UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

MASTER'S DEGREE FINAL THESIS

Master's degree in Interdisciplinary and Innovative Engineering

DIGITAL TWIN OF A GREENHOUSE FOR SMART FARMING

Report and Annexes

Author: Di Chiara Herrera, Christian
Director: Benítez Iglesias, Raul
Co-Director: Sacile, Roberto

Call: September 2022

Abstract

This document presents a digital twin framework in the smart farming domain. It designs a digital twin based on the autonomous predictive control of a greenhouse. The design of a digital environment to create a cyber-physical-system able to control the greenhouse ambient parameters remotely, and to allow the user to understand better the greenhouse and its environment. The digital twin presented uses model predictive control based on previewing weather forecasts, and services to retrieve and store the data in the cloud.

Resum

Aquest document presenta un marc de bessó digital dins del domini de l'agricultura intel·ligent. Dissenya un bessó digital basat en el control predictiu autònom d'un hivernacle. El disseny d'un entorn digital per crear un sistema ciberfísic capaç de controlar els paràmetres ambientals de l'hivernacle de forma remota i, per permetre a l'usuari entendre millor l'hivernacle i el seu entorn. El bessó digital presentat usa un control de model predictiu basat en l'anticipació de les previsions meteorològiques, i serveis per recuperar i guardar les dades del núvol.

Resumen

Este documento presenta un marco de gemelo digital dentro del dominio de la agricultura inteligente. Diseña un gemelo digital basado en el control predictivo autónomo de un invernadero. El diseño de un entorno digital para crear un sistema ciberfísico capaz de controlar los parámetros ambientales del invernadero de forma remota, y para permitir al usuario entender mejor el invernadero y su entorno. El gemelo digital presentado usa un control de modelo predictivo basado en la anticipación de las previsiones meteorológicas, y servicios para recuperar y guardar los datos de la nube.

Acknowledgements

This work was supported by the Universitat Politècnica de Catalunya (UPC) and the Università di Genova (UniGe). The author greatly acknowledges to both universities for the software and database licenses for free. The author also acknowledges to OpenWeather for the developer plan given for the present project.

Glossary

Initialism	Description
API	Application Programming Interface
CSV	Comma Separated Value
DMPC	Discrete Model Predictive Control
DT	Digital Twin
ECR	Equal Concern for Relaxation
GCP	Google Cloud Platform
GCS	Google Cloud Storage
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IEC	International Electrotechnical Commission
LCA	Life Cycle Assessment
LCIA	Life Cycle Impact Assessment
MD	Measured Disturbance
MIMO	Multi-Input Multi-Output
MO	Measured Output
MPC	Model Predictive Control
MV	Measured Variable
OPC	OLE for Process Control
PV	Photovoltaic
RTU	Remote Terminal Unit
SOA	Service-Oriented Architecture
TCP/IP	Transmission Control Protocol/Internet Protocol
TSDB	Time Series Database
UA	Unified Architecture
UD	Unmeasured Disturbance
UO	Unmeasured Output
URL	Uniform Resource Locator

Index

Figure index

Table index

1. Introduction

Agriculture plays a key role in the development of many countries, furthermore, due to the rapid increase of the global population, the agricultural output should increase by 20% in 2030 (United Nations 2019). In contraposition with manufacturing or distribution sectors, the agricultural one is more unwilling to adapt its infrastructure to new technologies (Pylianidis, Osinga and Athanasiadis 2021). In order to increase the agricultural output, the food security, and to deal with new factors as climate change, fresh water scarcity or an energy consumption reduction, agriculture should pursuit its transition to smart farming.

A digital twin model based on MPC and weather forecasts can be implemented in farms to adequately monitor its current status, optimize the control of the farm and have access to historical data to recognize patterns for future purposes (Verdouw, Tekinerdogan and Wolfert 2021). This report presents the primary development of a digital twin framework using MPC, future weather forecasts and cloud services to control autonomously the ambient temperature within a greenhouse. This report is arranged as following: section 2 defines the mathematical model of the plant, section 3 explains the design of the MPC controller, section 4 illustrates the physical domain requirements for the digital twin installation, section 5 specifies the cyber domain framework of the digital twin, section 6 presents the GUI application and, finally, section 7 outlines the environmental impact analysis of the whole project.

1.1. Objectives

The present document intends to:

- Define the design of a MIMO MPC controller for an actual greenhouse based on weather forecast previewing
- Define the design of a software architecture framework for the cyber domain of the digital twin
- Integrate the design of the MPC controller, the cyber domain and the GUI into an autonomous predictive digital twin

1.2. Scope

The present document covers:

- The design of a simplified thermal model for an actual greenhouse located in Savona (Italy) and its MATLAB implementation
- The design of a MIMO MPC controller for the greenhouse model, including its specifications and simulations in different scenarios
- The specification of the physical domain requirements to interface the digital twin application. Excluding the design and the selection of equipment
- The design of a software architecture framework of the digital twin's cyber domain
- The design and specification of the GUI application for the digital twin
- The analysis of the environmental impact of the digital twin
- The codes of the digital twin, including:
    - MATLAB simplified greenhouse model
    - MATLAB MPC controller
    - MATLAB control files (OPC UA client, initialization, loop and forecast update)
    - Python OPC UA server
    - Python intelligence layer
    - Python GUI
    - Telegraf configuration

1.3. Methodology

The development of a conceptual framework is typically a design-oriented methodology that aims at solving a certain type of problem by constructing a new artifact (Hevner, et al. 2004).

The project was organized in three phases:

1. Model design
2. MPC controller design
3. Digital twin framework design

Firstly, the project started with the analysis of the case of study (Boccalatte, Fossa and Sacile 2021). From the actual greenhouse, a mathematical model has been developed as a base for the future phases, specifying which are the inputs and outputs, and delimiting the capabilities of the future digital twin.

Secondly, using the mathematical model, the MPC controller was designed. The parameter specification and the tunning of the controller is made together with the simulation of the controller's response. The ranges of its inputs and outputs are fully specified, therefore, they are the inputs for the following phase.

Thirdly, the digital twin framework was designed. This phase is made up of three subphases: physical domain, cyber domain and GUI. The first subphase to be developed was the cyber domain. Its design was closely linked with the necessities of the MPC controller (inputs, outputs, data format and MATLAB integration). Once the cyber domain was defined, the interfaces with the physical domain and the GUI were stablished. The physical domain architecture was designed to fit the necessities of the MPC controller (sensors and actuators) and the interface with the cyber domain. Finally, the GUI application was developed to show to the user all the capabilities of the digital twin (cyber domain mirroring).

2. Greenhouse model

In the present chapter, the actual greenhouse in which the digital twin is based, and the mathematical model design shall be presented.

2.1. System description

The considered system is a mini-PV greenhouse prototype that will be located at the Savona Campus of the University of Genova (latitude: 44.31; longitude 8.48). Figure 2.1, presented below, shows the actual greenhouse prototype in a temporary location. Furthermore, the image shown, does not belong to the final version of the greenhouse due to there are modifications still pending (automatic opening/closure of the windows for ventilation and automatic reflecting curtains for the reduction of solar radiation). Equally, these pending modifications will be assumed in the design although there are not implemented yet.

Figure 2.1. Mini-PV greenhouse prototype

The complete description of the present greenhouse is presented in (Boccalatte, Fossa and Sacile 2021). The plan view of the building shows a squared structure of 5 m per side, being a total area of 25 m^2. The asymmetrical roof has a 4.2 m maximum height and 2 m gutter height. As stated before, the roof is asymmetrical, then the slopes have an inclination of 30° and 60°, with respect to the horizontal plane. The roof inclined 30° houses the PV modules and the other part of the roof will be used to house

the windows as well as the awnings (reflecting curtains). Figure 2.2, presented below, shows the drawings with the dimensions of the actual greenhouse.

The materials used to manufacture the greenhouse, as well as their properties and main parameters are gathered in Table 1 below. These parameters have been directly collected from (Boccalatte, Fossa and Sacile 2021).

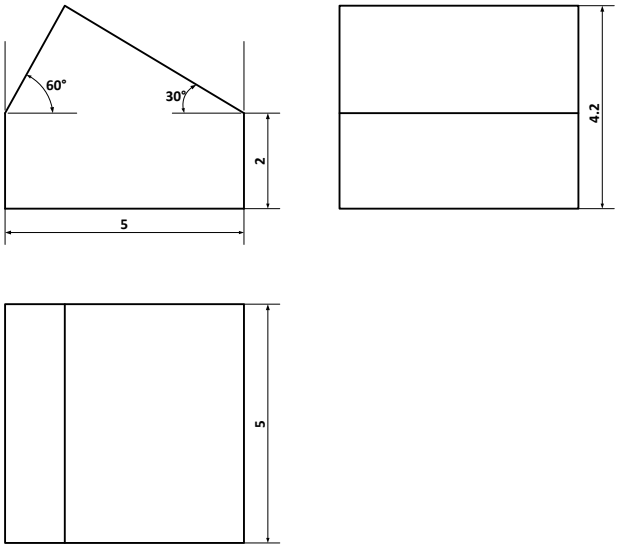


Figure 2.2. Greenhouse drawings (dimensions in meters)

Construction Elements		
Elements	**Physical properties**	**Values**
Glass	Thickness (m)	0.016
	U-factor (W/m^2·K)	1.10
	Solar Heat Gain Coefficient (-)	0.56
	Visible transmittance (-)	0.77
Aluminium frame	Thickness (m)	0.075
	Conductivity (W/m·K)	0.28
	Density (kg/m^3)	348
	Specific Heat (J/kg·K)	900
PV modules	Thickness (m)	0.004
	U-factor (W/m^2·K)	1.10
	Density (kg/m^3)	1069
	Thermal, Solar, Visible Absorptance (-)	0.85

Source: (Boccalatte, Fossa and Sacile 2021)

Table 1. Main parameters of the materials

2.2. Assumptions

The following assumptions are made:

- Aluminium structure is not considered. All the walls and the roof shall be glass or PV modules.
- All compounds and gasses are homogeneous and they have uniform temperature.
- All outdoor weather conditions are not influenced by the greenhouse climate conditions.
- The effect of longwave radiation on the materials is not considered.
- The reflecting curtains are able to reflect ideally the solar radiation.
- Reflecting curtains are only located in the glass roof. They do not actuate on the PV roof.
- The natural ventilation model only takes into account the difference of temperature between the outside and the inside of the greenhouse and the airflow related with the wind component.
- Air leakage due to imperfections of the building are not considered.
- The wind direction is not considered.
- Windows for natural ventilation are in the walls which are orthogonal to the ridge of the roof.
- The influence of the humidity is neglected.
- The influence of the crops within the greenhouse is neglected.
- The heater is also a cooler.
- The heater/cooler is considered ideal.

2.3. Model

The greenhouse model is written in state-space form in Eq 2.1:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{t}, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{d}) \qquad \textbf{Eq 2.1}$$

Where *t* is time, ***x*** are the states, ***u*** are the control inputs, ***d*** are external inputs, and ***f*** is a nonlinear function. The model has been developed by means of the description given by (Boccalatte, Fossa and Sacile 2021) and the mathematical models given by (Van Straten, et al. 2011). As stated previously, due to the purpose of the model, it has been simplified in comparison with the ones given in the references. The states, control inputs and external inputs are gathered in Table 2 and Figure 2.3.

Symbol	**Description**	**Unit**
States, *x*		
$\boldsymbol{T_g}$	Temperature roof (glass)	K

T_{pv}	Temperature roof (PV module)	K
T_r	Temperature indoor air below roof	K
T_a	Temperature indoor air	K
T_s	Temperature soil	K
Control inputs, *u*		
Q_{hc}	Heater/Cooler	W
$Cl_{1,\%}$	Window aperture lee-side	[0,1]
$Cl_{2,\%}$	Window aperture windward-side	[0,1]
τ_{AW}	Awning's extension	[0,1]
External Inputs, *d*		
T_o	Temperature outdoor air	K
I_o	Outdoor shortwave solar radiation	W/m^2
v_o	Outdoor wind speed	m/s

Table 2. States, control inputs and external inputs

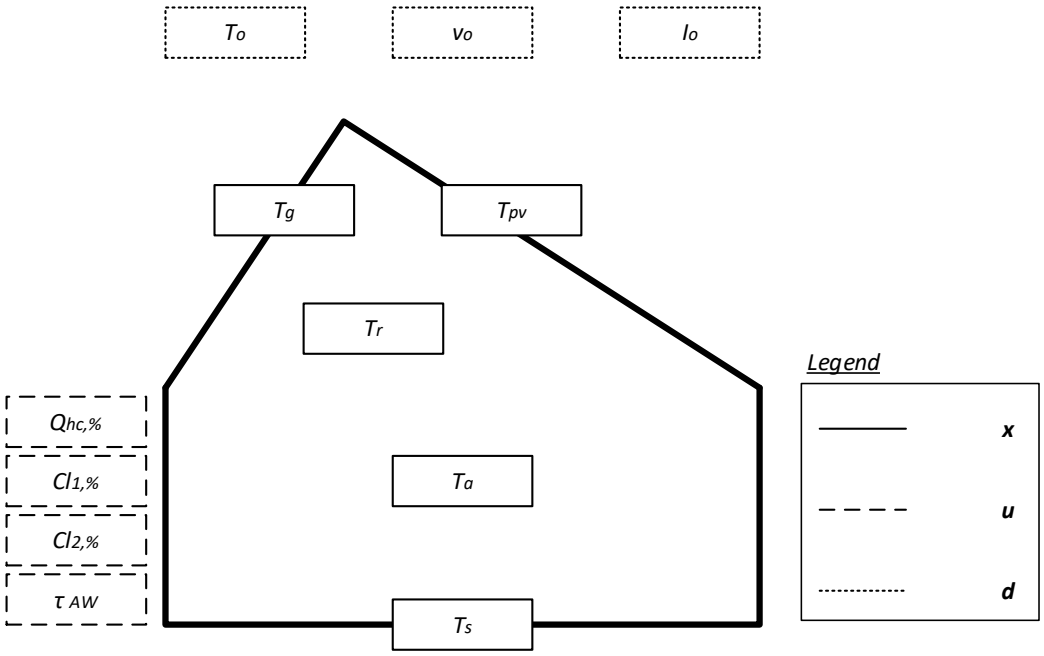


Figure 2.3. States *x* (solid box), control inputs *u* (dashed box) and external inputs *d* (dotted box) in the model

The state equations are:

Temperature indoor air [K/s] (Eq 2.2)

$$\frac{dT_a}{dt} = \frac{-Q_{ar} - Q_{as} + Q_{hc}}{\rho_a c_{p,a} V_a} \quad \textbf{Eq 2.2}$$

Temperature soil [K/s] (Eq 2.3)

$$\frac{dT_s}{dt} = \frac{Q_{as}}{\rho_s c_{p,s} V_s} \quad \textbf{Eq 2.3}$$

Temperature indoor air below the roof [K/s] (Eq 2.4)

$$\frac{dT_r}{dt} = \frac{Q_{ar} - Q_{ro} - Q_{rg} - Q_{rPV}}{\rho_r c_{p,r} V_r} \quad \textbf{Eq 2.4}$$

Temperature indoor side of the roof (glass) [K/s] (Eq 2.5)

$$\frac{dT_g}{dt} = \frac{Q_{radg} + Q_{rg} - Q_{go}}{\rho_g c_{p,g} V_g} \quad \textbf{Eq 2.5}$$

Temperature indoor side of the roof (PV module) [K/s] (Eq 2.6)

$$\frac{dT_{PV}}{dt} = \frac{Q_{radPV} + Q_{rPV} - Q_{PVo}}{\rho_{PV} c_{p,PV} V_{PV}} \quad \textbf{Eq 2.6}$$

The parameters of the main materials used in the present model have been gathered in Table 3 below.

Material	**Density**	**Spec. Heat Capacity**	**Area**	**Volume**	**Heat Capacity**
	ρ (kg/m^3)	**c_p (J/kg·K)**	**A (m^2)**	**V (m^3)**	**$c_p \cdot \rho \cdot V$ 10^6 (J/K)**
Air (ambient)	ρ_a = 1.29	$c_{p,a}$ = 1000	A_a = 25	V_a = 50	0.0645
Air (roof)	ρ_r = 1.29	$c_{p,r}$ = 1000	-	V_r = 27.94	0.036
Roof (glass)	ρ_g = 2500	$c_{p,g}$ = 840	A_g = 12.7	V_g = 0.203	0.4263
PV module	ρ_{PV} = 1069	$c_{p,PV}$ = 800	A_{PV} = 22	V_{PV} = 0.088	0.1126
Soil	ρ_s = 1600	$c_{p,s}$ = 800	A_s = 25	V_s = 12.5	16
Source: (Van Straten, et al. 2011) & (Boccalatte, Fossa and Sacile 2021)					

Table 3. Parameter values

2.3.1. Thermal model

The Figure 2.4 shown below, describes graphically the thermal model used for the greenhouse system model. The solid boxes represent the states, the dotted boxes the external inputs, and the lines between boxes the heat transfer between states by means of convection, shortwave radiation absorption or conduction. The presented thermal model is a simplified version of the actual one due to the limitations of processing real-time data in more complex models.

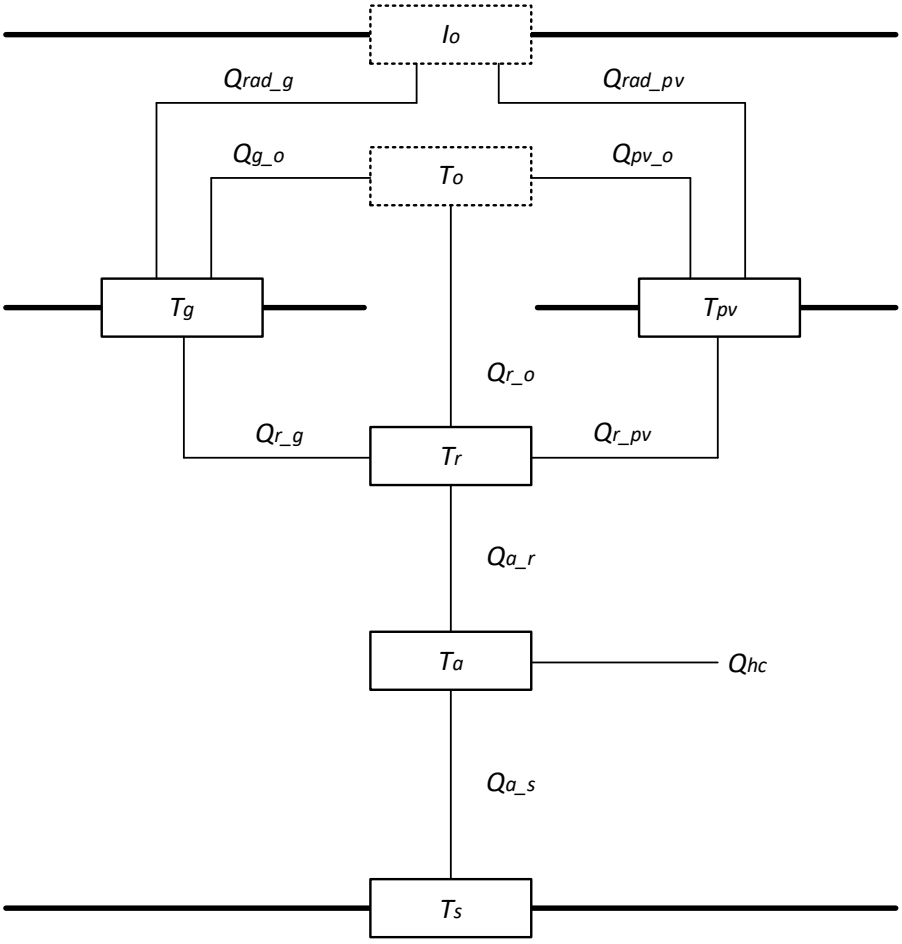


Figure 2.4. States *x* (solid box) and external inputs *d* (dotted box) in the thermal model

2.3.1.1. Convection

Convection is the heat transfer between a solid and a fluid. The heat transfer Q_{A_B} from A to B is described by the Newton's law of cooling, presented in Eq 2.7.

$$\boldsymbol{Q_{A\_B} = A_{A\_B}\alpha_{A\_B}(T_A - T_B)} \qquad \textbf{Eq 2.7}$$

Where A_{A_B} (m^2) is the surface area during the heat transfer, α_{A_B} (W/m^2K) is the heat transfer coefficient between A and B (U-factor), and T_A and T_B (K) are the temperatures of A and B respectively. The heat transfer due to convection is given in Table 4 for all the surfaces.

Q_{A_B}	T_A	T_B	A_{A_B}	α_{A_B}	**Source**
Q_{r_g}	T_r	T_g	A_g	α_{r_g} = 1.10	BFS*
Q_{r_pv}	T_r	T_{pv}	A_{pv}	α_{r_g} = 1.10	BFS*
Q_{a_s}	T_a	T_s	A_s	α_{a_s} = 1.00	BFS*
*Source: BFS** (Boccalatte, Fossa and Sacile 2021)					

Table 4. Convection values

The heat exchange between the ambient indoor temperature and the air below the roof is defined by Eq 2.8.

$$\boldsymbol{Q_{a\_r} = \overline{\rho_a} c_{p,a} F_{a\_r} (T_a - T_r)} \qquad \textbf{Eq 2.8}$$

Where ρ_a (kg/m^3) is the average density of air, $c_{p,a}$ (J/kg·K) is the specific heat capacity of air, F_{a_r} (m^3/s) is the flow of air from below to above, and T_a and T_r the indoor temperatures of the lower and upper part of the greenhouse respectively.

The heat exchange between the indoor temperature and the outdoor temperature by means of the ventilation is defined by Eq 2.9.

$$\boldsymbol{Q_{r\_o} = \overline{\rho_a} c_{p,a} F_{win} (T_r - T_o)} \qquad \textbf{Eq 2.9}$$

Where ρ_a (kg/m^3) is the average density of air, $c_{p,a}$ (J/kg·K) is the specific heat capacity of air, F_{win} (m^3/s) is the flow of air going through the greenhouse, and T_r and T_o the indoor temperature below the roof and the outdoor temperature respectively.

The airflow variables, F_{win} and F_{a_r}, are described in Ventilation model.

2.3.1.2. Shortwave radiation absorption

The shortwave radiation is the radiation received directly or indirectly from the sun integrated over all wavelengths in the shortwave interval. In this case, the heat Q_{rad_A} absorbed by object A is defined by Eq 2.10.

$$\boldsymbol{Q_{rad\_A} = A_A \eta_{A\_Is} I_o} \qquad \textbf{Eq 2.10}$$

Where A_A (m^2) is the surface area for the heat transfer, η_{A_Is} (-) is the shortwave radiation absorption coefficient for A, and *Io* (W/m^2) is the outdoor shortwave radiation. The heat transfer due to shortwave radiation is given in Table 5 for all the surfaces.

Q_{rad_A}	A_A	η_{A_Is}	**Source**
Q_{rad_g}	A_g	$\eta_{g\_Is} = 0.56 \cdot (1-\tau_{AW})$	BFS*
Q_{rad_pv}	A_{pv}	$\eta_{pv\_Is} = 0.85$	BFS*
*Source: BFS** (Boccalatte, Fossa and Sacile 2021)			

Table 5. Shortwave radiation values

Where τ_{AW} (-) is the control input to extend the reflecting curtains. Reefer Reflecting curtains model for more information regarding the correction coefficient and the screen model.

2.3.1.3. Longwave radiation absorption

Longwave radiation absorption is the heat transfer due to radiation between two materials. It is not considered in the present model, see Assumptions.

2.3.1.4. Conduction

Conduction is the heat transfer Q_{A_B} between the locations A and B in a homogeneous medium defined by Eq 2.11.

$$\boldsymbol{Q_{A\_B} = A_{A\_B} \frac{\lambda}{d} (T_A - T_B)} \qquad \textbf{Eq 2.11}$$

Where A_{A_B} (m^2) is the surface area for the heat transfer, λ (W/m·K) is the thermal conductivity of the homogenous medium, d is the thickness of the material (m), and T_A and T_B are the temperatures of A and B. The heat transfer due to conduction is given in Table 6 for all the surfaces.

Q_{A_B}	A_{A_B}	λ	d	T_A	T_B	**Source**
Q_{g_o}	A_g	0.9	0.016	T_g	T_o	BFS* & Y*
Q_{pv_o}	A_{pv}	148	0.004	T_{pv}	T_o	BFS* & Y*
*Source: for d values BFS** (Boccalatte, Fossa and Sacile 2021)*; for λ values Y** (Young and Sears 1992)						

Table 6. Conduction values

2.3.2. Ventilation model

The ventilation model is presented in Figure 2.5. Two airflows are considered in the model: an indoor airflow produced by the temperature difference and an external airflow that enters into the greenhouse through the windows.

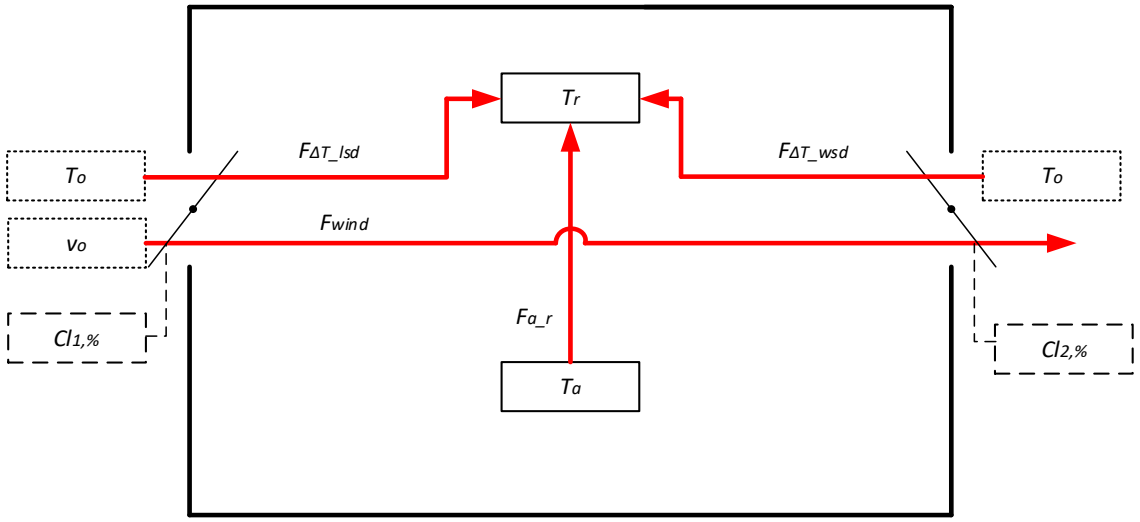


Figure 2.5. Ventilation model; states (solid boxes), external inputs (dotted box), control inputs (dashed boxes) and airflows (red arrows)

The parameters related with the windows (height, width and number) are presented in Table 7.

Type of window	h_{win} **(m)**	w_{win} **(m)**	n_{win} **(-)**
Window A	0.5	0.5	2

Table 7. Characteristics of the windows

2.3.2.1. Airflow through windows

By means of the opening of windows, the ventilation process takes place in the greenhouse. The following equations have been collected from the thesis (De Jong 1990). In order to simplify the ventilation model, only the wind-induced component and the component related to the difference of temperatures between the outdoor and the indoor have been considered. Therefore, the air-leakage will not be considered.

The airflow F_{win} through the windows is described by Eq 2.12.

$$F_{win} = \sqrt{F_{wind}{}^2 + \left(F_{\Delta T\_lsd} + F_{\Delta T\_wsd}\right)^2} \qquad \textbf{Eq 2.12}$$

Where F_{wind} (m^3/s) is the wind-induced component, and $F_{\Delta T\_lsd}$ and $F_{\Delta T\_wsd}$ (m^3/s) are the components determined by the temperature difference between indoor and outdoor for the lee- and windward-side respectively.

