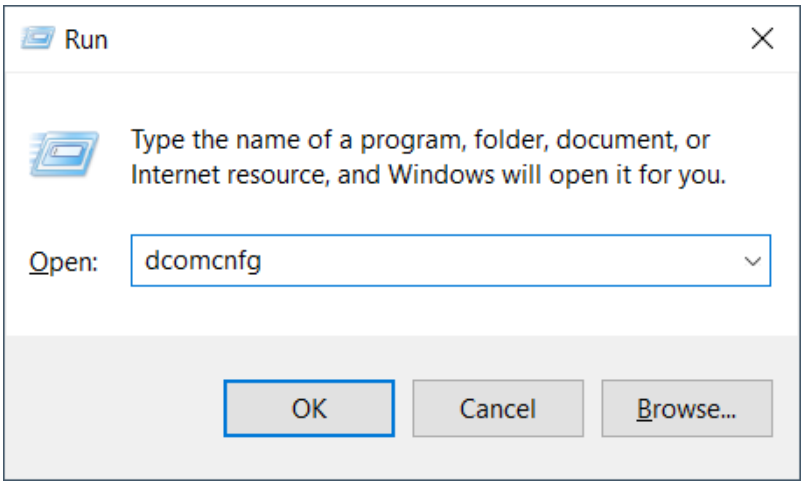


Figure 32: Open Component Services window

The Component Services Console appears, we have to configure General/Default Setting for DCOM configuration. For that, click on the Computers and then right click on “My Computer” and select “Properties”.

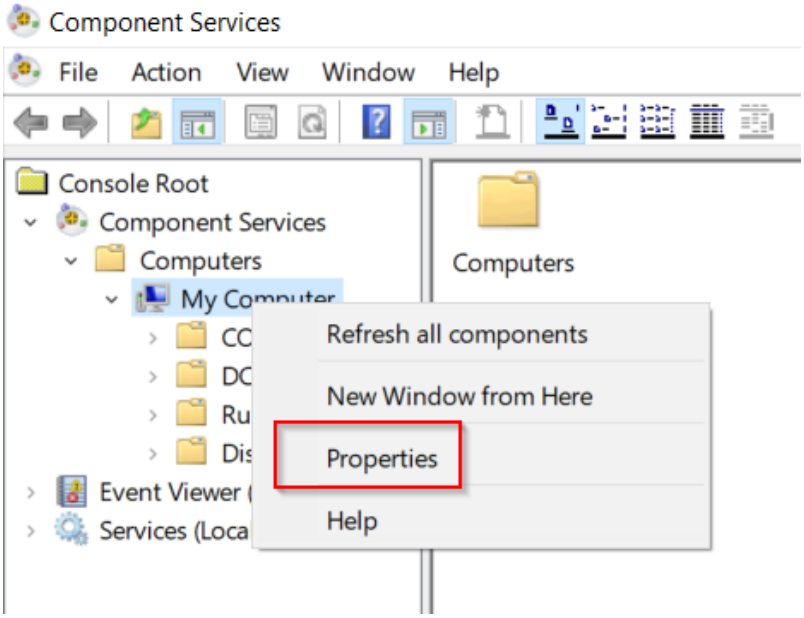


Figure 33: Component Services

The Default Properties tab: Configure the following options as follows:

- The Enable Distributed COM on this computer must be checked.
- The Default Authentication Level should be set to None . If an issue is observed with any Windows Components, please change this setting back to Connect.
- The Default Impersonation Level should be set to Identity.

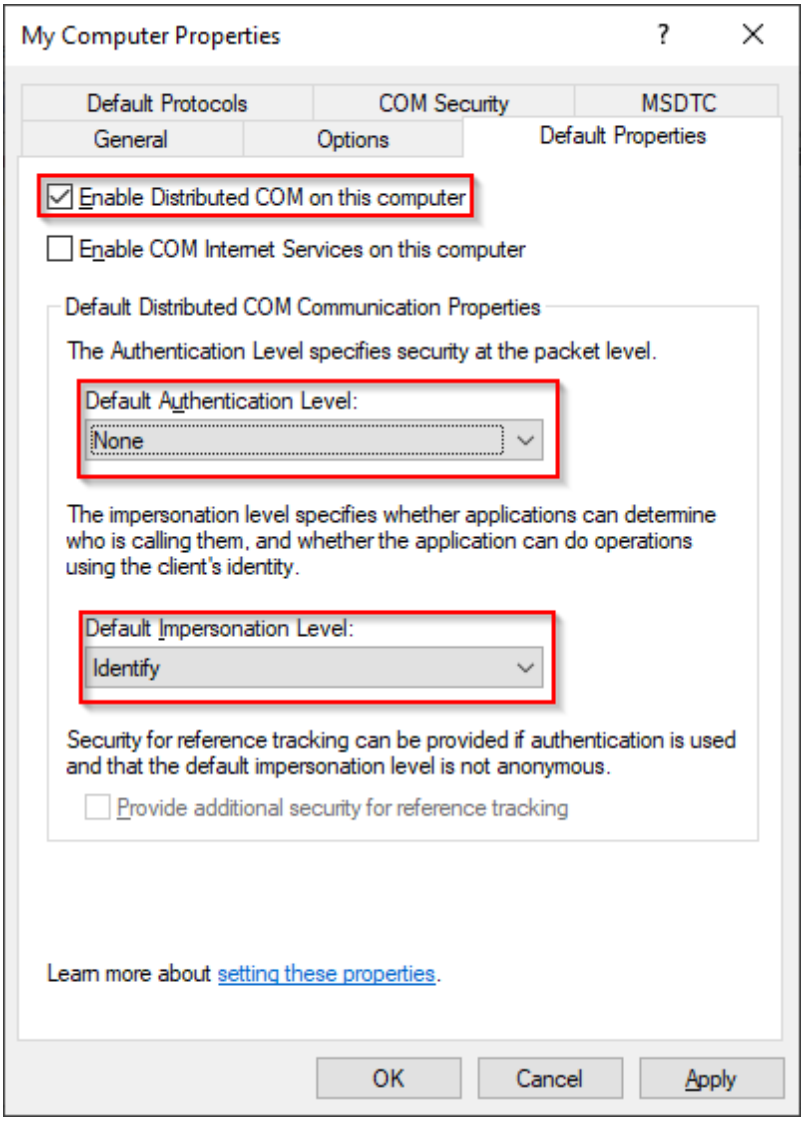


Figure 34: Default Properties Settings

The COM Security tab:

In this tab the access Rights could be given to Anonymous Logon, Everyone, Interactive, System accounts and the specific OPC user account which was created.

The launch permissions to control the list of users who are granted or denied permission to launch a particular server. The setting of access permissions, you must ensure that new user is included in the list of users that are granted access. The setting of access permissions is similar to setting launch permissions. Click on the Edit Default button within the Access Permissions and the Launch and Activation Permissions area and make the settings shown below.

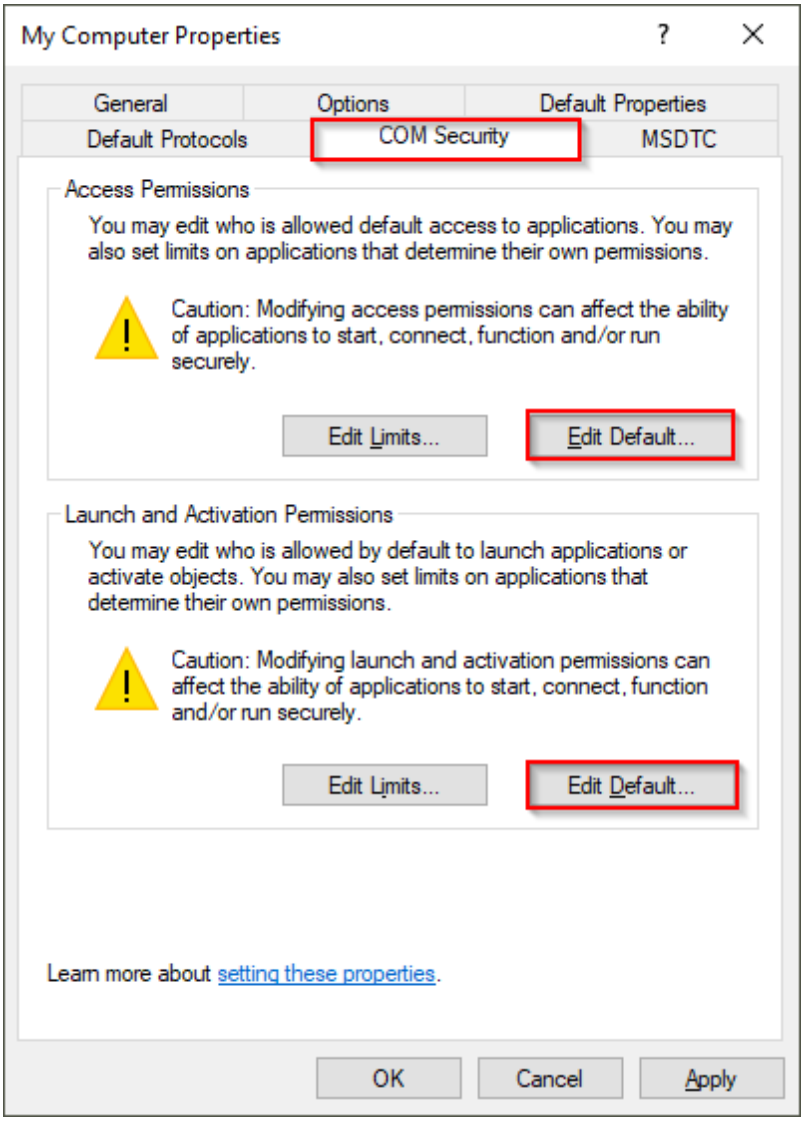


Figure 35: COM Properties Settings

Allow "Local Launch", "Remote Launch", "Local Activation" and "Remote Activation" for the user needed in the launch permissions tab .Allow "Local Access" and "Remote Access" for the user name in access permissions tab.

In this setting, you can also add a group or the user created for the DCOM configuration.

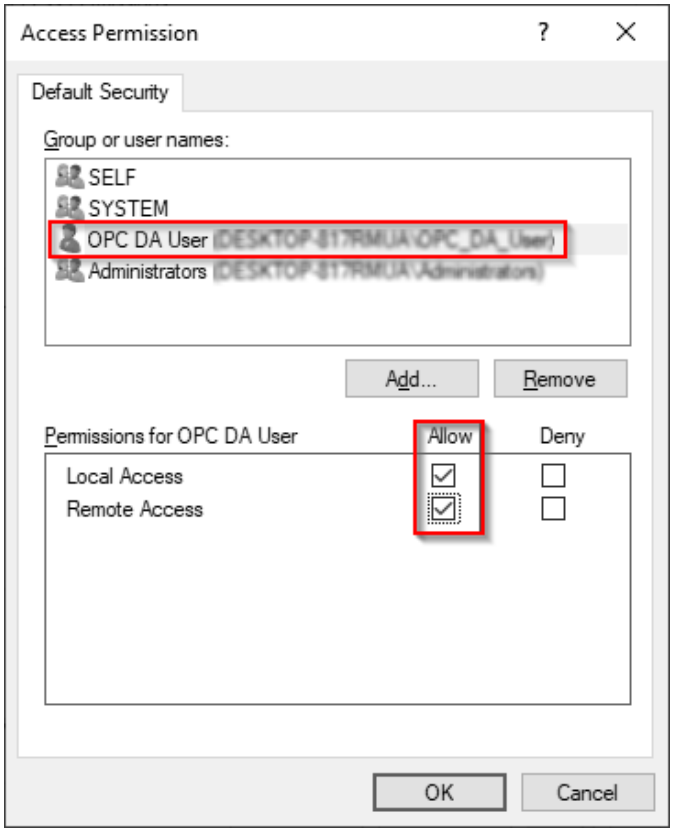


Figure 36: Access Permissions Settings

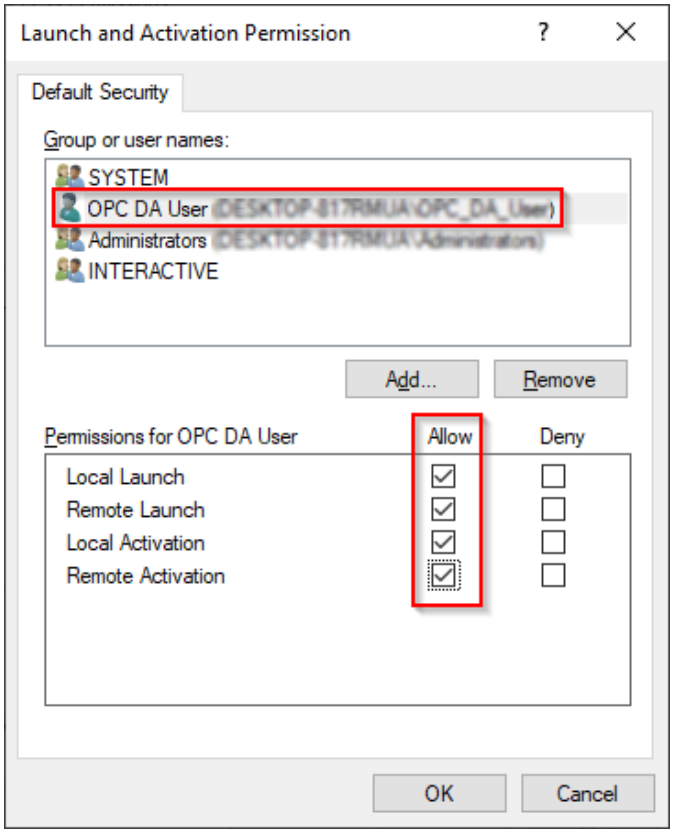


Figure 37: Launch and Activation Permission Settings

If a desired user name does not appear in the selection window, it can be added as follows:

- Click "Add".
- Click "Advanced.
- Click "Find Now".
- Search for the desired user in the list and Select the user.
- Click OK to save and close the window.

3.4.8 Configuring settings for OPCENUM

OPCEnum overrides DCOM settings and opens accessibility to everyone.
In the console of Component Services. Click on Computers>My Computer > DCOM Config> OPCEnum and display the Properties (by the right mouse button).

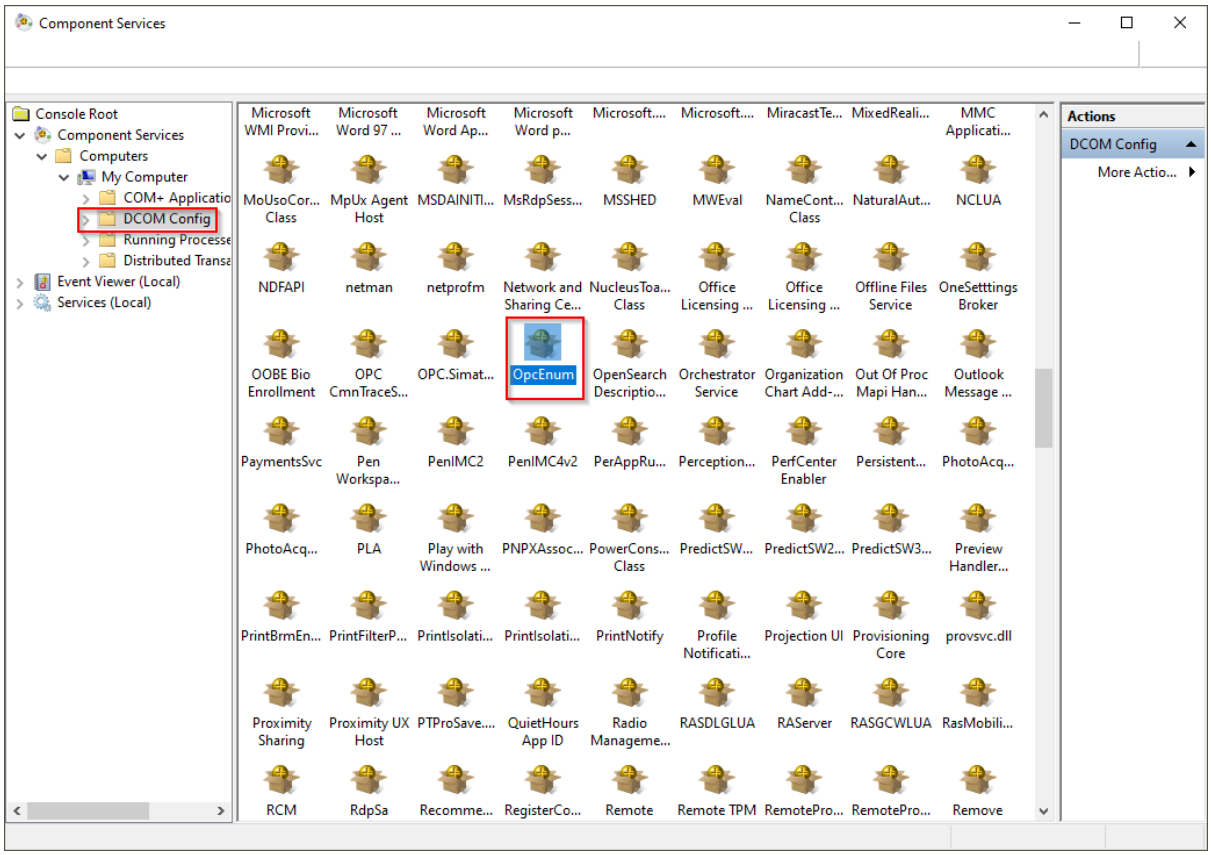


Figure 38: OpcEnum Component Services

On the "General" tab check if is set Authentication Level to None.

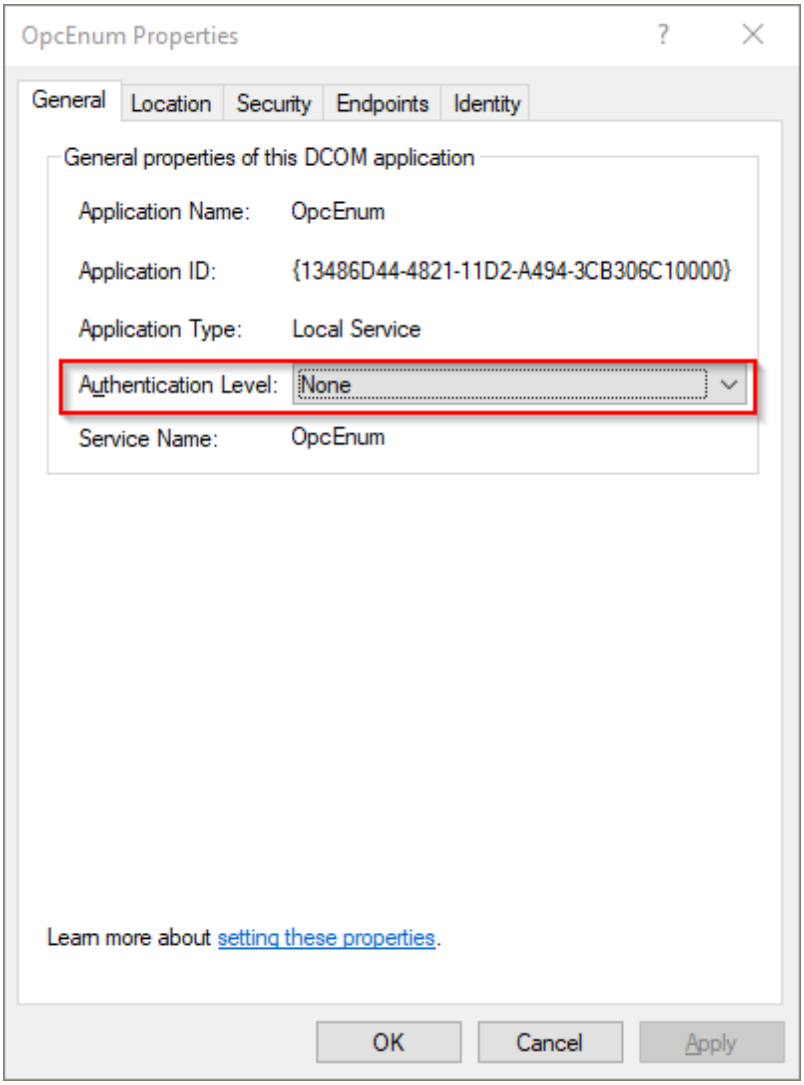


Figure 39: OpcEnum General settings

On the "Location" tab check if is set Run application on this computer.

