AN EMBEDDED INDUSTRIAL CONTROL SYSTEM FRAMEWORK FOR MODEL PREDICTIVE CONTROL OF A DISTRICT HEAT SUBSTATION

Joakim Örneskans [a,]*, Konstantinos Kyprianidis [b], Stavros Vouros [c], Gunnar Bengtsson [d]

[a] *Mälardalen University,* [b] *Mälardalen University,* [c] *Mälardalen University,* [d] *First Control Systems AB*
joakim.orneskans@mdu.se

Abstract

In this paper we present a standard platform XC05 for an Edge Controller based on an Industrial Control System, where functions made in Modelica and Python can be run as an integrated part of an automation system. We demonstrate how the platform is used to run a complex Model Predictive Control (MPC) strategy to optimize indoor heating in a residential building. MPC strategies have been increasingly popular due to their ability to handle nonlinear dynamics with constraints and multi-objective optimization. Since industrial control systems are real-time based, consideration must also be taken to running security and the real-time characteristics and timing of the overall system solution. We also show that heavy calculation, protected by the industrial control system operative, can run safely together within fast automation using standard electronics. The controlled variable in the MPC strategy is the supply water temperature (Space heating), and the objective is to keep the indoor temperature at a predefined setpoint despite variations in outdoor weather conditions by using local measurements and weather forecasts from the Swedish weather service SMHI. The model used in the MPC is trained automatically with real-time data during running. We describe the controller architecture and briefly the model predictive control algorithm, analyze the overall system performance regarding safety and real-time characteristics. The proposed model predictive control application showed stable operation and expected real-time characteristics during operation. Furthermore, a reduction in indoor temperature deviations was achieved.

1. Introduction

The XC05 automation platform was developed as an activity in the DISTRHEAT research project where the aim of the project is to demonstrate and test in real operating environment (MPC) applied to District Heating and Cooling networks. The automation platform is in fact an "Edge controller" as expressed in the standard Industry 4.0 i.e., a highly intelligent unit interfacing the process or machine. Such controllers are expected to replace conventional PLC systems in the future. The details of the software design are described below.

In the DISTRHEAT project we applied our ideas to Model predictive control (MPC) for optimizing the indoor temperature in residential buildings. Model predictive control (MPC) is an optimal control technique where the control actions minimize a cost function over a finite specified time horizon.[1]

Using MPC for controlling different processes in the heating sector has proven to be highly successful as shown in [6] [7] [8] and [9]. To implement an MPC in a conventional control system (PLC) has been done [2] but requires deep understanding of the underlying PLC language and is often limited to special use-cases. The novelty of the proposed solution is using well known languages and tools which interact in a safe way with the time-critical functions of the PLC software,

The research project includes all the basic steps of development such as control methods, simulations, and installation in a physical process. The automation platform was developed to cover all these aspects. It includes an industrial control system (ICS) which is integrated with commonly used research tools like Python scripts and Modelica simulations. In XC05 we introduce a new standard how to integrate research tools with an ICS system. For this purpose, we developed a graphical tool, FirstGraph where the objects in the ICS and the extended Python and Modelica libraries can be connected in a simple way. The tool is an extension of what we previously have used in the ICS and is understood by any process engineer. At each time step, the MPC controller receives or estimates the current state of the process being controlled. It then calculates the sequence of control actions that minimizes the cost over a specified time horizon by solving a constrained optimization problem that relies on an internal process model and the current system state. The controller then applies the first computed control action for the building and the procedure is then repeated each time instance. The internal model in the MPC algorithm is automatically updated from real-time data during periodical learning periods which are run in parallel to the control actions. Data are assembled and stored by the ICS part. The temperatures in the building are read from a local PLC via a standard Modbus TCP communication line and the weather forecast are read periodically from the Swedish Meteorological and Hydrological Institute (SMHI).

This MPC application was a good test of the XC05 platform since it consists of both heavy calculations in the MPC algorithm and fast real time data handling. Moreover, the platform had to integrate MPC software made by other researchers with the ICS functions and safety supervision. Before it was connected to the building, the MPC application was run in a real-time simulation inside the XC05 to verify the operation, capacity and safety in the simple standard electronics used (Raspberry 4). Linux with the RT_PREEMPT patch is chosen as the operative system because of its stable real-time performance [3][4][5]. The MPC installation has now been in operation for more than five months without a single operation failure.