The wind-induced component F_{wind} is described by Eq 2.13.

$$\boldsymbol{F_{wind} = n_{win}(f_{lsd} + f_{wsd})v_o w_{win} h_{win}} \quad \textbf{Eq 2.13}$$

Where f_{lsd} and f_{wsd} (-) are the ventilation functions for the lee-side and the windward-side respectively, w_{win} and h_{win} (m) are the width and the height of the windows, n_{win} (-) is the number of windows and v_o (m/s) is the wind speed.

The ventilation functions f_{lsd} and f_{wsd}, extracted from (De Jong 1990), are defined by Eq 2.14 (lee-side) and Eq 2.15 (windward-side).

$$\boldsymbol{f_{lsd} = c_{lsd3} Cl_{1,\%}{}^3 - c_{lsd2} Cl_{1,\%}{}^2 + c_{lsd1} Cl_{1,\%} + c_{lsd0}} \quad \textbf{Eq 2.14}$$

$$\boldsymbol{f_{wsd} = c_{wsd3} Cl_{2,\%}{}^3 - c_{wsd2} Cl_{2,\%}{}^2 + c_{wsd1} Cl_{2,\%} + c_{wsd0}} \quad \textbf{Eq 2.15}$$

Where $Cl_{1,\%} \in [0,1]$ and $Cl_{2,\%} \in [0,1]$ are the window aperture of both sides (control inputs). The ventilation coefficients c_{wsdx} and c_{lsdx} (-) have been gathered in Table 8.

Wind-side	**c_3**	**c_2**	**c_1**	**c_0**
Lee	$1.21774 \cdot 10^{-2}$	$3.77646 \cdot 10^{-2}$	$4.71176 \cdot 10^{-2}$	$1.26730 \cdot 10^{-4}$
Windward	$2.42856 \cdot 10^{-2}$	$4.48621 \cdot 10^{-2}$	$7.08828 \cdot 10^{-2}$	$5.91362 \cdot 10^{-4}$
Source: (De Jong 1990)				

Table 8. Ventilation coefficients for lee- and windward side

The airflow components $F_{\Delta T\_lsd}$ and $F_{\Delta T\_wsd}$ are defined by Eq 2.16 (lee-side) and Eq 2.17 (windward-side).

$$\boldsymbol{F_{\Delta T\_lsd} = n_{win} \frac{c_w}{3} w_{win} \sqrt{g \frac{|T_o - T_r|}{T_o}} L_{win\_lsd}{}^{1.5}} \quad \textbf{Eq 2.16}$$

$$\boldsymbol{F_{\Delta T\_wsd} = n_{win} \frac{c_w}{3} w_{win} \sqrt{g \frac{|T_o - T_r|}{T_o}} L_{win\_wsd}{}^{1.5}} \quad \textbf{Eq 2.17}$$

Where w_{win} (m) is the window width, c_w = 0.6 (-) is the discharge coefficient through the windows, g = 9.81 (m/s^2) is the gravity, L_{win_lsd} and L_{win_wsd} (m) are the lengths of the vertical projection of the windows opening, and T_o and T_r (K) are the temperatures of the air outside the greenhouse and below the roof.

2.3.2.2. Indoor airflow

The volume flow F_{a_r} exchange, from the lower sector of the greenhouse to the upper part (under the roof), is produced due to the difference of temperature between sectors. The expression is given by Eq 2.18.

$$\boldsymbol{F_{a\_r} = v_{a\_r} A_a} \qquad \textbf{Eq 2.18}$$

Where A_a (m^2) is the surface area of the heat exchange and v_{a_r} (m/s) is the air exchange rate (Van Straten, et al. 2011) given by Eq 2.19.

$$\boldsymbol{v_{a\_r} = 0.05\ \frac{m}{s}} \qquad \textbf{Eq 2.19}$$

2.3.3. Reflecting curtains model

In order to decrease the shortwave radiation influence into the indoor area, reflecting curtains shall be provided on the glass-side of the roof. The shortwave radiation absorption coefficient of the roof is defined by Eq 2.20.

$$\boldsymbol{\eta_{g\_Is} = 0.56(1 - \tau_{AW})} \qquad \textbf{Eq 2.20}$$

Where $\tau_{AW} \in [0,1]$ is the control input for the curtain's extension. The greenhouse effect produced by the addition of the reflecting curtains (longwave radiation absorption) is not considered.

2.3.4. Heating/cooling system model

The heating/cooling system modelled for the greenhouse is a simplification of the actual model. It only interacts with the indoor temperature state T_a, being insulated from other heat fluxes. In Figure 2.6 below, the heater/cooling system is represented.

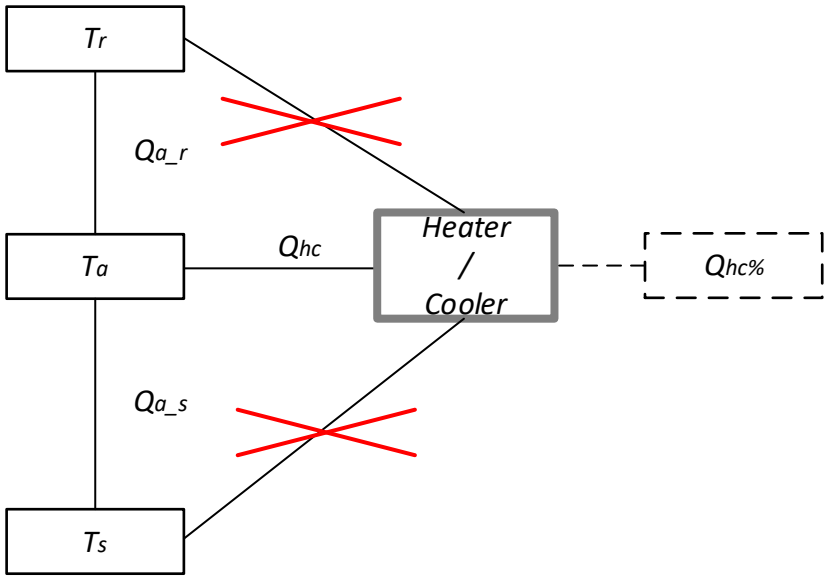


Figure 2.6. Heating/Cooling model; states (solid box), control inputs (dashed box)

2.4. MATLAB implementation

The model has been implemented using state-space representation in MATLAB. The matrices ***A***, ***B***, ***C*** and ***D*** which define our model are presented below:

$$\boldsymbol{A} = \begin{bmatrix} -0.02539 & 0.0003876 & 0.025 & 0 & 0 \\ 1.563 \cdot 10^{-6} & -1.563 \cdot 10^{-6} & 0 & 0 & 0 \\ 0.04474 & 0 & -0.07641 & 0.0003876 & 0.0006714 \\ 0 & 0 & 3.277 \cdot 10^{-5} & -0.001709 & 0 \\ 0 & 0 & 0.0003216 & 0 & -10.82 \end{bmatrix}$$

$$\boldsymbol{B} = \begin{bmatrix} 1.55 \cdot 10^{-5} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.0004017 & 0.002672 & 0.01563 & 0.0007178 & 0 \\ 0 & 0.003337 & 0 & 0 & 0.001676 & 0 & -1.668 \cdot 10^{-5} \\ 0 & 0 & 0 & 0 & 10.82 & 0 & 0.0002485 \end{bmatrix}$$

$$\boldsymbol{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{D} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The full formulation of the state-space, taking into account the states, the control inputs and the external inputs is presented below:

$$\begin{bmatrix} \dot{T}_a(t) \\ \dot{T}_s(t) \\ \dot{T}_r(t) \\ \dot{T}_g(t) \\ \dot{T}_{pv}(t) \end{bmatrix} = \boldsymbol{A} \cdot \begin{bmatrix} T_a(t) \\ T_s(t) \\ T_r(t) \\ T_g(t) \\ T_{pv}(t) \end{bmatrix} + \boldsymbol{B} \cdot \begin{bmatrix} Q_{hc\%}(t) \\ \tau_{AW}(t) \\ Cl_{1\%}(t) \\ Cl_{2\%}(t) \\ T_o(t) \\ I_o(t) \\ v_o(t) \end{bmatrix}$$

$$\boldsymbol{y(t)} = \boldsymbol{C} \cdot \begin{bmatrix} T_a(t) \\ T_s(t) \\ T_r(t) \\ T_g(t) \\ T_{pv}(t) \end{bmatrix}$$

In Figure 2.7 the plant response is presented against constant external inputs (T_o = 25 °C, I_o = 250 W/m² and v_o = 2 m/s)

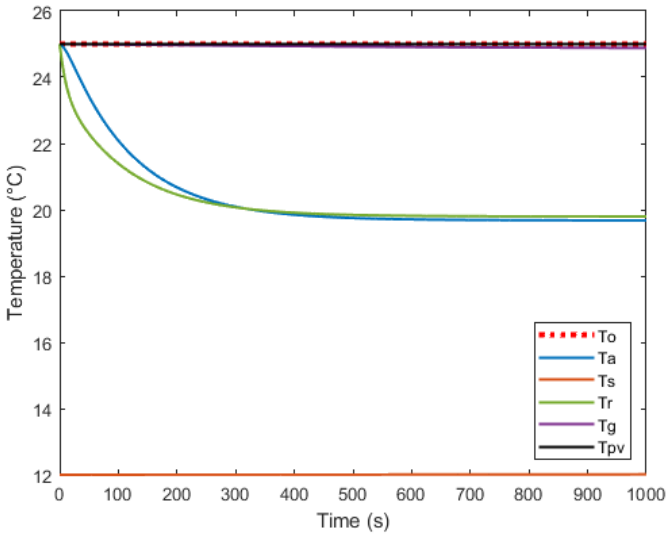


Figure 2.7. Plant response against constant *external inputs*

3. Model Predictive Control

In the present section, all the contents related with the model predictive control shall be presented. Including the design specifications and its MATLAB implementation.

3.1. Introduction to MPC

In this chapter, the controller based on Model Predictive Control (MPC) shall be introduced, developed and tunned. In addition, the results obtained shall be presented in section MPC simulation. Some important terms: moving horizon window, prediction horizon, receding horizon control and control objective are introduced below (Wang 2009).

1. **Moving horizon window:** it is the time-dependent window beginning from an arbitrary time t_i to $t_i + T_p$. The length of T_p remains constant.
2. **Prediction horizon:** it is the parameter which dictates how "far" the future shall be predicted. The prediction horizon equals the length of T_p (moving horizon window).
3. **Receding horizon control:** although the optimal trajectory of future control signal is completely described within the moving horizon window, the actual control input to the plant only takes the firsts *Nc* samples of the control signal, while neglecting the rest of the trajectory.
4. **Control objective:** in the present case, the objective is linked to an error function based on the difference between the desired and the actual response of the system. The optimal control action (control objective) is found by minimizing the cost function *J* within the optimization window.

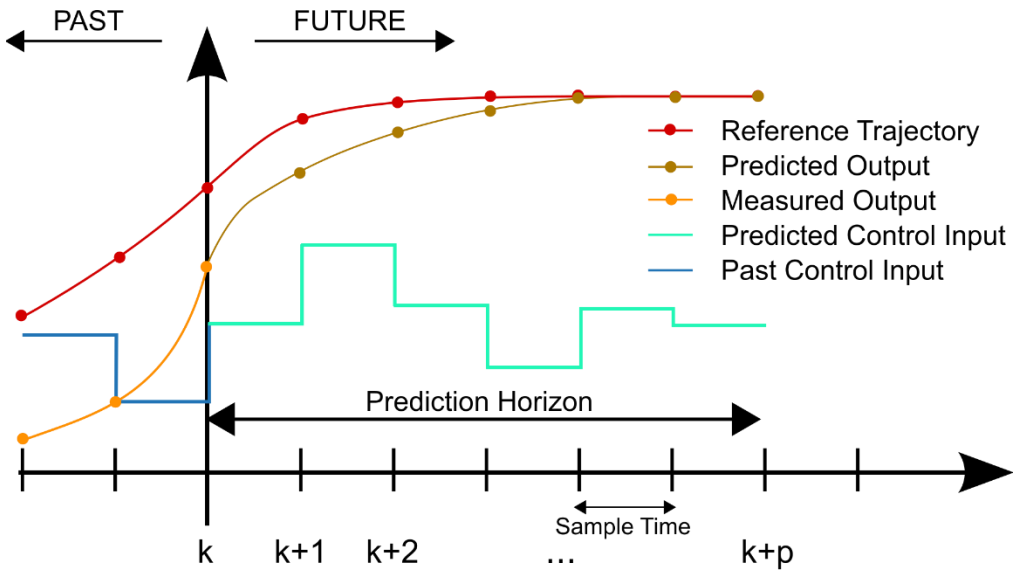


Figure 3.1. DMPC (Discrete Model Predictive Control) scheme

3.2. Predictive Control of the MIMO system

The design of MPC systems is based on the state-space model of the plant, see 2.4. MATLAB implementation.

3.2.1. Formulation of the model

The formulation of the MPC model has been developed according to (Wang 2009). The current plant, has *m* inputs (*m* = 7), *q* outputs (*q* = 5, but only one is measured) and n_1 states (n_1 = 5). The number of outputs shall be less or equal to the number of inputs in order to control each of the measured outputs independently without steady-state errors. Without taking into account the plant noise and the disturbances (they shall be neglected), the general formulation of the MIMO DMPC is described by Eq 3.1 and Eq 3.2.

$$\mathbf{x_m}(\boldsymbol{k}+\mathbf{1}) = \boldsymbol{A_m x_m}(\boldsymbol{k}) + \boldsymbol{B_m u}(\boldsymbol{k}) \qquad \textbf{Eq 3.1}$$

$$\boldsymbol{y}(\boldsymbol{k}) = \boldsymbol{C_m x_m}(\boldsymbol{k}) \qquad \textbf{Eq 3.2}$$

By defining $\Delta x_m(k) = x_m(k) - x_m(k-1)$ and $\Delta u(k) = u(k) - u(k-1)$, we obtain Eq 3.3 and Eq 3.4.

$$\mathbf{\Delta x_m}(\mathbf{k}+\mathbf{1}) = \boldsymbol{A_m}\mathbf{\Delta x_m}(\mathbf{k}) + \mathbf{B_m}\mathbf{\Delta u}(\mathbf{k}) \qquad \textbf{Eq 3.3}$$

$$\boldsymbol{\Delta y}(\boldsymbol{k}+\mathbf{1}) = \boldsymbol{C_m \Delta x_m}(\boldsymbol{k}+\mathbf{1}) \qquad \textbf{Eq 3.4}$$

Choosing a new state-space vector $x(k) = [\Delta x_m(k)^T \; y(k)^T]^T$, we obtain the following augmented state-space vector, described in Eq 3.5, which relates the states vector with the outputs vector

$$\begin{bmatrix} \boldsymbol{\Delta x_m}(\boldsymbol{k}+\mathbf{1}) \\ \boldsymbol{y}(\boldsymbol{k}+\mathbf{1}) \end{bmatrix} = \begin{bmatrix} \boldsymbol{A_m} & \boldsymbol{o_m^T} \\ \boldsymbol{C_m A_m} & \boldsymbol{I_{qxq}} \end{bmatrix} \begin{bmatrix} \boldsymbol{\Delta x_m}(\boldsymbol{k}) \\ \boldsymbol{y}(\boldsymbol{k}) \end{bmatrix} + \begin{bmatrix} \boldsymbol{B_m} \\ \boldsymbol{C_m B_m} \end{bmatrix} \boldsymbol{\Delta u}(\boldsymbol{k})$$

$$\boldsymbol{y}(\boldsymbol{k}) = \begin{bmatrix} \boldsymbol{o_m} & \boldsymbol{I_{qxq}} \end{bmatrix} \begin{bmatrix} \boldsymbol{\Delta x_m}(\boldsymbol{k}) \\ \boldsymbol{y}(\boldsymbol{k}) \end{bmatrix} \qquad \textbf{Eq 3.5}$$

Where I_{qxq} is the identity matrix with dimensions *q x q*; and om is a *q x* n_1 zero matrix. A_m, B_m and C_m have dimension n_1 *x* n_1, n_1 *x m* and *q x* n_1, respectively. In order to simplify the notation, we denote the state space model as defined in Eq 3.6.

$$\boldsymbol{x}(\boldsymbol{k}+\mathbf{1}) = \boldsymbol{A x}(\boldsymbol{k}) + \boldsymbol{B \Delta u}(\boldsymbol{k})$$

$$\boldsymbol{y(k) = Cx(k)} \qquad \textbf{Eq 3.6}$$

3.2.2. Prediction of state and output variables

After the formulation of the mathematical model, the next step is to calculate the predicted plant output with the future control signal as the adjustable variable. The current time is k_i, the length of the optimization window is N_p and the control horizon is N_c.

The future control trajectory is denoted by

$$\boldsymbol{\Delta u(k_i), \Delta u(k_i + 1), \dots, \Delta u(k_i + N_c - 1)}$$

The future state variables are denoted by

$$\boldsymbol{x(k_i + 1|k_i), x(k_i + 2|k_i), \dots, x\big(k_i + N_p\big|k_i\big)}$$

The future state variables shall be calculated sequentially using the set of future control parameters:

$$\begin{aligned}
\boldsymbol{x(k_i + 1|k_i)} &= \boldsymbol{Ax(k_i) + B\Delta u(k_i)} \qquad \textbf{Eq 3.7}\\
\boldsymbol{x(k_i + 2|k_i)} &= \boldsymbol{Ax(k_i + 1|k_i) + B\Delta u(k_i + 1)}\\
\boldsymbol{x(k_i + 2|k_i)} &= \boldsymbol{A^2 x(k_i) + AB\Delta u(k_i) + B\Delta u(k_i + 1)}\\
&\vdots\\
\mathbf{x}\big(\boldsymbol{k_i + N_p|k_i}\big) &= \boldsymbol{A^{N_p}}\mathbf{x}(\boldsymbol{k_i}) + \boldsymbol{A^{N_p-1}B\Delta u(k_i) + A^{N_p-2}B\Delta u(k_i + 1)} + \cdots\\
&\quad + \boldsymbol{A^{N_p-N_c}B\Delta u(k_i + N_c - 1)}
\end{aligned}$$

By means of the predicted state variables, we obtain by substitution the predicted outputs:

$$\begin{aligned}
\boldsymbol{y(k_i + 1|k_i)} &= \boldsymbol{CAx(k_i) + CB\Delta u(k_i)} \qquad \textbf{Eq 3.8}\\
\boldsymbol{y(k_i + 2|k_i)} &= \boldsymbol{CA^2 x(k_i) + CAB\Delta u(k_i) + CB\Delta u(k_i + 1)}\\
\boldsymbol{y(k_i + 3|l_i)} &= \boldsymbol{CA^3 x(k_i) + CA^2 B\Delta u(k_i) + CAB\Delta u(k_i + 1) + CB\Delta u(k_i + 2)}\\
&\vdots\\
\boldsymbol{y\big(k_i + N_p\big|k_i\big)} &= \boldsymbol{CA^{N_p} x(k_i) + CA^{N_p-1}B\Delta u(k_i) + CA^{N_p-2}B\Delta u(k_i + 1)} + \cdots\\
&\quad + \boldsymbol{CA^{N_p-N_c}B\Delta u(k_i + N_c - 1)}
\end{aligned}$$

The next step is to define the vectors Y and ΔU as shown in

$$\boldsymbol{\Delta U = [\Delta u(k_i)^T \Delta u(k_i + 1)^T \dots \Delta u(k_i + N_c - 1)^T]^T}$$

$$\boldsymbol{Y = \Big[y(k_i + 1|k_i)^T y(k_i + 2|k_i)^T \dots y\big(k_i + N_p\big|k_i\big)^T\Big]^T}$$

The vectors *ΔU* and *Y* shall be gathered in a matrix compact form described in Eq 3.9.

$$\boldsymbol{Y = Fx(k_i) + \Phi \Delta U} \qquad \textbf{Eq 3.9}$$

Where

$$\mathbf{F} = \begin{bmatrix} \boldsymbol{CA} \\ \boldsymbol{CA^2} \\ \mathbf{CA^3} \\ \vdots \\ \boldsymbol{CA^{N_p}} \end{bmatrix}; \boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{CB} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \boldsymbol{CAB} & \boldsymbol{CB} & \mathbf{0} & \dots & \mathbf{0} \\ \boldsymbol{CA^2B} & \boldsymbol{CAB} & \boldsymbol{CB} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{CA^{N_p-1}B} & \boldsymbol{CA^{N_p-2}B} & \boldsymbol{CA^{N_p-3}B} & \dots & \boldsymbol{CA^{N_p-N_c}B} \end{bmatrix}$$

3.2.3. Optimization

The objective of the predictive control system is to approximate the predicted output as much as possible to the set-point (SP) signal $r(k_i)$ at simple time k_i. Therefore, the controller has to find the vector *ΔU* which minimizes the error between the set-point signal and the predicted output.

The cost function *J* which reflects the control objective is described in Eq 3.10.

$$\boldsymbol{J = (R_s - Y)^T (R_s - Y) + \Delta U^T \bar{R} \Delta U} \qquad \textbf{Eq 3.10}$$

Where R_s is the data vector that contains all the set-point information and $\bar{R}$ is a diagonal matrix used to tune the closed-loop performance:

$$\boldsymbol{\bar{R} = r_w I_{N_c x N_c}}$$
$$\boldsymbol{r_w \geq 0}$$

The incremental optimal control within one optimization window is given by the optimal solution for the control signal:

$$\boldsymbol{\Delta U = (\Phi^T \Phi + \bar{R})^{-1} \left(\Phi^T \overline{R_s} r(k_i) - \Phi^T F x(k_i) \right)} \qquad \textbf{Eq 3.11}$$

3.2.4. Formulation of constraints

The predictive control problem needs to be handled as an optimization problem that takes into account the constraints of the design. In our case, these constraints are:

- Constraints on the incremental variation of the control variable: Δu^{max}, Δu^{min}
- Constraints on the amplitude of the control variable: u^{max}, u^{min}

Unconstrained output variable is assumed. The constraints shall be specified for each input independently.

The upper limits for the incremental variation shall be denoted as

$$[\boldsymbol{\Delta u_1^{max}} \boldsymbol{\Delta u_2^{max}} \dots \boldsymbol{\Delta u_m^{max}}]$$

The lower limits for the incremental variation shall be denoted as

$$[\boldsymbol{\Delta u_1^{min}} \boldsymbol{\Delta u_2^{min}} \dots \boldsymbol{\Delta u_m^{min}}]$$

The variables with rate of change are specified as

$$\begin{gathered} \left[\boldsymbol{\Delta u_1^{min}} \leq \boldsymbol{\Delta u_1(k)} \leq \boldsymbol{\Delta u_1^{max}}\right] \\ \left[\boldsymbol{\Delta u_2^{min}} \leq \boldsymbol{\Delta u_2(k)} \leq \boldsymbol{\Delta u_2^{max}}\right] \\ \vdots \\ \left[\boldsymbol{\Delta u_m^{min}} \leq \boldsymbol{\Delta u_m(k)} \leq \boldsymbol{\Delta u_m^{max}}\right] \end{gathered} \qquad \textbf{Eq 3.12}$$

The upper limits for the control signal shall be denoted as

$$[\boldsymbol{u_1^{max}} \boldsymbol{u_2^{max}} \dots \boldsymbol{u_m^{max}}]$$

The lower limits for the control signal shall be denoted as

$$[\boldsymbol{u_1^{min}} \boldsymbol{u_2^{min}} \dots \boldsymbol{u_m^{min}}]$$

Therefore, the amplitude of the control signal must satisfy the following constraints

$$\begin{gathered} \left[\boldsymbol{u_1^{min}} \leq \boldsymbol{u_1(k)} \leq \boldsymbol{u_1^{max}}\right] \\ \left[\boldsymbol{u_2^{min}} \leq \boldsymbol{u_2(k)} \leq \boldsymbol{u_2^{max}}\right] \\ \vdots \\ \left[\boldsymbol{u_m^{min}} \leq \boldsymbol{u_m(k)} \leq \boldsymbol{u_m^{max}}\right] \end{gathered} \qquad \textbf{Eq 3.13}$$

Having formulated the constraints, the following procedure is to convert them into linear inequalities by means of parametrizing the constrained variables using *ΔU*.

The constraints are imposed for all future sampling instants, and all constraints shall be expressed in as a function of *ΔU*.

$$\begin{bmatrix} \boldsymbol{u(k_i)} \\ \boldsymbol{u(k_i+1)} \\ \boldsymbol{u(k_i+2)} \\ \vdots \\ \boldsymbol{u(k_i+N_c-1)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{I} \\ \boldsymbol{I} \\ \vdots \\ \boldsymbol{I} \end{bmatrix} \mathbf{u}(\boldsymbol{k_i-1}) + \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \dots & \boldsymbol{0} \\ \boldsymbol{I} & \boldsymbol{I} & \boldsymbol{0} & \dots & \boldsymbol{0} \\ \boldsymbol{I} & \boldsymbol{I} & \boldsymbol{I} & \dots & \boldsymbol{0} \\ \vdots & \vdots & \vdots & \ddots & \boldsymbol{0} \\ \boldsymbol{I} & \boldsymbol{I} & \dots & \boldsymbol{I} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{\Delta u(k_i)} \\ \boldsymbol{\Delta u(k_i+1)} \\ \boldsymbol{\Delta u(k_i+2)} \\ \vdots \\ \boldsymbol{\Delta\, u(k_i+N_c-1)} \end{bmatrix} \qquad \textbf{Eq 3.14}$$

Re-writing the expression of above into a compact matrix form, where C_1 and C_2 correspond to the appropriate matrices, the constraints for the control movement are described in Eq 3.15 and in Eq 3.16 respectively.

$$-(\boldsymbol{C_1 u(k_i - 1) + C_2 \Delta U}) \leq -\boldsymbol{U^{min}} \quad \textbf{Eq 3.15}$$

$$(\boldsymbol{C_1 u(k_i - 1) + C_2 \Delta U}) \leq \boldsymbol{U^{max}} \quad \textbf{Eq 3.16}$$

Where U^{min} and U^{max} are column vectors with N_c elements of u^{min} and u^{max}. For the incremental variation of the signals, we have the constraints defined in Eq 3.17 and in Eq 3.18.

$$-\boldsymbol{\Delta U} \leq -\boldsymbol{\Delta U^{min}} \quad \textbf{Eq 3.17}$$

$$\boldsymbol{\Delta U} \leq \boldsymbol{\Delta U^{max}} \quad \textbf{Eq 3.18}$$

Where ΔU^{min} and ΔU^{max} are column vectors with N_c elements of Δu^{min} and Δu^{max}, respectively.

The model predictive control when there are hard constraints is proposed as finding ΔU that minimizes the expression defined in Eq 3.19.

$$\boldsymbol{J = \left(R_s - Fx(k_i)\right)^T \left(R_s - Fx(k_i)\right) - 2\Delta U^T \Phi^T \left(R_s - Fx(k_i)\right) + \Delta U^T (\Phi^T \Phi + \bar{R}) \Delta U} \quad \textbf{Eq 3.19}$$

Subject to the inequality constraints of Eq 3.20.

$$\begin{bmatrix} \boldsymbol{M_1} \\ \boldsymbol{M_2} \end{bmatrix} \boldsymbol{\Delta U} \leq \begin{bmatrix} \boldsymbol{N_1} \\ \boldsymbol{N_2} \end{bmatrix} \quad \textbf{Eq 3.20}$$

Where the data matrices are

$$\boldsymbol{M_1} = \begin{bmatrix} -\boldsymbol{C_2} \\ \boldsymbol{C_2} \end{bmatrix} \quad ; \boldsymbol{M_2} = \begin{bmatrix} -\boldsymbol{I} \\ \boldsymbol{I} \end{bmatrix};$$
$$\boldsymbol{N_1} = \begin{bmatrix} -\boldsymbol{U^{min}} + \boldsymbol{C_1 u(k_i - 1)} \\ \boldsymbol{U^{max}} - \boldsymbol{C_1 u(k_i - 1)} \end{bmatrix}; \boldsymbol{N_2} = \begin{bmatrix} -\boldsymbol{\Delta U^{min}} \\ \boldsymbol{\Delta U^{max}} \end{bmatrix}$$

3.3. MPC specification

3.3.1. MPC structure

The MPC structure is understood as the definition of the signals (type and size) that the MPC controller uses. The MPC structure block diagram is shown in Figure 3.2 below.