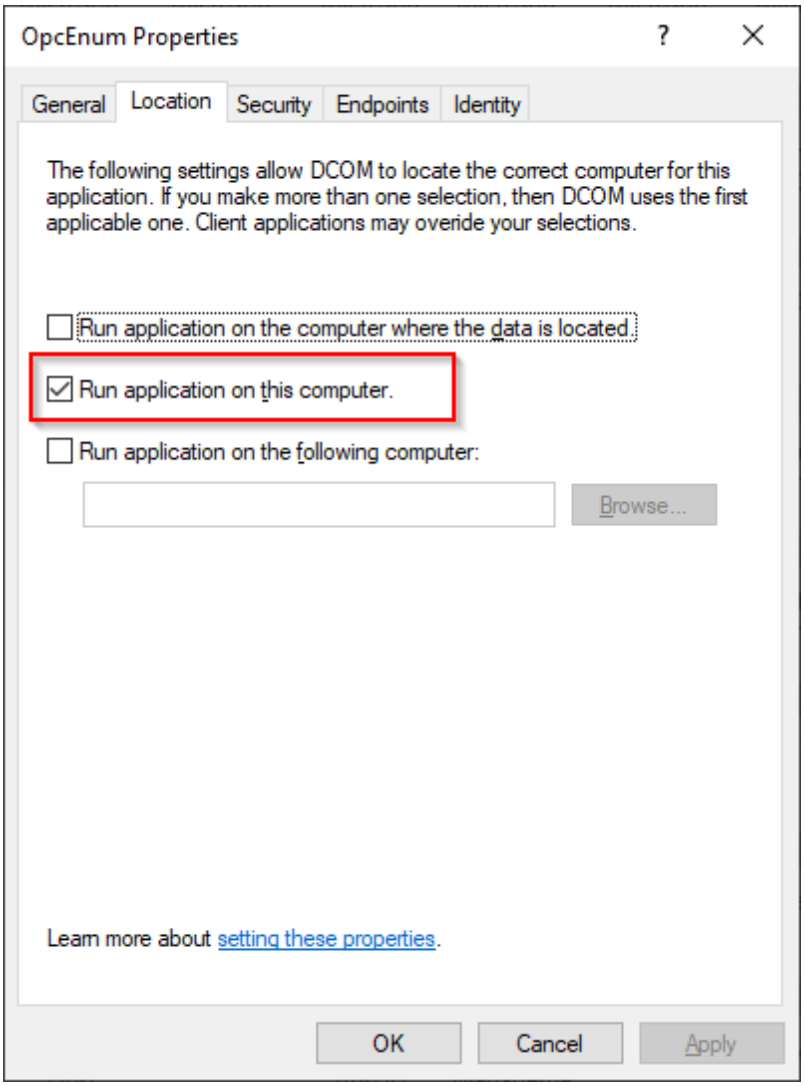


Figure 40: OpcEnum Location settings

On the "Security" tab check if options Launch and Activation Permissions, Access Permissions are set to Use Default and Configuration Permissions to Customize.
In Configuration Permissions when set to Customize.Click on edit and allow the following access.

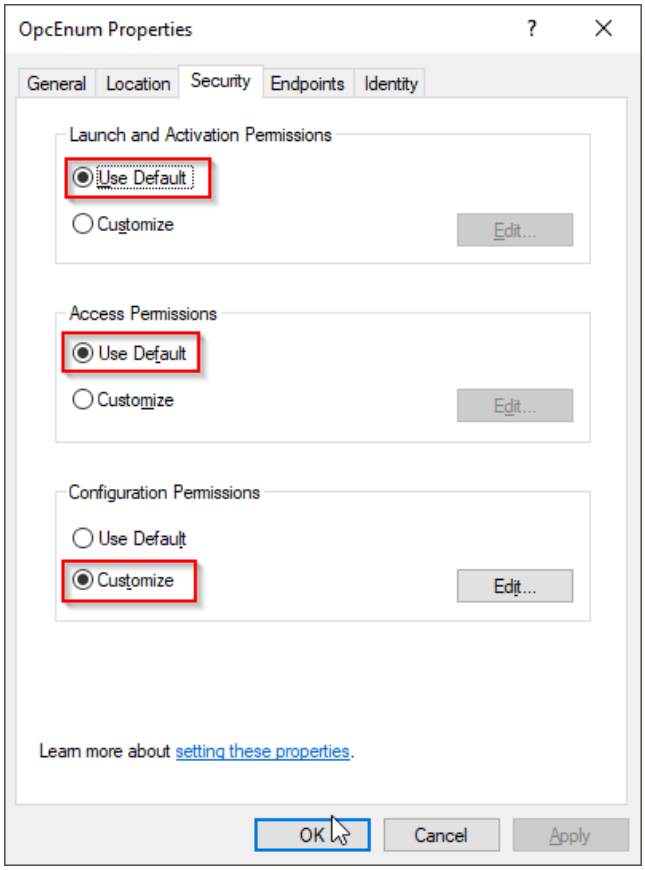


Figure 41: OpcEnum Security settings

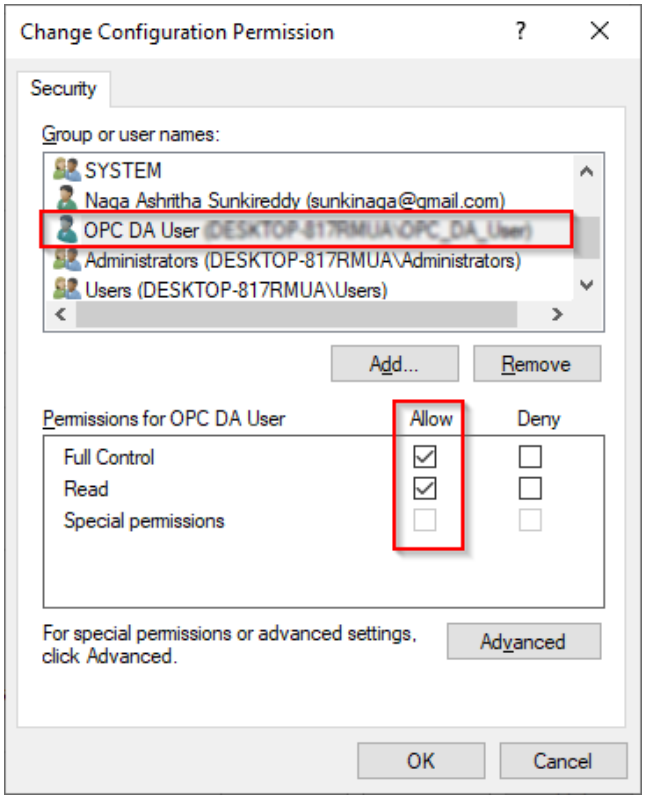


Figure 42: Configuration Permission

On the "Identity" tab check the The system account setting.

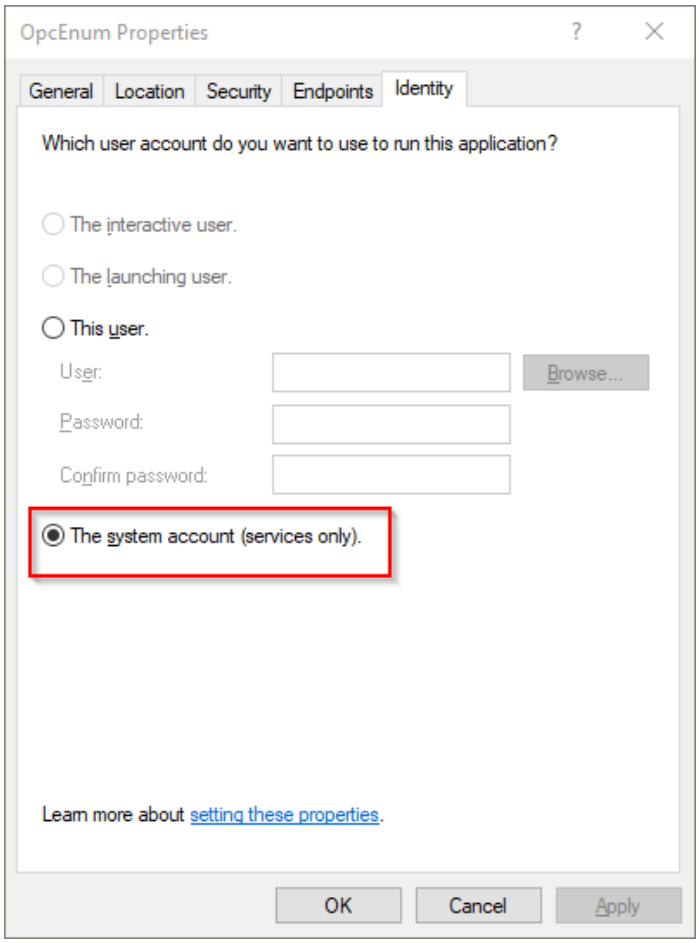


Figure 43: OpcEnum Identity settings

3.4.9 Configuring settings for Matrikon OPC DA

In the console of Component Services. Click on Computers > My Computer > DCOM Config > CoDeSysOPCDA and display the Properties (by the right mouse button).

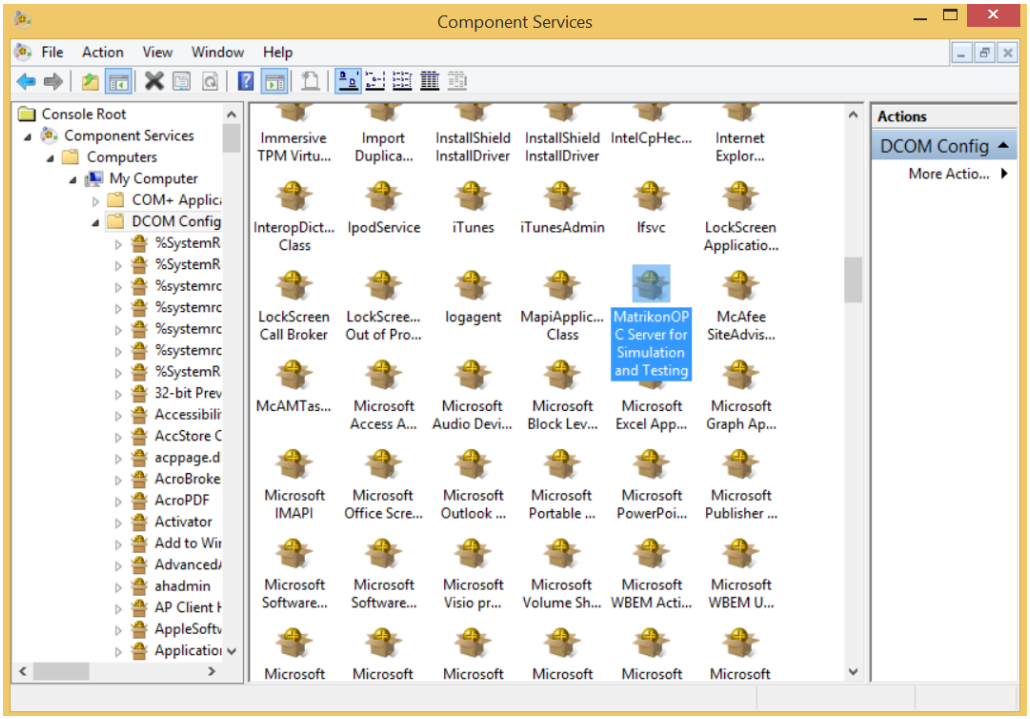


Figure 44: Matrikon OPC DA Component Services

In the "General" tab check if Authentication Level is set to Default.

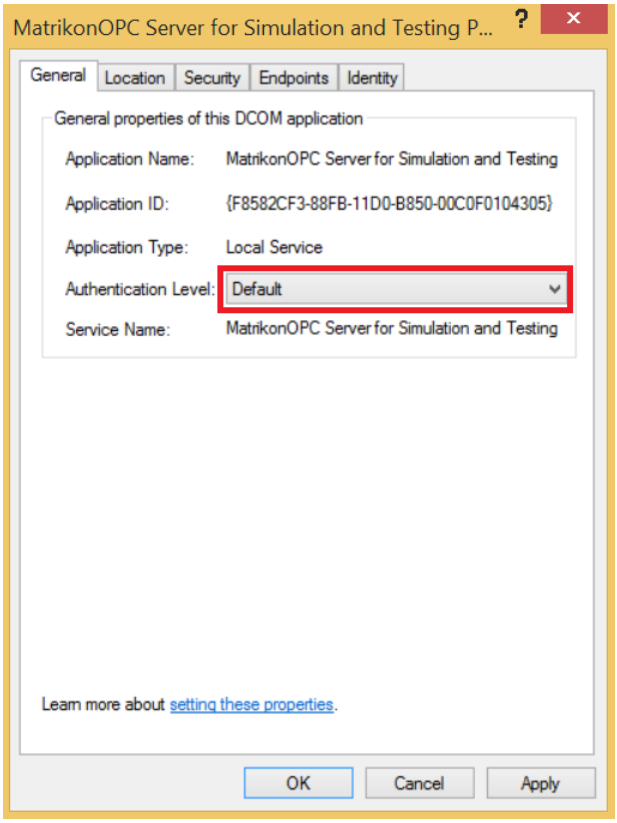


Figure 45: Matrikon OPC DA General Properties tab

In Security tab, these are the following changes

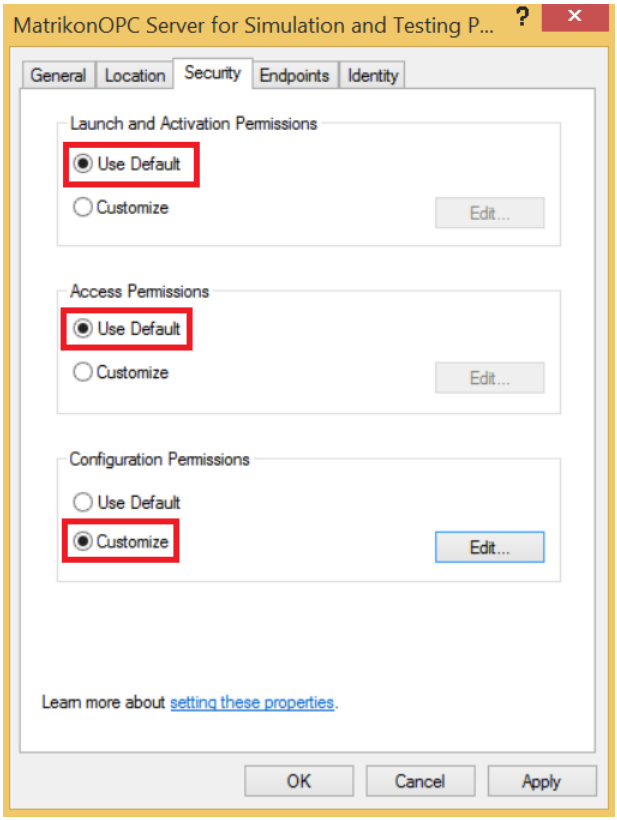


Figure 46: OPC DA Security tab

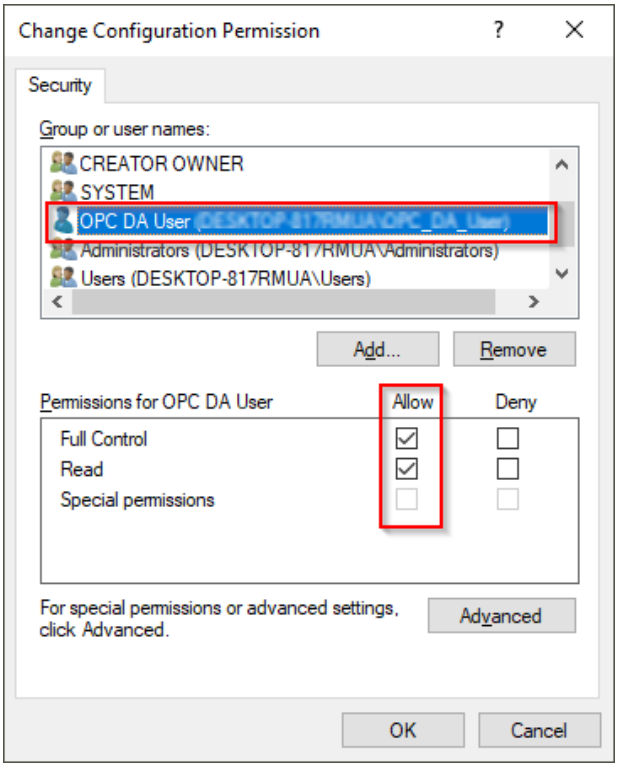


Figure 47: Fig:Configuration Permission

Under the Identity Tab, please ensure that the OPC Server is running as "The Interactive User".