2.Methodology

The basic task was to integrate an ICS system with functions created in Modelica and Python. The new platform is based on the same principles that were used in the ICS system, that is used as a base for the platform.

2.1. Industrial control system

We choose our own ICS which has been used in many industrial installations and therefore is very safe and contains all the functions needed in an automation task including our own version of adaptive control. It is based on the self-tuning concept and has been awarded by the IEEE. The ICS is then transformed to the Linux operative with necessary changes to preserve the original properties of the ICS platform.

2.2. Python scripts

The python scripts are made in the normal way in a standard PC and then loaded into the XC05 platform where they are stored in a special library dedicated to Python scripts. They must be supplied with simple standard input/output functions to interface the ICS.

2.3. Modelica simulation

The Modelica simulation is designed in the normal way on a standard PC and then loaded into the XC05 where it is compiled locally and stored in a special library dedicated to Modelica models.

2.4. Supervision and safety

The Python scripts and the Modelica models are external software which may contain errors. Therefore, they must be supervised and disconnected in case of malfunction or a software error to prevent disturbance in the automation system. The ICS part contains by itself safety protection developed for very sensitive processes in the steel and energy areas.

2.5. Functional integration

All functions in the platform are regarded as “modules” or “objects” with specified inputs and outputs according to the same standard as is used in the ICS system. This means that much of the software developed for the ICS can be reused in this case.

2.6 MPC algorithm

The core focus of this paper is the ICS edge controller, therefore only a brief description will be provided for the MPC kernel that runs inside. The MPC is a Python-based optimizer that is employing a machine learning model for building heat demands and thermal comfort. The objectives of the optimizer comprise multiple indices related to energy performance of the heating system or indoor temperature in the apartments.
Anyone familiar with control theory knows that PID technique developed for more than 50 years ago is far too limiting in this case since its internal structure is limited, not prepared for predictive control, and does not support feedforward. According to the internal model principle, any efficient control algorithm must contain an internal model of the process and its disturbances. A candidate would be a general adaptive controller based on the self-tuning principle, which is in fact an adaptive MPC based on a linear dynamical model. Such a regulator is available in the ICS part of XC05 and has been used in many installations. In this case, however, an MPC controller was favored since it is more general, can handle nonlinear physical models and the optimization algorithm can be adjusted to the specific case. Such a solution has also a higher development potential in the future. The drawback is the computational burden caused by the optimization and model training, but we have demonstrated it is quite feasible within the XC05Edge controller.

2.7 Experimental study

An experimental study was conducted in a residential building. The MPC algorithm was developed and customized for the building's district heat substation. It used real-time data from the local PLC, such as temperature and outdoor weather conditions, to optimize the operation of the system. The MPC algorithm determined the optimal setpoints for the forward temperature based on keeping the indoor temperature at a fixed setpoint. To evaluate the performance of the system, data were collected at one- and sixty-minutes intervals throughout the test period. Descriptive statistics, such as mean value and standard deviations were calculated before and after installation of the system.

3. Edge computing

The automation platform described in this paper is an "Edge Controller" as expressed in Industry 4.0. Such controllers are expected to replace conventional PLC systems in the future.

An important issue is to spread research results to ordinary process engineers. This means that the way the engineer interfaces the new technology should be according to the standards used in common automation.

Our previously developed tool FirstGraph for the ICS, which has been used by engineers for several years, has been extended to the generalized XC05 platform. It provides a standard format where all functions are regarded as "objects" which are connected to each other. The platform then unifies the automation functions in ICS with external functions created in Modelica or Python.

The FirstGraph project tree has been extended with two new libraries Modelica and Python.

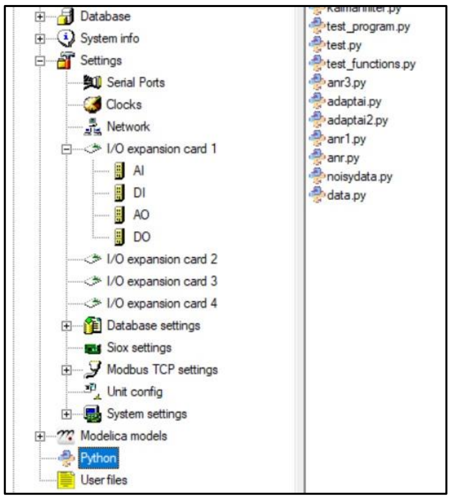


Figure 1: The XC05 graphical programming interface for engineers.