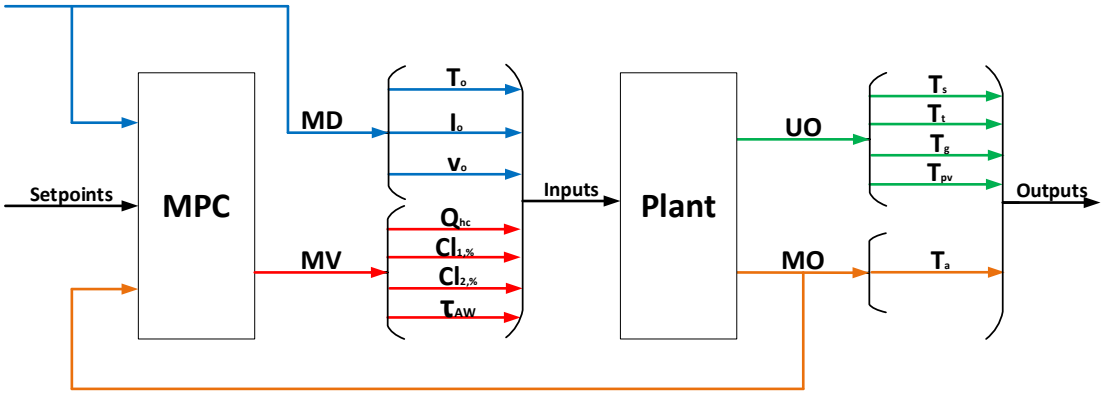


Figure 3.2. MPC structure block diagram

3.3.1.1. Plant inputs

Plant inputs are the independent variables affecting the plant. There are three types:

- **Measured disturbances (MD):** variables that cannot be adjusted by the controller, but they are used for feedforward compensation. Within this category, we find T_o, I_o, and v_o which are estimated variables by means of the weather forecast and the real-time measurements.
- **Manipulated variables (MV):** variables that can be adjusted by the controller in order to achieve its goals. The MV of the MPC are $Q_{hc\%}$, $Cl_{1\%}$, $Cl_{2\%}$, and τ_{AW}.
- **Unmeasured disturbances (UD):** variables of which the controller has no direct knowledge. In the present MPC design there are not UDs.

3.3.1.2. Plant Outputs

Plant outputs are the dependent variables to be controlled or monitored. There are two types:

- **Measured outputs (MO):** variables used by the controller to estimate the unmeasured variables and as feedback on the success of its adjustments. There is only one MO which is T_a.
- **Unmeasured outputs (UO):** variables estimated by the controller but not directly measured by a feedback loop. Within this kind of output, we find the following signals: T_s, T_r, T_g and T_{pv}.

3.3.2. MPC design

In this section, the main design parameters of the MPC are presented and justified.

3.3.2.1. Sample time and horizons

The first parameter to be set in the MPC shall be the sample time. This parameter should not be tunned afterwards. Qualitatively, as the sample time decreases, rejection of unknown disturbances improves

but, as it becomes small, the computational effort increases drastically. Therefore, the optimal choice of the sample time must be a balance of performance and computational effort.

For process control, the sample time is usually much greater than 1 second. Thus, only times much greater than 1 second shall be tested. Figure 3.3 shows the response of the ambient temperature, T_a, against a unitary step in the setpoint.

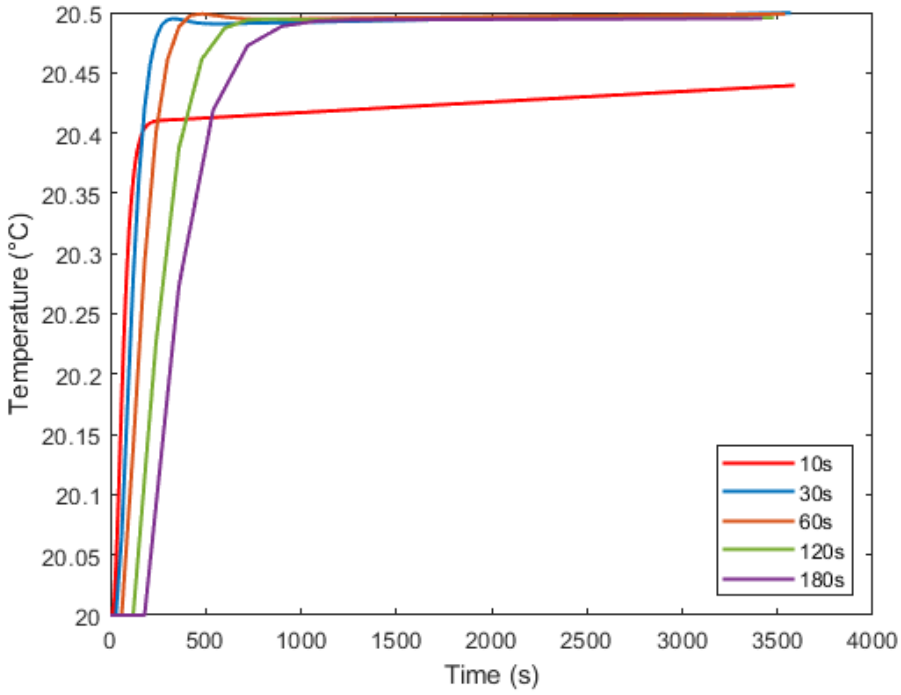


Figure 3.3. Sample time comparison: Ambient temperature response against unitary step setpoint; prediction horizon = 10; control horizon = 3

As we can observe, the response becomes slower as well as the sample time increases. Following the controller creation guidelines of MATLAB (The MathWorks Inc 2022), the sample time should be set between 10% and 25% of the minimum desired closed-loop response time. The desired closed-response time is 1800 seconds, thus, the sample time selected is 180 seconds.

Regarding the prediction horizon, the recommended practice is to select it early in the controller design and then hold it constant while tunning other controller settings. The value of the prediction horizon should be such that the controller is internally stable and anticipates constraint violations. Figure 3.4 shows a comparison between different prediction horizons. As we can observe, the variation is

minimum. The prediction horizon selected shall be 50 because the balance between performance and computational effort.

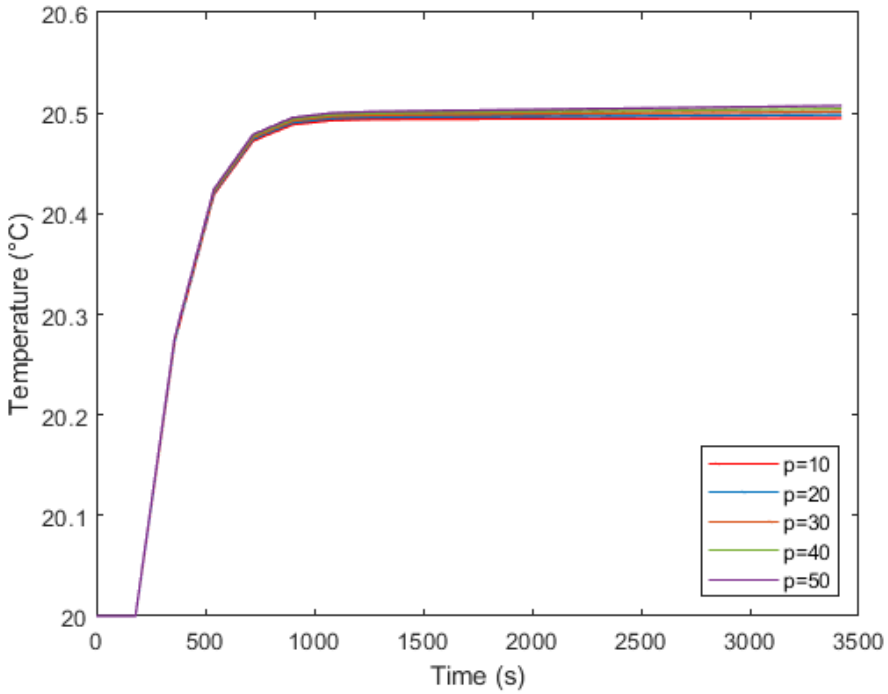


Figure 3.4. Prediction horizon comparison: Ambient temperature response against unitary step setpoint; step time = 180s; control horizon = 3

The control horizon should be kept much less than the prediction horizon in order to promote faster computations and internal stability of the controller.

The values of sample time, prediction and control horizons have been gathered in Table 9 below.

Parameter	**Value**
Sample time	180 s
Prediction horizon	50
Control horizon	5

Table 9. Sample time and horizons values

3.3.2.2. Specification of constraints

Constraints shall be included to not violate the physical limits of the plant. Only bounds for MV shall be applied due to there is no need to apply bounds to the output variables. Figure 3.5 shows the difference

between unconstrained (green solid line) and constrained (blue solid line) control. Unconstrained control violates the upper limits (red dotted-line). The constraints imposed are presented in Table 10.

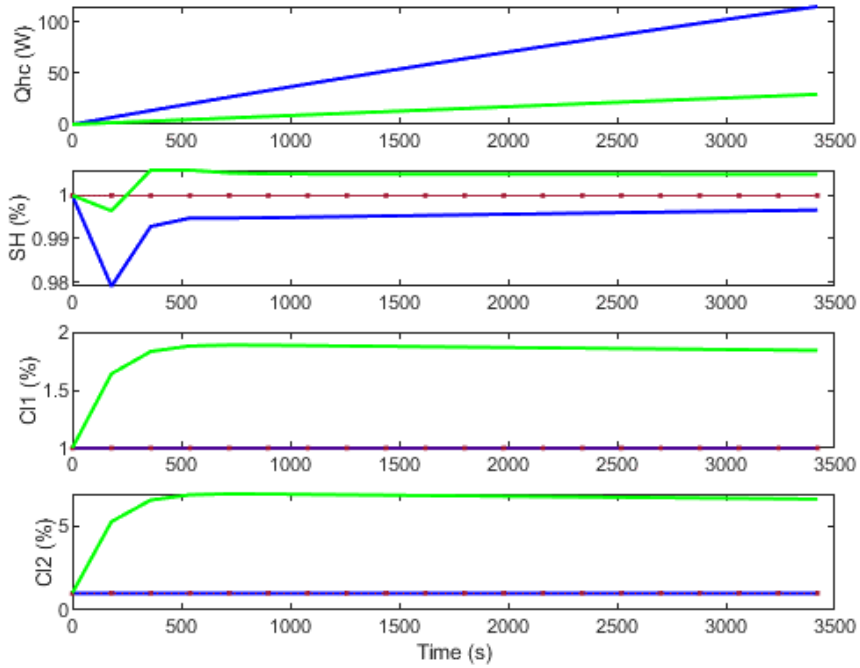


Figure 3.5. Unconstrained control (green) vs. constrained control (blue): Manipulated variables review

MV	u^{max}	u^{min}	Δu^{max}	Δu^{min}
Q_{hc}	5000	-5000	200	-200
τ_{AW}	1	0	0.1	-0.1
$Cl_{1\%}$	1	0	0.5	-0.5
$Cl_{2\%}$	1	0	0.5	-0.5

Table 10. Specification of constraints

Hard constraints are the constraints that the quadratic programming solution must satisfy. Applying hard constraints to the control bounds and to the control increment at the same time can lead to an infeasible solution. In order to avoid infeasibilities produced by hard constraints, the incremental limits shall be considered as soft constraints by means of setting an equal concern for relaxation (ECR). The ECR parameter is associated to each constraint and it indicates the relative magnitude of a tolerable violation. Table 11 shows the values of each ECR parameter.

MV	u^{max}	u^{min}	Δu^{max}	Δu^{min}
Q_{hc}	0	0	1	1
τ_{AW}	0	0	1	1

$Cl_{1\%}$	0	0	0.1	0.1
$Cl_{2\%}$	0	0	0.1	0.1

Table 11. Specification of ECR parameters

3.3.2.3. Specification of scale factors

The scale factor property is used to scale input and output variables when they have large differences in their magnitudes. By means of scaling the variables the MPC shall work with generated dimensionless signals. Potential benefits of scaling are as follows:

- Appropriate scale factors ease tuning and refinement
- Improved numerical conditioning. Round-off errors have less impact on calculations.

The scale factor should be approximated to the span of the variable. Table 12 presents the scale factors of the different variables.

Signal	Type	Unit	Nominal val.	Scale factor
Plant inputs				
Q_{hc}	MV	W	0	1000
τ_{AW}	MV	%	1	1
$Cl_{1\%}$	MV	%	1	100
$Cl_{2\%}$	MV	%	1	100
T_o	MD	°C	25	10
I_o	MD	W/m	300	600
v_o	MD	m/s	3	6
Plant outputs				
T_a	MO	°C	20	1
T_s	UO	°C	10	1
T_r	UO	°C	20	1
T_g	UO	°C	20	0.2
T_{pv}	UO	°C	20	0.001

Table 12. Specification of scale factor parameters

The scale factors od Q_{hc}, T_a, T_s, T_r, T_g and T_{pv} are quite different to their span in order to guarantee a positive-definite Hessian matrix of the quadratic programming problem.

3.3.2.4. Weights tuning

A model predictive controller requires the tuning of the cost function weights. The weights shall be set to the output variables (measured and unmeasured outputs) and to the manipulates variables. The weights tunning has been developed following the MathWork guidelines (MathWorks Inc 2022).Table 13 and Table 14 show the weight values for the input and the output signals respectively.

Signal	Type	Weight	Rate weight
Q_{hc}	MV	0	0.1
τ_{AW}	MV	0	0.1
$Cl_{1\%}$	MV	0	0.1
$Cl_{2\%}$	MV	0	0.1

Table 13. Input weights

As the manipulated variables do not have any target, their weight is 0. The rate weight of the manipulated variables has been set to 0.1 to avoid large increments.

Signal	Type	Weight
T_a	MO	1000
T_s	UO	0.1
T_r	UO	0.1
T_g	UO	0.1
T_{pv}	UO	0.1

Table 14. Output weights

Regarding the output variables, the reference tracking of T_a is prioritized above the other unmeasured outputs. Giving a weight of 1000 to T_a, the tracking error shall be reduced.

3.3.2.5. Look-Ahead (previewing)

To improve the performance of the controller, a look-ahead (previewing) function on the reference and the measured disturbances has been added to the controller.

By means of previewing, the controller is able to actuate over the manipulated values before a change in the setpoint or the measured disturbances.

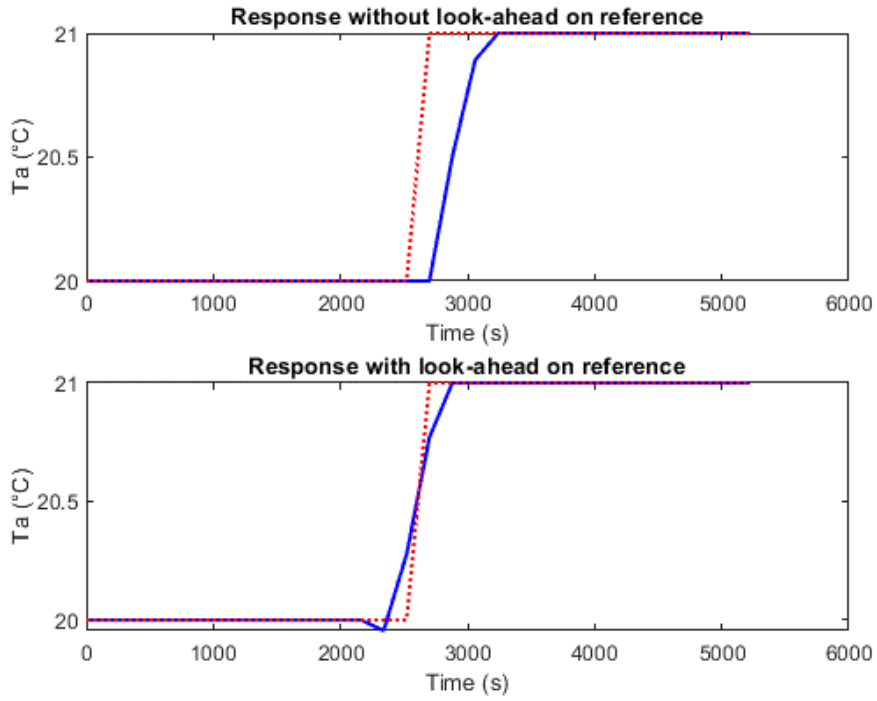


Figure 3.6. Behaviour of the response with and without look-ahead on reference

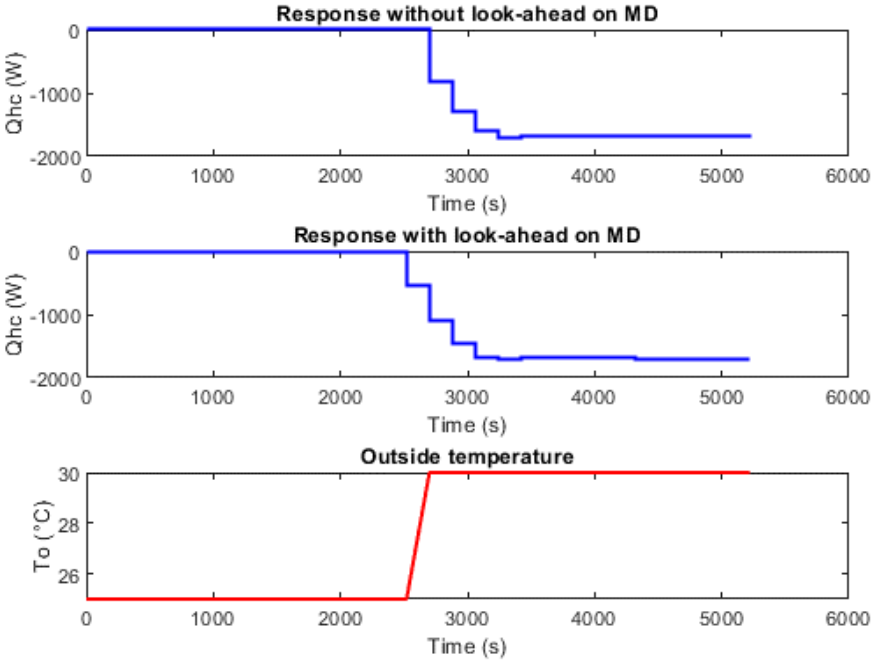


Figure 3.7. Behaviour of the response with and without look-ahead on MDs

3.4. MPC simulation

The MPC, specified in section MPC design, has been tested in four different simulation scenarios in order to study its response against different weather conditions. All the scenarios are located at Savona using historical weather data. The simulation scenarios are the following:

- Winter (18th February 2021)
- Spring (18th May 2021)
- Summer (18th August 2021)
- Autumn (18th November 2021)

The objective of the simulation of these four scenarios is to ensure the performance of the controller against any realistic scenario.

3.4.1. Winter scenario

The historical data of the measured disturbances of this scenario is presented in Figure 3.8. The outdoor temperature does not present considerable variations throughout the day, its value is around 12 °C. The shortwave solar radiation peak takes place at 13:00h with a value that is below than 500 W/m^2. Regarding the wind speed, its value oscillates around 5 m/s.

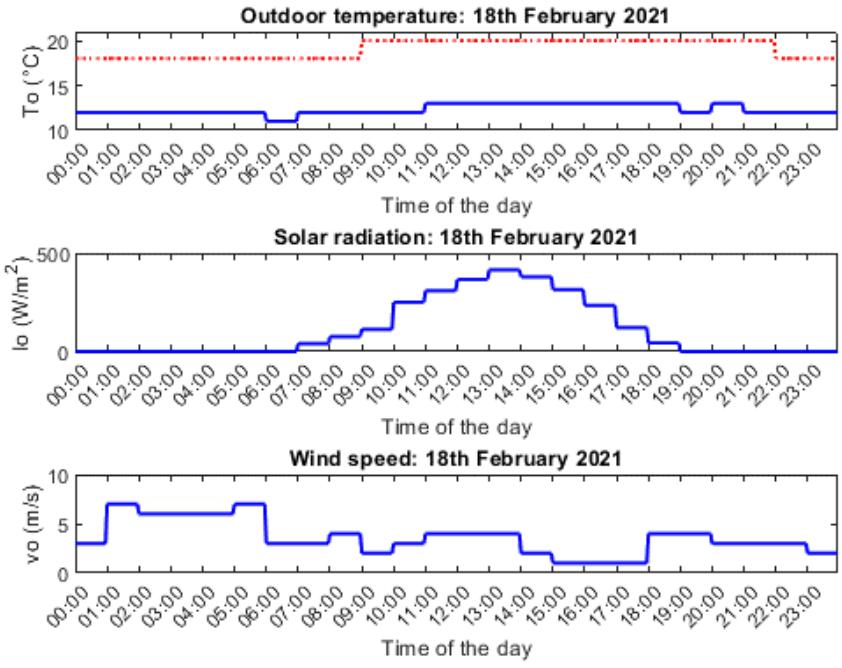


Figure 3.8. Winter scenario: measured disturbances

3.4.1.1. Ambient temperature

The ambient temperature response within this scenario is presented in Figure 3.9.

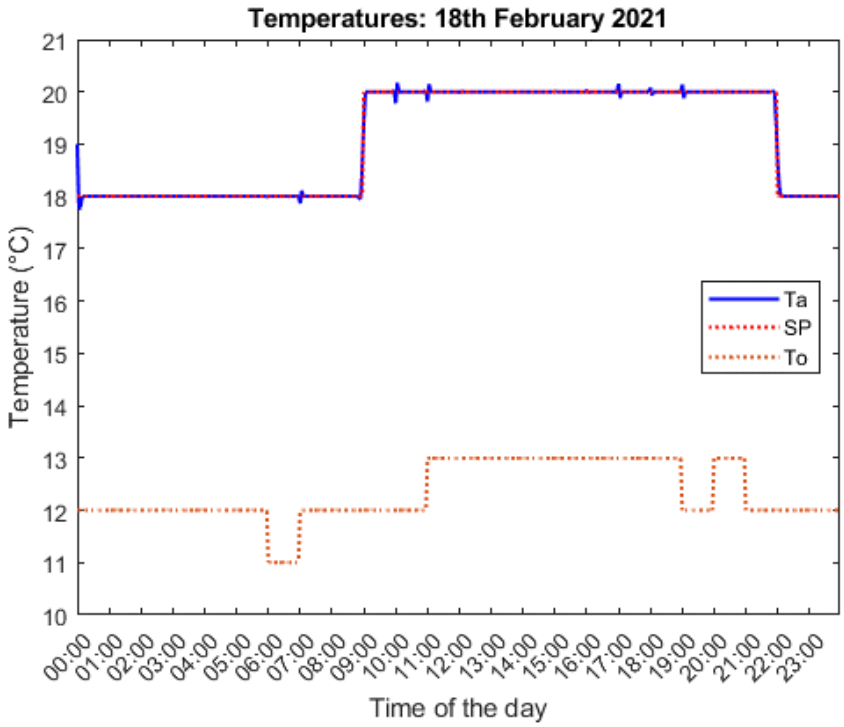


Figure 3.9. Winter scenario: ambient temperature response

3.4.1.2. Manipulated Variables

The Q_{hc} simulation is presented in Figure 3.10 and the simulation of τ_{AW}, $Cl_{1,\%}$ and $Cl_{2,\%}$ is presented in Figure 3.11.

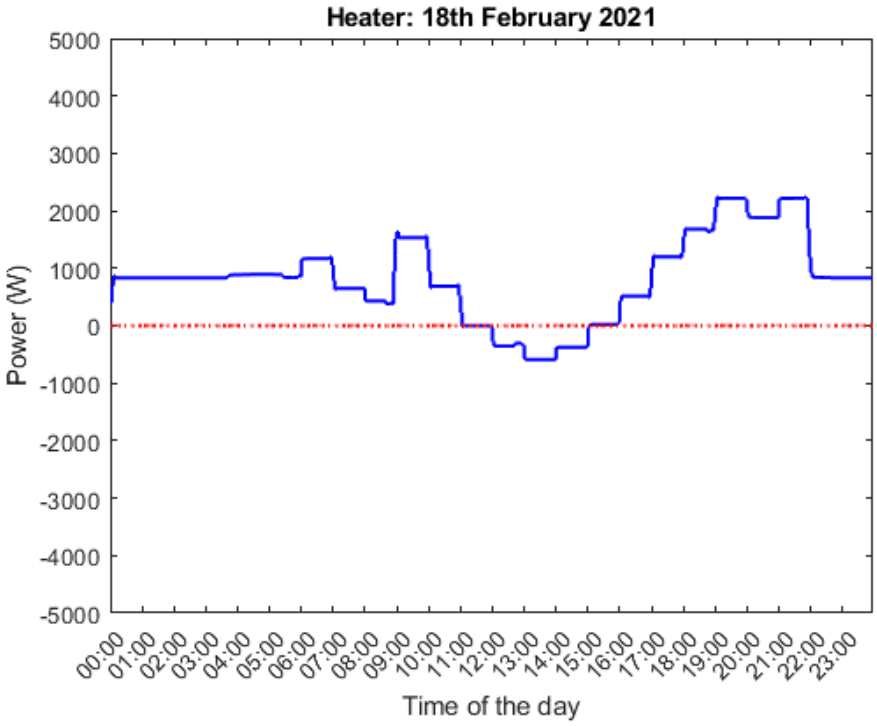


Figure 3.10. Winter scenario: Q_{hc}

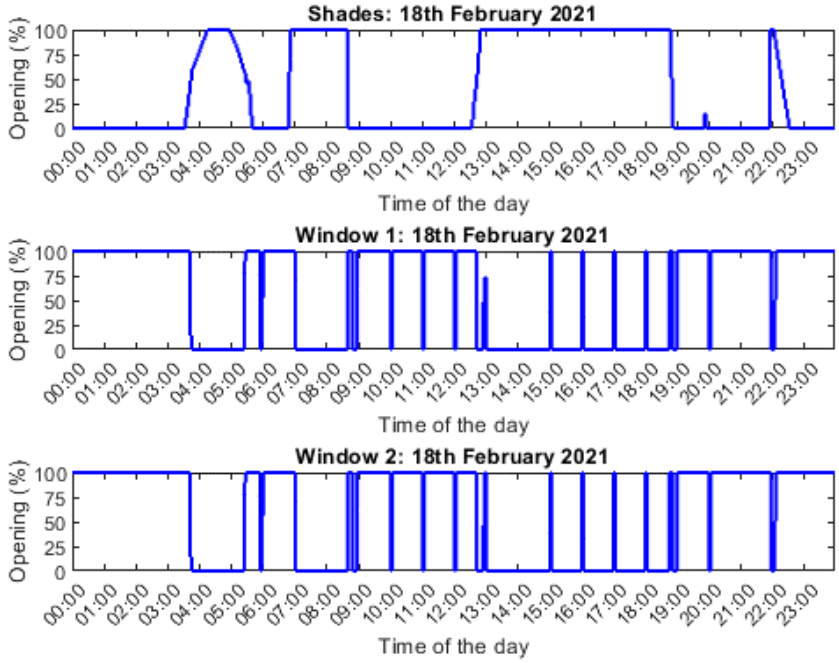


Figure 3.11. Winter scenario: τ_{AW}, $Cl_{1\%}$, $Cl_{2\%}$

3.4.2. Spring scenario

The historical data of the measured disturbances of this scenario is presented in Figure 3.12. The outdoor temperature raises during the day reaching 20 °C at 18:00h, the difference between the maximum and the minimum temperature is 5 °C. In this scenario, the shortwave solar radiation is lower than expected – reaching only 200 W/m^2 in its peak. The wind speed oscillates throughout the day around 3 m/s.