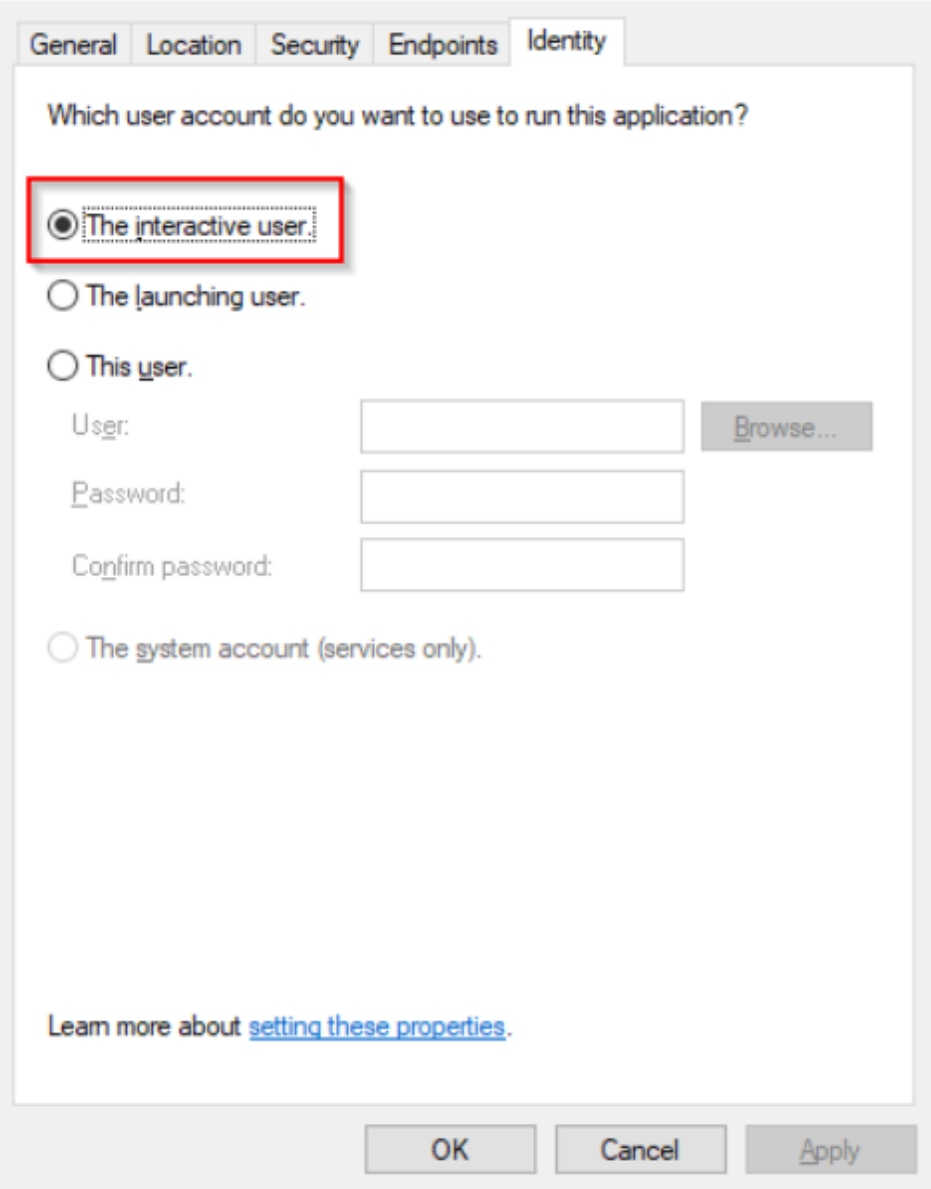


Figure 48: Fig:OPC DA Identity tab

3.4.10 Local Security Policy

The following settings must be done on both the OPC Server and the OPC Client computers. The Local Security Settings can be found under: Control Panel − > System and Security − > Administrative Tools− >Local Security Policy.

In the Local Policies folder, open the Security Options folder and locate the following options:

- DCOM: Machine Access Restrictions in Security Descriptor Definition Language (SDDL) syntax.
- DCOM: Machine Launch Restrictions in Security Descriptor Definition Language (SDDL) syntax.

Both options should be set to Not Defined

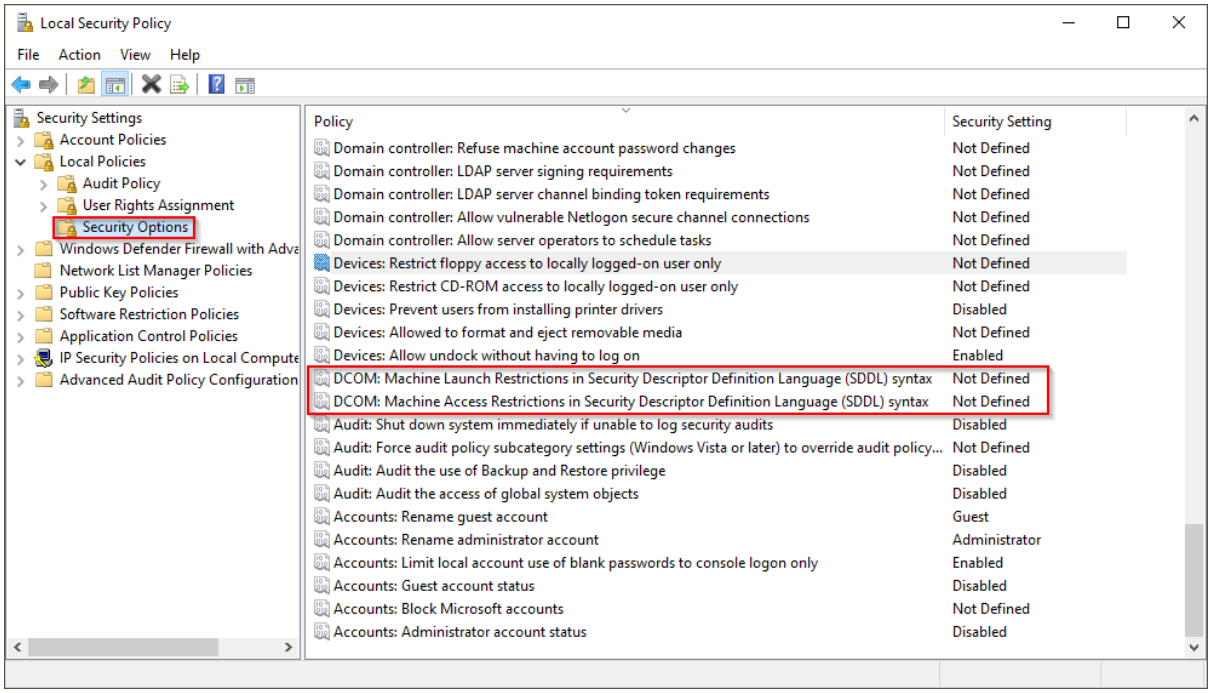


Figure 49: Local Security Policy Settings

In next section, we shall discuss use and implementation of GIT in this case study.

3.5 GIT Hub

GitHub, Inc. is a provider of Internet hosting for software development and version control using Git. It offers the distributed version control and source code management (SCM) functionality of Git, plus its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, continuous integration and wikis for every project.

3.5.1 Features and Importance of GIT

- Simultaneous Editing in cooperation with different Contributors.
- Individual contributors can create their own branch or forks and if code is running as desired then can push in main brach. This avoids ruining main code due to errors in individual codes. Below screenshot shows the history of pushing individual codes by individuals to main code repository. We can also title the small changes in code.

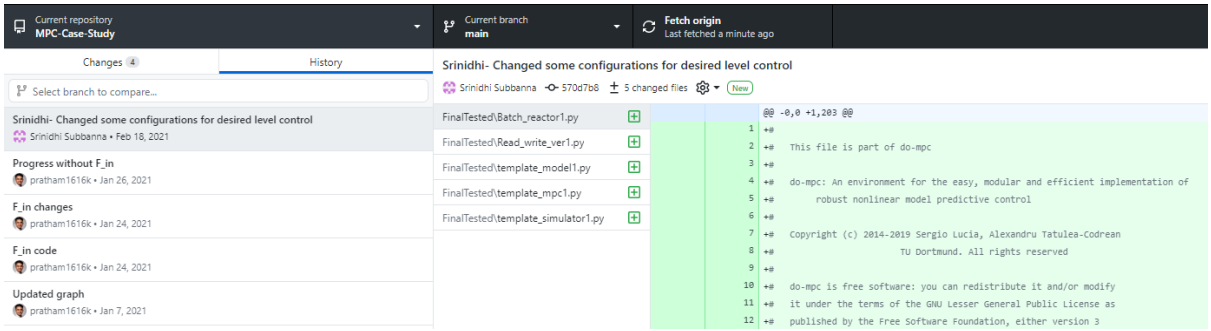


Figure 50: Main repository in GIT

- Online platform for change management. Below screenshot shows MPC Github project created by owner and different sections of codes for better change management and effective operation.

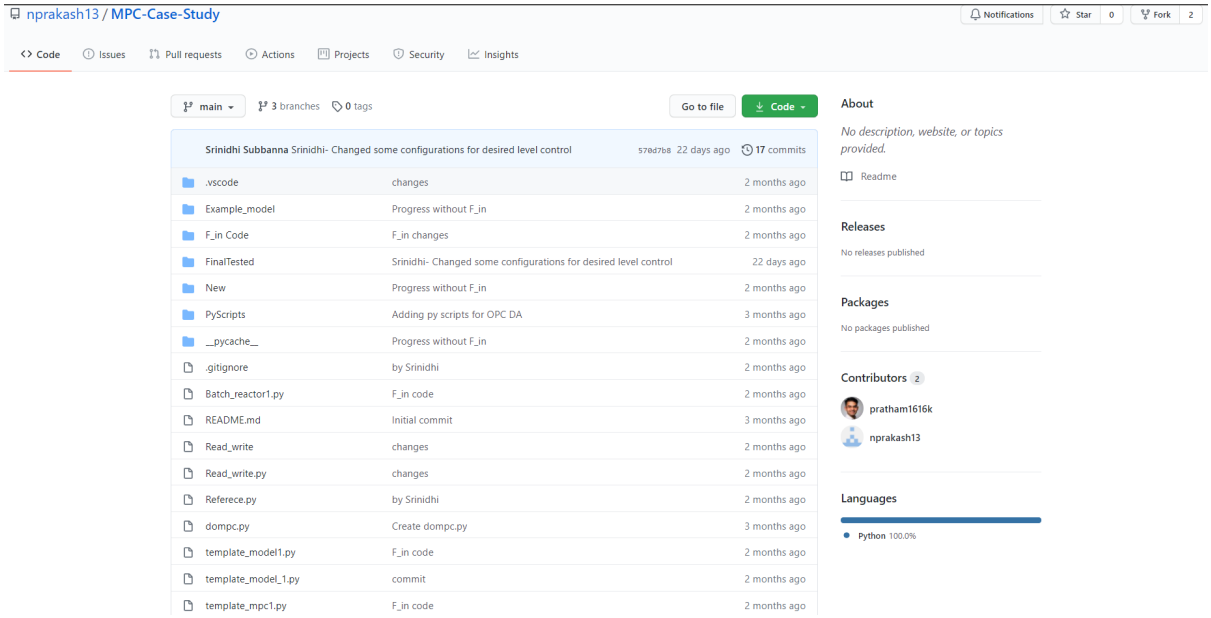


Figure 51: GIT Dashboard

- Contributors can pull the main code and thereby does not have to write the main code again.
- Easy to link to Visual studio code thereby pushing and pulling codes to branch become easy.

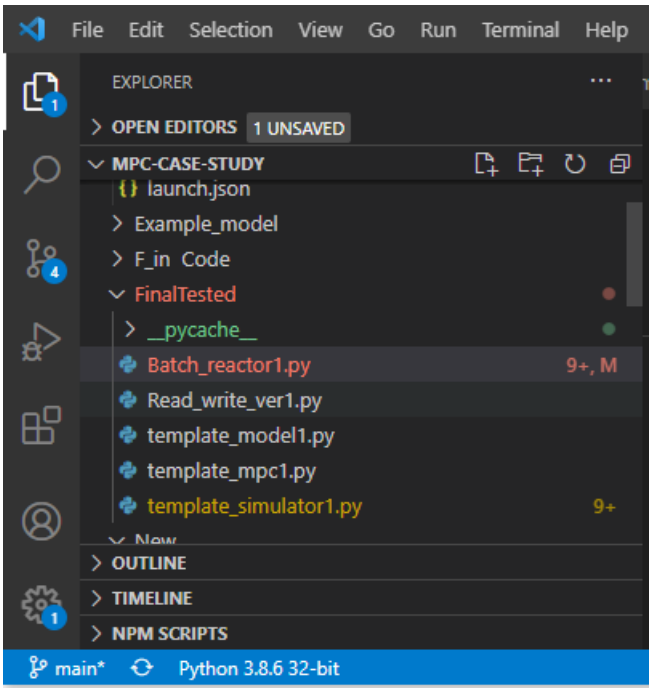


Figure 52: Visual studio linked with GIT Project

In next section, we shall discuss override control strategy in detailed.

4 Override Control strategy

Override Control is a popular control strategy when there are control problems with process optimization and prevention of abnormal operating regions. A typical Override structure has 2 or more PID controllers trying to control a same Actuator. There is always a selector to choose the right control output from the PID controllers based on a selection criteria (eg: min, max).

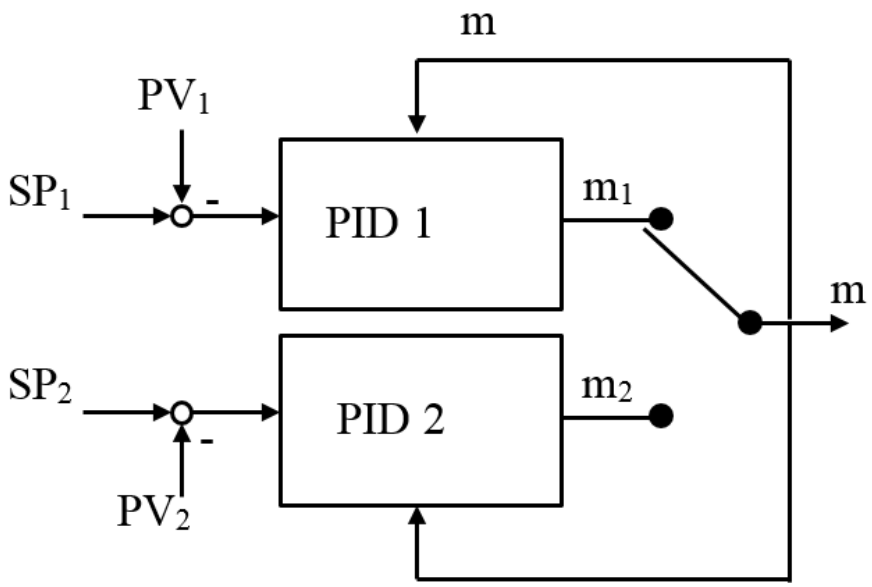


Figure 53: Override Control structure

The main problem in such control structures is to synchronize the outputs of all controllers such that when there is a switch in control, there are no sudden surges in the control output. A simple trick can resolve this issue and to achieve that we should consider what kind of selector is used. For a max selector, the output of the selector should be wired to the MV_LOW_LIMIT parameter of all PID controllers with a variance of 3 to 5% MV. In case of a min selector, the MV_HIGH_LIMIT parameter should be wired as described above.

To implement the Override structure in Unisim, the PI controller with external reset is used. The Integrator of the standard PI controller is replaced with a first order Lag element. The Lag element holds the stationary setpoint.

4.1 Control Structure implementation in Unisim

Our Override structure* consists of three PI controllers. A flow controller, a Low Level Controller and a High Level Controller. The output of the controllers are chosen based on a selection criteria. The output of the selection is cascaded to a Flow PI controller. The Flow Controller receives the setpoint from Override controller and controls the outlet valve of the tank.

* The Unisim model and the Override control application is provided by Prof. Dr. Rainer Scheuring as an input to the Case Study.

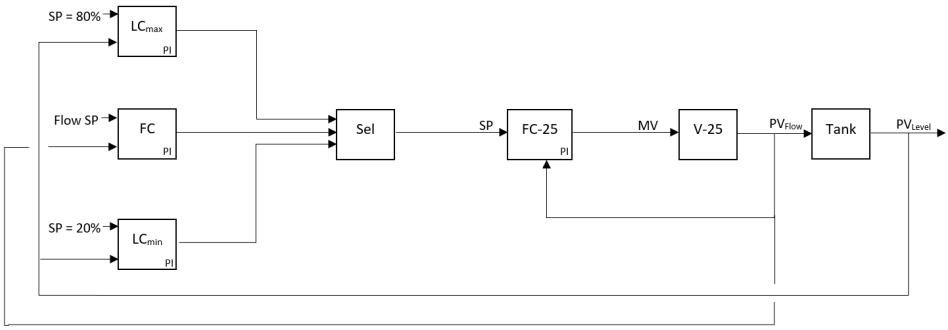


Figure 54: Override Control for Level Control

This structure is implemented in the Honeywell Unisim using controller, spread sheets and transfer function blocks as shown in the below figure.

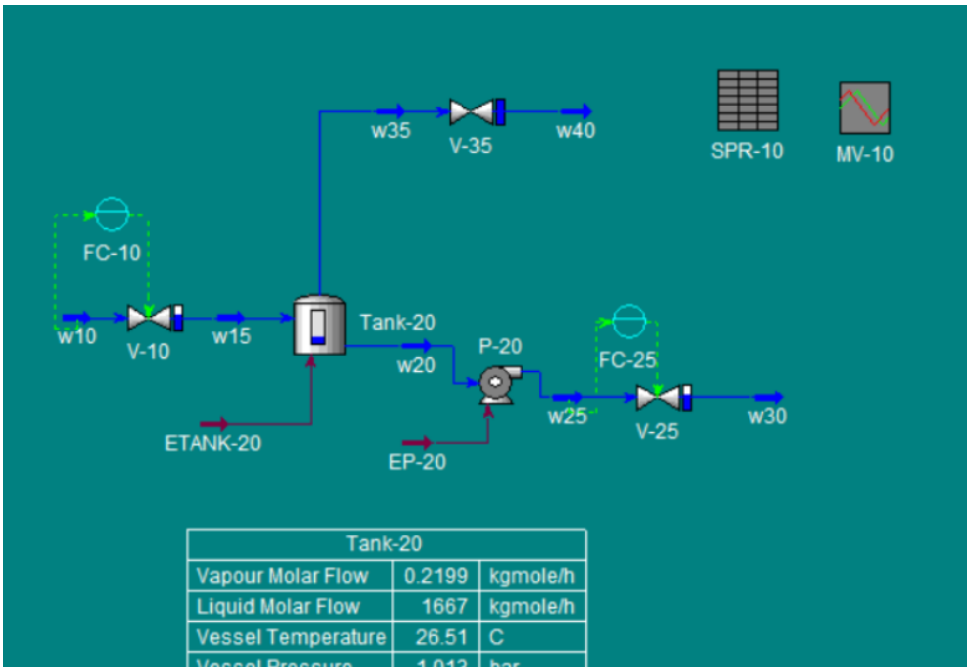


Figure 55: Override Control implementation

As described in the earlier sections we are implementing the Override controllers using External Reset. The Transfer Function block configured as a first order lag gives the stationary setpoint to all three controllers.