The process engineer selects elements from the extended libraries and connects them graphically to the automation system as is illustrated in figure 7. This can be done without stopping the running control task. Note that a program change will be active within about 0.5 msec after being loaded since it is handled locally by the operative.

If there is an MPC function or a Modelica function loaded to the library, the process engineer may directly use it in the automation system which creates a direct link between researchers and users. This was an important factor in this development.

4.Proposed architecture

4.1. Embedded system Architecture

To make the entire system portable to different hardware architectures all software are developed in C99 targeting different Linux distributions. It is also possible to port the framework to other operating systems by using a simple API to interface the new operating system.

The system consists of four main modules and a supervisor where all are strictly prioritized based on their different functionalities.

These modules are:

- Main controller task (ICS)
- Simulation executor
- Supervisor task
- Communication task
- Python executor

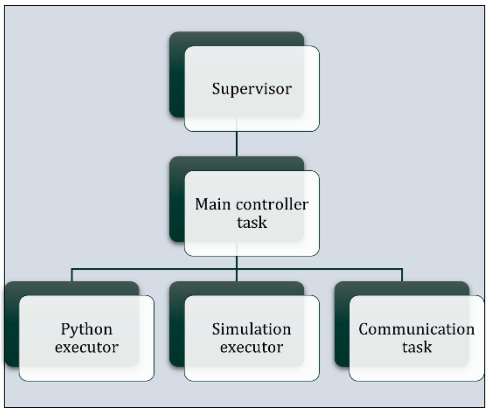


Figure 2: Embedded system architecture

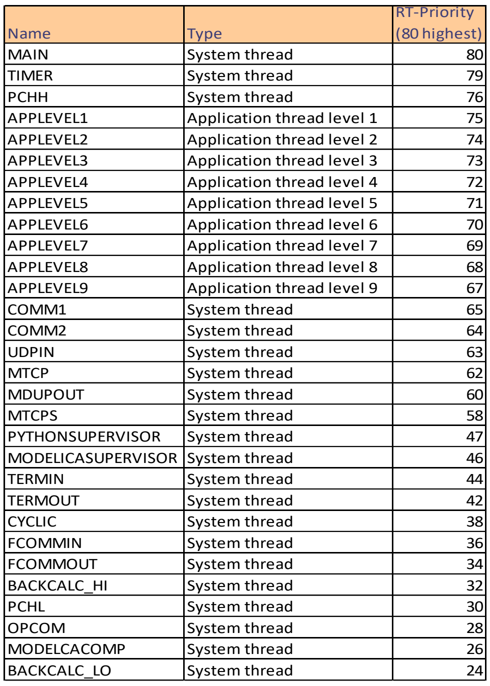
The main controller task (ICS) handles all PLC functionality and executes all user applications. These applications could be run on eight different priority levels depending on their purpose. It's also responsible for creating the different subtasks and supervising these.

The simulation executor is responsible for running simulation of user-defined models created in the OpenModelica language and communication with the main controller task by using shared memory.

All external communication is handled by the communication task which supports a variety of industrial protocols like Modbus TCP/IP, Modbus RTU and others. The Python executor handles the execution of user defined python scripts and communicates with the main controller task by using POSIX queues. Each parent task in the systems supervises and receives errors from subtasks. All errors are forwarded to the supervisor task which takes different actions depending on the severity of the error. More detailed information about the principles in the XC05 platform [10].

4.2 ICS task

The threads in the main controller task are strictly prioritized with respect to their functionality. Application level indicates nine different levels (priority 67 to 75, 75 being the highest) where user-applications could be executed at different priorities. The system is strictly event based and the threads are only woken when some external- or timer event occurs. The main controller task has the highest thread priorities in the system except for the Supervisor task main thread. The communication task main thread will inherit the same priority as the thread handling the communication in the main controller task (COMM1 and COMM2). To minimize latency times, all threads which belong to the main controller are directed to a single CPU-core.