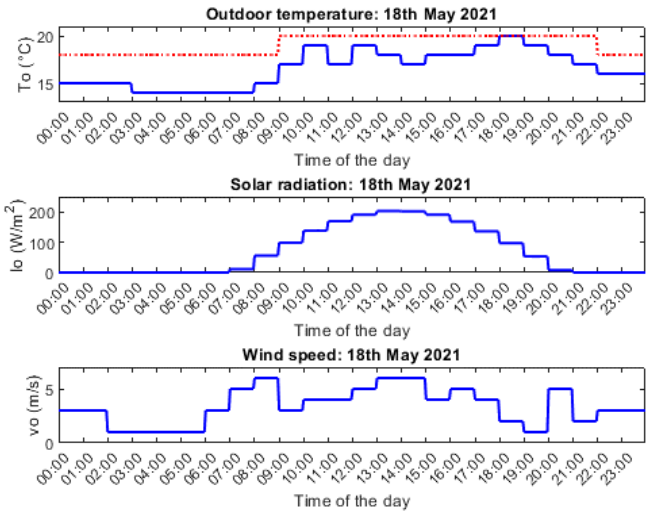


Figure 3.12. Spring scenario: measured disturbances

3.4.2.1. Ambient temperature

The ambient temperature response within this scenario is presented in Figure 3.13.

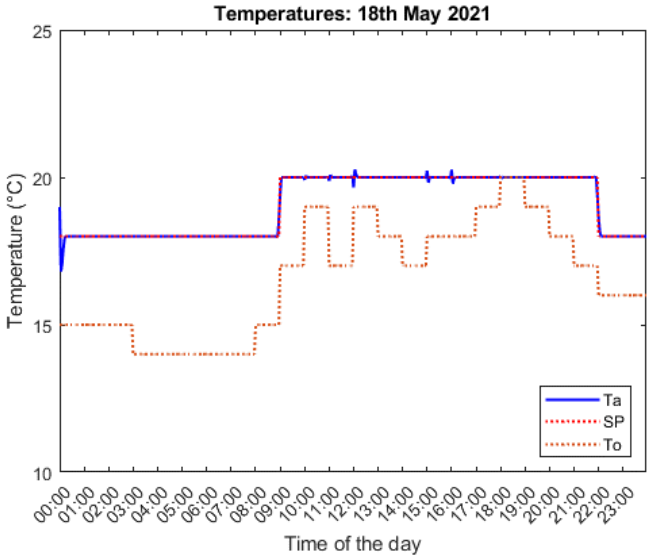


Figure 3.13. Spring scenario: ambient temperature response

3.4.2.2. Manipulated Variables

The Q_{hc} simulation is presented in Figure 3.14 and the simulation of τ_{AW}, $CI_{1,\%}$ and $CI_{2,\%}$ is presented in Figure 3.15.

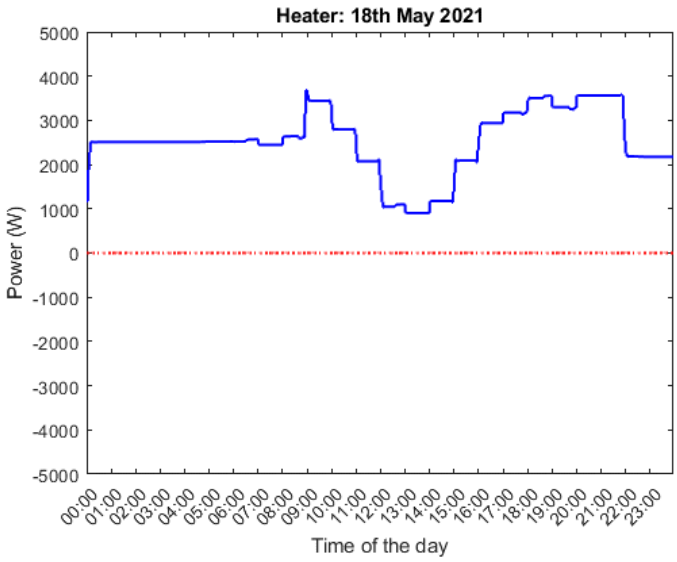


Figure 3.14. Spring scenario: Q_{hc}

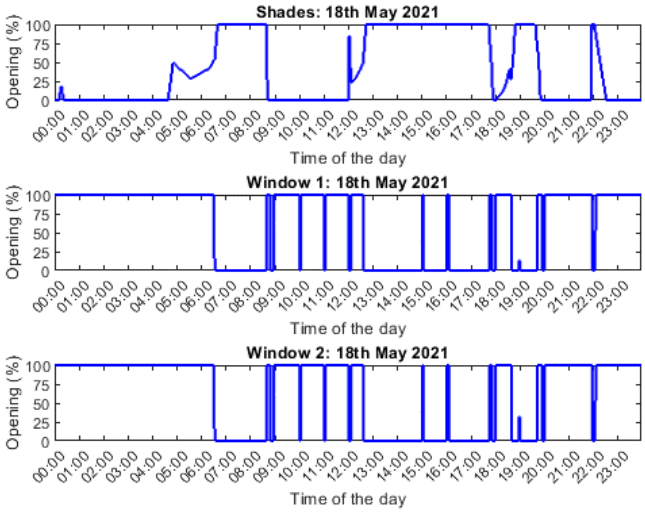


Figure 3.15. Spring scenario: τ_{AW}, $Cl_{1\%}$, $Cl_{2\%}$

3.4.3. Summer scenario

The historical data of the measured disturbances of this scenario is presented in Figure 3.16. During summer scenario, the outdoor temperature surpasses 30 °C in some instants of the day. Furthermore, the temperature does not lower 25 °C. Regarding the solar irradiance, it reaches 870 W/m^2 of peak value.

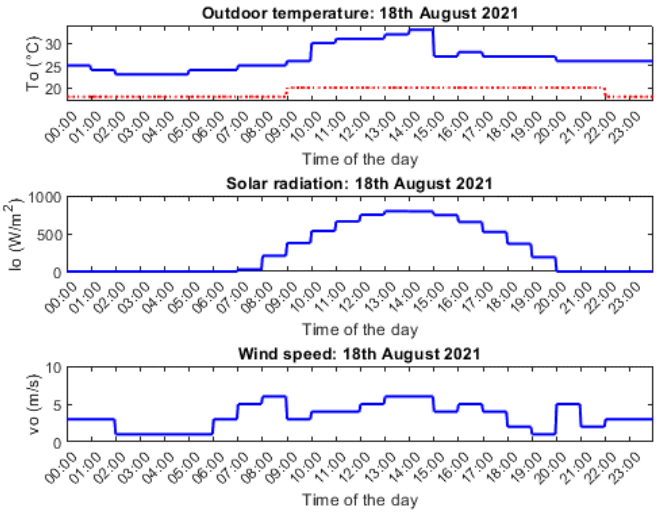


Figure 3.16. Summer scenario: measured disturbances

3.4.3.1. Ambient temperature

The ambient temperature response within this scenario is presented in Figure 3.17.

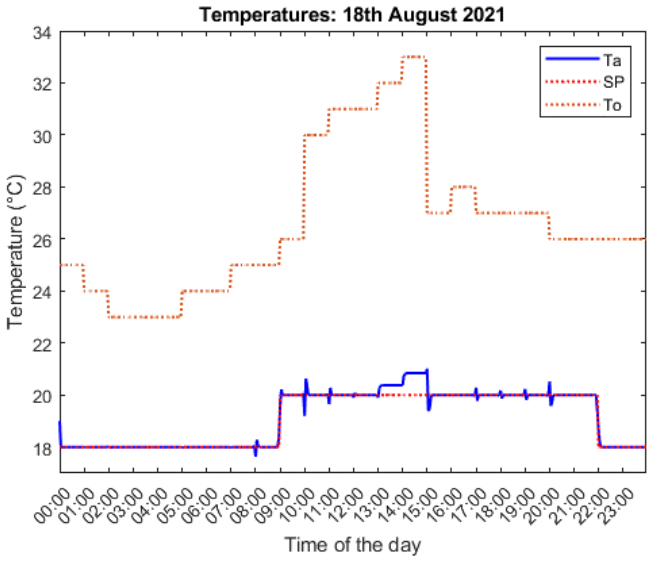


Figure 3.17. Summer scenario: ambient temperature response

3.4.3.2. Manipulated Variables

The Q_{hc} simulation is presented in Figure 3.18 and the simulation of τ_{AW}, $Cl_{1,\%}$ and $Cl_{2,\%}$ is presented in Figure 3.19.

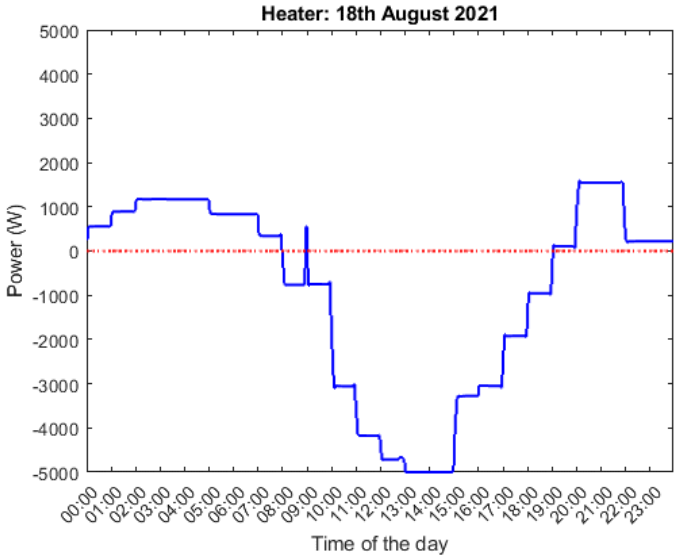


Figure 3.18. Summer scenario: Q_{hc}

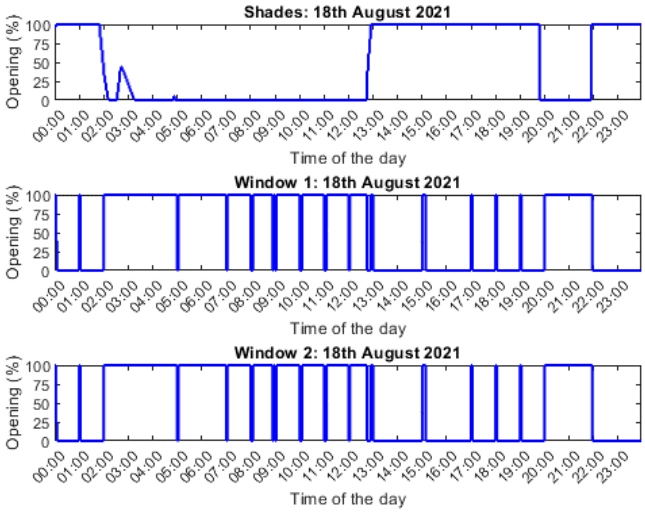


Figure 3.19. Summer scenario: τ_{AW}, $Cl_{1\%}$, $Cl_{2\%}$

3.4.4. Autumn scenario

The historical data of the measured disturbances of this scenario is presented in Figure 3.20. The outdoor temperature does not present considerable variations throughout the day, its value is around 12 °C. The shortwave solar radiation peak takes place at 13:00h with a value that is below than 500 W/m^2. Regarding the wind speed, its value increases throughout the day.

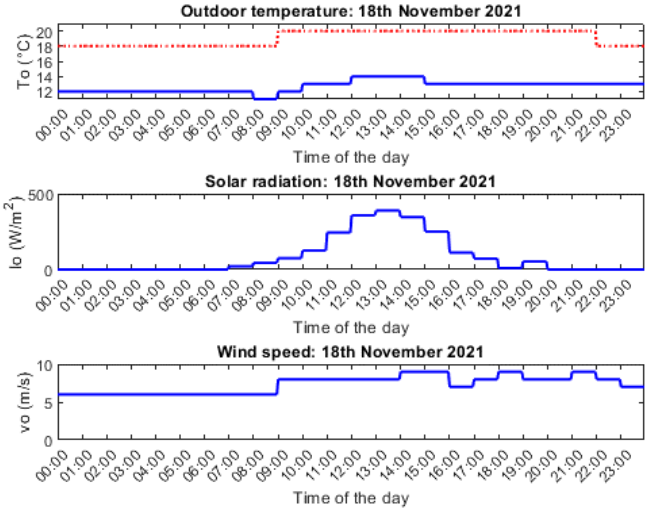


Figure 3.20. Autumn scenario: measured disturbances

3.4.4.1. Ambient temperature

The ambient temperature response within this scenario is presented in Figure 3.21.

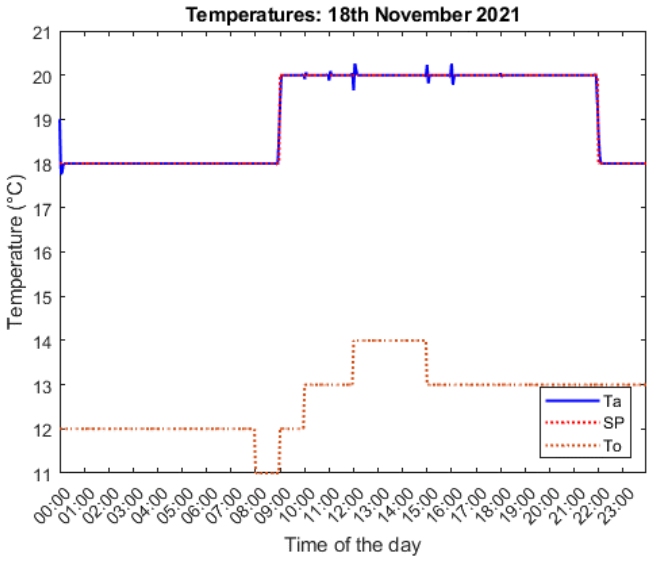


Figure 3.21. Autumn scenario: ambient temperature response

3.4.4.2. Manipulated Variables

The Q_{hc} simulation is presented in Figure 3.22 and the simulation of τ_{AW}, $Cl_{1,\%}$ and $Cl_{2,\%}$ is presented in Figure 3.23.

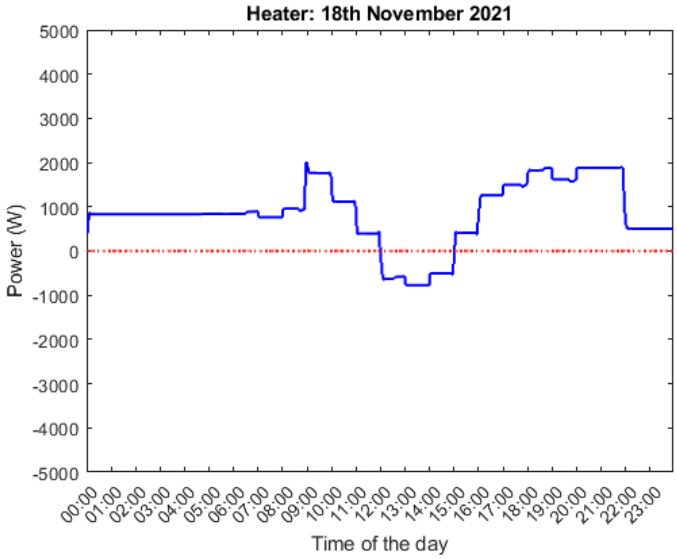


Figure 3.22. Autumn scenario: Q_{hc}

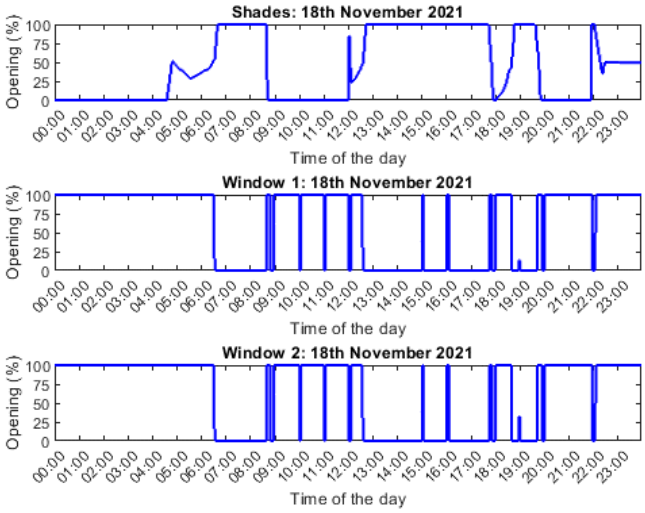


Figure 3.23. Autumn scenario: τ_{AW}, $Cl_{1\%}$, $Cl_{2\%}$

4. Physical domain

The physical domain is the environment related to the actual physical plant. In the present chapter, the minimum architecture of the system and the needed requirements to interface the physical domain with the digital twin are presented.

4.1. Physical domain architecture

The hardware architecture of the physical domain, as well as its data flow, is presented in Figure 4.1 below.

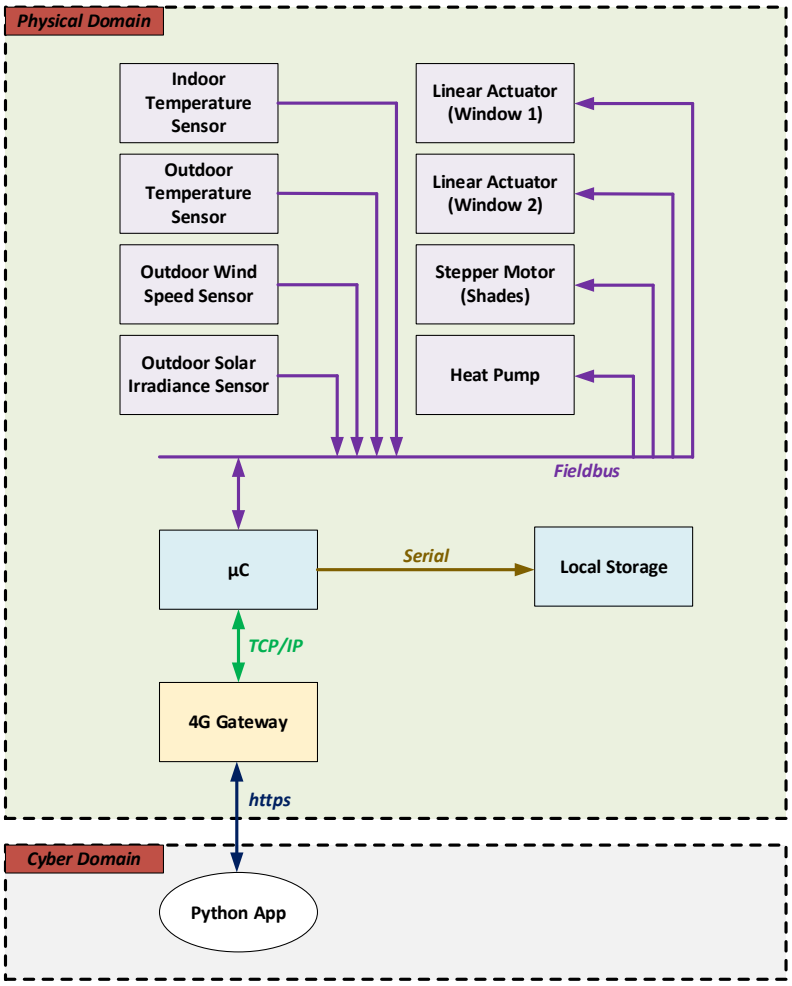


Figure 4.1. Diagram block of the physical domain architecture

The communication network follows a linear bus topology. Sensors, actuators and the microcontroller are connected to the Modbus RTU fieldbus. The microcontroller is directly connected to a 4G gateway by means of TCP/IP communication in order to have access to internet. The microcontroller stores the transferred data into a flash memory to have a backup of the information in case of an internet cut off.

The minimum operation of the microcontroller is the following:

1. The microcontroller reads the data from the sensors via Modbus RTU
2. The microcontroller sends sensor data to the cyber domain via HTTPS through the 4G gateway
3. The microcontroller reads actuator setpoints from the cyber domain via HTTPS through the 4G gateway
4. The microcontroller sends setpoint data to the actuators vis Modbus RTU
5. The microcontroller stores sensor and actuator data in a flash memory

4.2. Physical domain requirements

In the present section, the design requirements for the component selection shall be presented. These are the minimum requirements needed to fit with the exigencies of the digital twin design (refer to 2. Greenhouse model, 3. Model Predictive Control and 5. Cyber domain).

4.2.1. Sensors

The list of necessary sensors with their minimum specifications is shown in Table 15.

Sensor	**Use**	**Min range**
Temperature	Indoor	[10, 35] °C
Temperature	Outdoor	[-10, 50] °C
Anemometer	Outdoor	[0, 15] m/s
PV pyranometer	Outdoor	[0, 2000] W/m²

Table 15. Sensor requirements

The indoor temperature sensor shall be used for measuring the ambient temperature (T_a) within the greenhouse. The outdoor temperature sensor shall be used for measuring the temperature outside the greenhouse (T_o). The anemometer shall sense the wind speed (v_o) and the PV pyranometer shall sense the shortwave solar irradiance (I_o). The minimum ranges have been selected according to the average maximum and minimum values of the historical weather data in Savona (World Weather Online 2022).

4.2.2. Actuators

The list of necessary actuators with their minimum specifications is shown in Table 16.

Actuator	Use	Min range	Min precision
Linear actuator (x2)	Indoor	[0, 0.5] m	0.05 m
Servomotor	Indoor	N/A	0.045 m
Heat pump	Indoor	[-5000, 5000] W	200 W

Table 16. Actuator requirements

Linear actuators are used for opening and closing both windows ($Cl_{1,\%}$ and $Cl_{2,\%}$). The servomotor shall be used for controlling shades extension (τ_{AW}). The heat pump is the actuator used for heating/cooling the greenhouse (Q_{hc}). The minimum ranges and the minimum precision have been set to satisfy the specifications of the MPC controller.

5. Cyber domain

The cyber domain is the environment where the data processing takes place, comprising networks, devices, all software, processes, information in storage or in transit, applications and services (Maurer and Morgus 2014).

5.1. Cyber domain architecture

The software architecture of the cyber domain, as well as its data flow, is presented in Figure 5.1 below.

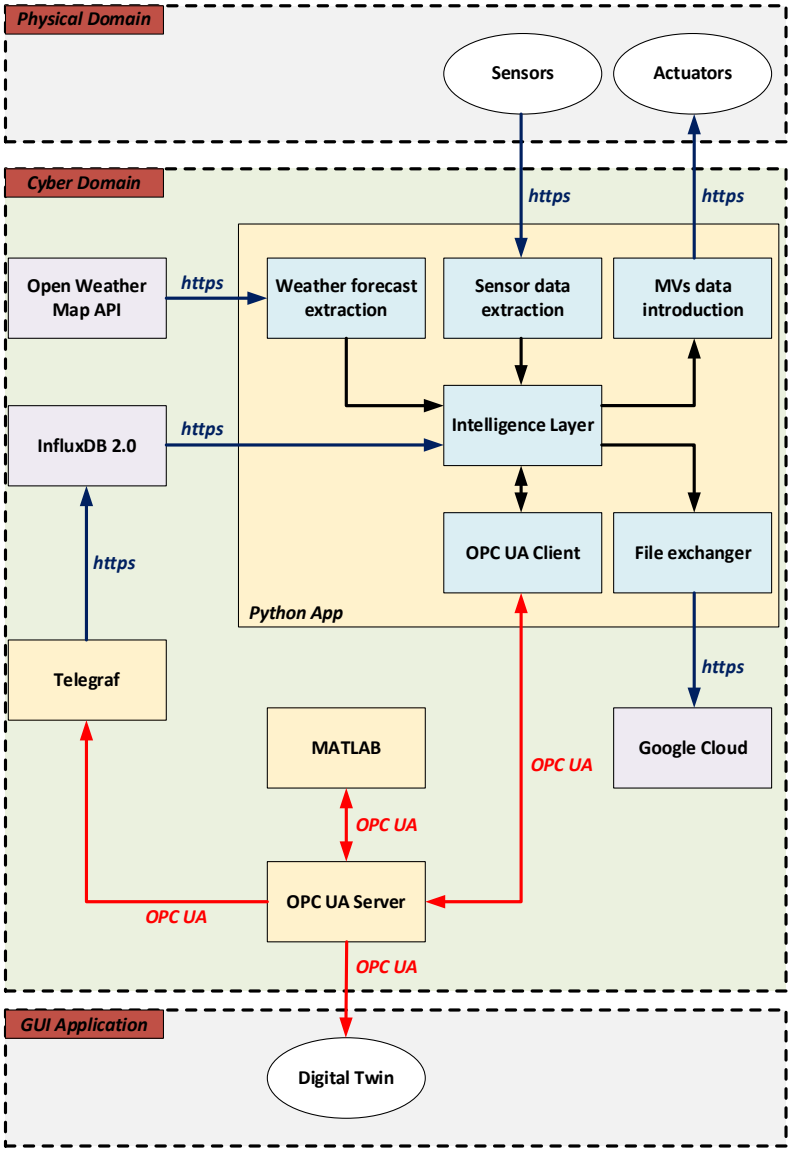


Figure 5.1. Diagram block of the cyber domain architecture

Within the presented architecture, based on (Gomes, et al. 2019), (Chaux, Sanchez-Londono and Barbieri 2021) and (Jang-Singh, et al. 2020) architectures, three main components can be distinguished:

- **Communication protocols:** systems of rules that allow two or more entities of a communication system to transmit information. OPC UA is the communication protocol used to transmit data between the different stand-alone applications. HTTPS is the communication protocol used to transmit information between stand-alone applications and cloud services.
- **Stand-alone applications:** applications able to operate independently of other software. These applications are located within a physical server where they run simultaneously and independently. The present category is composed of the python application, the OPC UA server, MATLAB and Telegraf.
- **Cloud services:** services delivered on demand over the internet. InfluxDB 2.0 is an open-source time series database (TSDB) for storage and retrieval of time series data. Google Cloud Platform (GCP) is used to store and retrieve files (csv, xml, scripts and MATLAB data files). OpenWeatherMap API is the service to retrieve the weather forecast used for the control of the plant.

The OPC UA Server initialises its service in order to enable the OPC UA communication between the server and its clients (Python application, Telegraf, MATLAB and the Digital Twin). Once the OPC UA communication is stablished, Telegraf begins its operation converting the OPC UA data into HTTPS data able to be read for InfluxDB 2.0. The database stores all the data in a time series format in order to be retrieved on demand by the application.

The python application retrieves the sensor data from the physical domain and the weather forecast data from OpenWeatherMap API. Furthermore, the python application gathers both data and processes it in order to send them to the OPC UA server through the OPC UA client.

MATLAB saves the current state of the controller in a local file in order to be retrieved when needed. As OPC UA client, MATLAB reads the processed data from the sensors, the processed forecasted weather data and the current and future setpoints. The MPC controller processes the input data to calculate the values for the manipulated variables. The output data is sent via OPC UA and the new current state of the controller is saved.

The python application retrieves the MATLAB output data and processes it in order to send the manipulated variable values to the actuators located in the physical domain.

The intelligence layer of the python application retrieves the historical data from the InfluxDB service. This data is processed and it is sent to Google Cloud Platform (GCP) in order to have backup files. MATLAB files are also sent to GCP.

The Digital Twin application receives all the data through OPC UA.

5.2. Communication protocols

In this section, the communication protocols used within the cyber domain shall be presented.

5.2.1. OPC UA

The OPC Unified Architecture (UA) is a platform independent service-oriented architecture that integrates all the functionality of the individual OPC Classic specifications into one extensible framework. The most significant difference between classical OPC and OPC UA is that it does not rely on OLE or DCOM technology from Microsoft that makes it possible to implement it on any platform (OPC UA Foundation 2021).

OPC UA is an IEC (International Electromechanical Commission) standard developed by OPC Foundation in collaboration with users, manufacturers and researchers.

Characteristics of OPC UA:

- Oriented to data collection and control systems
- Open protocol – there are no restrictions in its usage
- Cross-platform
- Service-oriented architecture (SOA)
- Robust security

5.2.2. HTTPS

Hypertext Transfer Protocol Secure (HTTPS) is an extension of the Hypertext Transfer Protocol (HTTP). It is used for secure communication over a computer network, and it is widely used on the Internet. The protocol uses TLS/SSL encryption over the HTTP to encrypt communications by using an asymmetric public key infrastructure. The security system uses two keys to encrypt the communication between two different parties. There is a private key, owned by the server, that decrypts the information encrypted by the public keys. In the other hand, there are public keys, available to everyone who wants to interact with the server, that only can be decrypted by private keys.