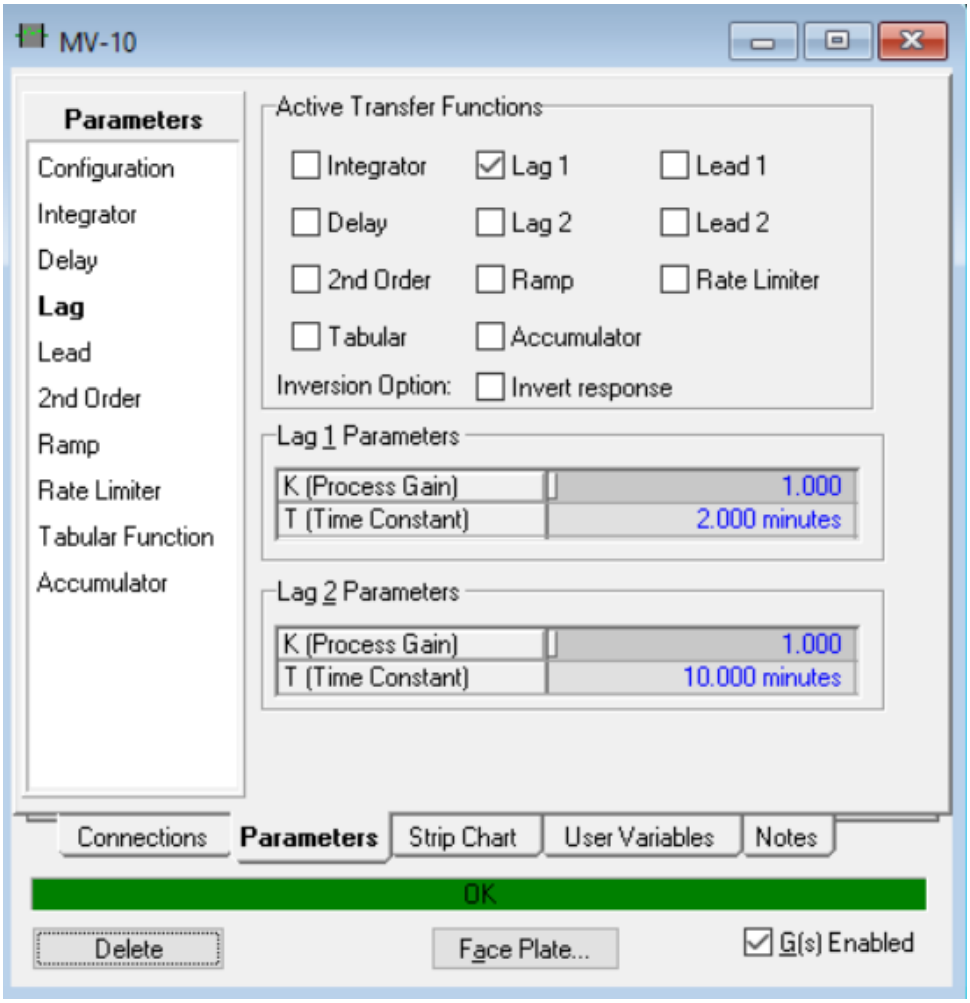
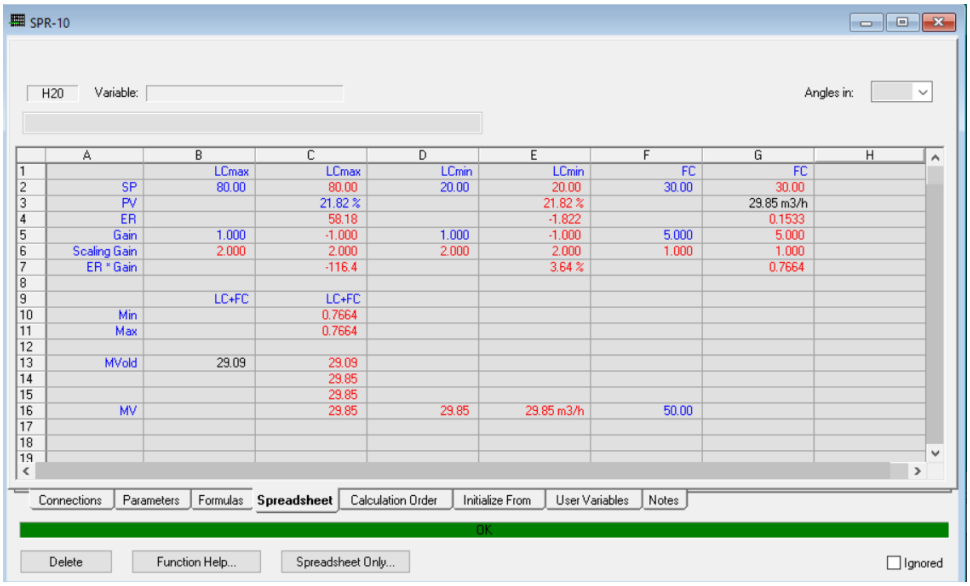


Figure 56: 1st Order lag as external reset

The key idea for the override control strategy is that in normal operation range, the outflow should be free to adjust. Hence, at centre we place a PI flow controller. The Level should not, in any case, increase above 80%. The High Level controller is configured to ascertain this. Likewise a Low Level controller is configured to maintain the Liquid Level above 20%. The proportional part of these controllers are implemented in a spreadsheet in Unisim as shown below.



	A	B	C	D	E	F	G	H
1		LCmax	LCmax	LCmin	LCmin	FC	FC	
2	SP	80.00	80.00	20.00	20.00	30.00	30.00	
3	PV		21.82 %		21.82 %		29.85 m3/h	
4	ER		58.18		-1.822		0.1533	
5	Gain	1.000	-1.000	1.000	-1.000	5.000	5.000	
6	Scaling Gain	2.000	2.000	2.000	2.000	1.000	1.000	
7	ER * Gain		-116.4		3.64 %		0.7664	
8								
9		LC+FC	LC+FC					
10	Min		0.7664					
11	Max		0.7664					
12								
13	MVold	29.09	29.09					
14			29.85					
15			29.85					
16	MV		29.85	29.85	29.85 m3/h	50.00		
17								
18								
19								

Figure 57: Override Controllers SpreadSheet

The setpoints are configured for all the controllers and respective errors are calculated. The errors are mutliplied with respecctive gains to obtain the P part of the controllers. The P Part output is then selected based on the selection criteria and then added to the past MV taken from the 1st order Lag block. The new value is written back to the Lag block. The new value is clamped between the flow range of 0 to 50 m^3/h and fed as setpoint for the Flow controller secondary loop.

Selection Criteria:

Once the P part output of all controllers are ready, the selection criteria gives the optimum setpoint for the secondary controller based on the current state of the plant. When the Level of tank is below 20%, the Low Level Controller should kick in and start to bring the level back up to 20%. Similarly when the level crosses 80%, the High level controller should take over and bring the Level back to 80%. In the intermediate range, the flow setpoint can be set in the spread sheet to operate at desirable outflow. The gains of Low Level controller are chosen such that if the level decreases below the low limit, the P part will become negative. Hence, we use a min() comparison between flow controller's P part and Low Level Controller's P part. Similarly the High Level Controller's gain is chosen such that the P part is positive only when the Level rises above the High set point. Thus we do a max() comparison between High Level controller's P part and the output of previously described min() comparison. Thus it is made sure that the right override actions are available at the right situations.

4.2 Results

The model is tested after it has been completely implemented. Different scenarios are tested and recorded here. We have observed the control system working in normal operating region, Low Level and High level beyond the operating points.

The Unisim Simulation starts with a Level of 65%. The in flow to the tank is increased to 30 m^3/h while the outflow is 10 m^3/h. This causes the Level to raise above the upper limit of 80%. The Override controller (LCmax) takes over and increases the outflow accordingly such that the level is maintained at the upper limit. We can see this experiment in the below figure.

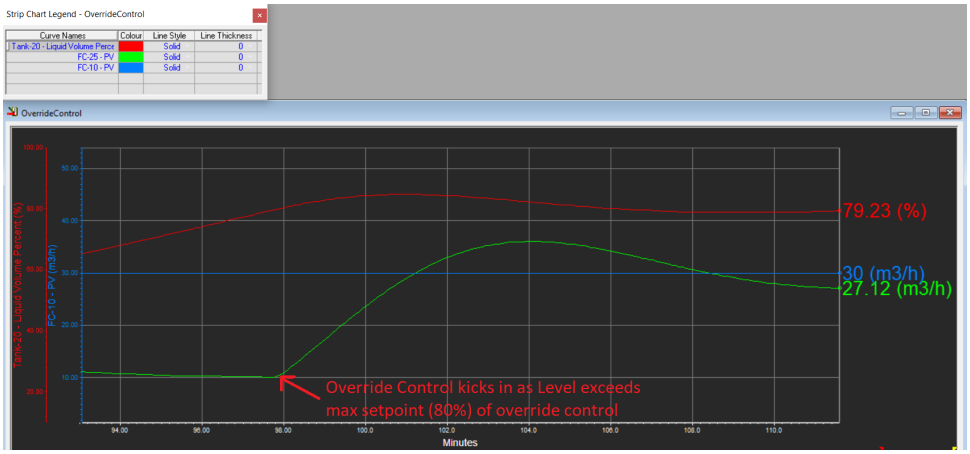


Figure 58: Override Control in High Level Control mode

In a different setting, we started the simulation with Level at 30% and the outflow at 30 m^3/h while the inflow is 10 m^3/h. This will reduce the Level sharply until the level drops below the low Level limit of 20%. The Low Level override control (LCmin) takes over and reduces the outflow such that the level rises to the low set point.

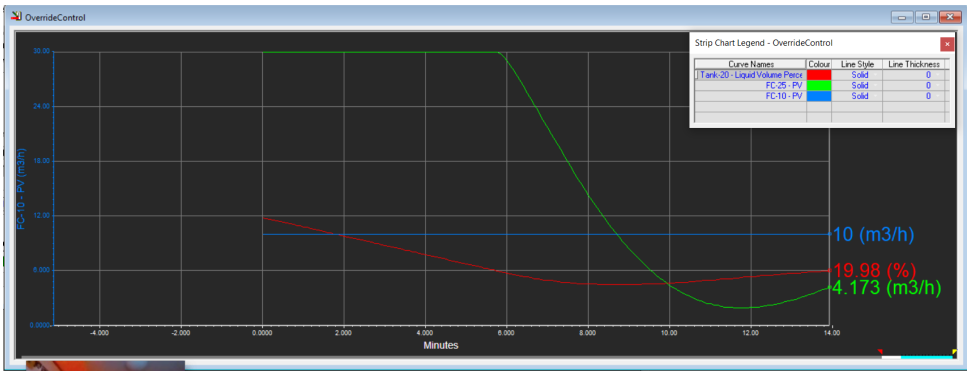


Figure 59: Override Control in Low Level Control mode

From the above experiments we have tested the Override Controller and we have seen that it does a great job for our control problem. The Override controller responds quickly to the change of states due to change of disturbances. In the coming sections we have shared how to put together an MPC controller for the same control Problem and tested.

In next section, we shall discuss Model Predictive Control Strategy in python in detailed.

5 Model Predictive Control

5.1 Model Predictive Control : Basics

Model predictive control (MPC) is a control scheme where a model is used for predicting the future behavior of the system over finite time window, the horizon. Based on these predictions and the current measured/estimated state of the system, the optimal control inputs with respect to a defined control objective and subject to system constraints is computed. After a certain time interval, the measurement, estimation and computation process is repeated with a shifted horizon. This is the reason why this method is also called **receding horizon control (RHC).** Major advantages of MPC in comparison to traditional reactive control approaches, e.g. PID, etc. are

- **Proactive control action:** The controller is anticipating future disturbances, set-points etc.
- **Non-linear control:** MPC can explicitly consider non-linear systems without linearization.
- **Arbitrary control objective:** Traditional set-point tracking and regulation or economic MPC.
- **constrained formulation:** Explicitly consider physical, safety or operational system constraints

[7] The codes are referred from do-mpc documentation and the parameters are updated to suit our application

5.2 Model Predictive Control : Getting Started with MPC

Importing various packages and libraries required to initialize the setup. The short description of each package is below:

- **Numpy**:Numerical python based computational library.
- **CasADi**: CasADi is an open-source tool for nonlinear optimization and algorithmic differentiation.
- **Matplotlib**: Library to plot Graphs.
- **pdb**:PDB stands for "Python Debugger", and is a built-in interactive source code debugger with a wide range of features, like pausing a program, viewing variable values at specific instances, changing those values, etc.
- **sys**: The python **sys** module provides functions and variables which are used to manipulate different parts of the Python Runtime Environment.
- **do_mpc:** do-mpc is a comprehensive open-source toolbox for robust model predictive control (MPC) and moving horizon estimation (MHE).
- **os**: The OS module in Python provides functions for interacting with the operating system.
- **time**: time() The time() function returns the number of seconds passed since epoch.
- **pywintypes**: pywintypes is part of the Python for Windows extensions, otherwise known as pywin32.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from casadi import *
from casadi.tools import *
import pdb
import sys
sys.path.append('../../')
import do_mpc
import matplotlib.axes as maxes
```

```
from matplotlib.animation import FuncAnimation, FFMpegWriter, ImageMagickWriter
import pdb
import os
from do_mpc.tools import IndexedProperty, Structure

import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import time

from template_model1 import template_model1
from template_mpc1 import template_mpc1
from template_simulator1 import template_simulator1
import OpenOPC
from time import sleep
import pywintypes
pywintypes.datetime = pywintypes.TimeType
```

[ ]:
```
""" User settings: """
show_animation = True
store_results = False
```

[ ]:
```
"""
Get configured do-mpc modules:
"""

model1 = template_model1()
mpc = template_mpc1(model1)
simulator = template_simulator1(model1)
estimator = do_mpc.estimator.StateFeedback(model1)
```

5.2.1 Connecting to OPC-DA server and setting up initial state

The aim is to communicate OP(MPC.FC_Out) and PV(MPC.Level) from Unisim to Python and vice-versa through Matrikon OPC-DA server. Below code shows-

- Importing OpenOPC library
- Read the PV(MPC.Level)
- Write the OP(MPC.F_Out)

[ ]:
```
#Connection as Python as Client and Matrikon OPC Simulation server using OpenOPC␣
 ↪library.
opc = OpenOPC.client()
opc.servers()

#Connecting to Matrikon OPC simulation server named PRATHAM; This is a OPC-DA␣
 ↪communication which is operational on two different machines
opc.connect('Matrikon.OPC.Simulation.1', 'PRATHAM')
value = opc.read('MPC.Level')

#Storing value with format 'Float' in variable 'L'
L = float(value[0])

#Setup initial guess in form of an Array which is used to Simulate and Estimate the␣
 ↪process
```

```
x0 = np.array([L])
mpc.x0 = x0
simulator.x0 = x0
estimator.x0 = x0
mpc.set_initial_guess()
```

5.2.2 Setup Graphics and Running MPC in a Loop

Running MPC code is structured in four different python files, main file being 'Batch Reactor1' and this file calls other three files namely,

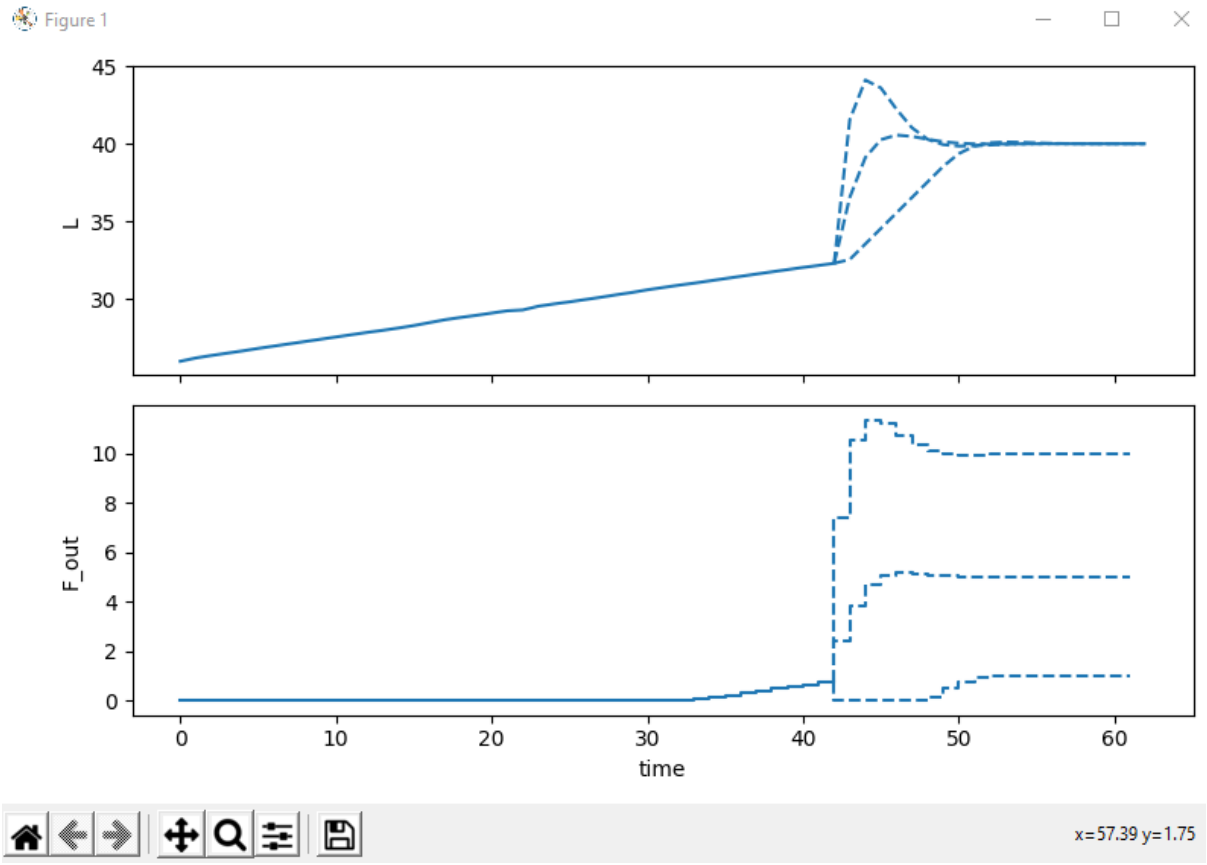
- Batch_Reactor1.py : This is main file which calls/imports other python files running in background.
- Template_Model1.py : This file builds Dynamic system model which is representing real-time process running in Unisim using ODE (Ordinary Differential Equation)
- Template_MPC1.py : This file configures MPC controller setting up different MPC parameters
- Template_Simulator1.py : For testing purpose, before connecting to Unisim, Simulation of MPC can be viewed. But for final implementation, we shall not use simulation tool in Python, but in Unisim.

Setting up the graphics includes use of in-built function of do-mpc called do_mpc.graphics.default_plot which plots the data of MPC.Level, MPC.F_Out with time axis, The dotted lines in plot shows the prediction.

[ ]:
```
"""
Setup graphic:
"""
# Initialize graphic:
graphics = do_mpc.graphics.Graphics(mpc.data)

fig, ax, graphics = do_mpc.graphics.default_plot(mpc.data,figsize=(8,5))