Name	Type	RT-Priority (80 highest)
MAIN	System thread	80
TIMER	System thread	79
PCHH	System thread	76
APPLEVEL1	Application thread level 1	75
APPLEVEL2	Application thread level 2	74
APPLEVEL3	Application thread level 3	73
APPLEVEL4	Application thread level 4	72
APPLEVEL5	Application thread level 5	71
APPLEVEL6	Application thread level 6	70
APPLEVEL7	Application thread level 7	69
APPLEVEL8	Application thread level 8	68
APPLEVEL9	Application thread level 9	67
COMM1	System thread	65
COMM2	System thread	64
UDPIN	System thread	63
MTCP	System thread	62
MDUPOUT	System thread	60
MTCPS	System thread	58
PYTHONSUPERVISOR	System thread	47
MODELICASUPERVISOR	System thread	46
TERMIN	System thread	44
TERMOUT	System thread	42
CYCLIC	System thread	38
FCOMMIN	System thread	36
FCOMMOUT	System thread	34
BACKCALC_HI	System thread	32
PCHL	System thread	30
OPCOM	System thread	28
MODELCACOMP	System thread	26
BACKCALC_LO	System thread	24

Figure 3: Threads and priorities main controller task

The applications written in the ICS by the user are executed on APPLEVEL1 – 9 which have among the highest priorities in the system.

4.3 Simulation executor

The simulation executor is responsible for executing code written in the Modelica language which is used in various academic institutes as well in industry. In order to accomplish this the complete OpenModelica package is preinstalled on the system. This includes the OpenModelica compiler, the OpenModelica simulator and a large library of predefined objects.
The simulation executor consists of two parts, the Modelica compiler and the Modelica executor.

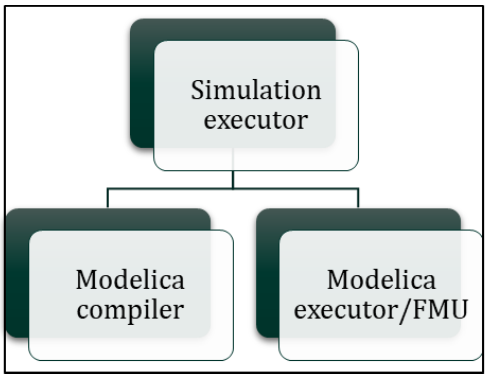


Figure 4: The simulation executor

The Modelica executor compiles the Modelica code defined by the user into a shared library according to the Function Mock-up Interface (FMI) into a Functional Mock-up Unit (FMU). The main controller task communicates with the simulation executor using shared memory protected by POSIX semaphores.

4.4. Python executor

The application thread of the main controller task is responsible for creating each corresponding subtask to the Python code that should be executed. Each subtask is then supervised from python supervision thread in the main controller.

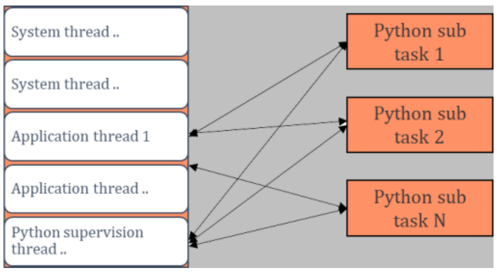


Figure 5: Python sub task creation

In order to support different scenarios of how the python code should be executed, two different modes are introduced, synchronous and asynchronous. In synchronous mode each subtask is created at the same priority level as the calling application thread which waits for an answer or a

user defined timeout before it continues execution. In asynchronous mode each subtask is created on priority level 0 (background) and the calling thread immediately continues execution and checks periodically if an answer has been received. A fixed limit of how many sub tasks that can be created has been set in the system due to system limitations.
The python code is called by an interface function defined in the python script by the user. The function can have an arbitrary name but must be defined with a specific number of arguments that corresponds to the executing block in the ICS. The function consists of a list of four which contains float, integer, Boolean and text values. The last three arguments are prepared to be returned by the python script and contains real, integers and Booleans.
The data exchange between the ICS and python executor is handled by POSIX message queues. Each sample, set by the user in the ICS, the corresponding ICS block checks if the python subtask is Idle. If the Python subtask is idle a message is sent to the message queue with the last values from the ICS. The Python subtask then receives the values, executes the script and returns the result to the ICS. Depending on if the python sub task is created in synchronous or asynchronous mode the corresponding block in the ICS waits for the answer or continues execution immediately.
The errors in the python script are handled by the corresponding block in the ICS. Any syntax error in the python script causes the python sub task to end execution.

4.5. Supervisor

The main functionality of the supervisor is to monitor all other sub tasks in the system. The supervisor has the highest priority in the system and is able to take predefined actions depending on if an error state exists in some of the sub tasks. This could be, for example, a controlled shutdown of all system tasks, restart of a certain sub task or a complete halt of the system. Each task communicates with the supervisor task using a shared memory which contains the current state of the specific task.