5.3. Stand-alone applications

In this section, the stand-alone application (python application, OPC UA server, MATLAB and Telegraf) shall be presented.

5.3.1. Python application

The libraries used in the python application are gathered in Table 17.

Name	Version	Description
json	v3.10.7	Allows to work with json files
requests	v2.28.1	HTTP library
numpy	v1.23.0	Adds support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions
opcua	v0.98.13	Python OPC UA/IEC-62541 client and server
solarpy	v0.1.3	Provides a reliable solar radiation model
csv	v3.10.7	Allows python to read and write CSV files
subprocess	v3.10.7	Allows to spawn new processes, connect to their input/output/error pipes, and obtain their return codes
storage	v2.0	Allows the access to Google Cloud Storage from python
datetime	v3.10.7	Supplies classes for manipulating dates and times

Table 17. Libraries used in the python application

5.3.1.1. Sensor data extraction

The purpose of this submodule is to retrieve the sensor data (T_a, T_o, I_o and v_o) from Google Cloud Storage (refer to GCS configuration).

Variable name	Type	Size	Description
Ta	Float	1	Ambient temperature readout from sensor
To	Float	1	Outdoor temperature readout from sensor
Io	Float	1	Solar irradiance readout from sensor
vo	Float	1	Wind-speed readout from sensor
bucket_name	String	13	GCP bucket ID
blob_sensor	String	19	GCP path of sensor.csv
path_sensor	String	58	Local path for sensor.csv

Table 18. Global variables used to extract sensor data

The application accesses the blob where the sensor data is stored. Then, the submodule downloads the CSV file and gives the corresponding value to the global variables.

5.3.1.2. MVs data introduction

The purpose of this submodule is to send the MV data values to Google Cloud Storage. Hence, the physical domain shall be able to retrieve the MVs data.

Variable name	Type	Size	Description
Qhc	Float	1	Power value for the actuator
SH	Float	1	Extension percentage for the shades
Cl1	Float	1	Opening percentage for 1st window
Cl2	Float	1	Opening percentage for 2nd window
bucket_name	String	13	GCP bucket ID
blob_actuator	String	19	GCP path for actuator.csv
path_sensor	String	58	Local path of actuator.csv

Table 19. Global variables used to send MVs data

The application creates a local CSV file with the values of the manipulated variables for the actuators (Q_{hc}, τ_{AW}, $Cl_{1,\%}$ and $Cl_{2,\%}$). Therefore, the CSV file is uploaded to Google Cloud Storage to be accessible for the physical domain.

5.3.1.3. Weather forecast extraction

The purpose of this submodule is to retrieve the weather forecast (temperature and wind) from OpenWeatherMap API (see OpenWeatherMap API), to retrieve the solar beam irradiance by means of solarpy python library and to process the data in order to be readable for other stand-alone applications.

Variable name	Type	Size	Description
fore_solar	List	100	5-hour prediction of the irradiance
fore_temp	List	100	5-hour forecast of the temperature
fore_wind	List	100	5-hour forecast of the wind-speed
h	Int	1	Height of Savona compared to the sea-level
lat	Float	1	Latitude of Savona
url_forecast	String	126	URL direction of the OpenWeatherMap API
vnorm	List	3	Plane pointing zenith

Table 20. Global variables used by weather forecast extraction module

The application, by means of requests library (see Table 17), retrieves the temperature (in °C) and wind-speed (in m/s) hourly-data for the following 5 hours from OpenWeatherMap API. The data obtained is

parsed using json library. Then, the solarpy library, predicts the hourly solar irradiance (in W/m^2) for the next 5 hours.

Finally, the forecasted and predicted data is transformed to agree with the MATLAB requirements.

5.3.1.4. OPC UA Client

This aim of this submodule is to manage the data traffic between the physical domain (sensors and actuators) and the OPC UA server (stand-alone application).

Variable name	Type	Size	Description
opc_url	String	27	OPC UA server endpoint URL
client	Client	1	Client object of opcua.client.client module
Ta	Float	100	Ambient temperature readout from sensor
To	Float	1	Outdoor temperature readout from sensor
Io	Float	1	Solar irradiance readout from sensor
vo	Float	126	Wind-speed readout from sensor
fore_solar	List	100	5-hour prediction of the irradiance
fore_temp	List	100	5-hour forecast of the temperature
fore_wind	List	100	5-hour forecast of the wind-speed
Qhc	Float	1	Power value for the actuator
SH	Float	1	Extension percentage for the shades
Cl1	Float	1	Opening percentage for 1st window
Cl2	Float	1	Opening percentage for 2nd window

Table 21. Global variables used by OPC UA client

OPC UA client has two modes of operation:

- **OPC UA client input:** the client stablishes connection with the OPC UA server. Then, it writes the values of the sensor's readout, the forecasted weather data and the predicted irradiance to the corresponding nodes of the OPC UA server. Finally, the client ends the connection with the server.
- **OPC UA client output:** the client stablishes connection with the OPC UA server. Then, it reads the values of the manipulated variables from the OPC UA server and saves its values into global variables. Finally, the client ends the connection with the server.

5.3.1.5. Intelligence layer

The intelligence layer is the higher-level layer of the python application. Thus, it coordinates the execution of the other python functions and batch files. Its flow diagram is shown in Figure 5.2. The

MATLAB operation, shown in Figure 5.3, is also managed by the intelligence layer. Refer to A8. Python intelligence layer for the code.

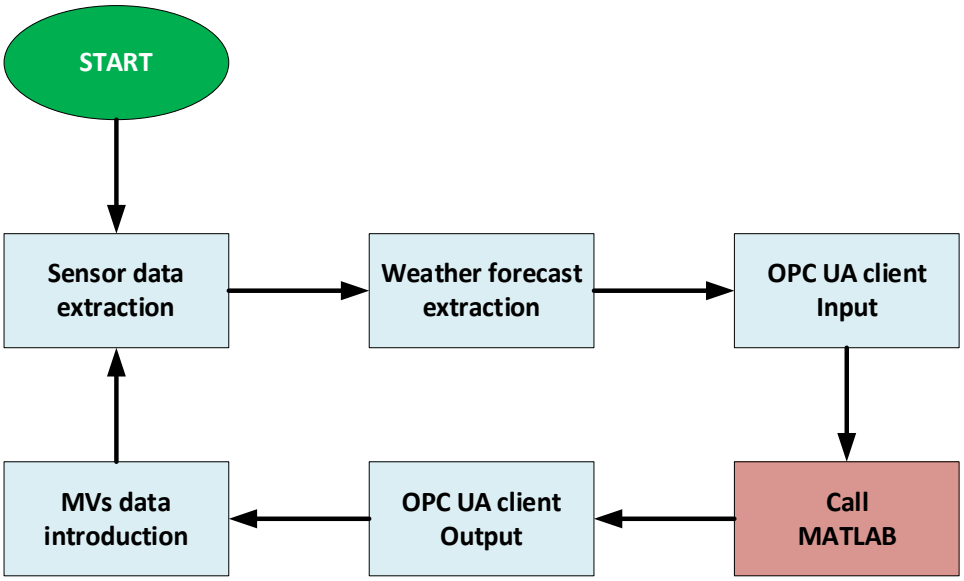


Figure 5.2. Flow diagram of the intelligence layer (python)

The python file shall be executed after the OPC UA server. The execution is cyclical:

1. Retrieves sensor data from the cloud and saves the data locally.
2. Retrieves weather forecast data from the cloud and processes it to fit the OPC UA requirements.
3. Writes the retrieved data to their OPC UA nodes.
4. Calls MATLAB (see MATLAB operation).
5. Reads from OPC UA the new estimated states and manipulated variables, and saves the data locally.
6. Processes and sends the MVs data read in the previous step to the cloud.

Each cycle shall be executed every three minutes to fit with the controller specifications (see Sample time and horizons).

5.3.2. OPC UA server

The OPC UA server is a python stand-alone application which manages access to OPC UA communication within the cyber domain. The server defines the OPC UA data structure (nodes), manages the access to the OPC UA clients (Python application, Telegraf, MATLAB and Digital Twin application) and gives them reading/writing permissions. The OPC UA server has been developed by means of opcua python library (see Table 17).

5.3.2.1. OPC UA server node configuration

The OPC UA node structure is presented in Table 22. Its structure is divided in objects which group variables of similar purpose. The server allows the client to read and write any object or variable excepting the Server object.

Object	Variable	NsIndex	ID	Description
Server	Server	0	2253	Object container for Server
MVs	MVs	2	1	Object container for MVs
	Qhc	2	2	Nominal power of heater/cooler
	SH	2	3	Extension percentage of shades
	Cl1	2	4	Opening percentage of 1st window
	Cl2	2	5	Opening percentage of 2nd window
States	States	2	6	Object container for States
	Ta	2	7	Ambient temperature (sensor)
	Ts	2	8	Soil temperature (estimated)
	Tr	2	9	Roof air temperature (estimated)
	Tg	2	10	Glass temperature (estimated)
	Tpv	2	11	PV module temperature (estimated)
Refs	Refs	2	12	Object container for Refs
	r1	2	13	Ambient temperature's setpoint
MDs	MDs	2	14	Object container for MDs
	To	2	15	Outdoor temperature (sensor)
	Io	2	16	Solar irradiance (sensor)
	vo	2	17	Wind-speed (sensor)
Vectors	Vectors	2	18	Object container for Vectors
	To_vec	2	19	Forecasted outdoor temperature
	Io_vec	2	20	Predicted solar irradiance
	vo_vec	2	21	Forecasted wind-speed
Flags	Flags	2	22	Object container for Flags
	Flag1	2	23	Utility flag 1 (Boolean)
	Flag2	2	24	Utility flag 2 (Boolean)
	Flag3	2	25	Utility flag 3 (Boolean)

Table 22. OPC UA node list

5.3.2.2. OPC UA server operation

The OPC UA server is executed by means of a batch file (OPCUA_server.bat) called in first place by the python application. The server stops its operation only during maintenance. In A7. Python OPC UA server the OPC UA server python code is presented.

5.3.3. MATLAB

MATLAB is a proprietary multi-paradigm programming language and numeric computing environment developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages.

Software	Version	Company
MATLAB	R2021b	MathWorks Inc.

Table 23. MATLAB software specification

MATLAB needs the following toolboxes to accomplish the purpose of the application:

- **Industrial Communication Toolbox:** allows the data exchange over OPC UA, Modbus, MQTT, and other industrial protocols.
- **Model Predictive Control Toolbox:** allows the design and simulation of model predictive controllers.
- **Symbolic Math Toolbox:** allows to perform symbolic math computations.

5.3.3.1. OPC UA client

The purpose of the OPC UA client is to connect MATLAB with the OPC UA endpoint in order to stablish a communication with the server. Once the client is connected to the server, MATLAB retrieves the node list (see Table 22) and save in its workspace the nodes that shall use afterwards (MVs, States, MDs, Vectors and References). Then, MATLAB loads the file (GH.mat) in its workspace which contains the information related to the MPC controller and the plant.

This script must always be run before other MATLAB scripts (initialization, loop or update). During the OPC UA client execution, MATLAB does not end the connection with the server because MATLAB shall disconnect from the server in the other MATLAB instances. Refer to A3. MATLAB OPC UA client for the code.

5.3.3.2. MATLAB initialization

The MATLAB initialization must be run only once. The script initializes the MPC controller and save its initial state into a file (save.mat). Therefore, the initialization script reads (through OPC UA) the reference and the MDs' forecast and saves their values in offline files (references.mat and MDs.mat respectively). Finally, the script disconnects the client from the server. Refer to MATLAB initialization code.

5.3.3.3. MATLAB loop

The MATLAB loop is the script that shall run iteratively. Its purpose is to calculate the MVs' values for one sample time. Thus, it shall be executed in each sample time.

The script loads in its workspace the data of the following files:

- save.mat
- references.mat
- MDs.mat

Then, MATLAB has the information of the current state of the controller, the reference (setpoint) vector and the forecast of the measured disturbances. Afterwards, MATLAB retrieves the current data of the sensors (*Ta*, *To*, *Io* and *vo*) and the estimation of the states (*Ts*, *Tr*, *Tg* and *Tpv*). Using the current value of the states, the script updates the outputs.

Afterwards, the MPC controller calculates the next values for the manipulated variables. MATLAB updates the states by means of the estimations of the controller and sends via OPC UA the value of the manipulated variables (*Qhc*, τ_{AW}, $Cl_{1,\%}$ and $Cl_{2,\%}$) and the estimated states (*Ts*, *Tr*, *Tg* and *Tpv*).

The vectors containing the references and the measured disturbances shift one position to be ready for the next iteration. Finally, the current status, the shifted references and the shifted measured disturbances shall be saved in save.mat, references.mat and MDs.mat respectively. Refer to A5. MATLAB loop code.

5.3.3.4. MATLAB reference and forecast update

The purpose of the present script is to update the reference and the measured disturbances vectors. The before mentioned vectors are useless after fifty iterations of the script loop. Thus, this script reads the new forecasted data as well as the new references values. In order to be more accurate with the final result, the update script shall be executed after 20 iterations of the loop script. Refer to MATLAB update code.

5.3.3.5. MATLAB operation

MATLAB scripts are executed by means of batch files (init.bat, loop.bat and update.bat) called by the python application. All batch files begin executing the OPC UA client (see OPC UA client) and then they execute their particular function (initialise MATLAB, calculate the next controller move or update the weather forecast inputs).

Figure 5.3 presents the flow diagram of the MATLAB execution.

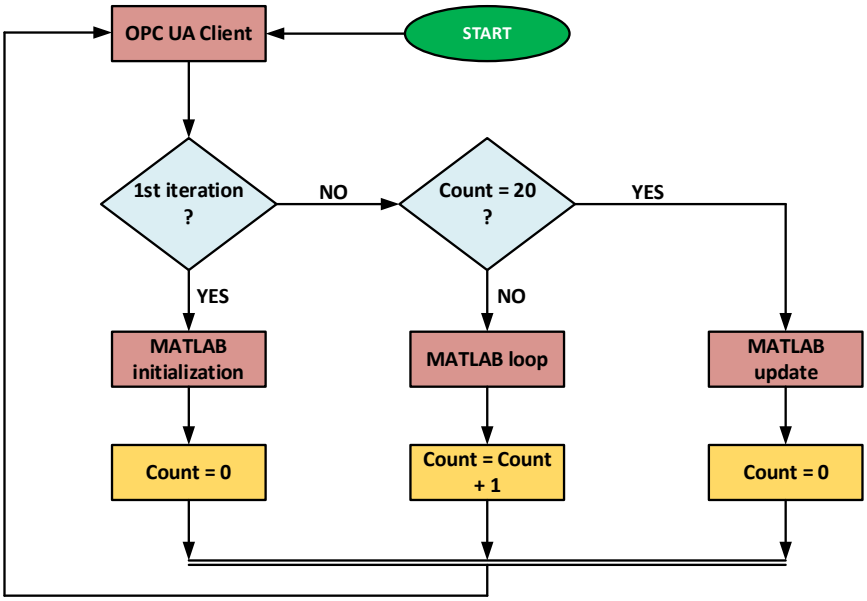


Figure 5.3. Flow diagram of the MATLAB operation

5.3.4. Telegraf

Telegraf is a server-based agent which collects and sends metrics and events from databases, systems, and IoT sensors. Written in Go, Telegraf compiles into a single binary with no external dependencies, requiring very minimal memory.

Software	Version	Company
Telegraf	v1.23	InfluxData Inc.

Table 24. Telegraf software specification

Furthermore, Telegraf is a plugin-driven agent. It supports four categories of plugins:

- **Input plugins:** collect metrics from the services, or third-party APIs.
- **Output plugins:** write metrics to various destinations.
- **Aggregator plugins:** create aggregate metrics (for example, mean, min, max, quantiles, etc)

- **Processor plugins:** transform, decorate and filter metrics.

Plugin	Type	Description
OPC UA	Input	Gathers metrics from client devices using the OPC UA machine-to-machine communication protocol for industrial automation.
InfluxDB v2	Output	Writes metrics to InfluxDB 2.x OSS or Cloud

Table 25. List of Telegraf plugins used

Telegraf is configured by means of a configuration file (see A10. Telegraf configuration code). The configuration file sets the needed values for the Telegraf agent, the input plugin and the output plugin.

5.3.4.1. Agent configuration

The configuration parameters of the agent are the following:

- **interval:** default data collection interval for all inputs
- **round_interval:** rounds collection interval to "interval"
- **metric_batch_size:** Telegraf will send metrics to outputs in batches of at most metric_batch_size metrics.
- **metric_buffer_limit:** maximum number of unwritten metrics per output.
- **collection_jitter:** is used to jitter the collection by a random amount.
- **flush_interval:** default flushing interval for all outputs.
- **flush_jitter:** jitter the flush interval by a random amount.

5.3.4.2. OPC UA plugin configuration

The configuration parameters of the OPC UA plugin are the following:

- **endpoint:** OPC UA endpoint URL.
- **connect_timeout:** maximum time allowed to establish a connection to the endpoint.
- **request_timeout:** maximum time allowed for a request over a stablished connection.
- **nodes:** node ID configuration. Where:
  - **name:** field name to use in the output.
  - **namespace:** OPC UA namespace of the node.
  - **identifier_type:** OPC UA ID type.
  - **identifier:** OPC UA ID.

5.3.4.3. InfluxDB v2 plugin configuration

The configuration parameters of the InfluxDB v2 plugin are the following:

- **urls:** URL of the InfluxDB cluster nodes.

- **token:** token for authentication.
- **organization:** the name of the organization you wish to write to.
- **bucket:** destination bucket to write into.

5.3.4.4. Telegraf operation

Telegraf agent is executed by means of a batch file (telegraf.bat) called by the python application after the initialization of the OPC UA server. The agent stops its operation only during maintenance or during an internet outage (the agent can store some metrics in its buffer, see Agent configuration).

5.4. Cloud services

In this section, the infrastructure, platforms and software that are hosted by third-party providers and made available to users through the internet are presented.

5.4.1. InfluxDB Cloud

InfluxDB is an open-source time series database. It is written in Go for storage and retrieval of time series data in fields such as operations monitoring, application metrics, Internet of Things, and real-time analytics. InfluxDB Cloud is built as a cloud-native, multi-tenant, elastic scale, serverless platform. Furthermore, InfluxDB Cloud is architected to run on any cloud (Amazon Web Services, Google Cloud Platform and Microsoft Azure).

Table 26 shows the Cloud specification for the current InfluxDB Cloud.

Provider	Location	URL
GCP	Belgium	https://europe-west1-1.gcp.cloud2.influxdata.com/

Table 26. InfluxDB Cloud specification

5.4.1.1. InfluxDB Cloud configuration

The creation of a bucket is mandatory to store the time-series data sent by the agent. All buckets have a retention policy, a duration of time that each data point persist.

Name	Retention Policy	Schema Type	ID
OPCUA	30 days	Implicit	718b85b4aa1d3b23

Table 27. Bucket specification (InfluxDB Cloud)

After the creation of the bucket, an API token shall be generated in order to link the agents and clients with the InfluxDB Cloud. The API token gives permissions for reading/writing the selected buckets.

Name	Type	Bucket	Permissions
Telegraf	Agent	OPCUA	Writing
Python	Client	OPCUA	Reading

Table 28. Custom API token generation parameters

5.4.2. Google Cloud Storage

Google Cloud Storage (GCS) is a RESTful online file storage web service for storing and accessing data on Google Cloud Platform.

5.4.2.1. GCS configuration

A bucket shall be created in order to retrieve and store files. The configuration parameters of the GCS bucket are presented in Table 29.

Name	Location	Default storage class	Control access	Public access
gh_data_cloud	Multi-region (eu)	Standard	Uniform	Not public

Table 29. Bucket specification (Google Cloud Storage)

Within the gh_data_cloud bucket, different folders shall be created in order to organize the files and the data. Table 30 presents the folders and their purpose.

Folder name	Description
Actuators/	Contains a CSV file with the current values that shall be sent to the actuators.
Codes/	Contains all the codes of the cyber domain as backup.
Historical_data/	Contains CSV files of the historical data retrieved from InfluxDB Cloud.
MATLAB/	Contains MATLAB data files.
Sensors/	Contains a CSV file with the current values read by the sensors.

Table 30. Folder hierarchy of the bucket

5.4.3. OpenWeatherMap API

OpenWeatherMap, by means of geographical coordinates (latitude and longitude) and an API key, provides current and forecasted weather data for any location on demand. The plan used is specified in Table 31.

Name	Description	Price plan
Weather	Current weather and forecast	Developer plan

Table 31. Services available on OpenWeatherMap API

The services provided by the developer plan are the following:

- Current weather
- 3-hour forecast for 5 days
- Hourly forecast for 4 days
- Daily forecast for 16 days
- Climatic forecast for 30 days
- Advanced weather maps
- Historical maps

The number of calls is limited to 3000 calls/minute and to 100000000 calls/month.

6. Graphical User Interface

In the present chapter, the GUI (Graphical User Interface) shall be presented. Refer to A9. Python GUI for the code.

6.1. CustomTkinter

The libraries used in the python GUI are gathered in Table 32.

Name	Version	Description
ctkinter	v4.6.3	UI-library based on Tkinter
json	v3.10.7	Allows to work with json files
requests	v2.28.1	HTTP library
numpy	v1.23.0	Adds support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions
opcua	v0.98.13	Python OPC UA/IEC-62541 client and server
solarpy	v0.1.3	Provides a reliable solar radiation model
matplotlib	v3.6.0	Allows the creation of static, animated, and interactive visualizations in Python
pandas	v1.5.0	Supplies data analysis and manipulation tools
storage	v2.0	Allows the access to Google Cloud Storage from python
io	v3.10.7	Provides Python's main facilities for dealing with various types of I/O
PIL	v9.2.0	Adds image processing capabilities to Python
Influxdb	v1.32.0	Contains the Python client library for InfluxDB 2.0
datetime	v3.10.7	Supplies classes for manipulating dates and times

Table 32. Libraries used in the python GUI

The GUI has been developed using CustomTkinter which is a python UI-library based on Tkinter, which provides new, modern and fully customizable widgets. The widgets used in the GUI application are presented in Table 33.

Widget	Icon example	Description
Frame		Frames are containers of other widgets

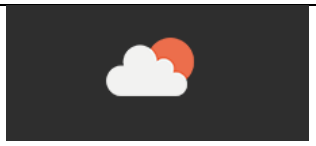
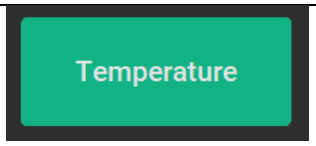
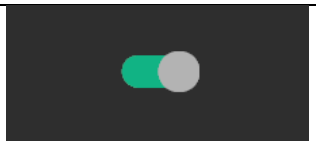
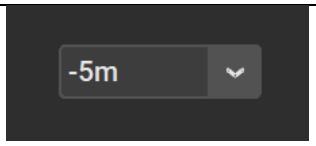
Text label		Text labels are used for displaying static or dynamic text
Image label		Image labels are used for displaying images
Button		Buttons call an associated function when they are pressed
Switch		Switches toggle the value of a Boolean variable
Combo box		Combo boxes are the combination of a drop-down list and a textbox, allowing the user to either type or select a value

Table 33. Basic widgets description

The graphical user interface is made up of two main frames: navigation and panel. The navigation frame is located at the left of the window and it always remains visible. The panel frame is located at the right part of the window, and the different widgets and panels (Current status, Weather forecast and Historical data) are children off this frame. Figure 6.1 shows the size and the position of the frames within the application.

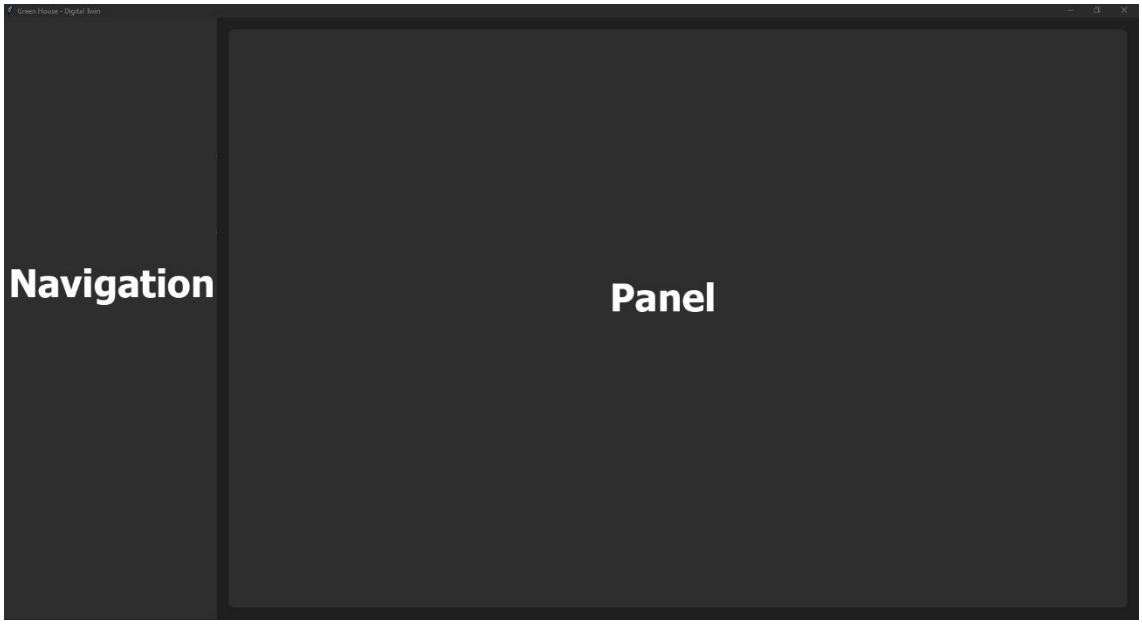


Figure 6.1. Location of the main frames in the GUI application

6.2. Frames

In the present section, the following frames shall be presented:

- **Navigation:** used for navigation among frames
- **Current status:** displays the current values of sensor readouts, manipulated variable instructions and state variable temperature ranges.
- **Weather forecast:** displays current weather conditions and weather forecast for different time periods.
- **Historical data:** retrieves historical data from InfluxDB database.

6.2.1. Navigation

The navigation frame allocates the widgets used for the navigation among panels. As stated before, the frame always remains visible for the user. The list of widgets is shown in Table 34. Figure 6.2 shows the navigation frame containing its widgets.

Button widget	Function
Current Status	Raises Current Status panel
Weather Forecast	Raises Weather Forecast panel
Historical Data	Raises Historical Data panel

Table 34. Button widgets description of the navigation frame

Figure 6.2. Navigation frame containing its widgets

6.2.2. Current status

The Current status panel has the objective to show to the user the current values of the main parameters (sensors and actuators). Furthermore, the panel represents the state variables within the greenhouse graphically. The present panel is read-only intended and the user cannot interact with it.

The widgets within the Current status panel are listed in Table 35 (frames) and in Table 36 (text and image labels). The location of the frames and widgets is shown in Figure 6.3.