plt.ion()
```

Running MPC in a loop includes

- Running Model, MPC controller and Read/write tags from/on Matrikon server.
- Initially, we shall read the level of tank from Unisim model from Matrikon server (This value of level will be written from Unisim to Matrikon) untill health of tag is 'Good'.
- Iteration of MPC loop will happen every after 5 seconds and calculated MPC.F_out will be written to Matrikon server and Value of MPC.Level will be read from Matrikon to Python.
- Sleep time of 5 seconds will reduce computational time of running the optimization problem.
- After every 5 seconds, x0 or initial value will get updated and optimization problem will thus find a new optimal solution of MPC.F_out and this solution will be written to Matrikon server.
- Animation will be shown during every simulation step and it comprises of current value of MPC.Level and estimate value of MPC.Level would be represented by dotted line as shown in figure below.



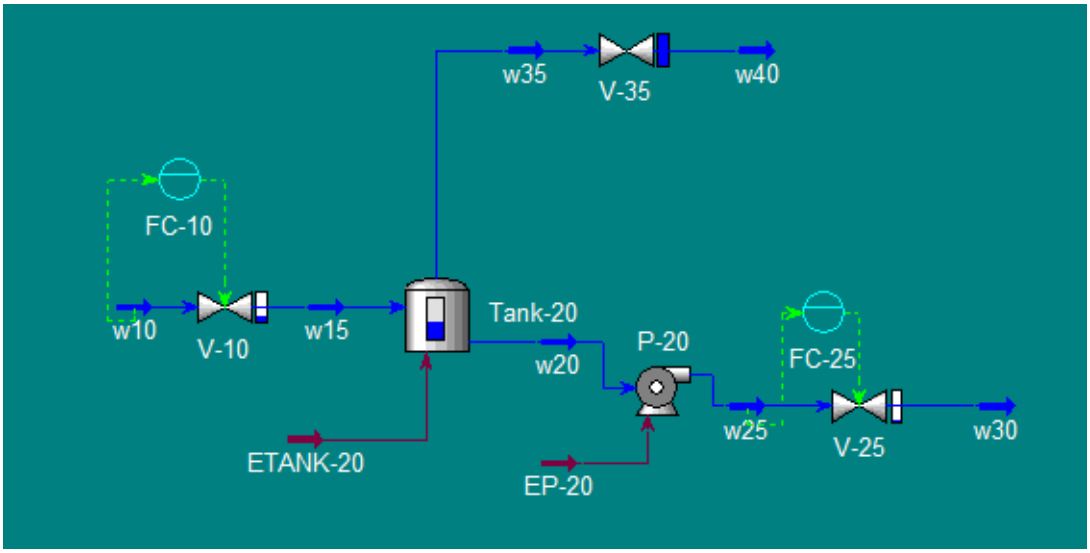
- Similarly, MPC.F_Out will be calculated and shown in graph. Also Value of MPC.F_out will also be estimated in prediction horizon.
- To stop iteration of loop, user-defined key-board interruption command is also enabled by pressing 'Ctrl-C', otherwise this MPC loop will run and simulate the result for infinite time if Matrikon continues to receive 'Good' health of MPC.Level tag.
- Previous results of steps including current simulation result would get stored.

[ ]:
```
"""
Run MPC main loop:
"""
try :
    P = 0

    while (opc.read('MPC.Level')[1]) == 'Good':
        time.sleep(5)
        u0 = mpc.make_step(x0)
        opc.write( ('MPC.F_out', float(u0[0][0])))
        y_next = simulator.make_step(u0)
        y_next1 = np.array([[float(opc.read('MPC.Level')[0])]])  #reading value␣
↪from OPC server
        x0 = estimator.make_step(y_next1)
        if show_animation:
            graphics.plot_results(t_ind=P)
            graphics.plot_predictions(t_ind=P)
            #graphics.add_line(var_type='_p', var_name='F_in', axis=ax[1])
            graphics.reset_axes()
            plt.show()
```

```
            plt.pause(0.01)
        P = P+1
except KeyboardInterrupt:
    print('Program was stopped by interrupt')
    opc.close()

    # Store results:
if store_results:
    do_mpc.data.save_results([mpc, simulator], 'batch_reactor_MPC')
```

5.3 Model Predictive Control : Defining Model in Python

As discussed in earlier chapter, We can define whole process in mathematical equation by which process parameters behave or change. In our case study, we are dealing with override tank level control in conjunction of cascade control of output flow loop.



Above screenshot shows W-10 is inlet flow steam with F_in as inlet flow which might be treated as a disturbance to tank level as generally in Process plats, inlet flow is generally varies and not of fixed flow rate. Tank-20 is vertical tank with Height = 4m and Diameter = 1.7 m with volume of 10m3. Level of tank is state parameter which changes with time and is defined by MPC.Level in Matrikon Alias configuration. It is measured in % of liquid level.
Output flow from tank is controlled parameter defined by MPC.F_Out. This value is OP and should be calculated and communicated by MPC controller in Python. Maximum of F-25 is 50 m3/hr defined. Inlet of flow F-10 is a disturbance to process and thus uncertain parameter with certain range of 1-10 m3/hr Generally we do not desire level of tank to fall below 20% and overflow above 80% of height of tank corresponding to level of tank. therefore, controllable lower limit of level would be 20% and higher limit of level be 80%. We expect level should not exceed these boundries. Certain parameters include time constant of tank which is calculated as 1hr. T = 1.
The above whole process can be captured in mathematical equation, which can be calculated in ODE (Ordenary differential Equation) format.
$\frac{dl}{dt} = \frac{1}{T} \cdot (F_{in} - F_{Out})$

Below figure shows different types of variables available under do-mpc.

Long name	short name	Remark
states	_x	Required
inputs	_u	Required
algebraic	_z	Optional
parameter	_p	Optional
timevarying_parameter	_tvp	Optional

Identification of States and parameters

There is one state x in model = Level of tank (l) = L

There is one input u in model = outlet flow = F_out

There is one parameter p in model = inlet flow (which is uncertain parameter) = F_in

There is one time constant in model = T = 60 mins. or 1 hr.

Below code defines template for model which will be called for every iteration of MPC controller.

[ ]:
```
import numpy as np
from casadi import *
from casadi.tools import *
import pdb
import sys
sys.path.append('../../')
import do_mpc


def template_model1():

    """
    --------------------------------------------------------------------------
    template_model1: Variables / RHS / AUX
    --------------------------------------------------------------------------
    """
    model_type = 'continuous' # either 'discrete' or 'continuous'
    model1 = do_mpc.model.Model(model_type)

   # Certain parameters
    T = 1

    # States struct (optimization variables):
    L = model1.set_variable('_x', 'L') # Level of tank

    # Input struct (optimization variables):
    F_out = model1.set_variable('_u', 'F_out') # Outlet flow of tank

    # parameters:
    F_in = model1.set_variable('_p', 'F_in') # Inlet flow (FC10)

    # Differential equations
    model1.set_rhs('L', 1/T*(F_in-F_out))

    # Build the model
    model1.setup()

    return model1
```

5.4 Model Predictive Control : Configuration of MPC Controller

Configuration and setup of MPC controller involves following steps

- set_param() function to setup the parameters of MPC controller. The only required parameters are n_horizon and t_step. All other parameters are optional.
- Set the objective of the control problem with set_objective() and set_rterm()
- Set upper and lower bounds with bounds (optional)
- Set further (non-linear) constraints with set_nl_cons() (optional).
- Use high level API (set_uncertainty_values()) to create scenarios for robust MPC (optional).
- Finally, call setup().

Detailed description of individual code can be found in comment section below.

[ ]:
```
import numpy as np
from casadi import *
from casadi.tools import *
import pdb
import sys
sys.path.append('../../')
import do_mpc


def template_mpc1(model1):
    """
    --------------------------------------------------------------------------
    template_mpc: tuning parameters
    --------------------------------------------------------------------------
    """
    mpc = do_mpc.controller.MPC(model1)

    setup_mpc = {
        'n_horizon': 20,  #Prediction horizon of the optimal control problem
        'n_robust': 1, #Robust horizon for robust scenario-tree MPC.Optimization␣
↪problem grows exponentially with n_robust.
        'open_loop': 0, #If the parameter is False, for each timestep AND scenario␣
↪an individual control input is computed.
        't_step': 1,    #Timestep of the mpc
        'state_discretization': 'collocation',  #state discretization for␣
↪continuous models.
        'collocation_type': 'radau', # collocation type for continuous models,␣
↪currently only 'Radau' is available.
        'collocation_deg': 2,  #collocation degree for continuous models with␣
↪collocation as state discretization.
        'collocation_ni': 2,  #number of finite elements for the states within a␣
↪time-step.
        'store_full_solution': True  #required for animating the predictions in␣
↪post processing.

    }

    mpc.set_param(**setup_mpc)
```

```
#Terminal cost - scalar symbolic expression with respect to _x and _p This is␣
 ↪expression leads to maintain mterm to 50  and thus would maintain Level to 50
    mterm = (((model1._x['L']) - 50)**2)

#Stage cost - scalar symbolic expression with respect to _x, _u, _z, _tvp, _p. This␣
 ↪is expression leads to maintain lterm to 50 and thus would maintain Level to 50
    lterm = (((model1._x['L']) - 50)**2)

#Sets the objective of the optimal control problem (OCP) using lterm and mterm
    mpc.set_objective(mterm= mterm, lterm=lterm)


#Call this function with keyword argument refering to the input names in model and␣
 ↪the penalty factor as the value
    mpc.set_rterm(F_out = 1)

#defining lower and upper bound of controlled input and state variable
    mpc.bounds['lower', '_x', 'L'] = 20.0
    mpc.bounds['upper', '_x', 'L'] = 80.0
    mpc.bounds['lower', '_u', 'F_out'] = 0.0
    mpc.bounds['upper', '_u', 'F_out'] = 50.0

    #Define scenarios for the uncertain parameters. High-level API method to set␣
 ↪all possible scenarios for MPC
    mpc.set_uncertainty_values(F_in = np.array([1.0, 5.0, 10.0]))

#Setup the MPC class. Internally, this method will create the MPC optimization␣
 ↪problem.
    mpc.setup()

    return mpc
```

5.5 Model Predictive Control : Optimization solver

Initially we have defined objective function of level and outflow to be solved. It involves the langrage term(lterm) and meyer term(mterm).
Objective of model is to Maximize the output flow keeping the level of tank within bound of 20-80 .
The Lagrangian term is the stage cost (which is added at each time step) and the Mayer term is the terminal cost only added at the final prediction step. The formulation of objective function C and thereby lterm and mterm can be solved as below.
$C = \sum_{k=0}^{n-1} \left(l\left(x_k, u_k\right) + \Delta u_k^T R \Delta u_k \right) + m\left(x_n\right)$

$l\left(x_k, u_k\right) = lterm = \left(x_k - 50\right)^2$

$m\left(x_n\right) = mterm = \left(x_k - 50\right)^2$

$Wherein, Level = x_k, F_Out = u_k$

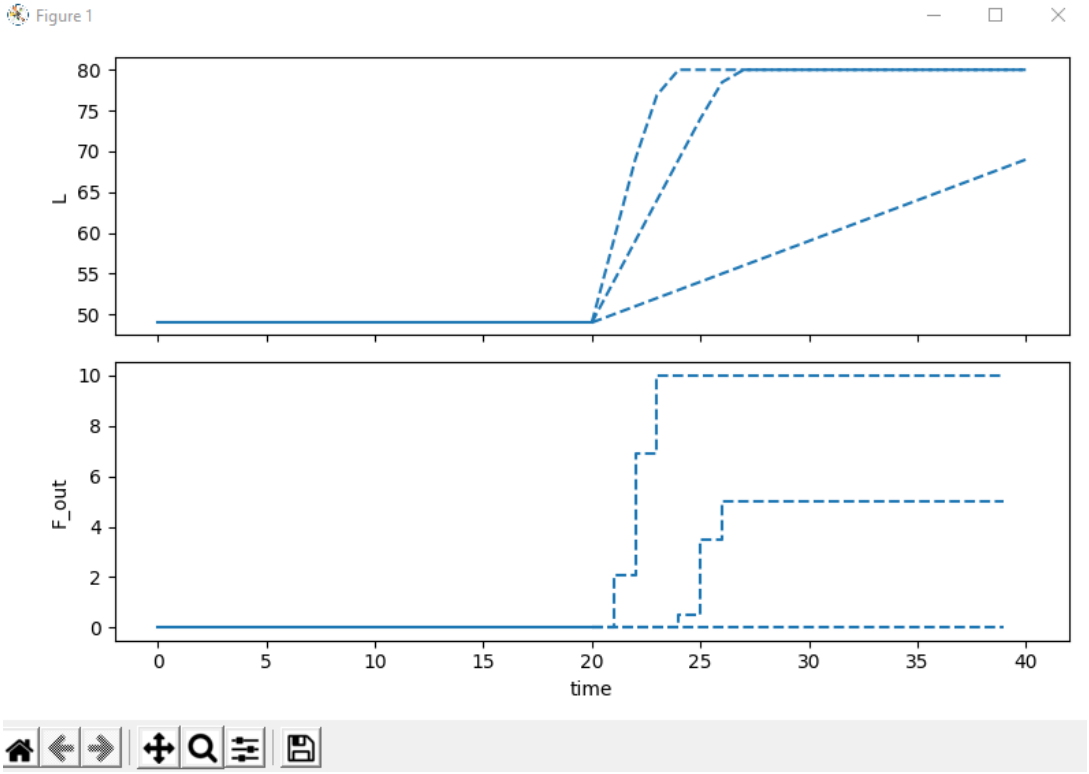
$\Delta u_k^T R \Delta u_k = rterm = F_Out = 1$

Part of the objective function is also the penalty for the control inputs which is rterm. This penalty can often be used to smooth the obtained optimal solution and is an important tuning parameter.

Below screenshot shows the solution of each step of optimization problem and its associated evaluations, equality constraints, Lagrangian hessian matrix evaluation at each step and time required by CPU to calculate the solution of optimization problem.

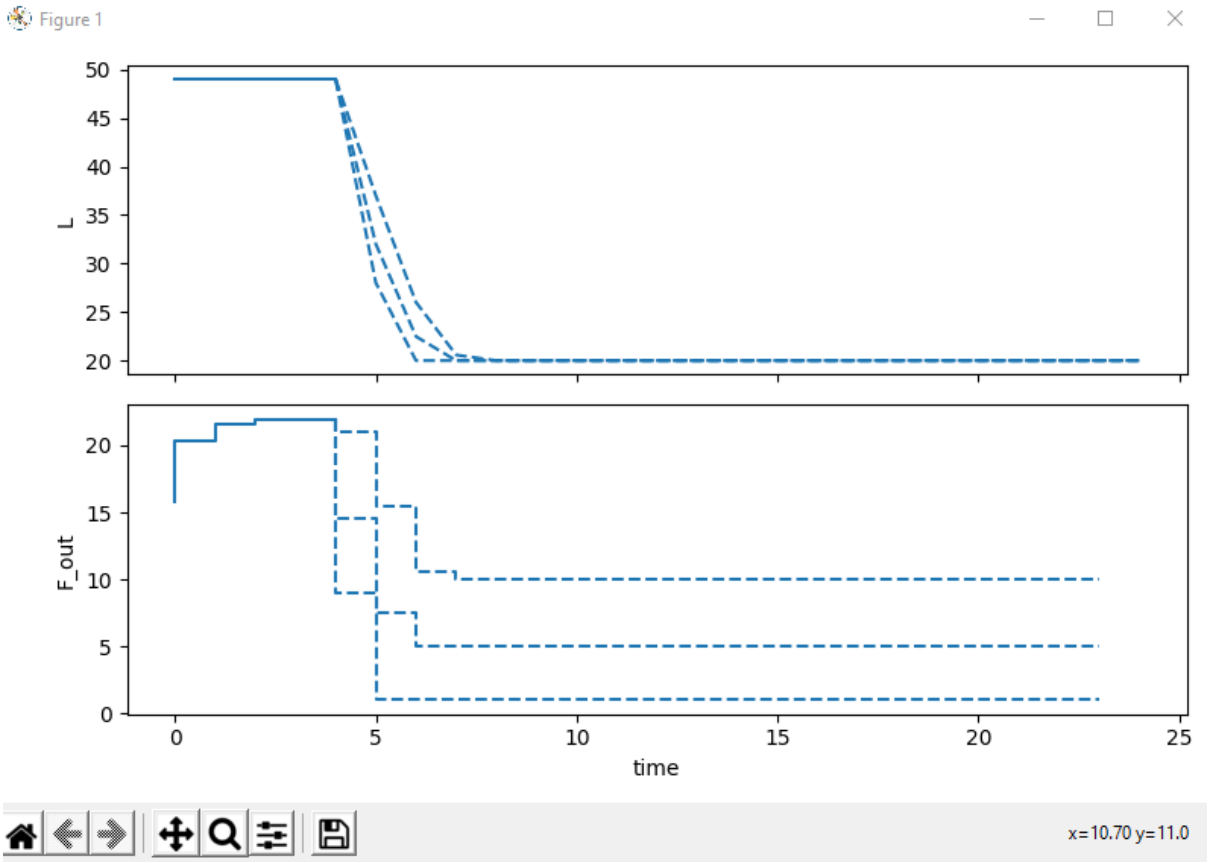
```
OUTPUT  TERMINAL  DEBUG CONSOLE  ...                    1: Python
Objective...............:  9.5456115477187410e+001   9.5456115477187410e+001
Dual infeasibility......:  6.2356319665693224e-013   6.2356319665693224e-013
Constraint violation....:  5.3290705182007514e-014   5.3290705182007514e-014
Complementarity.........:  2.6048214243479841e-009   2.6048214243479841e-009
Overall NLP error.......:  2.6048214243479841e-009   2.6048214243479841e-009


Number of objective function evaluations             = 8
Number of objective gradient evaluations             = 8
Number of equality constraint evaluations            = 8
Number of inequality constraint evaluations          = 0
Number of equality constraint Jacobian evaluations   = 8
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations             = 7
Total CPU secs in IPOPT (w/o function evaluations)   =      0.114
Total CPU secs in NLP function evaluations           =      0.004

EXIT: Optimal Solution Found.
          S  :   t_proc      (avg)   t_wall      (avg)    n_eval
       nlp_f  |        0 (       0)        0 (       0)         8
       nlp_g  |   3.00ms (375.00us)   1.99ms (249.12us)         8
    nlp_grad  |        0 (       0)        0 (       0)         1
  nlp_grad_f  |        0 (       0)        0 (       0)         9
  nlp_hess_l  |        0 (       0)        0 (       0)         7
   nlp_jac_g  |   1.00ms (111.11us)   1.01ms (112.00us)         9
       total  | 185.00ms (185.00ms) 185.00ms (185.00ms)         1
```

Inclusion of equality constraints on objective function makes optimization problem complete. below screenshot shows the upper bound constraint of 80 percent with current level being 50 percent. To test the operation of upper bound, we have chosen the optimal level controlled at 90 percent, which is more than upper bound, but certainly below screenshot shows maximum level predicted to be 80 percent. Hence, we can conclude that upper bound of 80 percent would work.



Similarly, providing optimal solution of 15 percent, which is lower than lower bound of 20 percent,

minimum level tank can predict is 20 percent. Hence, we can conclude that lower bound of 20 percent would also work.



5.6 Model Predictive Control : Simulation

Initially, we made use of Simulation tool from do-mpc is used to test the MPC controller operation without connecting to Unisim. This simulator simulates true values at each time steps with integration method as 'cvodes'. At every timestep, it simulates with new value and history would logged and replaced with new results. below is the code for simulation in Python. This simulation will be called in main python Batch rector file.

[ ]:
```
import numpy as np
from casadi import *
from casadi.tools import *
import pdb
import sys
sys.path.append('../../')
import do_mpc


def template_simulator1(model1):
    """
    --------------------------------------------------------------------------
    template_simulator: tuning parameters
    --------------------------------------------------------------------------
    """
#he simulator is parametrized to simulate with the "true" values at each timestep
    simulator = do_mpc.simulator.Simulator(model1)
    params_simulator = {
        'integration_tool': 'cvodes',
```