5. Test installation in a residential building

Figure 6: Test installation site

The experimental setup is performed at a district heat substation in a residential building in an urban area. The experiment is conducted in the heat season to get reasonable data for the MPC algorithm.
Hardware and Technical Stack: The framework runs on Linux (Debian 10) on a Raspberry Pi4. The detailed system setup and technical specifications to run the experiment are presented in Table 1.

Table: 1.

Parameters	**Values**
OS	ARM based Raspbian
OS Name	Debian 10
OS Version	Linux 5.11 RT-PREEMPT
Processor	Broadcom BCM2711 SoC with a 1.5 GHz 64-bit quad-core ARM Cortex-A72 processor
System RAM	2GB

We implement and evaluate the MPC algorithm using the Scikit-learn2 library, SciPy library and NumPy library for Python3. The data from different sensors are fetched from the local PLC by using Modbus TCP/IP with a time resolution of 50 milliseconds. The prediction of the local outdoor temperature for the next 10 hours is fetched from the Swedish weather service using SMHI API. The MPC algorithm then calculates the setpoint for each hour and writes it back to the local PLC by using Modbus TCP/IP. The data is saved locally with one- and sixty minutes time resolution for further analysis and to be processed by the MPC algorithm. The internal sample time of the ICS is set 10 milliseconds for each application task to register the real-time behavior of the overall system. Timestamps are recorded for each sample interval to be used for further analysis.

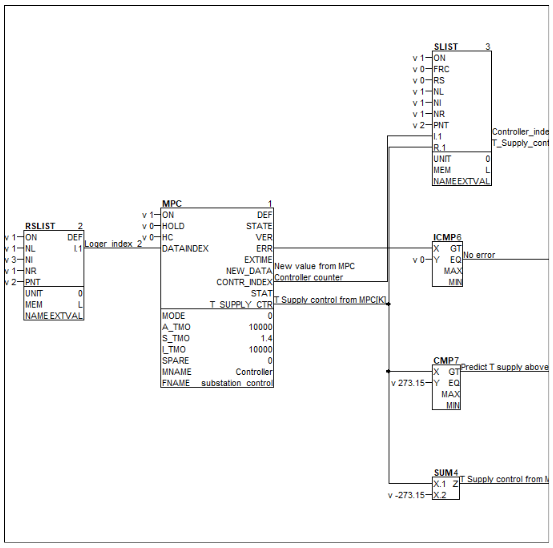


Figure 7: Application program in the ICS

6. Results of the test installation

The data gathering started at the end of January and the controller was turned on February 6th.

The complete system showed stable real-time characteristics during the whole test period regardless of the heavy background load due to the MPC computations. The ICS system functions were performed within the time intervals set by the user, which in this case was a sampling time of 10 milliseconds without any disturbances.

The Python-implemented MPC algorithm was executed as expected on the proposed platform and without interfering with the ICS system's functionality. This includes updates of python software, application changes including adding new applications programs and functions. When comparing the time required for a cycle's calculations, these were performed on a normal PC with a time consumption of approximately 4 minutes and on the industrial control system, the time consumption was approximately 8 minutes. This depends on hardware capabilities and process architecture.

The MPC algorithm performed very well and was stable throughout the whole test period. The purpose was to keep the indoor temperature at a predefined set-point, in this case 21.5°C and minimize the variations. The controlled variable was the supply temperature (space heating) which was set to the local PLC at site. As seen in figure 8 the indoor temperature was reduced from a mean value of 22.5°C to 21.5°C about one day after the controller was switched on. The variations were about +-0.2K.

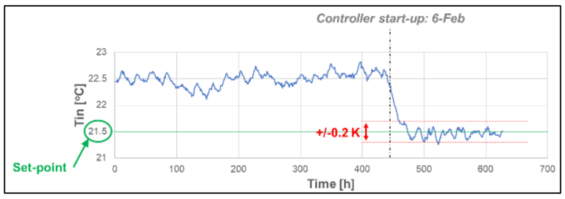


Figure 8: MPC performance

The response in temperature can of course be made much faster, but that would not have been convenient for inhabitants in the building. In this case, the control action is deliberately limited to achieve an "acceptable change in temperature" as was requested by the end user. The limit is only active when large changes are required for instance in a start-up as is shown in figure 8. During normal operation, there are no limits in control action.