Frame widget	**Parent**	**Function**
Sensors	Panel	Allocates widgets related to sensor info
Actuators	Panel	Allocates widgets related to MVs info

Table 35. Frame widgets description of the Current status panel

Label widget	**Parent**	**Function**
Text label (x5)	Sensor	Description labels (static)
Text label (x4)	Sensors	Value indicators
Text label (x5)	Actuators	Description labels (static)
Text label (x4)	Actuators	Value indicators
Image label	Panel	GH widget (Refer to Greenhouse widget)

Table 36. Label widgets description of the Current status panel

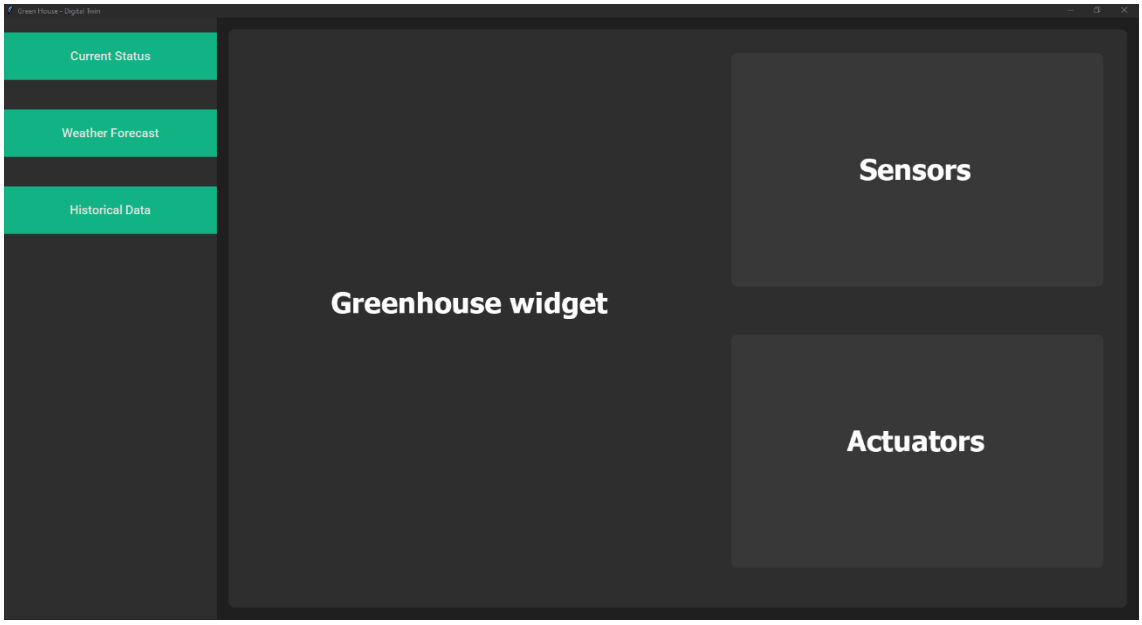


Figure 6.3. Location of the frames and widgets in the Current status panel

The python GUI calls the function *READ(client)* every 5 seconds. The *READ(client)* function operation is the following:

1. The OPC UA client accesses the intended nodes (refer to Table 37)
2. The function reads their values and stores them into global variables
3. Depending on state variable values, the function assigns the corresponding image to each sub-image variable of Table 38
4. The function updates the non-static text labels with the values retrieved from the OPC UA server

After calling *READ(client)* function, the GUI calls *sim()* function which constructs the greenhouse widget. The *sim()* function operation is the following:

1. The function copies the base image of the greenhouse to work on it
2. The sub-image variables of the bottom layer are pasted over the base image applying a mask
3. The sub-image variables of the intermediate layer are pasted over the bottom layer applying a mask
4. The sub-image variables of the top layer are pasted over the intermediate layer applying a mask

Type	**Variable**	**Units**
Sensor data	Ambient air temperature	[°C]
	Outdoor temperature	[°C]
	Solar irradiance	[W/m²]

	Wind speed	[m/s]
Actuator data	Heater/Cooler power	[W]
	Shades extension	[%]
	Window 1 opening	[%]
	Window 2 opening	[%]
States	Soil temperature	[°C]
	Roof air temperature	[°C]
	Glass temperature	[°C]
	PV module temperature	[°C]

Table 37. Variables displayed within "Current status" panel

The Current status panel is shown in Figure 6.4.

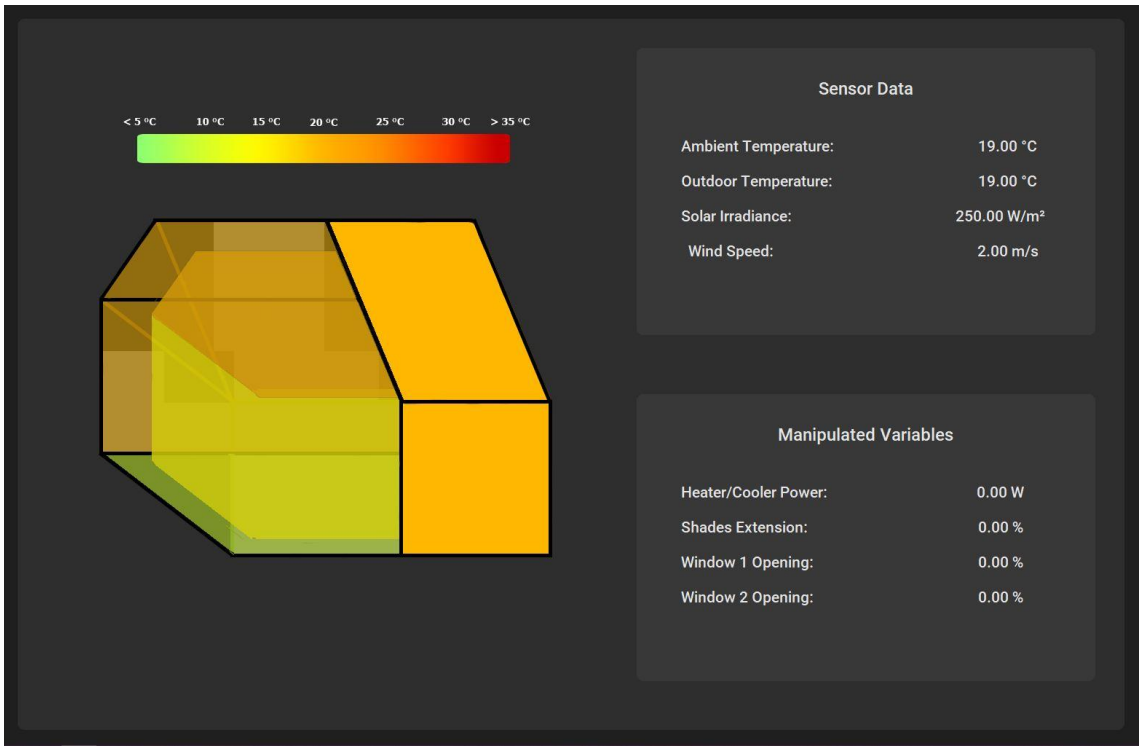


Figure 6.4. Current status panel

6.2.2.1. Greenhouse widget

The purpose of the greenhouse widget is to represent graphically the temperatures of the greenhouse model (see Model). The widget constructs an image which is made up of other six sub-images, shown in Table 38, to represent the current state of the greenhouse. The widget overlays the sub-images

depending on their layer, that is to say, the images corresponding to the bottom layer are the base of the final image, then, the images of the intermediate layer are pasted over the ones of the bottom layer and, finally, the images of the top layer are pasted over the images of the intermediate one.

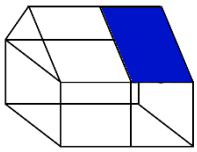
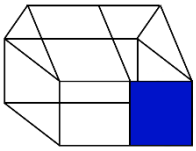
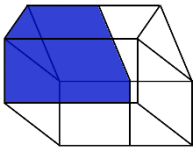
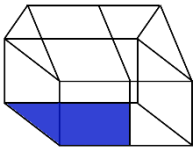
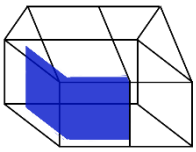
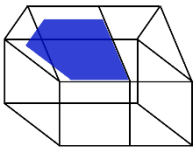
Image	Variable	Layer
	PV module temperature	Top layer
	Glass temperature	Top layer
	Glass temperature	Bottom layer
	Soil temperature	Bottom layer
	Ambient air temperature	Intermediate layer
	Roof air temperature	Intermediate layer

Table 38. Greenhouse widget layer specification

There are 42 different images to build the representation of the greenhouse model. The application assigns a colour to each temperature range (see Table 39). There are seven images for each temperature zone, thus, there are 42 different images to build the representation of the greenhouse model.

Colour image	Hex colour code	Temperature range
	#8EFF71	(-∞, 10) °C
	#CFFF30	[10, 15) °C
	#FFF600	[15, 20) °C
	#FFB700	[20, 25) °C
	#FF7C00	[25, 30) °C
	#FF4000	[30, 35) °C
	#C90000	[35, +∞) °C

Table 39. Greenhouse widget colour specification

Figure 6.5 shows the greenhouse widget representing the following information:

- Ambient air temperature within [15, 20) °C range
- Soil temperature within [10, 15) °C range
- Glass temperature within [20, 25) °C range
- Roof air temperature within [20, 25) °C range
- PV module temperature within [25, 30) °C range

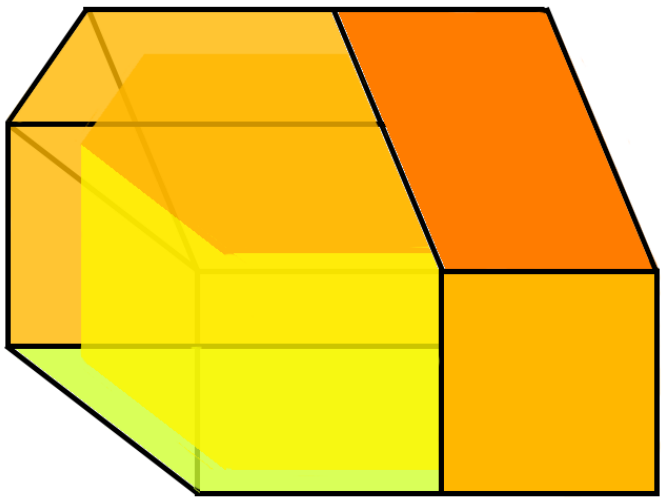


Figure 6.5. Greenhouse widget detail

6.2.3. Weather forecast panel

The Weather forecast has the purpose to inform the user of the weather forecast in Savona (location of the greenhouse). The measured disturbances (outdoor temperature, wind speed and solar irradiance) have a big impact on the plant as well as on the controller. Therefore, it is important for the user to know how the climatic conditions will change in the future.

The widgets within the Weather forecast panel are listed in Table 40 (frames), Table 41 (buttons) and in Table 42 (text and image labels). The location of the frames and widgets is shown in Figure 6.6.

Frame widget	**Parent**	**Function**
Forecast	Panel	Allocates widgets related to weather forecast

Table 40. Frame widgets description of the Weather forecast panel

Button widget	**Parent**	**Function**
Temperature	Panel	Displays TA2 map
Wind Speed	Panel	Displays WND map
Cloudiness	Panel	Displays CL map
Get daily forecast	Panel	Displays daily forecast (8 days)
Get 3-hour forecast	Panel	Displays 3-hour forecast (8 3-hour periods)

Table 41. Button widgets description of the Weather forecast panel

Label widget	**Parent**	**Function**
Text label (x24)	Forecast	Description labels (static)
Text label (x32)	Forecast	Value indicators
Image label (x8)	Forecast	Weather forecast indicators
Image label	Panel	Weather map base
Image label	Panel	Weather map legend

Table 42. Label widgets description of the Weather forecast panel

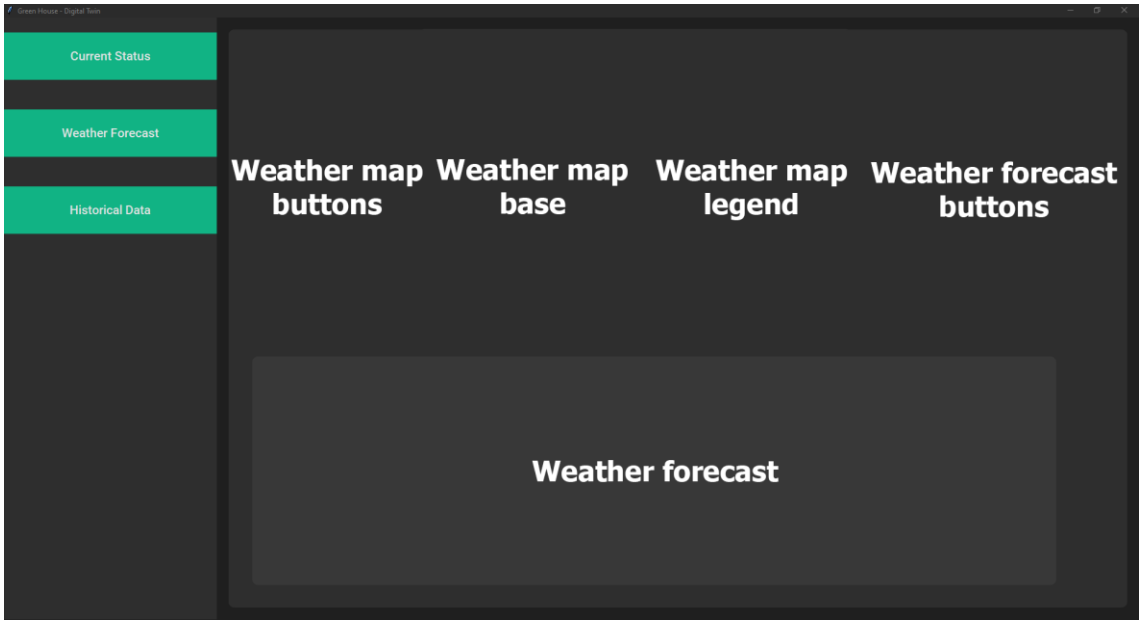


Figure 6.6. Location of the frames and widgets in the Weather forecast panel

The weather map buttons are linked to weather map image labels, and weather forecast buttons are linked to weather forecast frame.

When a weather map button is pressed:

1. The application sends an HTTPS request to OpenWeatherMap API
2. The application receives PNG images of the requested advanced weather maps
3. The weather maps are pasted over the base image applying a mask
4. The weather map legend is updated to fit the corresponding colour palette of the selected weather map

When a weather forecast button is pressed:

1. The application sends an HTTPS request to OpenWeatherMap API
2. The application receives a json file of the requested weather forecast
3. The date time, weather, maximum and minimum temperatures, cloudiness and the wind speed are extracted from the json file
4. The solar irradiance is estimated by means of the cloudiness using solarpy library
5. A weather icon is assigned to each image label depending on its weather variable
6. Non-static text and image labels are updated

Figure 6.7 presents the Weather forecast panel with all the widgets.

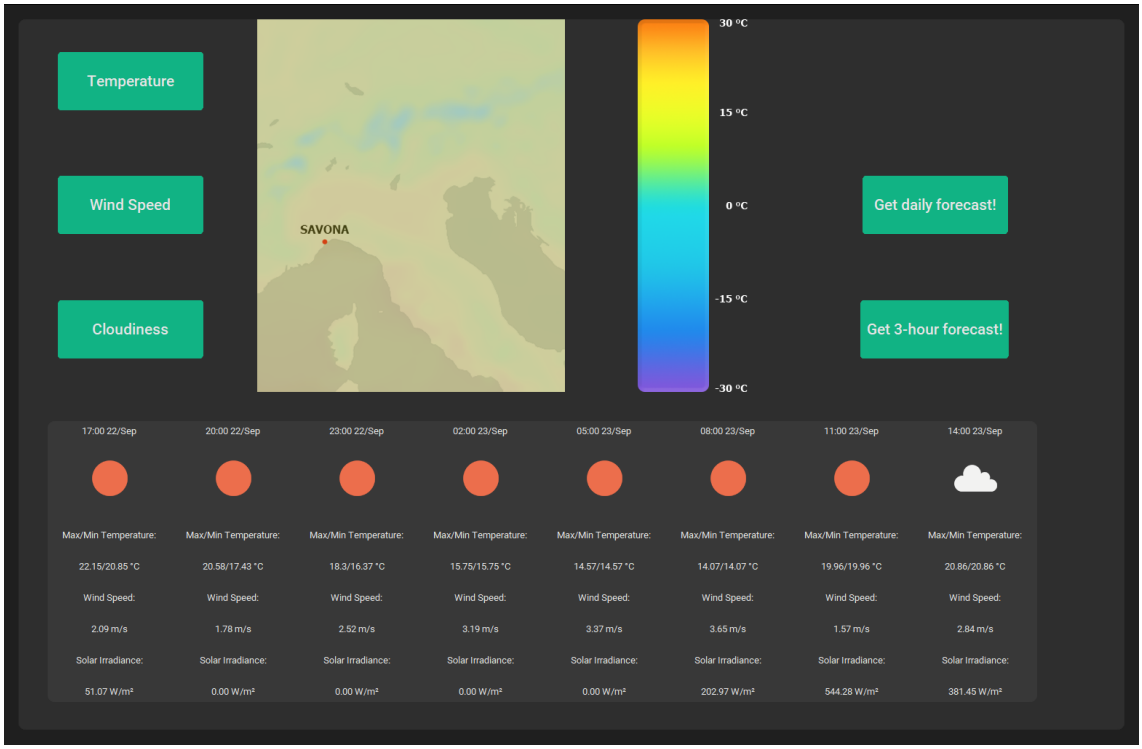


Figure 6.7. Weather forecast panel

6.2.3.1. Current weather maps

The selected weather maps, that are present in the application, are the following:

- **TA2:** Air temperature at a height of 2 meters (shown in Figure 6.8). Measured in °C
- **WND:** Joint display of wind speed (colour) and wind direction (arrows), received by U and V components (shown in Figure 6.9). Measured in m/s
- **CL:** Cloudiness (shown in Figure 6.10). Measured in %

Figure 6.8. TA2 map detail

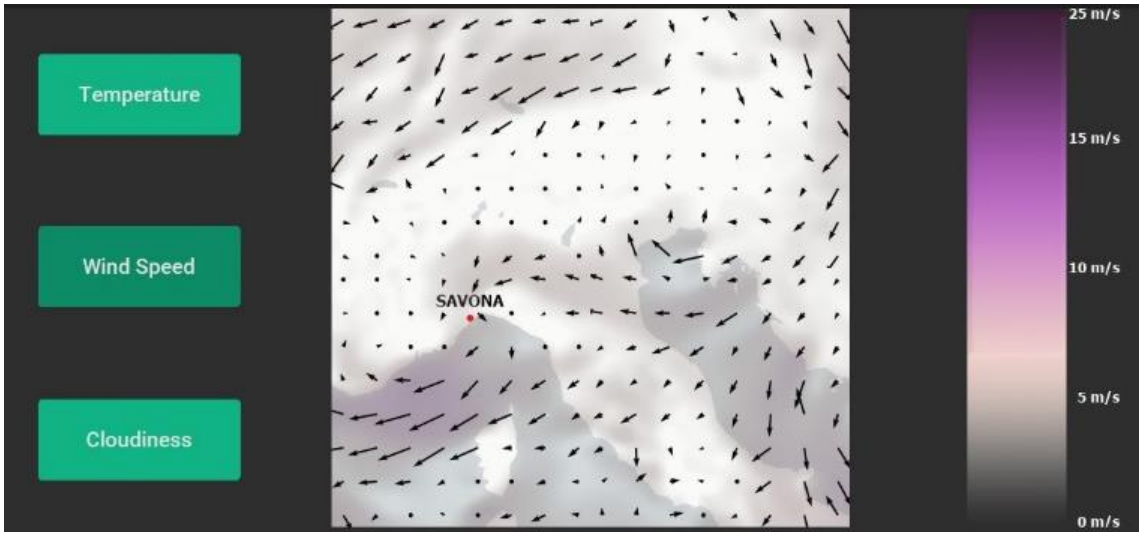


Figure 6.9. WND map detail

Figure 6.10. CL map detail

6.2.3.2. Weather forecast

There are two modes of operations:

- **Daily forecast:** receives the forecast of the present day and the following 7 days.
- **3-hour forecast:** receives the forecast of the current time and the following 21 hours.

The application sends a request to OpenWeatherMap API when the buttons *Get daily forecast* or *Get 3-hour forecast* are pressed. Then, it receives the parameters shown in Table 43 depending the selected mode of operation. The text label widgets, linked to the parameters, are updated every 5 seconds.

A detail of the 3-hour weather forecast is presented in Figure 6.11.

Parameter	**Units**
Date time	N/A
Weather	N/A
Maximum temperature	°C
Minimum temperature	°C
Wind speed	m/s
Solar irradiance	W/m²

Table 43. Weather forecast parameters

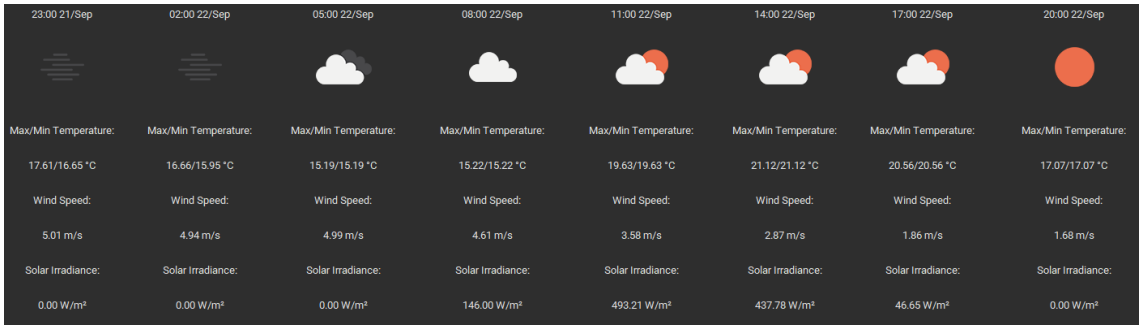


Figure 6.11. 3-hour weather forecast detail

6.2.4. Historical data panel

The Historical data panel has the objective to display the historical data of the digital twin. The on/off switch widgets of the panel activate and deactivate the variables displayed in the plot. By means of a combo box, the user is able to specify the time range of the query.

The widgets within the Historical data panel are listed in Table 44 (frames), Table 45 (switches), in Table 46 (buttons), in Table 47 (labels) and in Table 48 (combo box). The location of the frames and widgets is shown in Figure 6.12.

Frame widget	**Parent**	**Function**
Query	Panel	Allocates switch widgets and time frame
Time	Query	Allocates widgets to specify time range

Table 44. Frame widgets description of the Historical data panel

Switch widget	**Parent**	**Function**
Switches (x12)	Query	Selects the variables to query

Table 45. Switch widgets description of the Historical data panel

Button widget	**Parent**	**Function**
Get query	Panel	Creates the query and sends the request to InfluxDB

Table 46. Button widgets description of the Historical data panel

Label widget	**Parent**	**Function**
Text label	Time	Description label (static)
Image label	Panel	Plot of the queried data

Table 47. Label widgets description of the Historical data panel

Combo widget	**Parent**	**Function**

Combo box	Time	Specifies the time range of the query

Table 48. Combo box widgets description of the Historical data panel

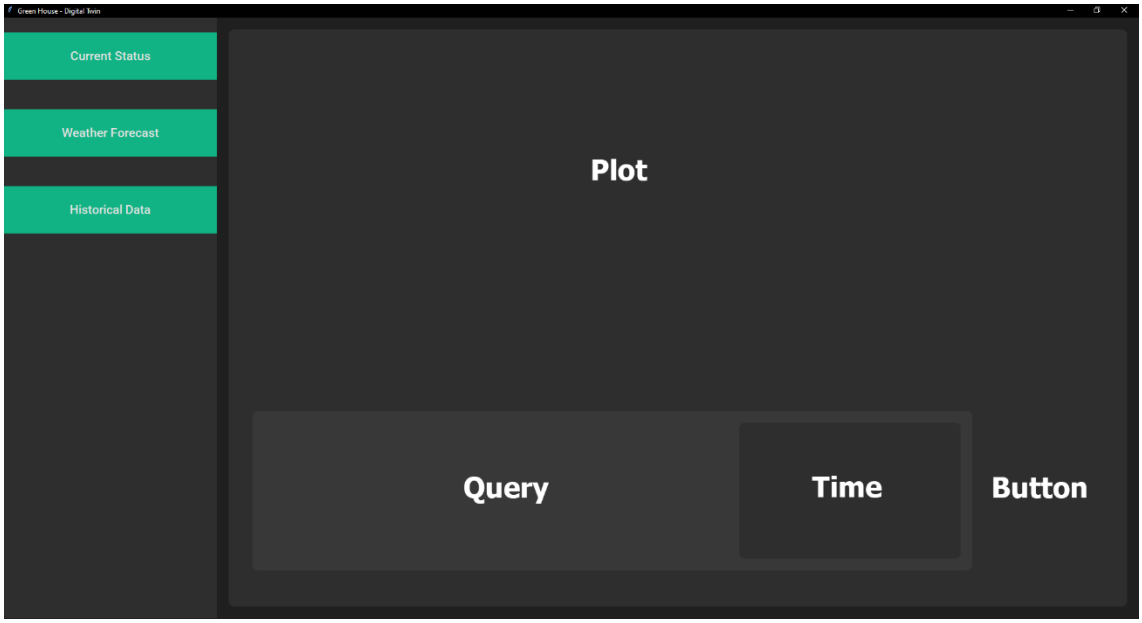


Figure 6.12. Location of the frames and widgets in the Historical data panel

The button widget calls *get_query()* function:

1. The function reads which switches are in ON state
2. The function reads the time range specified in the combo box
3. By means of the combination of switches and time range, the function creates a string query in Flux language
4. The function sends an HTTPS request to InfluxDB Cloud with the string query
5. The data received is processed using pandas library
6. The processed data is plotted using matplotlib.pyplot library
7. The image label is updated with the image of the plot

Figure 6.13 presents the Historical data panel with all its widgets.

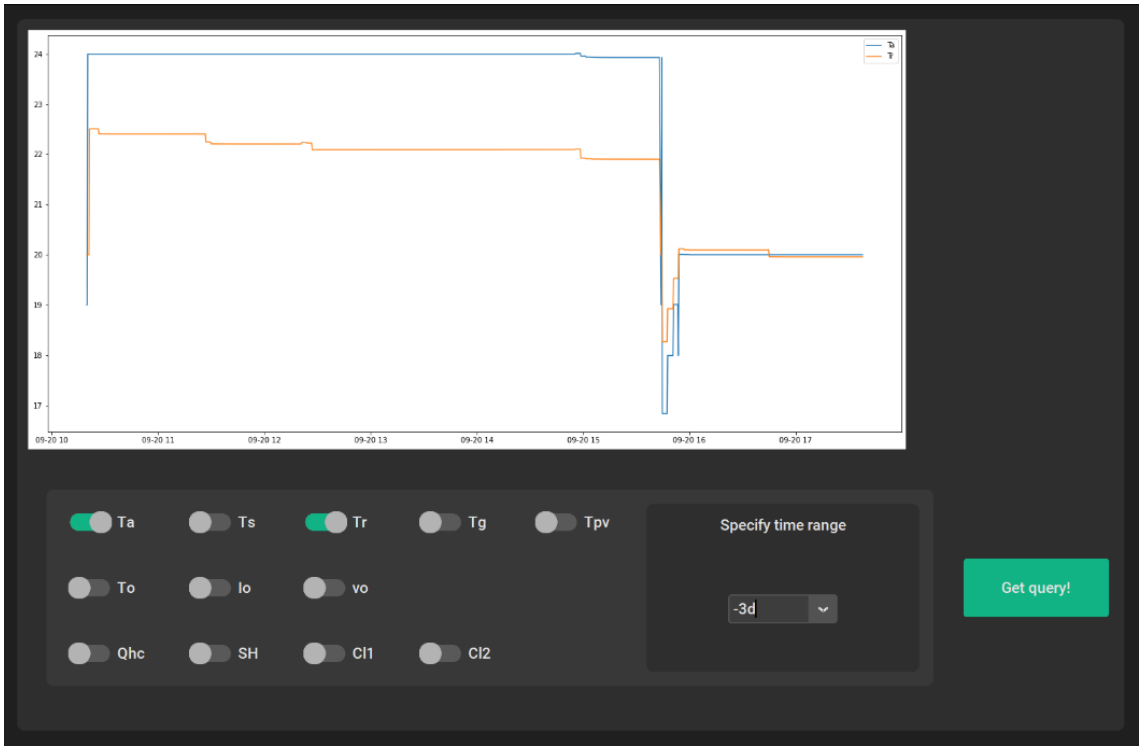


Figure 6.13. Historical data panel

7. Environmental impact analysis

In this chapter the environmental impact analysis of the digital twin is presented. The impact assessment has been developed using openLCA software (v1.11.0).

7.1. LCA study

The presented digital twin pertains exclusively to cyber domain but it must be hosted in a server with internet connection to work appropriately. Therefore, a server and a router shall be included as necessary hardware for the application.