```
        'abstol': 1e-10,
        'reltol': 1e-10,
        't_step': 1
    }

    simulator.set_param(**params_simulator)

    p_num = simulator.get_p_template()  #Obtain output template for set_p_fun()
    p_num['F_in'] = 10

    def p_fun(t_now):
        return p_num

    simulator.set_p_fun(p_fun)  #Set function which returns parameters.

    simulator.setup()  #Setup the Simulation.

    return simulator
```

5.7 Model Predictive Control : Order of Execution

In order to operate whole process, we should perform following steps.

- Running the Matrikon Simulation server and client for exchanging data between Python and Unisim clients
- Running Unisim Simulation by connection Unisim to Matrikon
- Running MPC in Python by connecting Python to Matrikon

5.8 Model Predictive Control : Result

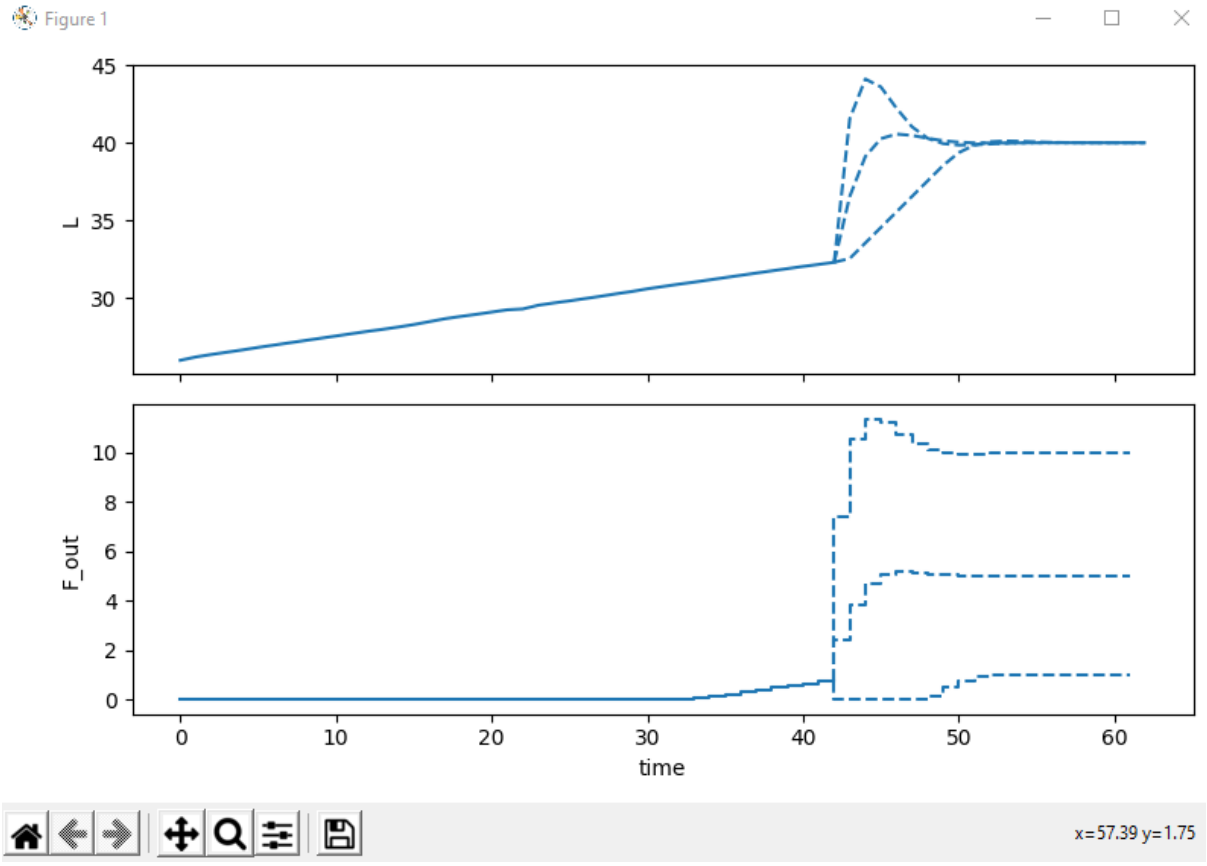
We started simulation in sync with simulation time with unisim. Steps involved were:

- Reading level value from Unisim
- Running optimization problem with current level and flow value
- Finding optimal solution of Output flow value which changes very slowly and gradually with change in level.
- As per objective function, we aimed to maintain level of tank within bound of 20 to 80 percent and optimal value of level of tank maintained being 50 percent level of tank.

As per above scenario, we tested our model for following cases.

5.8.1 Experiment 1: Starting with lower level bound

Initial level of the tank was set to 24% and optimal solution was set around 40% of the tank. Below screenshot predicts the level of 40% and F_ out around 10 m^3/hr, 6 m^3/hr and 0 m^3/hr



After simulation time of 80 minutes, we could see the level of tank is optimized to 48% and output flow of 9 m^3/hr. Below screenshot represents the result of the same.

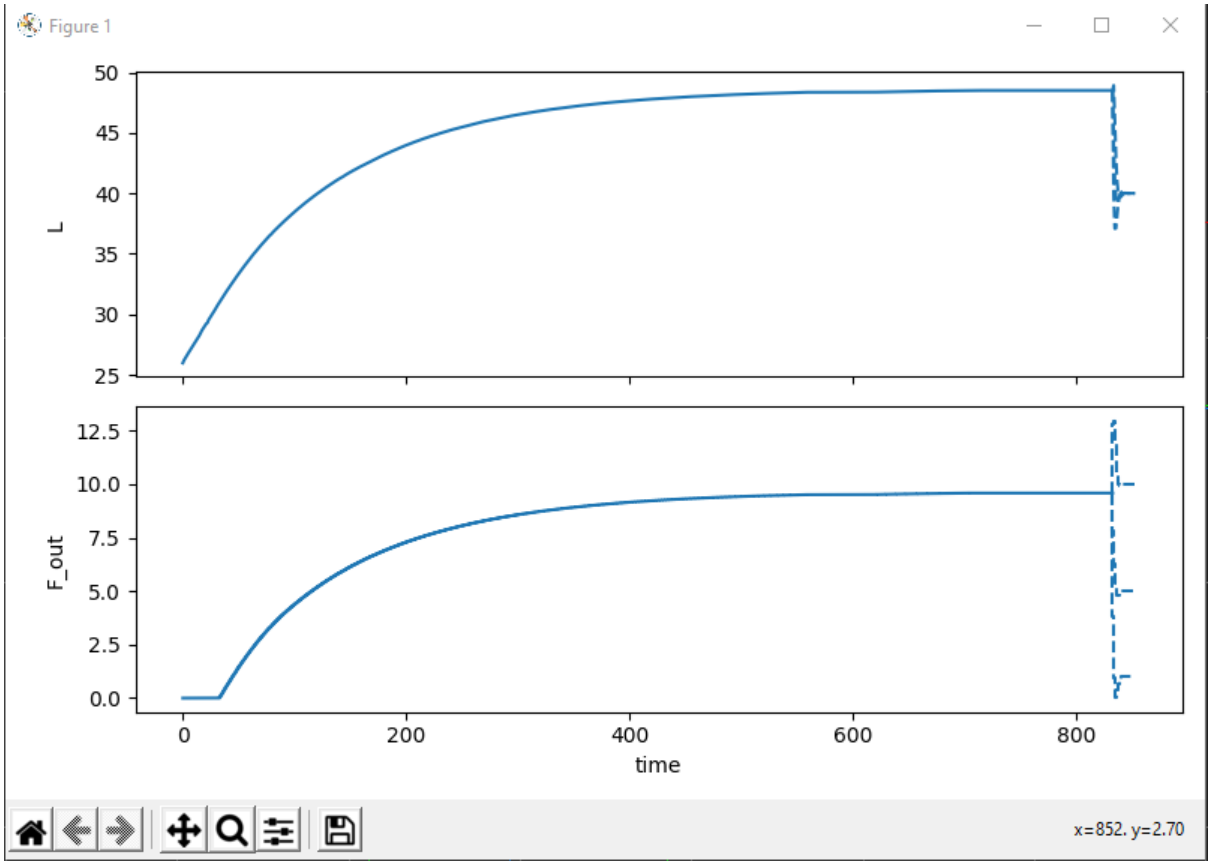
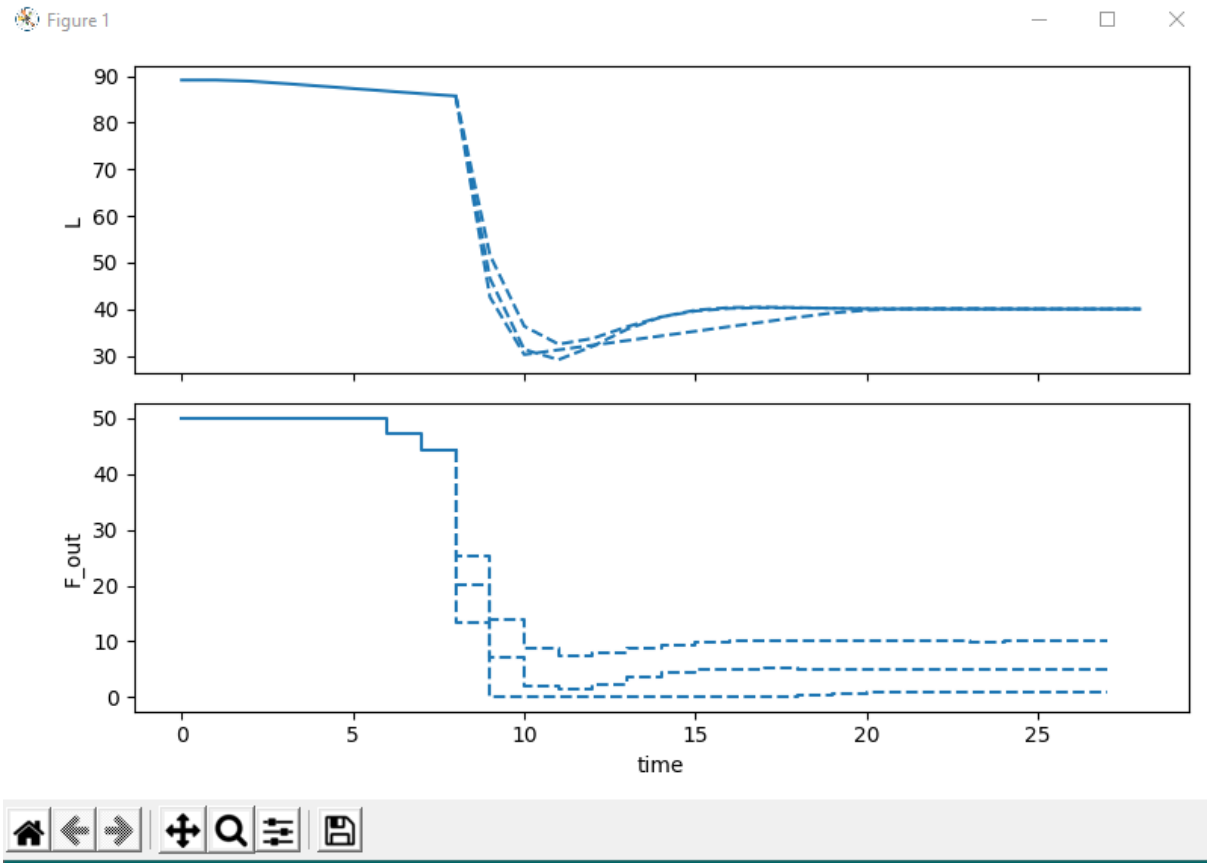


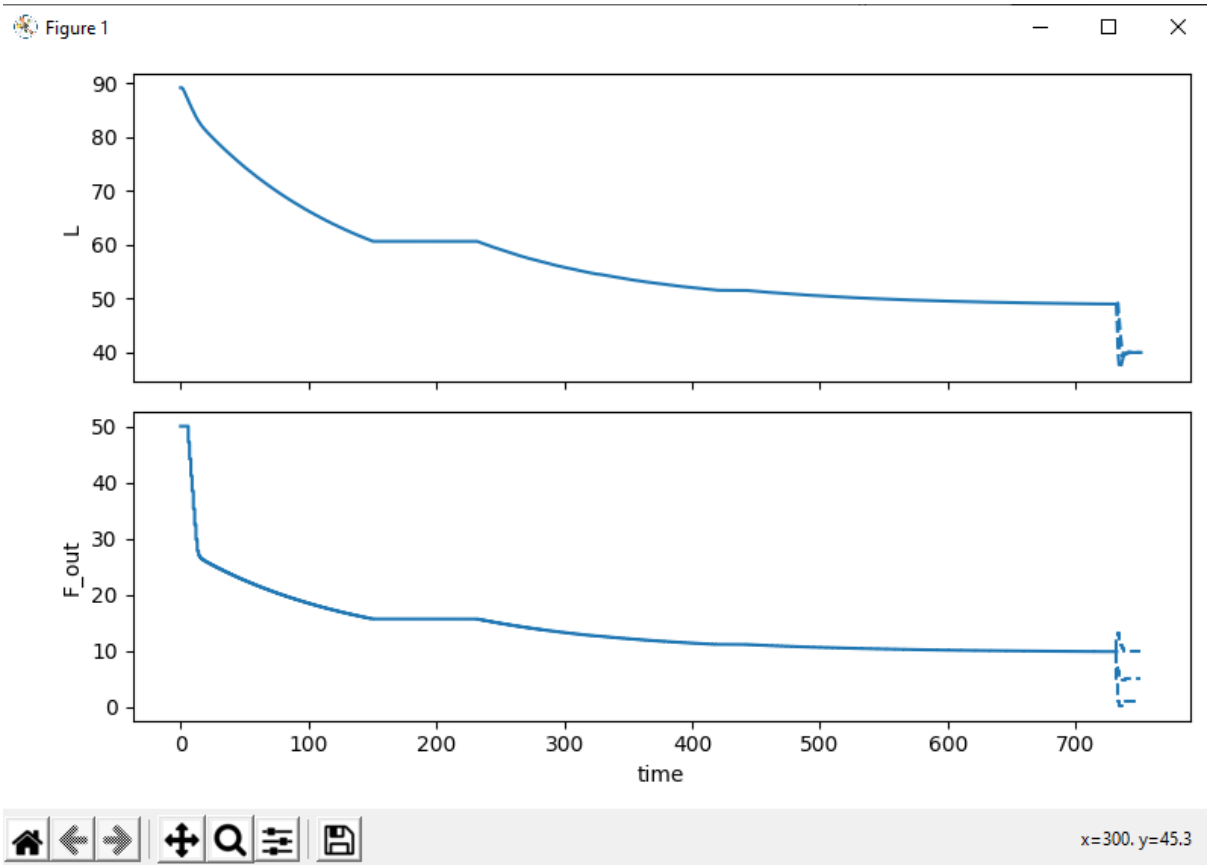
Figure 1
L
F_out
time
50
45
40
35
30
25
12.5
10.0
7.5
5.0
2.5
0.0
0
200
400
600
800
x=852. y=2.70

5.8.2 Experiment 2: Starting with higher level bound

Initial level of the tank was set to 90% and optimal solution was set around 40% of the tank. Below screenshot predicts the level of 40% and F_ out around 10 m^3/hr, 6 m^3/hr and 0 m^3/hr.



After simulation time of 70 minutes, we could see the level of tank is optimized to 48% and output flow of 10 m^3/hr. Hence, With Initial high level of tank, MPC Controller makes higher output flow so that level of the tank can be reduced and with gradual decrease in F_ out makes the level goes to 50% of tank. Below screenshot represents the result.



In the next section, we shall discuss the OPC communication settings in Unisim.

6 OPC

6.1 Settings in Unisim

6.1.1 Creation of Spreadsheet

Once the DCOM configuration had been established between both the client and the server PC, we try to experiment the communication between Matrikon (Server) and Python (Client), UNISIM (Client). As the first step in this process, we create two spreadsheets, OPC_READ and OPC_SEND. From the controller perspective, we set the names in UNISIM, that is OPC_READ would include the tags which are to be sent to Python. OPC_SEND would include the tags which are received from Python to UNISIM.

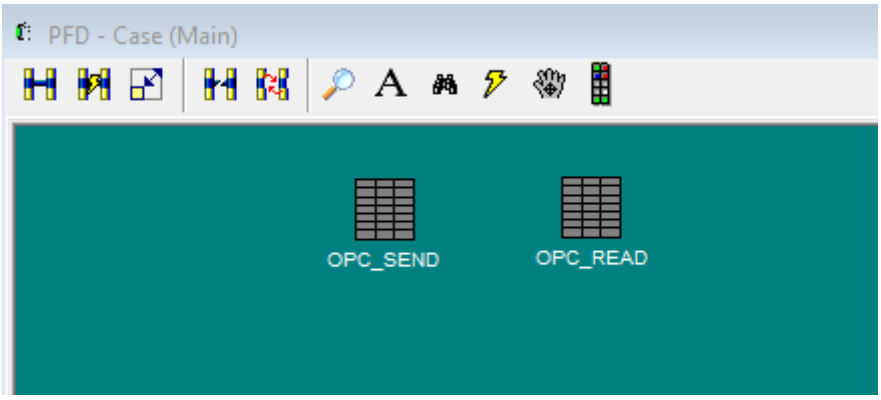


Figure 60: Creation of Spreadsheet

6.1.2 Establishing a connection to the OPC Server

It is to be noted that the earlier versions of Unisim like R390 do not support OPC and we will not be able to make the Unisim as a client. Therefore, it is required to update the version of Unisim to R451 or higher. We go to Tools →Databook or the Ctrl+D combination keys are pressed to open the Databook.

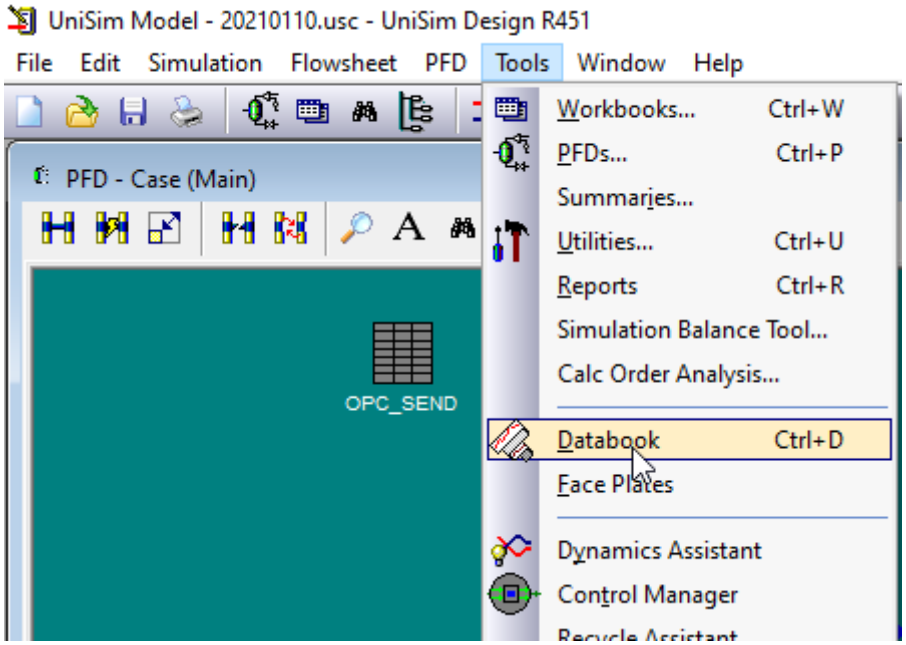


Figure 61: Choosing Databook option from Tools menu

In the Databook dialogue box that appears, we will be able to find the OPC Client tab and we click that tab. We would be able to check that the communication had been done correctly when the Connect

option changes to Disconnect for the OPC Server and there would be an option to choose the OPC Server.

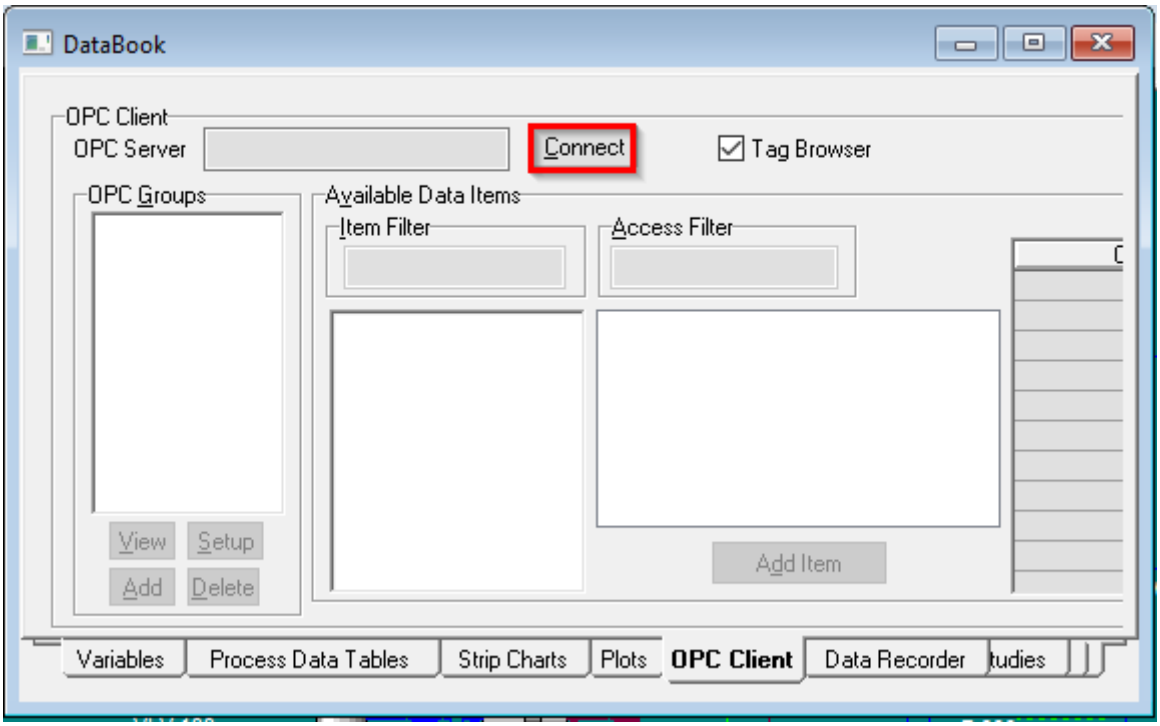


Figure 62: DataBook OPC Client configuration

The Matrikon.OPC.Simulation.1 server is chosen.

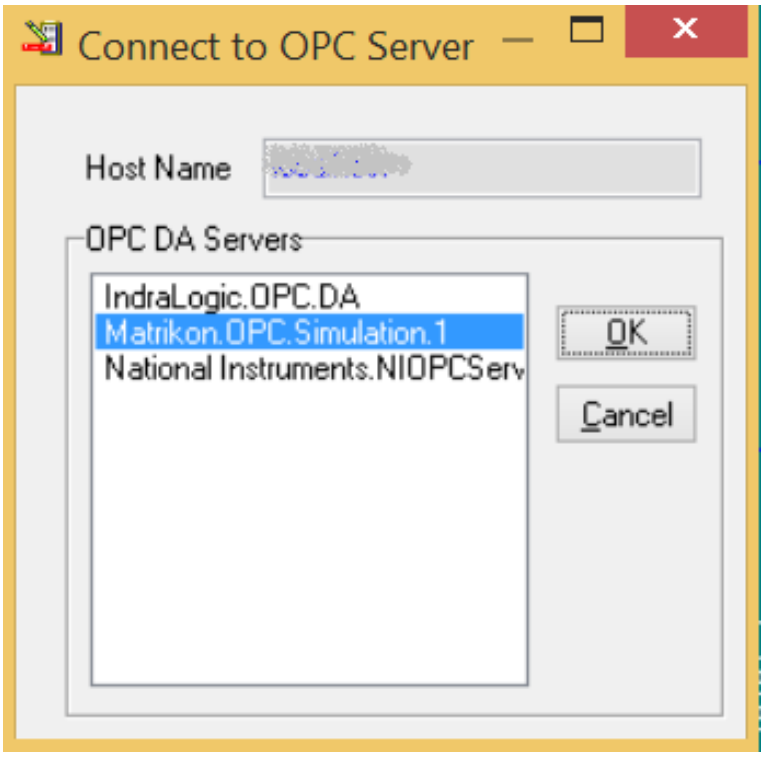


Figure 63: Selecting the OPC DA Server

6.1.3 Managing the tags from Python

We create two OPC Excel sheets, OPC READ and OPC WRITE, so that we can sort the incoming tags accordingly. To do this we choose the Group1 and then click the Setup option below. We change the name in the Group properties and check the update rate if it is 0.5. Also we added the tags of MPC.Level and MPC.F_out under Configured Alias name MPC.

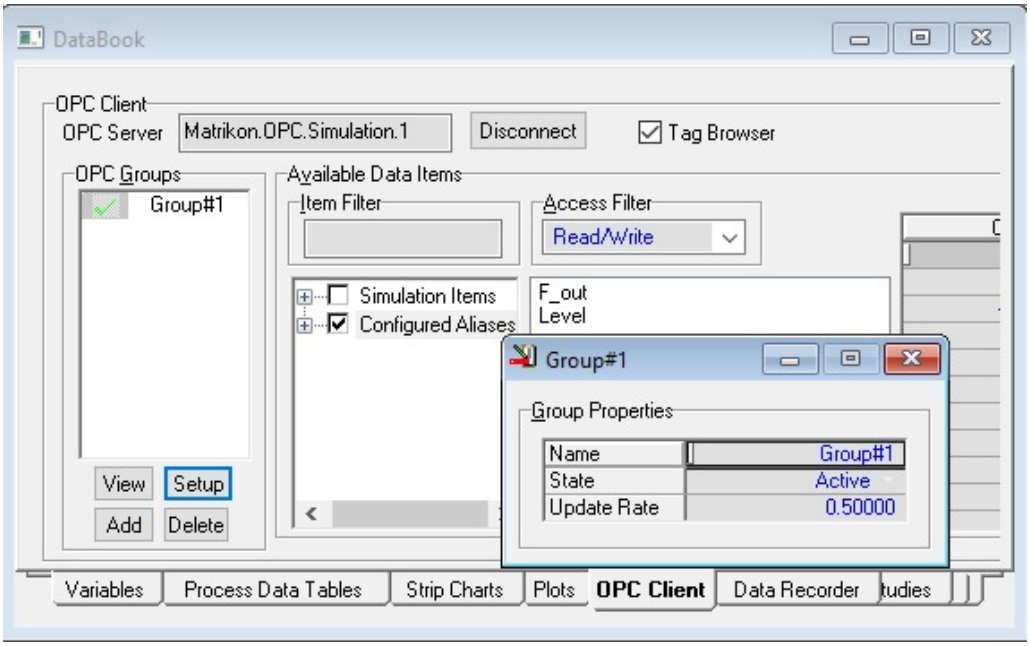


Figure 64: Setup name of OPC Group and Alias

After addition of tags from configured Alias name MPC, we can view these tags by clicking View button.

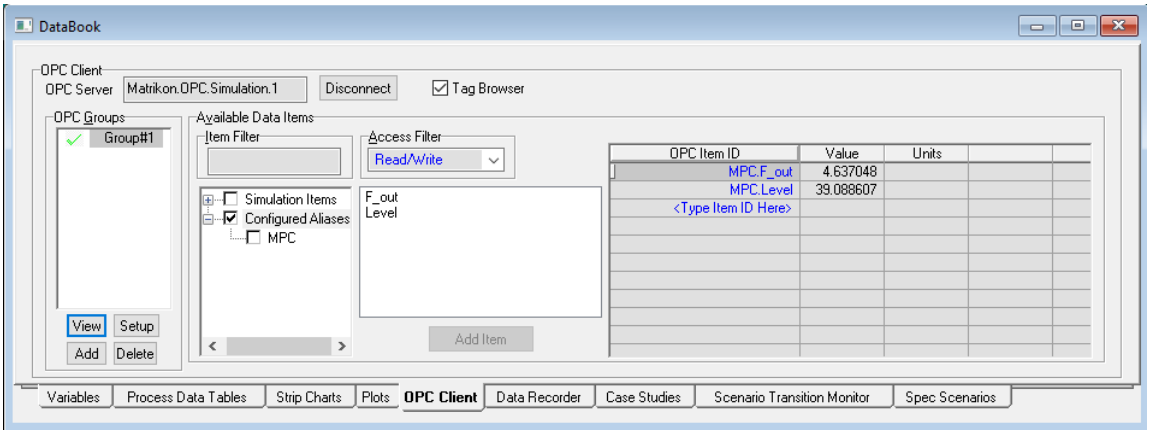


Figure 65: OPC tags after being added to the respective groups

6.1.4 Creation of variables

We then create the variables, which are to be imported or exported via the OPC DA communication.Therefore, we go to the Databook and then choose the Variables tab. Here, the Insert option is clicked and we select the range of cells from both the spreadsheets which should contain the OPC tags.

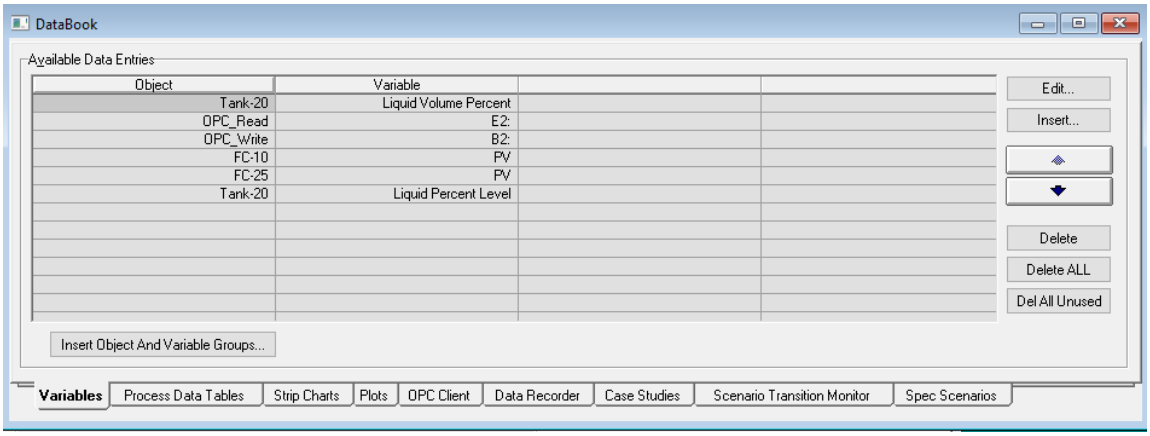


Figure 66: Inserting Variables from Spreadsheet

For our case, we are importing value of level of tank, FC-10 and FC-25 values by selecting the proper spreadsheet as shown below. Below is the example to select liquid level percentage of tank-20.

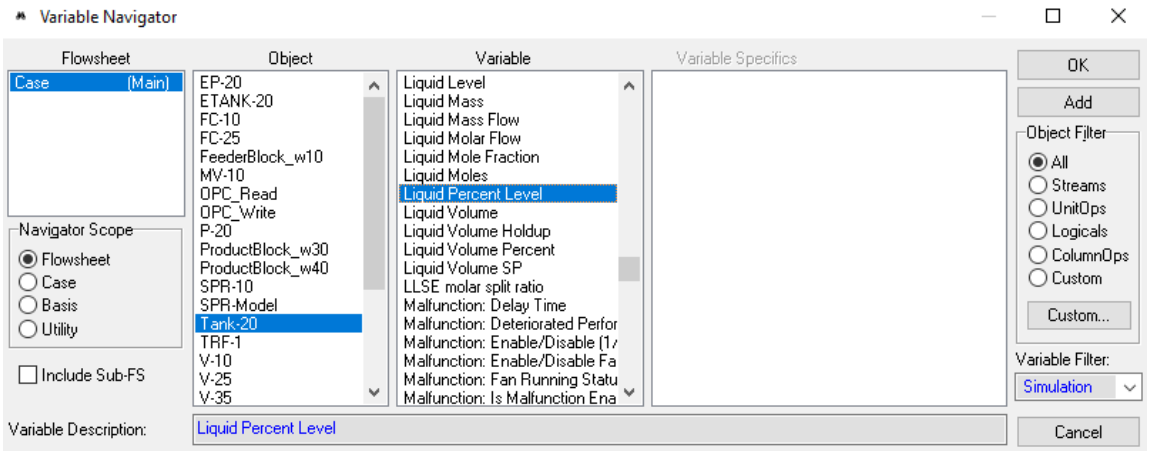


Figure 67: Adding the cell range into the Databook