7. Summary and Discussions

In this paper we have attempted to form a standard for "Edge computing" i.e. how advanced functions created in research tools like Modelica and Python can be included in automation. Such a standard is necessary for broader use. As far as we know, the attempts that have been done so far consist of special programming in each individual case which is much more costly and requires help from a software specialist. With a standard setup as is described here, the process engineer can use external functions made by researchers in the automation system without the assistance of software experts.

The proposed solution showed that it was feasible to implement a MPC strategy together with an existing industrial control system without interfering with its critical functionality and real-time behavior. The execution time for the MPC algorithm was reasonable and it performed well for the specific process. The Modelica simulator was not used in the experimental setup in this case but may be used for more complicated physical-based models as a local "Digital Twin". We have shown that the capacity is sufficient for such solutions.

In this experiment the dynamics of the process being controlled were slow which meant that the CPU capacity was enough to calculate the setpoint once per hour. Processes with faster dynamics may require other approaches, for example other hardware solutions, such as dedicated hardware, a more simplified model for the MPC or different solvers and optimizers. The Python interface to the ICS could also be improved regarding data conversion, messages, optimization etc.

The main idea about using python as a base for MPC, and other types of control algorithms is that implementations easily can be transferred from construction and design phase to an ICS without any further or very little modifications. Python is also a well-known language which has many libraries available for different purposes and also has a huge amount of support among the developer community. Many other use-cases exist that can take advantages of the introduction of Python in an ICS, for example different ML-implementations which will directly appear as new blocks in the ICS. Using ML in the ICS with data-driven models could be an alternative to the physics-based models implemented in the simulation task. Future research should focus on further validating the framework's performance in different industrial processes and explore the scalability and reliability of the proposed solutions.

Acknowledgment
This work is supported by the District Heat project. Further Thanks to the team at Mimer for their help and support.

References

[1] Schwenzer M, Ay M, Bergs T, Abel D (2021) 'Review on predictive control: an engineering perspective', *The International Journal of Advanced Manufacturing Technology (*2021) 117:1327–1349 .doi:10.1007/s00170-021-07682-3

[2] Krupa P, Limon D,Alamo T (2021) 'Implementation of Model Predictive Control in Programmable Logic Controllers´, *IEEE Transations on control systems technology,* vol. 29, no. 3, May 2021 pp 1117–1130. doi:10.1109/TCST.2020.2992959

[3] de Oliveira DB, de Oliveira RS (2016) 'Timing analysis of the PREEMPT RT Linux kernel'.*Software-practice & experience* , vol.46 issuse 6 pp 789-819. doi:10.1002/spe.2333

[4] Adam, GK, Petrellis, N, Doulos, LT (2021). 'Performance Assessment of Linux Kernels with PREEMPT_RT on ARM-Based Embedded Devices´. *Electronics 2021,* 10, 1331. doi:10.3390/electronics10111331

[5] Carvalho AA, Machado CLD,, Moraes FS (2019). 'Raspberry Pi performance analysis in real-time applications with the RT-Preempt patch´. *2019 Latin american robotics symposium, 2019 Brazilian symposium on robotics (SBR) and 2019 workshop on robotics in education*. pp 162-167 doi:10.3390/electronics10111331

[6] Saletti C, Zimmerman N, Morini M, Kyprianidis K, Gambarotta A (2021). 'Enabling smart control by optimally managing the State of Charge of district heating networks´. *Applied Energy, 283,* p.116286. doi:10.1016/j.apenergy.2020.116286

[7] Saletti C, Zimmerman N, Morini M, Kyprianidis K, Gambarotta A (2022). 'A control-oriented scalable model for demand side management in district heating aggregated communities'. *Applied Thermal Engineering, 201*, p.117681 doi:10.1016/j.applthermaleng.2021.117681

[8] Zimmerman N, Kyprianidis K, Lindberg CF (2019). 'Achieving lower district heating network temperatures using feed-forward MPC.'. *Materials, 12(15*), p.2465. doi:10.3390/ma12152465

[9] Monghasemi N, Vouros S, Kyprianidis K, Vadiee A (2022) 'A non-linear gray-box model of buildings connected to district heating systems'. *Energy Proceedings* ,Vol 29 doi.org:10.46855/energy-proceedings-10497

Additional information
[10] First Control (2023) 'XC05 operation manual for researchers and engineers'