The scope of the present LCA study covers the following:

- Environmental impact of the normal operation of the digital twin during a whole year. Excluding the hardware manufacturing and its transport to the end location.

7.1.1. Inventory analysis

The inventory of the normal operation has been developed by means of ecoinvent (v3.7.1.) LCA database. The processes used in this inventory are the following:

1. Internet access, work, 0.2 Mbit/s |APOS, S - CH
2. Market for electricity, low voltage | APOS, S - IT

ID	Product	Flow property	Unit	Target amount
1	Electricity, LV	Energy	kWh	7008 kWh
2	Internet access	Duration	h	7008 h

Table 49. Inventory of normal operation

7.1.2. Impact assessment

The life cycle impact assessment (LCIA) method used is IMPACTWorld+ (midpoint). The impact categories, as well as the values for the assessed processes are shown in Table 50.

Impact category	Units	Electricity, LV	Internet access	Total
Climate Change Long Term	kg CO_2-eq	$2.84 \cdot 10^3$	$9.77 \cdot 10^0$	$2.85 \cdot 10^3$
Climate Change Short Term	kg CO_2-eq	$3.01 \cdot 10^3$	$1.01 \cdot 10^1$	$3.02 \cdot 10^3$
Fossil and Nuclear Energy Use	MJ deprived	$5.13 \cdot 10^4$	$4.53 \cdot 10^2$	$5.18 \cdot 10^4$

Freshwater Acidification	kg SO_2-eq	$7.50 \cdot 10^{-11}$	$3.06 \cdot 10^{-13}$	$7.53 \cdot 10^{-11}$
Freshwater Ecotoxicity	CTUe	$4.98 \cdot 10^{7}$	$5.65 \cdot 10^{5}$	$5.04 \cdot 10^{7}$
Freshwater Eutrophication	kg PO_4-eq	$7.68 \cdot 10^{-3}$	$4.08 \cdot 10^{-5}$	$7.73 \cdot 10^{-3}$
Human Toxicity Cancer	CTUh	$1.96 \cdot 10^{-4}$	$2.09 \cdot 10^{-6}$	$1.98 \cdot 10^{-4}$
Human Toxicity Non-Cancer	CTUh	$3.24 \cdot 10^{-4}$	$3.98 \cdot 10^{-6}$	$3.28 \cdot 10^{-4}$
Ionizing Radiations	Bq C-14-eq	$3.65 \cdot 10^{4}$	$2.12 \cdot 10^{3}$	$3.86 \cdot 10^{4}$
Land Occupation Biodiversity	m^2 arable land eq	$8.77 \cdot 10^{0}$	$1.20 \cdot 10^{-1}$	$8.89 \cdot 10^{0}$
Land Transformation Biodiversity	m^2 arable land eq	$8.12 \cdot 10^{-1}$	$3.91 \cdot 10^{-3}$	$8.15 \cdot 10^{-1}$
Marine Eutrophication	kg N-eq	$1.15 \cdot 10^{-1}$	$8.04 \cdot 10^{-4}$	$1.15 \cdot 10^{-1}$
Mineral Resources Use	kg deprived	$9.05 \cdot 10^{0}$	$5.15 \cdot 10^{-1}$	$9.56 \cdot 10^{0}$
Ozone Layer Depletion	kg CFC-11-eq	$5.18 \cdot 10^{-4}$	$9.78 \cdot 10^{-7}$	$5.19 \cdot 10^{-4}$
Particulate Matter Formation	kg PM2.5-eq	$7.13 \cdot 10^{-1}$	$4.00 \cdot 10^{-3}$	$7.17 \cdot 10^{-1}$
Photochemical Oxidant Formation	kg NMOC-eq	$6.91 \cdot 10^{0}$	$2.68 \cdot 10^{-2}$	$6.94 \cdot 10^{0}$
Terrestrial Acidification	kg SO_2-eq	$4.73 \cdot 10^{-5}$	$1.96 \cdot 10^{-7}$	$4.75 \cdot 10^{-5}$
Water Scarcity	m^3 world-eq	$2.54 \cdot 10^{3}$	$4.09 \cdot 10^{1}$	$2.58 \cdot 10^{3}$

Table 50. Impact analysis of the digital twin in normal operation

From the relative results, presented in Figure 7.1, we can extract that the environmental impact of the energy consumption of the server is much higher than the environmental impact of the internet access.

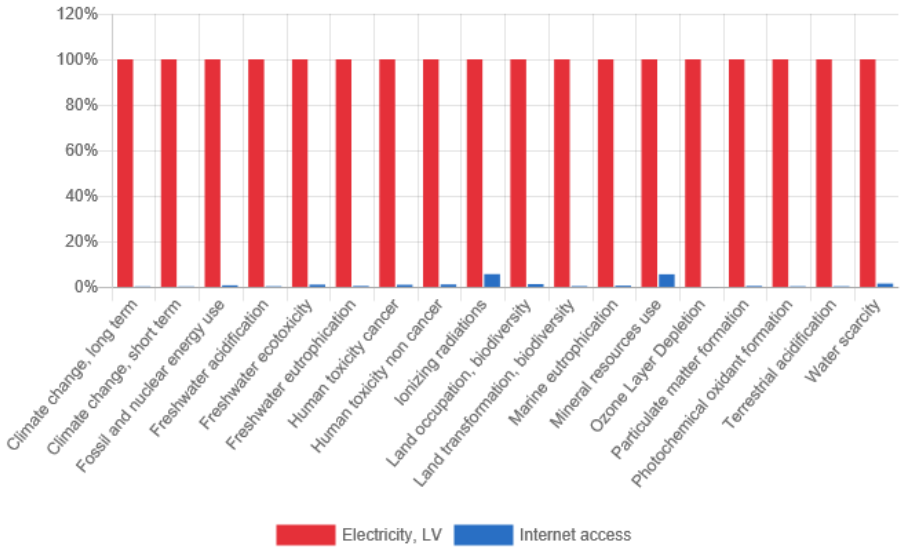


Figure 7.1. LCA relative results

Conclusions

This document has defined a digital twin framework for smart farming purposes using an actual case of study. More specifically, it addressed the three objectives as mentioned in the introduction as follows.

Firstly, the report has defined the design of a MIMO MPC controller for the actual greenhouse located in Savona. The controller has been able to control the ambient temperature of the simulated plant in four different scenarios which were modelled with actual historical data. By means of adding a look-ahead (previewing) feature in the MPC controller, the response anticipates the measured disturbances (weather forecasts) as well as the changes on the setpoints.

Secondly, the document has defined the design of a software architecture framework for the digital twin's cyber domain. The framework comprises the remote autonomous control and the monitoring of the current state of the greenhouse, the retrieving of weather forecast data to know the future state of the plant, and the upload of all data to cloud-oriented services (database and storage repository) to have access to it at any time.

Thirdly, the integration between the controller, the digital twin's cyber domain and the graphical user interface has been successful. The result of the integration is made up of an autonomous predictive digital twin, allowing the user to be aware of the greenhouse status in the present (sensor readouts), the past (historical data) and the future (weather forecasts).

To conclude, the project outlines that the system can be scaled or modified easily due to the modularity of this digital twin framework. Furthermore, the same framework can be used for other smart farming purposes by means of replacing the model, the controller or the GUI application, due to the cyber domain of the digital twin remains invariable.

Economic analysis

The project design has been broken down into working units. These working units have an associated number of worked hours that depend on their necessary completion time and complexity.

Working units	**Number of hours**
Greenhouse model design	160 h
MPC controller design	160 h
Physical domain specification	15 h
Cyber domain design	160 h
Graphical User Interface design	80 h
Digital twin implementation	80 h
Environmental impact analysis	15 h
Preparation of documentation	80 h
TOTAL HOURS:	**750 h**

The labour cost of one technical engineer for the amount of total hours worked, according to the Spanish legislation, is broken down below.

Engineering labour	**Cost per hour**	**Total cost**
Gross salary	12.50 €/h	9375.00 €
Common contingency contribution	2.95 €/h	2212.50 €
Training contribution	0.075 €/h	56.25 €
Unemployment contribution	0.8375 €/h	628.13 €
FOGASA contribution	0.025 €/h	18.75 €
Occupational accident contribution	0.1875 €/h	140.62 €
TOTAL ENGINEERING LABOUR COST:		**12434.25 €**

The paid software required for the design of the project is listed below. Open-source software have been excluded from the list.

Software licenses	**Cost per year**
MATLAB	840.00 €
Symbolic Math Toolbox	420.00 €
Model Predictive Control Toolbox	840.00 €
Control System Toolbox	480.00 €
Industrial Communication Toolbox	520.00 €
TOTAL SOFTWARE COST:	**3100.00 €**

The paid cloud services required for the design of the project are listed below. InfluxDB has been excluded due to the necessary pricing plan is for free.

Cloud service	**Cost per month**	**Total cost**
OpenWeatherMap Developer plan	160.00 €/month	800.00 €
Storage GCP	0.023 €/month	0.12 €
Operative charges GCP	0.05 €/month	0.25 €
TOTAL CLOUD SERVICES PRICE:		**800.37 €**

The total cost of the project is broken down below.

Concept	**Total cost**
Engineering labour	12434.25 €
Software licenses	3100.00 €
Cloud services	800.37 €
TOTAL PROJECT COST:	**16334.62 €**

The economic cost increases due to the software licenses and the OpenWeatherMap Developer plan. The economic analysis comprises the project design costs, excluding the maintenance and operation costs of the digital twin.

The cost associated to MATLAB could be reduced at the expense of using open-source software or designing an MPC controller with python. MATLAB have been selected over other options due to its robustness, the quantity of documentation available and because it is a proved software for control engineering.

The cost associated to OpenWeatherMap could only be reduced at the expense of reducing the features of the digital twin (advanced weather maps, precise forecast).

Bibliography

Boccalatte, A., M. Fossa, and R. Sacile. “Modeling, Design and Construction of a Zero-Energy PV Greenhouse for Applications in Mediterranean Climates.” *Thermal Science and Engineering Progress*, 2021: 101046.

Chaux, Jesus David, David Sanchez-Londono, and Giacomo Barbieri. “A DIgital Twin Architecture to Optimize Productivity within Controlled Environment Agriculture.” *Applied Sciences* 11, no. 19 (2021): 8875.

De Jong, T. *Natural Ventilation of Large Multi-span Greenhouse.* Wageningen: Wageningen Agricultural University, 1990.

Gomes, Rafael, Gilberto Souza, Rodrigo Filev Maia, Carlos Kamienski, Juha-Pekka Soininen, and Plinio Thomaz Aquino-Jr. “A digital twin for smart farming.” *IEEE GLobal Humanitarian Technology Conference (GHTC).* Seattle: IEEE, 2019. 1-4.

Hevner, Alan R., T. March Salvatore, Park Jinsoo, and Ram Sudha. “Design Science in Information Systems Research.” *MIS Quarterly*, 2004: 75-105.

Jang-Singh, Melanie, Kathryn Leeming, Ruchi Choudhary, and Mark Girolami. “Digital twin of an urban-integrated hydroponic farm.” *Data-Centric Engineering*, 2020: e20.

MathWorks Inc. *Tune Weights.* 2022. https://es.mathworks.com/help/mpc/ug/tuning-weights.html.

Maurer, Tim, and Robert Morgus. “SPACE. In Compilation of Existing Cybersecurity and Information Security Related Definitions.” *New America*, 2014: 18-24.

OPC UA Foundation. *What is OPC?* 2021. https://opcfoundation.org/about/what-is-opc/.

Pylianidis, Christos, Sjoukje Osinga, and Ioannis N. Athanasiadis. “Introducing digital twins to agriculture.” *Computers and Electronics in Agriculture*, 2021: 105942.

Tekinerdogan, Bedir, y Cor Verdouw. «Systems Architecture Design Pattern Catalog for Developing Digital Twins.» *Sensors*, 2020: 5103.

The MathWorks Inc. “Choosing Sample Times and Horizons.” 2022. https://it.mathworks.com/help/mpc/ug/choosing-sample-time-and-horizons.html.

United Nations. *Goal 2: Zero Hunger.* 2019. https://www.un.org/sustainabledevelopment/hunger/.

Van Straten, Gerrit, Gerard Van Willigenburg, Eldert Van Henten, and Rachel Van Ooteghem. *Optimmal Control of Greenhouse Cultivation.* Boca Raton: CRC Press, 2011.

Verdouw, Cor, Bedir Tekinerdogan, and Sjaak Wolfert. "Digital twins in smart farming." *Agricultural Systems*, 2021: 1-19.

Wang, Liuping. *Model Predictive Control System Design and Implementation Using MATLAB®.* London: Springer, 2009.

World Weather Online. *Savona Historical Weather.* 2022. https://www.worldweatheronline.com/savona-weather-history/liguria/it.aspx (accessed 2022).

Young, Hugh D., and Francis Weston Sears. *University Physics.* Addison Wesley, 1992.

Annex A

The annex A comprises all the codes used in the DT application.

A1. MATLAB greenhouse model

```
syms Ta Ts Tr Tg Tpv            % symbolic output variables
syms Qhc SH Cl1 Cl2 To Io vo    % symbolic input variables

%% Parameters
% Density (kg/m^3)
ro_a = 1.29;        % density of ambient air
ro_r = 1.29;        % density of roof air
ro_g = 2500;        % density of glass
ro_pv = 1069;       % density of PV roof
ro_s = 1600;        % density of soil
% Specific heat capacity (J/kg·K)
cp_a = 1000;        % cp of ambient air
cp_r = 1000;        % cp of roof air
cp_g = 840;         % cp of glass
cp_pv = 800;        % cp of PV roof
cp_s = 800;         % cp of soil
% Area (m^2)
Aa = 25;            % area of ambient air
Ar = Aa;            % area of roof air
Ag = 12.7;          % area of glass
Apv = 22;           % area of PV roof
As = 25;            % area of soil
% Volume (m^3)
Va = 50;            % volume of ambient air
Vr = 27.94;         % volume of roof air
Vg = 0.203;         % volume of glass
Vpv = 0.088;        % volume of PV roof
Vs = 12.5;          % volume of soil
% Heat transfer coefficient (W/K·m^2)
aph_rg = 1.1;       % aph between roof and glass
aph_as = 1;
% Shortwave radiation absorption coefficient (-)
eta_g = 0.56*(1-SH);   % eta of glass
eta_pv = 0.85;          % eta of PV roof
% Thermal conductivity (W/m·K)
l_g = 0.9;          % l of glass
l_pv = 148;         % l of PV roof
% Thickness (m)
dg = 0.016;         % thickness of glass
dpv = 0.004;        % thicknes of PV roof
% Air exchange rate (m/s)
v_ar = 0.05;        % air exchange rate between ambient and roof air
% Acceleration (m/s^2)
g = 9.81;           % gravity
% Discharge coeffient (-)
cw = 0.6;           % Discharge coeffient
```

```
%% Ventilation model
% Ventilation parameters
nwin = 2;           % number of windows
wwin = 0.5;         % width of the windows
hwin = 0.5;         % height of the windows
c3l = 1.21774e-2;   % coefficient 3 lee-side
c2l = 3.77646e-2;   % coefficient 2 lee-side
c1l = 4.71176e-2;   % coefficient 1 lee-side
c0l = 1.26730e-4;   % coefficient 0 lee-side
c3w = 2.42856e-2;   % coefficient 3 wind-side
c2w = 4.48621e-2;   % coefficient 2 wind-side
c1w = 7.08828e-2;   % coefficient 1 wind-side
c0w = 5.91362e-4;   % coefficient 0 wind-side
flsd = c3l*Cl1^3 - c2l*Cl1^2 + c1l*Cl1 + c0l;   % ventilation func lee-side
fwsd = c3w*Cl2^3 - c2w*Cl2^2 + c1w*Cl2 + c0w;   % ventilation func wind-side
% Airflow (m^3/s)
Far = v_ar*Aa;           % volume flow exchange between ambient and roof airs
Fwind = nwin*(flsd+fwsd)*vo*wwin*hwin;              % wind-induced airflow
F_tlsd = nwin*(cw/3)*wwin*sqrt(g*(abs(To-Tr)/To));  % lee-side airflow
F_twsd = nwin*(cw/3)*wwin*sqrt(g*(abs(To-Tr)/To));  % wind-side airflow
Fwin = sqrt((Fwind)^2+(F_tlsd+F_twsd)^2);           % airflow though windows

%% Thermal model
Qrg = Ag*aph_rg*(Tr-Tg);        % heat transfer between roof air and glass
Qrpv = Apv*aph_rg*(Tr-Tpv);     % heat transfer between roof air and PV roof
Qas = As*aph_as*(Ta-Ts);        % heat transfer between ambient air and soil
Qar = ro_a*cp_a*Far*(Ta-Tr);    % heat transfer between ambient air and roof
Qro = ro_a*cp_a*Fwin*(Tr-To);   % heat transfer between roof air and outside
Qradg = Ag*eta_g*Io;            % heat absorbed by glass from SW radiation
Qradpv = Apv*eta_pv*Io;         % heat absorbed by PV roof from SW radiation
Qgo = Ag*(l_g/dg)*(Tg-To);      % heat transfer between glass and outside
Qpvo = Apv*(l_pv/dpv)*(Tpv-To); % heat transfer between PV roof and outside

%% State variables
dTa = (-Qar -Qas +Qhc)/(ro_a*cp_a*Va);              % Ta state-function
dTs = (+Qas)/(ro_s*cp_s*Vs);                        % Ts state-function
dTr = (+Qar -Qro -Qrg -Qrpv)/(ro_r*cp_r*Vr);        % Tr state-function
dTg = (+Qradg + Qrg -Qgo)/(ro_g*cp_g*Vg);           % Tg state-function
dTpv = (+Qradpv + Qrpv -Qpvo)/(ro_pv*cp_pv*Vpv);    % Tpv state-function

%% State space model
eqs = [dTa;dTs;dTr;dTg;dTpv];               % state equations
nominal_state = [20,10,20,20,20,1,1,25,5];  % nominal-states
nominal_input = [0,0,1,1,20,21,5,200];      % nominal inputs

% Continuous state space model
Ac = double(subs(jacobian(eqs, [Ta Ts Tr Tg Tpv]),[Ta Ts Tr Tg Tpv Cl1 Cl2 To
vo],nominal_state));          % A matrix
Bc = double(subs(jacobian(eqs, [Qhc SH Cl1 Cl2 To vo Io]),[Qhc SH Cl1 Cl2 Tr To vo
Io],nominal_input));     % B matrix
Cc = eye(size(Ac));
% C matrix
Dc = zeros(5,7);
% D matrix
```

```
plant = ss(Ac,Bc,Cc,0);
```

A2. MATLAB MPC controller

```
% set names
plant.InputName = {'Qhc','SH','Cl1','Cl2','To','Io','vo'};      % set names of
input variables
plant.OutputName = {'Ta','Ts','Tr','Tg','Tpv'};                 % set names of
output variables
plant.StateName = {'dTa','dTs','dTr','dTg','dTpv'};             % set names of
state variables

% set units
plant.InputUnit = {'W','%','%','%','°C','W/m','m/s'};           % set units of
input variables
plant.OutputUnit = {'°C','°C','°C','°C','°C'};                  % set units of
output variables
plant.StateUnit = {'°C/s','°C/s','°C/s','°C/s','°C/s'};         % set units of
state variables

% Assign I/O signals to different MPC categories
plant = setmpcsignals(plant,'MV',[1 2 3 4],'MD',[5 6 7],'MO',1,'UO',[2 3 4 5]);

% Create controller
old_status = mpcverbosity('off');
ts = 180;                                   % sample time

%% MPC object definition
mpcobj = mpc(plant,ts);
mpcobj.PredictionHorizon = 50;         % set prediction horizon
mpcobj.ControlHorizon = 5;             % set control horizon

mpcobj.Model.Nominal.U = [0;0.5;0.5;0.5;18;200;3];      % set nominal MV values
Spring
%mpcobj.Model.Nominal.U = [0;0.5;0.5;0.5;13;200;3];     % set nominal MV values
Autumn
%mpcobj.Model.Nominal.U = [0;0.5;0.5;0.5;13;200;3];     % set nominal MV values
Winter
%mpcobj.Model.Nominal.U = [0;0.5;0.5;0.5;25;350;3];     % set nominal MV values
Summer
mpcobj.Model.Nominal.Y = [23;10;20;20;20];              % set nominal output
values Spring
%mpcobj.Model.Nominal.Y = [19;10;20;20;20];             % set nominal output
values Autumn
%mpcobj.Model.Nominal.Y = [19;10;20;20;12];             % set nominal output
values Winter
%mpcobj.Model.Nominal.Y = [19;10;20;20;20];             % set nominal output
values Summer

mpcobj.MV(1).ScaleFactor = 200;         % set Qhc scale factor
mpcobj.MV(3).ScaleFactor = 100;         % set Cl1 scale factor
mpcobj.MV(4).ScaleFactor = 100;         % set Cl2 scale factor
mpcobj.DV(1).ScaleFactor = 10;          % set To scale factor
mpcobj.DV(2).ScaleFactor = 600;         % set Io scale factor
mpcobj.DV(3).ScaleFactor = 6;           % set vo scale factor
```

```
mpcobj.OV(4).ScaleFactor = 0.2;         % set Tg scale factor
mpcobj.OV(5).ScaleFactor = 0.001;       % set Tpv scale factor

mpcobj.MV(1).Min = -5000;               % set Qhc min
mpcobj.MV(1).Max = 5000;                % set Qhc max
mpcobj.MV(1).RateMin = -200;            % set inc Qhc min
mpcobj.MV(1).RateMax = 200;             % set inc Qhc max
mpcobj.MV(2).Min = 0;                   % set SH min
mpcobj.MV(2).Max = 1;                   % set SH max
mpcobj.MV(2).RateMin = -0.1;            % set inc SH min
mpcobj.MV(2).RateMax = 0.1;             % set inc SH max
mpcobj.MV(3).Min = 0;                   % set Cl1 min
mpcobj.MV(3).Max = 1;                   % set Cl1 max
mpcobj.MV(3).RateMin = -0.5;            % set inc Cl1 min
mpcobj.MV(3).RateMax = 0.5;             % set inc Cl1 max
mpcobj.MV(4).Min = 0;                   % set Cl2 min
mpcobj.MV(4).Max = 1;                   % set Cl2 max
mpcobj.MV(4).RateMin = -0.5;            % set inc Cl2 min
mpcobj.MV(4).RateMax = 0.5;             % set inc Cl2 max

mpcobj.MV(1).RateMinECR = 1;            % set ECR  inc Qhc min
mpcobj.MV(1).RateMaxECR = 1;            % set ECR  inc Qhc max
mpcobj.MV(2).RateMinECR = 1;            % set ECR  inc SH min
mpcobj.MV(2).RateMaxECR = 1;            % set ECR  inc SH max
mpcobj.MV(3).RateMinECR = 0.1;            % set ECR  inc Cl1 min
mpcobj.MV(3).RateMaxECR = 0.1;            % set ECR  inc Cl1 max
mpcobj.MV(4).RateMinECR = 0.1;            % set ECR  inc Cl2 min
mpcobj.MV(4).RateMaxECR = 0.1;            % set ECR  inc Cl2 max

mpcobj.Weights.MV = [0 0 0 0];              % set MV weights
mpcobj.Weights.MVRate = [0.1 0.1 0.1 0.1];  % set MV rate weights
mpcobj.Weights.OV = [1000 0.1 0.1 0.1 0.1]; % set OV weights
mpcobj.Weights.ECR = 100000;                % set ECR weight

options = mpcsimopt();
options.Constraints = 'on';
options.RefLookAhead = 'on';
options.MDLookAhead = 'on';
options.OpenLoop = 'off';
```

A3. MATLAB OPC UA client

```
%% OPC UA Client
uaClient = opcua("192.168.1.69",4840);                % OPC UA endpoint
uaClient.connect()                                  % connection client-server

%% Node definition
NodeList = getNamespace(uaClient);                    % Node List generation

Node_u1 = NodeList(1,2).Children(1,1);                % Node = Qhc
Node_u2 = NodeList(1,2).Children(1,2);                % Node = SH
Node_u3 = NodeList(1,2).Children(1,3);                % Node = Cl1
Node_u4 = NodeList(1,2).Children(1,4);                % Node = Cl2
Node_u = [Node_u1;Node_u2;Node_u3;Node_u4];           % MV vector Node
```

```
Node_x1 = NodeList(1,3).Children(1,1);                  % Node = Ta
Node_x2 = NodeList(1,3).Children(1,2);                  % Node = Ts
Node_x3 = NodeList(1,3).Children(1,3);                  % Node = Tr
Node_x4 = NodeList(1,3).Children(1,4);                  % Node = Tg
Node_x5 = NodeList(1,3).Children(1,5);                  % Node = Tpv
Node_x = [Node_x1;Node_x2;Node_x3;Node_x4;Node_x5];     % States vector Node

Node_MD1 = NodeList(1,5).Children(1,1);                 % Node = To
Node_MD2 = NodeList(1,5).Children(1,2);                 % Node = Io
Node_MD3 = NodeList(1,5).Children(1,3);                 % Node = vo
Node_MD = [Node_MD1;Node_MD2;Node_MD3];                 % MD vector Node

Node_vec1 = NodeList(1,6).Children(1,1);                % Node = To forecast
Node_vec2 = NodeList(1,6).Children(1,2);                % Node = Io forecast
Node_vec3 = NodeList(1,6).Children(1,3);                % Node = vo forecast
Node_vec = [Node_vec1;Node_vec2;Node_vec3];          % Forecast vector Node

Node_ref = NodeList(1,4).Children;                      % Node = Reference

%% Load plant and MPC
load('GH.mat');                                         % Load required data
old_status = mpcverbosity('off');
```

A4. MATLAB initialization code

```
r1 = readValue(uaClient,Node_ref);      % Read reference value
r2 = [10*ones(100,1)];
r3 = [20*ones(100,1)];
r4 = [20*ones(100,1)];
r5 = [20*ones(100,1)];
r = [r1' r2 r3 r4 r5];                  % References [Ta Ts Tr Tg Tpv]

v1 = readValue(uaClient,Node_vec1);     % Read To forecast
v2 = readValue(uaClient,Node_vec2);     % Read Io forecast
v3 = readValue(uaClient,Node_vec3);     % Read vo forecast
v = [v1' v2' v3'];                      % MDs [To Io vo]

xmpc = mpcstate(mpcobj);                % initilization of MPC states [0 0 0 0 0]

save('save.mat','xmpc')                 % save xmpc
save('references.mat','r')              % save references
save('MDs.mat','v')                     % save MDs

uaClient.disconnect()
```

A5. MATLAB loop code

```
load('save.mat');                           % Load MPC state (xmpx)
load('references.mat');                     % Load references
load('MDs.mat');                            % Load MDs

x = cell2mat(readValue(uaClient,Node_x));   % Read states
y = plant.C*x;                              % Update outputs
```

```
% Calculate control action
u = mpcmove(mpcobj,xmpc,y(1),r(2:51,:),v(1:51,:));
% Send U values to OPC UA
writeValue(uaClient,Node_u,{u(1),u(2),u(3),u(4)});
% Update States estimation
x = [19;10;20;20;20] + xmpc.Plant;
% Send States values to OPC UA
writeValue(uaClient,Node_x,{x(1),x(2),x(3),x(4),x(5)});

r = circshift(r,-1);        % Shift references
v = circshift(v,-1);        % Shift MDs

save('save.mat','xmpc')     % Save MPC state
save('references.mat','r')  % Save references
save('MDs.mat','v');        % Save MDs

uaClient.disconnect()
```

A6. MATLAB update code

```
r1 = readValue(uaClient,Node_ref);      % Read reference value
r2 = [10*ones(100,1)];
r3 = [20*ones(100,1)];
r4 = [20*ones(100,1)];
r5 = [20*ones(100,1)];
r = [r1' r2 r3 r4 r5];                  % References [Ta Ts Tr Tg Tpv]

v1 = readValue(uaClient,Node_vec1);     % Read To forecast
v2 = readValue(uaClient,Node_vec2);     % Read Io forecast
v3 = readValue(uaClient,Node_vec3);     % Read vo forecast
v = [v1' v2' v3'];                      % MDs [To Io vo]

save('references.mat','r')              % save references
save('MDs.mat','v')                     % save MDs

uaClient.disconnect()
```