That is, in OPC READ OPC Group, the variables from Unisim are exported. So we create the variable level by including the last cell E2 from OPC READ spreadsheet. Similarly, in OPC WRITE OPC Group, the values are being imported from Python which is of flow-out, so we create the variables by including the cell B2 OPC WRITE spreadsheet. So value of MPC.F-out from OPC server is written in cell B2 of Unisim OPC WRITE speadsheet. Before that we have to assign created variables to the tags. Procedure is explained in below section.

6.1.5 Assigning the created variables to the OPC Tags

The databook is opened and in the tab OPC Client, the View option is clicked for any of the two OPC Groups. The table that appears next would contain the OPC tags that have been added to that Group. The user would have to then assign the Object (name of spreadsheet) and the corresponding variable to it.

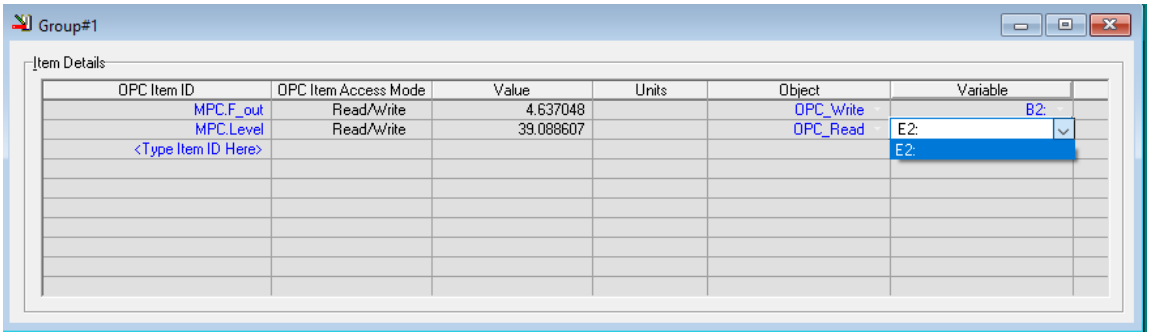


Figure 68: Assigning the object and variable to the OPC Tag

6.1.6 Reading and sending OPC values

In both the spreadsheets we assign a parameter SIM in Cell C, which helps to toggle between the manual and automatic mode for assigning values. 0 - UNISIM model (Automatic mode) 1 - Values fed to the spreadsheet (Manual mode). In the OPC READ spreadsheet, we import the values from the Unisim model in the column B and then export it through the column E. The following condition has been implemented to all the rows in cell E. E2=IF(C2=0,B2,D2),

Similarly, In the OPC WRITE spreadsheet, we use the values from OPC server in the column B, and then we export the value to the suitable objects in the Unisim model. The following condition has been implemented to the rows in cell E and has been exported from the same cell. E2=IF(C2=0,B2,D2)

Below are screenshots of entire OPC READ and OPC WRITE spreadsheet.

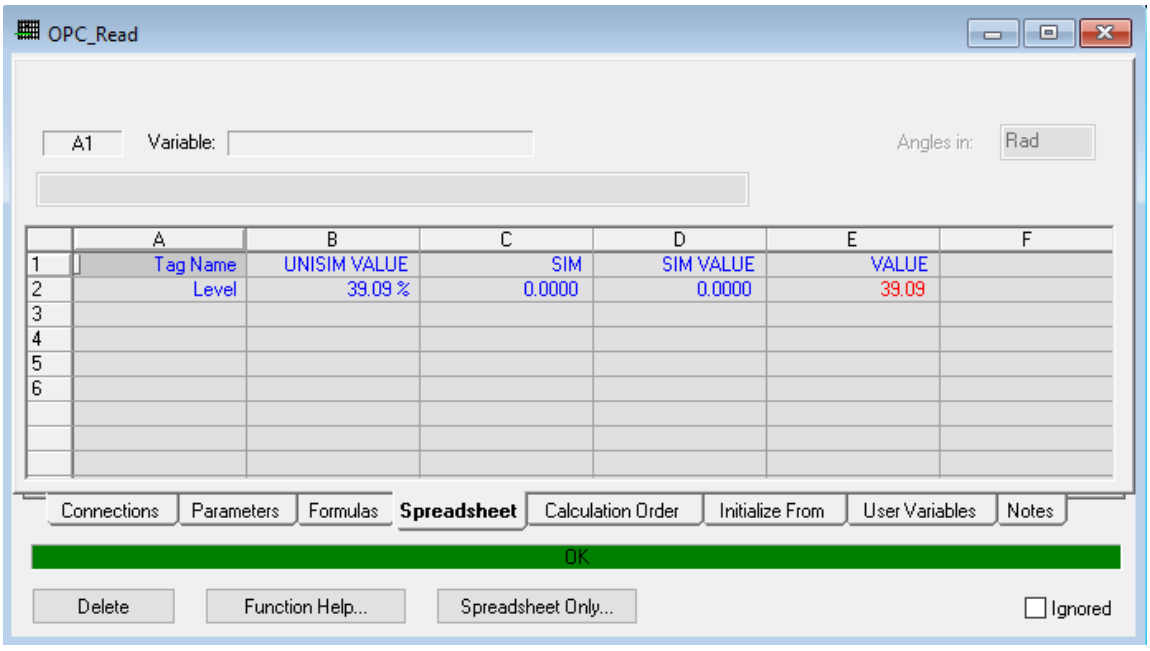


Figure 69: OPC tags after being added to the spreadsheet OPC READ

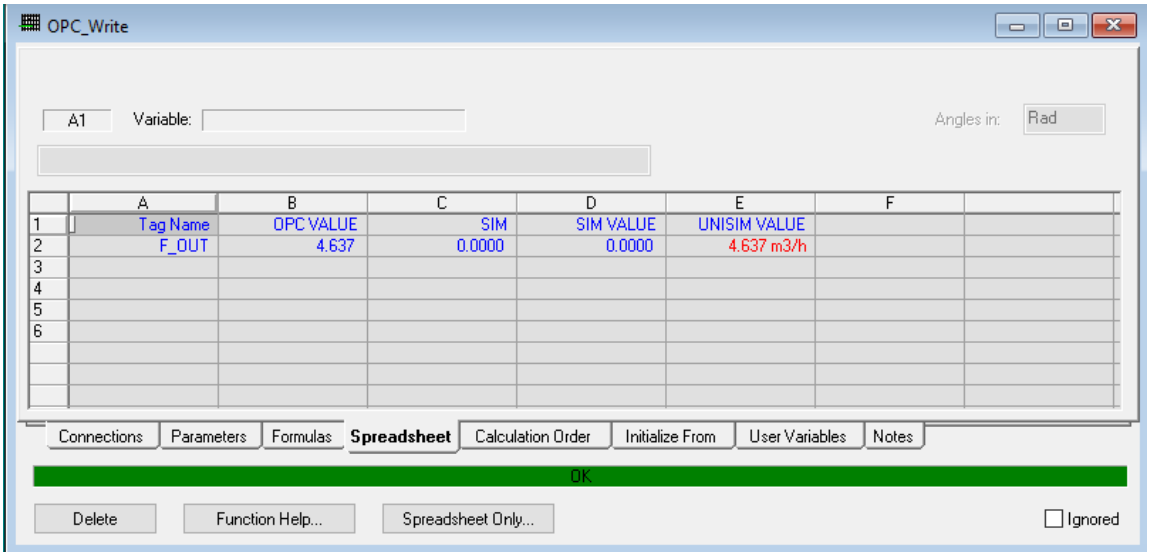


Figure 70: OPC tags after being added to the spreadsheet OPC WRITE

It is very important for the OPC READ variables which are being sent to Python need to be unitless for the communication to take place effectively. The values are made Unitless in the OPC READ spreadsheet, by selecting the Unitless option under the Variable Type drop down list

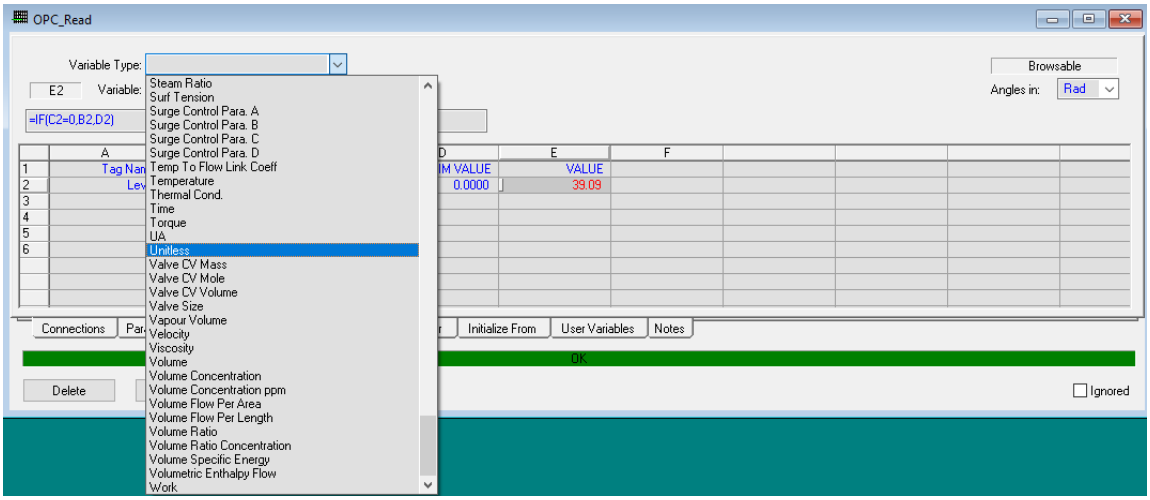


Figure 71: Making the OPC READ variables unitless

The Unisim variables are thus being set for the OPC communication to take place.

In next section, we tried to compare Override and MPC strategy along with their result.

7 Comparison of Override and MPC

Due to the change of PVs and MVs when we move from MPC controlled system to Override Controlled system, it is hard to compare the performance of these controllers. Thus, some observations are made of both strategies and shared in the following section.

7.1 MPC

- We can choose desired set point of level within the upper limit and lower limit, but we can not control the flow set-point
- Change in disturbance variable (F_in) can be compensated through MPC controller action which shows robustness of MPC. This can be realized in an example below.

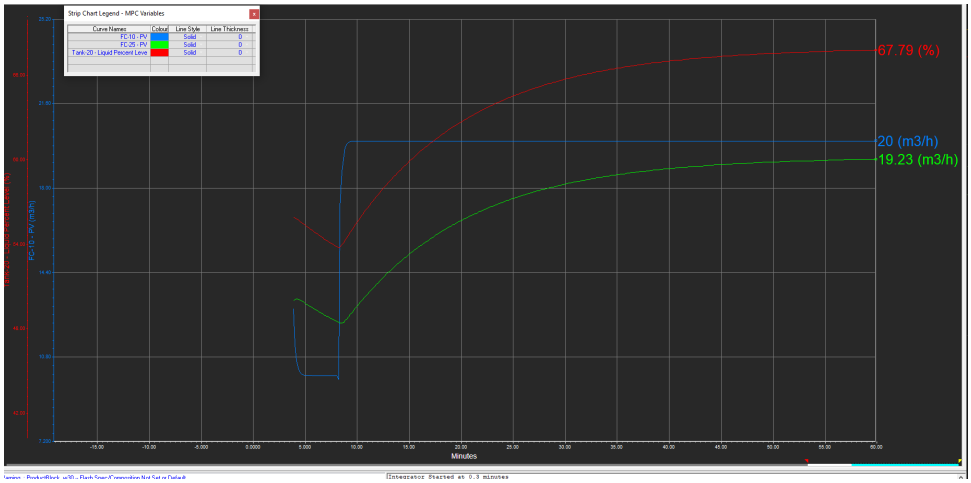


Figure 72: Robustness of MPC to disturbances

Above screenshot, we can see that initially input flow was 10 m3/hr during the operation, it changed to 20m3/hr. Still MPC tried to keep level within the bounds of 20-80 % and maintained the level of tank at 68 % by exceeding more output flow F_ out till 19.23 m3/hr.

This shows robustness of MPC to uncertain disturbances.

- Through objective function of MPC, we can choose the desired optimum value of level, when there is change in disturbance variable (F_in) the controller finds out the optimal solution by reiterating the differential equation so that for every iteration we get optimal (F_out) solution. Therefore, change in disturbance would not affect change in desired level.

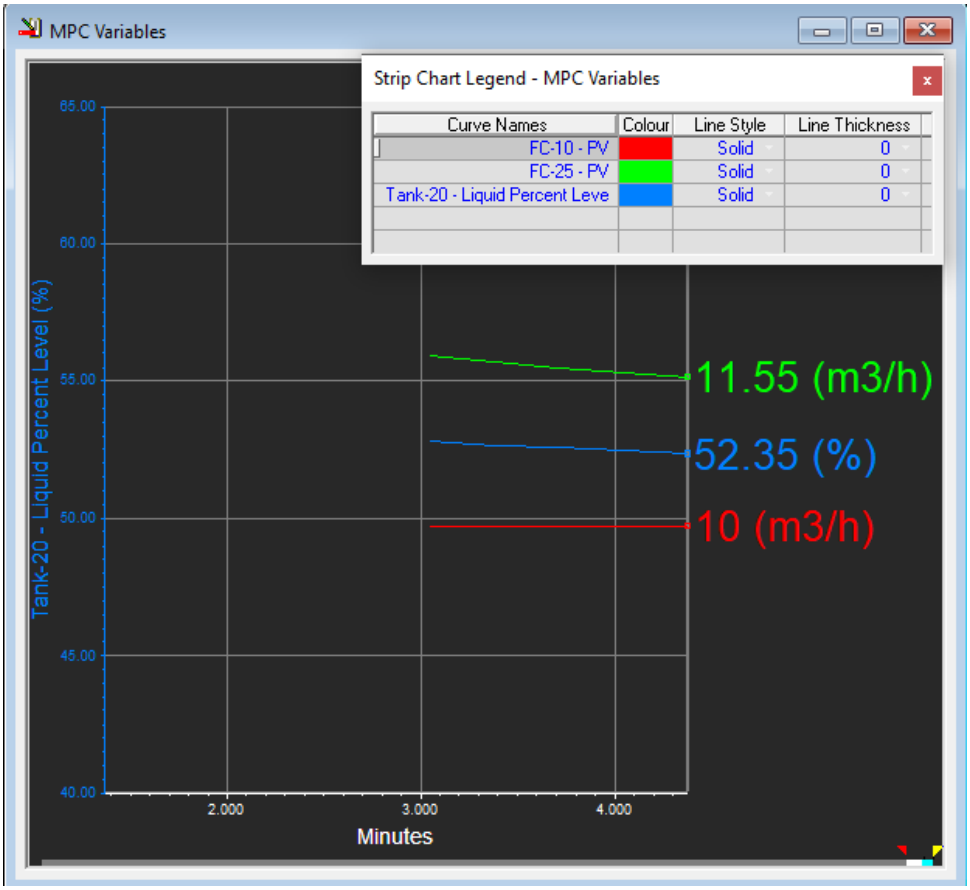


Figure 73: MPC Predictions

Set point of level chosen is 40% and initial PV is 53%. Below screenshot shows MPC control action with OP as F_out with value of 11.55, with level of 55.35% until it reaches 40%

7.2 Override Control

We have tested some scenarios for the Override controller. The Override controller is configured as follows.

Override controller configuration:

LCmin setpoint: 20

LCmax setpoint: 80

FC25 flowsetpoint: 30 m^3/h

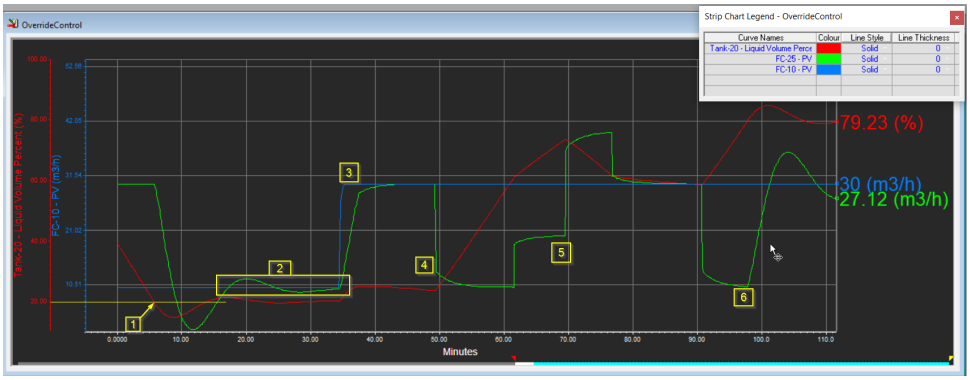


Figure 74: Testing the Override Control strategy

1. The level falls below the low override control limit (20%) and Low level Override control (LCmax) takes over and reduces the outflow so that the level of tank can rise above 20%

2. The Override controller trying to set the Controlled variable (out flow) around the value of disturbance variable.

3. Increasing the disturbance variable to $30\,m^3/h$ will give the flow controller the freedom to operate at its own setpoint ($30\,m^3/h$). Hence the outflow value rises to 30 and level value(25%) is now in between low (20%) and high (80%) limit.

4, 5. Since the level is in the range between low and high limit, the flow controller setpoint can be freely varied.

6. The setpoint of the out flow controller is reduced to $10\,m^3/h$ while the distutbance (in flow) is $30\,m^3/h$. Due to this, the tank level shoots over 80% at point 6 and High level override control (LCmax) takes over. The override control picks up the out flow or controlled variable such that the level stays below 80%.

Both the control strategies hold the process value within the safe operation regions. MPC algorithms are Computationally Expensive and for small control problems like Level control, MPCs are not a popular choice. Our MPC controller uses basic optimizers but it can be configured to use more complex optimizers/solvers from MA27 library for Python. As our optimization problem is linear optimization problem, we necessarily should not involve in more computational efforts using MPC. For more complex plants, MPC with non-linear solvers could be effectively used to solve the optimization problem.

Since process of level control is slow when compared to pressure or flow control, the MPC algorithm does not actually have to run every second or some milliseconds. From our experiments we have found out that running MPC once in every 5 to 10 seconds is sufficient to control the process and handle the disturbances.

8 Conclusion

When we started the case study, we had lot of doubts regarding building an MPC controller in Python. Although we had theoretical knowledge on Override Controller, implementing it for a real problem at hand seemed little tricky. During the course of the project, we have learnt much more than MPC and Override controllers. Working with a team of three on a version control system (GitHub), interfacing the software running on different machines, building models and running simulations to test accuracy, have allowed us to evolve ourselves into multi-skilled individuals.

Building the MPC controller took a large portion of the time spent on the case study. We had experimented with a lot of ideas to get the MPC controller up and running. The process was exiting as we hurdled through the challenges and road blocks.

The only justification to use OPC-DA at this point in time is to provide a connection to Unisim that only supports OPC-DA which cannot be replaced or updated. From the realistic experience gained during the case study, OPC UA can be used regardless of whether or not OPC DA servers are available. Since there are solutions such as OPC tunnellers which allow information from OPC DA products, new OPC UA based products can still be obtained.

References

[1] Barracuda Campus. (May 02, 2015). OPC DA (DCOM) Firewall Configuration. Barracuda Campus. Retrieved February 24, 2021 from https://campus.barracuda.com/product/archiveone/doc/46206124/how-to-configure-the-firewall-to-allow-dcom-connections/.

[2] do-mpc. do-mpc: Model Predictive Control in Python. www.do-mpc.com. Retrieved March 05, 2021 from https://www.do-mpc.com/en/v4.0.0-beta/getting_started.html.

[3] Honeywell Process Solutions (January, 2017). UniSim Design Suite. www.honeywellprocess.com. Retrieved March 05, 2021 from https://www.honeywellprocess.com/library/marketing/notes/unisimdesign-pin-r451.pdf.

[4] Joel A E Andersson and Joris Gillis and Greg Horn and James B Rawlings and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. Springer. Retrieved March 05, 2021 from https://web.casadi.org/.

[5] Kepware. (July, 2019). Quick Start Guide, OPC DA (DCOM). Kepware. Retrieved February 24, 2021 from https://www.kepware.com/support/resource-library/connectivity-guides/remote-opc-da-dcom.pdf.

[6] LSoft Technologies Inc. (n.d.). DCOM Configuration. ActiveActive UNDELETEUNDELETE. Retrieved February 24, 2021 from http://active-undelete.com/dcom-configuration.htm.

[7] Lucia, Sergio Tatulea, Alexandru (December 18, 2020). do-mpc documentation release 4.0.0. https://www.do-mpc.com/en/v4.0.0/. Retrieved February 24, 2021 from https://www.do-mpc.com/en/v4.0.0/.

[8] Meijer, Erick. (July 21, 2013). The apacite package[PDF file]. TeXdoc Online. Retrieved February 24, 2021 from http://www.texdoc.net/texmf-dist/doc/bibtex/apacite/apacite.pdf.

[9] n.d. (April 29, 2010). Inserting a PDF file in LaTeX. StackOverflow. Retrieved February 24, 2021 https://stackoverflow.com/questions/2739159/inserting-a-pdf-file-in-latex.

[10] n.d. (November 10, 2007). Landscape in Latex. TexBlog. Retrieved February 24, 2021 from https://texblog.org/2007/11/10/landscape-in-latex/.

[11] n.d. (October 07, 2017). Bibliography in LaTeX with Bibtex/Biblatex. LaTeX-Tutorial. Retrieved February 24, 2021 from https://www.latex-tutorial.com/tutorials/bibtex/.

[12] Oberdiek, Heiko. (August 30, 2014). Headheight Problem. StackExchange. Retrieved February 24, 2021 from https://tex.stackexchange.com/questions/198692/headheight-problem#198694.

[13] Overleaf. (2020). Paragraph formatting. Overleaf. Retrieved February 24, 2021 from https://www.overleaf.com/learn/latex/Paragraph_formatting.

[14] Overleaf. (n.d.). Understanding underfull and overfull box warnings. Overleaf. Retrieved February 24, 2021 from https://www.overleaf.com/learn/how-to/Understanding_underfull_and_overfull_box_warnings.

[15] PROMOTIC. (n.d.). DCOM OPC DA Server Configuration. SCADA system documentation. Retrieved February 24, 2021 from https://www.promotic.eu/en/pmdoc/Directions/CfgDcomOpc.htm.

[16] Sascha, Frank. (n.d.). LaTeX table fixed width. Retrieved February 24, 2021 from http://www.sascha-frank.com/Faq/tables_six.html.

[17] Sullivan. (November 22, 2011). How do I use APA-style citations with BibTeX? Stack Exchange Inc. Retrieved February 24, 2021 from https://tex.stackexchange.com/questions/35809/how-do-i-use-apa-style-citations-with-bibtex.

[18] TeX. (November 02, 2008). How does one insert a backslash or a tilde ( ) into LaTeX? StackExchange. Retrieved February 24, 2021 from https://tex.stackexchange.com/questions/9363/how-does-one-insert-a-backslash-or-a-tilde-into-latex.

[19] TeXBlog. (December 21, 2012). Multi-column and multi-row cells in LaTeX tables, multiple columns. TeXBlog. Retrieved February 24, 2021 from https://texblog.org/2012/12/21/multi-column-and-multi-row-cells-in-latex-tables/.

[20] Vellage, Claudio. (November 01, 2017). LaTeX list - Enumerate and Itemize. LaTeX-Tutorial.com. Retrieved February 24, 2021 from https://www.latex-tutorial.com/tutorials/lists/.

[21] WEBfactory). Establishing OPC Communication. www.WEBfactory.com. Retrieved March 05, 2021 from https://docs.webfactory-i4.com/webfactory2010/en/establishing-opc-communication-on-different-windows-versions.html.

[22] Wickerson, John. (May 08, 2013). Appendix package, making appendix subsections. TEX StackExchange. Retrieved February 24, 2021 from https://tex.stackexchange.com/questions/113195/appendix-package-making-appendix-subsections.

[23] Wikia Org. (n.d.). List of LaTeX symbols. LaTeX Wiki. Retrieved February 24, 2021 from https://latex.wikia.org/wiki/List_of_LaTeX_symbols.