A7. Python OPC UA server

```
from opcua import Server

ref = [24.0]*100
v1 = [17.0]*100
v2 = [250.0]*100
v3 = [2.5]*100

server = Server()

url = "opc.tcp://192.168.1.69:4840"
server.set_endpoint(url)

name = "OPCUA_SIMULATION_SERVER"
addspace = server.register_namespace(name)
```

```
node = server.get_objects_node()

### MANIPULATED VARIABLES ###
MV = node.add_object(addspace, "MVs")

Qhc = MV.add_variable(addspace, "Qhc", 0.0)
Sh = MV.add_variable(addspace, "SH", 0.0)
Cl1 = MV.add_variable(addspace, "Cl1", 0.0)
Cl2 = MV.add_variable(addspace, "Cl2", 0.0)

Qhc.set_writable()
Sh.set_writable()
Cl1.set_writable()
Cl2.set_writable()

### STATES ###
States = node.add_object(addspace, "States")

Ta = States.add_variable(addspace, "Ta", 19.0)
Ts = States.add_variable(addspace, "Ts", 10.0)
Tr = States.add_variable(addspace, "Tr", 20.0)
Tg = States.add_variable(addspace, "Tg", 20.0)
Tpv = States.add_variable(addspace, "Tpv", 20.0)

Ta.set_writable()
Ts.set_writable()
Tr.set_writable()
Tg.set_writable()
Tpv.set_writable()

### REFERENCES ###
References = node.add_object(addspace, "Refs")

r1 = References.add_variable(addspace, "r1", ref)
r1.set_writable()

### MEASURED DISTURBANCES ###
MD = node.add_object(addspace, "MDs")

To = MD.add_variable(addspace, "To", 19.0)
Io = MD.add_variable(addspace, "Io", 250.0)
vo = MD.add_variable(addspace, "vo", 2.0)

To.set_writable()
Io.set_writable()
vo.set_writable()

### VECTORS ###
Vectors = node.add_object(addspace, "Vectors")

To_vec = Vectors.add_variable(addspace, "To_vec", v1)
Io_vec = Vectors.add_variable(addspace, "Io_vec", v2)
vo_vec = Vectors.add_variable(addspace, "vo_vec", v3)

To_vec.set_writable()
```

```
Io_vec.set_writable()
vo_vec.set_writable()


### FLAGS ###
Flags = node.add_object(addspace, "Flags")

Flag1 = Flags.add_variable(addspace, "Flag1", False)
Flag2 = Flags.add_variable(addspace, "Flag2", False)
Flag3 = Flags.add_variable(addspace, "Flag3", False)

Flag1.set_writable()
Flag2.set_writable()
Flag3.set_writable()

try:
    server.start()
except:
    print("ERROR: Server cannot run!")
```

A8. Python intelligence layer

```
import csv, json, requests, time, schedule
import numpy as np
from google.cloud import storage
from opcua import Client
from solarpy import irradiance_on_plane
from datetime import datetime
from subprocess import Popen

bucket_name = "gh_data_cloud"
blob_sensor = "Sensors/sensors.csv"
blob_actuator = "Actuators/actuators.csv"
path_sensor = "C:\\Christian\\Personal\\Erasmus\\TFM\\Codes\\Python\\sensors.csv"
path_actuator =
"C:\\Christian\\Personal\\Erasmus\\TFM\\Codes\\Python\\actuators.csv"

url_forecast =
"https://pro.openweathermap.org/data/2.5/forecast/hourly?lat=44.29&lon=8.48&appid=
3f7d5cda9bddd63b2f1a73456fb51fe0&units=metric"
vnorm = np.array([0, 0, -1])  # plane pointing zenith
h = 0  # sea-level
lat = 44.299999     #latitude of Savona

opc_url = "opc.tcp://192.168.1.69:4840"
client = Client(opc_url)


def sensor_data(bucket_name, source_blob_name, destination_file_name):
    global Ta, To, Io, vo

    storage_client = storage.Client()
    bucket = storage_client.bucket(bucket_name)
    blob = bucket.blob(source_blob_name)
    blob.download_to_filename(destination_file_name)
```

```
    file = open('sensors.csv')
    csvreader = csv.reader(file)
    rows = []
    for row in csvreader:
        rows.append(row)
    Ta = float(rows[1][0])
    To = float(rows[1][1])
    Io = float(rows[1][2])
    vo = float(rows[1][3])

def actuator_data(bucket_name, source_blob_name,
location_file_actuator,Qhc,SH,Cl1,Cl2):
    header = ['Qhc','SH','Cl1','Cl2']
    data = [str(Qhc),str(SH),str(Cl1),str(Cl2)]

    with open('actuators.csv','w', encoding='UTF8',newline='') as f:
        writer = csv.writer(f)
        writer.writerow(header)
        writer.writerow(data)

    storage_client = storage.Client()
    bucket = storage_client.bucket(bucket_name)
    blob = bucket.blob(source_blob_name)
    blob.upload_from_filename(location_file_actuator)

def OPCUA_client_Input(client,Ta,To,Io,vo,vecTo,vecIo,vecvo):
    #root = Client.get_objects_node()

    node_Ta = client.get_node("ns=2;i=7")
    node_To = client.get_node("ns=2;i=15")
    node_Io = client.get_node("ns=2;i=16")
    node_vo = client.get_node("ns=2;i=17")
    node_vecTo = client.get_node("ns=2;i=19")
    node_vecIo = client.get_node("ns=2;i=20")
    node_vecvo = client.get_node("ns=2;i=21")

    try:
        client.connect()

        node_Ta.set_value(Ta)
        node_To.set_value(To)
        node_Io.set_value(Io)
        node_vo.set_value(vo)
        node_vecTo.set_value(vecTo)
        node_vecvo.set_value(vecvo)
        node_vecIo.set_value(vecIo)

        client.disconnect()
    except:
        print("Could not connect to server!1")

def OPCUA_client_Output(client):
    #root = Client.get_objects_node()
    node_Qhc = client.get_node("ns=2;i=2")
    node_SH = client.get_node("ns=2;i=3")
```

```
    node_Cl1 = client.get_node("ns=2;i=4")
    node_Cl2 = client.get_node("ns=2;i=5")

    global Qhc, SH, Cl1, Cl2

    try:
        client.connect()

        Qhc = node_Qhc.get_value()
        SH = node_SH.get_value()
        Cl1 = node_Cl1.get_value()
        Cl2 = node_Cl2.get_value()

        client.disconnect()
    except:
        print("Could not connect to server!")

def weather_forecast(url_forecast,lat,h):
    global fore_temp
    global fore_wind
    global fore_solar

    fore_temp = [1]*100
    fore_wind = [1]*100
    fore_solar = [1]*100

    temp = []
    wind = []
    solar = []
    ts = []
    dt = []

    response = requests.get(url_forecast)
    forecast = response.json()

    for i in range(0,5,1):
        temp.append(forecast['list'][i]['main']['temp'])
        wind.append(forecast['list'][i]['wind']['speed'])
        ts.append(forecast['list'][i]['dt'])
        dt.append(datetime.fromtimestamp(ts[i]))
        solar.append(float(irradiance_on_plane(vnorm, h, dt[i], lat)))

    for i in range(0,100,1):
        if i < 20:
            fore_temp[i] = temp[0]
            fore_wind[i] = wind[0]
            fore_solar[i] = solar[0]
        elif (i > 19)and(i < 40):
            fore_temp[i] = temp[1]
            fore_wind[i] = wind[1]
            fore_solar[i] = solar[1]
        elif (i > 39)and(i < 60):
            fore_temp[i] = temp[2]
            fore_wind[i] = wind[2]
            fore_solar[i] = solar[2]
        elif (i > 59)and(i < 80):
```

```
            fore_temp[i] = temp[3]
            fore_wind[i] = wind[3]
            fore_solar[i] = solar[3]
        else:
            fore_temp[i] = temp[4]
            fore_wind[i] = wind[4]
            fore_solar[i] = solar[4]



counter = 0

sensor_data(bucket_name,blob_sensor,path_sensor)
weather_forecast(url_forecast,lat,h)
OPCUA_client_Input(client, Ta, To, Io, vo, fore_temp, fore_solar, fore_wind)
init = Popen("init.bat", cwd=r"C:\Christian\Personal\Erasmus\TFM\Codes\Python")
time.sleep(15)
OPCUA_client_Output(client)
OPCUA_client_simulation(client)
actuator_data(bucket_name, blob_actuator, path_actuator, Qhc, SH, Cl1, Cl2)
simulation_data(bucket_name, blob_sensor, path_sensor, Ta, To, Io, vo)
print("Initilization DONE")

while True:
    if counter == 10:

        counter = 0
        print("Forecast has been updated")
    else:
        now = datetime.now().time()
        print("Init time: ",now)
        sensor_data(bucket_name,blob_sensor,path_sensor)
        weather_forecast(url_forecast,lat,h)
        OPCUA_client_Input(client, Ta, To, Io, vo, fore_temp, fore_solar,
fore_wind)
        loop = Popen("loop.bat",
cwd=r"C:\Christian\Personal\Erasmus\TFM\Codes\Python")
        time.sleep(15)
        OPCUA_client_Output(client)
        OPCUA_client_simulation(client)
        actuator_data(bucket_name, blob_actuator, path_actuator, Qhc, SH, Cl1,
Cl2)
        simulation_data(bucket_name, blob_sensor, path_sensor, Ta, To, Io, vo)
        counter = counter + 1
        print("Iteration number: ",counter)
        time.sleep(163.3)
```

A9. Python GUI

```
import customtkinter as ctk
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import requests, json, influxdb_client
```

```
from PIL import ImageTk, Image
from io import BytesIO
from datetime import datetime
from solarpy import irradiance_on_plane
from opcua import Client
############################################################
global lines
lines = Image.open("./images/sim/lines/lines_new.png")
#global front_Tg_5, front_Tpv_5, air_Ta_5, air_Tr_5, back_Ts_5, back_Tg_5
front_Tg_5 = Image.open("./images/sim/front/Tg_5.png")
front_Tpv_5 = Image.open("./images/sim/front/Tpv_5.png")
air_Ta_5 = Image.open("./images/sim/air/Ta_5.png")
air_Tr_5 = Image.open("./images/sim/air/Tr_5.png")
back_Ts_5 = Image.open("./images/sim/back/Ts_5.png")
back_Tg_5 = Image.open("./images/sim/back/Tg_5.png")

front_Tg_10 = Image.open("./images/sim/front/Tg_10.png")
front_Tpv_10 = Image.open("./images/sim/front/Tpv_10.png")
air_Ta_10 = Image.open("./images/sim/air/Ta_10.png")
air_Tr_10 = Image.open("./images/sim/air/Tr_10.png")
back_Ts_10 = Image.open("./images/sim/back/Ts_10.png")
back_Tg_10 = Image.open("./images/sim/back/Tg_10.png")

front_Tg_15 = Image.open("./images/sim/front/Tg_15.png")
front_Tpv_15 = Image.open("./images/sim/front/Tpv_15.png")
air_Ta_15 = Image.open("./images/sim/air/Ta_15.png")
air_Tr_15 = Image.open("./images/sim/air/Tr_15.png")
back_Ts_15 = Image.open("./images/sim/back/Ts_15.png")
back_Tg_15 = Image.open("./images/sim/back/Tg_15.png")

front_Tg_20 = Image.open("./images/sim/front/Tg_20.png")
front_Tpv_20 = Image.open("./images/sim/front/Tpv_20.png")
air_Ta_20 = Image.open("./images/sim/air/Ta_20.png")
air_Tr_20 = Image.open("./images/sim/air/Tr_20.png")
back_Ts_20 = Image.open("./images/sim/back/Ts_20.png")
back_Tg_20 = Image.open("./images/sim/back/Tg_20.png")

front_Tg_25 = Image.open("./images/sim/front/Tg_25.png")
front_Tpv_25 = Image.open("./images/sim/front/Tpv_25.png")
air_Ta_25 = Image.open("./images/sim/air/Ta_25.png")
air_Tr_25 = Image.open("./images/sim/air/Tr_25.png")
back_Ts_25 = Image.open("./images/sim/back/Ts_25.png")
back_Tg_25 = Image.open("./images/sim/back/Tg_25.png")

front_Tg_30 = Image.open("./images/sim/front/Tg_30.png")
front_Tpv_30 = Image.open("./images/sim/front/Tpv_30.png")
air_Ta_30 = Image.open("./images/sim/air/Ta_30.png")
air_Tr_30 = Image.open("./images/sim/air/Tr_30.png")
back_Ts_30 = Image.open("./images/sim/back/Ts_30.png")
back_Tg_30 = Image.open("./images/sim/back/Tg_30.png")

front_Tg_35 = Image.open("./images/sim/front/Tg_35.png")
front_Tpv_35 = Image.open("./images/sim/front/Tpv_35.png")
air_Ta_35 = Image.open("./images/sim/air/Ta_35.png")
air_Tr_35 = Image.open("./images/sim/air/Tr_35.png")
back_Ts_35 = Image.open("./images/sim/back/Ts_35.png")
```

```
back_Tg_35 = Image.open("./images/sim/back/Tg_35.png")

base_map = Image.open("./images/maps/base_map.png")
base_bar = Image.open("./images/maps/base_bar.png")
TA2_bar = Image.open("./images/maps/TA2_bar.png")
WND_bar = Image.open("./images/maps/WND_bar.png")
CL_bar = Image.open("./images/maps/CL_bar.png")
ini_bar = Image.open("./images/maps/bar_ini.png")

broken_clouds = Image.open("./images/maps/icons/broken_clouds.png")
clear_sky = Image.open("./images/maps/icons/clear_sky.png")
few_clouds = Image.open("./images/maps/icons/few_clouds.png")
mist = Image.open("./images/maps/icons/mist.png")
rain = Image.open("./images/maps/icons/rain.png")
scattered_clouds = Image.open("./images/maps/icons/scattered_clouds.png")
shower_rain = Image.open("./images/maps/icons/shower_rain.png")
snow = Image.open("./images/maps/icons/snow.png")
thunderstorm = Image.open("./images/maps/icons/thunderstorm.png")

base_plt = Image.open("./plot.png")

is_daily = True
is_base = True
is_TA = False
is_WND = False
is_CL = False

lat = 44.299999
lon = 8.483333
url_forecast_daily =
"https://pro.openweathermap.org/data/2.5/forecast/daily?lat=44.29&lon=8.48&cnt=8&a
ppid=3f7d5cda9bddd63b2f1a73456fb51fe0&units=metric"
url_forecast_3h =
"https://pro.openweathermap.org/data/2.5/forecast?lat=44.29&lon=8.48&cnt=8&appid=3
f7d5cda9bddd63b2f1a73456fb51fe0&units=metric"
vnorm = np.array([0, 0, -1])
h = 0

query_list = [0,0,0,0,0,0,0,0,0,0,0,0] #[Ta,Ts,Tr,Tg,Tpv,To,Io,vo,Qhc,Sh,Cl1,Cl2]
query_names = ["Ta","Ts","Tr","Tg","Tpv","To","Io","vo","Qhc","SH","Cl1","Cl2"]

bucket = "OPCUA"
org = "jato11channel@gmail.com"
token = "kJXsqZWs4DZZ51gK1VeuxbHeRw6vtjbikCj7tN9-
pLy_wFUAjLBB2QFblAa4dNlZbp1fuaBjvKyXRxWJ459Stg=="
influx_url = "https://europe-west1-1.gcp.cloud2.influxdata.com"

influx_client = influxdb_client.InfluxDBClient(
    url = influx_url,
    token = token,
    org = org)

query_api = influx_client.query_api()

opc_url = "opc.tcp://192.168.1.69:4840"
client = Client(opc_url)
```

```
############################################################
ctk.set_appearance_mode("Dark")
ctk.set_default_color_theme("green")

root = ctk.CTk()
root.title("Green House - Digital Twin")
root.geometry("1920x1080")

root.grid_columnconfigure(1, weight = 1)
root.grid_rowconfigure(0, weight = 1)

frame_left = ctk.CTkFrame(master=root, width=360, corner_radius=0)
frame_left.grid(row=0, column=0, sticky="nswe")

frame_1 = ctk.CTkFrame(master=root)
frame_1.grid(row=0, column=1, sticky="nswe", padx=20, pady=20)

frame_1.rowconfigure(3, weight=10)
frame_1.columnconfigure(3, weight=10)

frame_2 = ctk.CTkFrame(master=root)
frame_2.grid(row=0, column=1, sticky="nswe", padx=20, pady=20)

frame_2.rowconfigure(14, weight=10)
frame_2.columnconfigure(9, weight=10)

frame_3 = ctk.CTkFrame(master=root)
frame_3.grid(row=0, column=1, sticky="nswe", padx=20, pady=20)

frame_3.rowconfigure(14, weight=10)
frame_3.columnconfigure(9, weight=10)

frame_sensor = ctk.CTkFrame(frame_1)
frame_sensor.grid(column=3, row=1, padx=40, pady=40, sticky="nswe")

frame_sensor.rowconfigure(5, weight=10)
frame_sensor.columnconfigure(2, weight=10)

frame_actuator = ctk.CTkFrame(frame_1)
frame_actuator.grid(column=3, row=2, padx=40, pady=40, sticky="nswe")

frame_actuator.rowconfigure(5, weight=10)
frame_actuator.columnconfigure(2, weight=10)

frame_forecast = ctk.CTkFrame(frame_2)
frame_forecast.grid(column=0, row=3, padx=40, pady=40, sticky="nswe",
columnspan=12)

frame_forecast.rowconfigure(8, weight=10)
frame_forecast.columnconfigure(8, weight=10)

frame_query = ctk.CTkFrame(frame_3)
frame_query.grid(column=0, row=6, padx=40, pady=40, sticky="nswe",
                 rowspan=3, columnspan=6)

frame_query.rowconfigure(3, weight=10)
```

```
frame_query.columnconfigure(14, weight=10)

frame_time = ctk.CTkFrame(frame_query)
frame_time.grid(column=14, row=0, padx=20, pady=20, sticky="nswe", rowspan=3,
                columnspan=1)

frame_time.rowconfigure(2, weight=10)
frame_time.columnconfigure(1, weight=10)

def show_frame(frame):
    frame.tkraise()

show_frame(frame_1)

############################################################
def nav_frame1():
    show_frame(frame_1)

def nav_frame2():
    show_frame(frame_2)

def nav_frame3():
    show_frame(frame_3)

def daily_fore():
    global is_daily
    if is_daily == False:
        is_daily = True
    else:
        is_daily = is_daily

def hourly_fore():
    global is_daily
    if is_daily == True:
        is_daily = False
    else:
        is_daily = is_daily
def TA():
    global is_base, is_WND, is_TA, is_CL
    if (is_base == True)or(is_WND == True)or(is_CL == True):
        url_img1 =
"http://maps.openweathermap.org/maps/2.0/weather/TA2/6/33/22?appid=3f7d5cda9bddd63
b2f1a73456fb51fe0"
        url_img2 =
"http://maps.openweathermap.org/maps/2.0/weather/TA2/6/34/22?appid=3f7d5cda9bddd63
b2f1a73456fb51fe0"
        url_img3 =
"http://maps.openweathermap.org/maps/2.0/weather/TA2/6/33/23?appid=3f7d5cda9bddd63
b2f1a73456fb51fe0"
        url_img4 =
"http://maps.openweathermap.org/maps/2.0/weather/TA2/6/34/23?appid=3f7d5cda9bddd63
b2f1a73456fb51fe0"

        response1 = requests.get(url_img1)
        response2 = requests.get(url_img2)
        response3 = requests.get(url_img3)
```

```
        response4 = requests.get(url_img4)

        img1 = Image.open(BytesIO(response1.content))
        img2 = Image.open(BytesIO(response2.content))
        img3 = Image.open(BytesIO(response3.content))
        img4 = Image.open(BytesIO(response4.content))

        TA_map = base_map.copy()

        TA_map.paste(img1, (0,0), mask = img1)
        TA_map.paste(img2, (256,0), mask = img2)
        TA_map.paste(img3, (0,256), mask = img3)
        TA_map.paste(img4, (256,256), mask = img4)

        TA_img = ImageTk.PhotoImage(TA_map)
        lbl_map.configure(image=TA_img)
        lbl_map.image = TA_img
        lbl_bar.configure(image=TA2bar_img)
        lbl_bar.image = TA2bar_img
        is_base = False
        is_WND = False
        is_TA = True
        is_CL = False
    else:
        base_img = ImageTk.PhotoImage(base_map)
        lbl_map.configure(image=base_img)
        lbl_map.image = base_img
        lbl_bar.configure(image=bar_img)
        lbl_bar.image = bar_img
        is_base = True
        is_WND = False
        is_TA = False
        is_CL = False

def WND():
    global is_base, is_WND, is_TA, is_CL
    if (is_base == True)or(is_TA == True)or(is_CL == True):
        url_img1 =
"http://maps.openweathermap.org/maps/2.0/weather/WND/6/33/22?appid=3f7d5cda9bddd63
b2f1a73456fb51fe0"
        url_img2 =
"http://maps.openweathermap.org/maps/2.0/weather/WND/6/34/22?appid=3f7d5cda9bddd63
b2f1a73456fb51fe0"
        url_img3 =
"http://maps.openweathermap.org/maps/2.0/weather/WND/6/33/23?appid=3f7d5cda9bddd63
b2f1a73456fb51fe0"
        url_img4 =
"http://maps.openweathermap.org/maps/2.0/weather/WND/6/34/23?appid=3f7d5cda9bddd63
b2f1a73456fb51fe0"

        response1 = requests.get(url_img1)
        response2 = requests.get(url_img2)
        response3 = requests.get(url_img3)
        response4 = requests.get(url_img4)

        img1 = Image.open(BytesIO(response1.content))
```

```
        img2 = Image.open(BytesIO(response2.content))
        img3 = Image.open(BytesIO(response3.content))
        img4 = Image.open(BytesIO(response4.content))

        WND_map = base_map.copy()

        WND_map.paste(img1, (0,0), mask = img1)
        WND_map.paste(img2, (256,0), mask = img2)
        WND_map.paste(img3, (0,256), mask = img3)
        WND_map.paste(img4, (256,256), mask = img4)

        WND_img = ImageTk.PhotoImage(WND_map)
        lbl_map.configure(image=WND_img)
        lbl_map.image = WND_img
        lbl_bar.configure(image=WNDbar_img)
        lbl_bar.image = WNDbar_img
        is_base = False
        is_WND = True
        is_TA = False
        is_CL = False
    else:
        base_img = ImageTk.PhotoImage(base_map)
        lbl_map.configure(image=base_img)
        lbl_map.image = base_img
        lbl_bar.configure(image=bar_img)
        lbl_bar.image = bar_img
        is_base = True
        is_WND = False
        is_TA = False
        is_CL = False

def CL():
    global is_base, is_WND, is_TA, is_CL
    if (is_base == True)or(is_TA == True)or(is_WND == True):
        url_img1 =
"http://maps.openweathermap.org/maps/2.0/weather/CL/6/33/22?appid=3f7d5cda9bddd63b
2f1a73456fb51fe0"
        url_img2 =
"http://maps.openweathermap.org/maps/2.0/weather/CL/6/34/22?appid=3f7d5cda9bddd63b
2f1a73456fb51fe0"
        url_img3 =
"http://maps.openweathermap.org/maps/2.0/weather/CL/6/33/23?appid=3f7d5cda9bddd63b
2f1a73456fb51fe0"
        url_img4 =
"http://maps.openweathermap.org/maps/2.0/weather/CL/6/34/23?appid=3f7d5cda9bddd63b
2f1a73456fb51fe0"

        response1 = requests.get(url_img1)
        response2 = requests.get(url_img2)
        response3 = requests.get(url_img3)
        response4 = requests.get(url_img4)

        img1 = Image.open(BytesIO(response1.content))
        img2 = Image.open(BytesIO(response2.content))
        img3 = Image.open(BytesIO(response3.content))
        img4 = Image.open(BytesIO(response4.content))
```

```
        CL_map = base_map.copy()

        CL_map.paste(img1, (0,0), mask = img1)
        CL_map.paste(img2, (256,0), mask = img2)
        CL_map.paste(img3, (0,256), mask = img3)
        CL_map.paste(img4, (256,256), mask = img4)

        CL_img = ImageTk.PhotoImage(CL_map)
        lbl_map.configure(image=CL_img)
        lbl_map.image = CL_img
        lbl_bar.configure(image=CLbar_img)
        lbl_bar.image = CLbar_img
        is_base = False
        is_WND = False
        is_TA = False
        is_CL = True
    else:
        base_img = ImageTk.PhotoImage(base_map)
        lbl_map.configure(image=base_img)
        lbl_map.image = base_img
        lbl_bar.configure(image=bar_img)
        lbl_bar.image = bar_img
        is_base = True
        is_WND = False
        is_TA = False
        is_CL = False

def query_Ta():
    global query_list
    if query_list[0] == 1:
        query_list[0] = 0
    else:
        query_list[0] = 1
def query_Ts():
    global query_list
    if query_list[1] == 1:
        query_list[1] = 0
    else:
        query_list[1] = 1

def query_Tr():
    global query_list
    if query_list[2] == 1:
        query_list[2] = 0
    else:
        query_list[2] = 1

def query_Tg():
    global query_list
    if query_list[3] == 1:
        query_list[3] = 0
    else:
        query_list[3] = 1

def query_Tpv():
```

```
    global query_list
    if query_list[4] == 1:
        query_list[4] = 0
    else:
        query_list[4] = 1

def query_To():
    global query_list
    if query_list[5] == 1:
        query_list[5] = 0
    else:
        query_list[5] = 1

def query_Io():
    global query_list
    if query_list[6] == 1:
        query_list[6] = 0
    else:
        query_list[6] = 1
def query_vo():
    global query_list
    if query_list[7] == 1:
        query_list[7] = 0
    else:
        query_list[7] = 1

def query_Qhc():
    global query_list
    if query_list[8] == 1:
        query_list[8] = 0
    else:
        query_list[8] = 1

def query_SH():
    global query_list
    if query_list[9] == 1:
        query_list[9] = 0
    else:
        query_list[9] = 1

def query_Cl1():
    global query_list
    if query_list[10] == 1:
        query_list[10] = 0
    else:
        query_list[10] = 1

def query_Cl2():
    global query_list
    if query_list[11] == 1:
        query_list[11] = 0
    else:
        query_list[11] = 1

def get_query():
    global plot
```

```
    number = sum(query_list)
    counter = 0
    query_ini = ' from(bucket:"OPCUA")\
    |> range(start: '
    query_end = ')\
    |> filter(fn:(r) => r._measurement == "opcua")\
    |> filter(fn:(r) => r._field == '

    current_time = combo_query.get()
    query = query_ini + current_time + query_end
    if number == 0:
        print("Select at leat one variable")
    else:

        for i in range(0,12,1):
            if query_list[i] == 1:
                query = query + '"' + query_names[i] + '"'
                counter = counter + 1
                if counter < number:
                    query = query + " or r._field == "
                else:
                    query = query + ")"
        result = query_api.query(org = org, query = query)

        results = []
        for table in result:
          for record in table.records:
            results.append((record.get_field(), record.get_value(),
record.get_time()))

        df = pd.DataFrame(results, columns = ['Variable','Value','Time'])
        dfTa = df.where(df['Variable']=='Ta').dropna()
        dfTs = df.where(df['Variable']=='Ts').dropna()
        dfTr = df.where(df['Variable']=='Tr').dropna()
        dfTg = df.where(df['Variable']=='Tg').dropna()
        dfTpv = df.where(df['Variable']=='Tpv').dropna()
        dfTo = df.where(df['Variable']=='To').dropna()
        dfIo = df.where(df['Variable']=='Io').dropna()
        dfvo = df.where(df['Variable']=='vo').dropna()
        dfQhc = df.where(df['Variable']=='Qhc').dropna()
        dfSH = df.where(df['Variable']=='SH').dropna()
        dfCl1 = df.where(df['Variable']=='Cl1').dropna()
        dfCl2 = df.where(df['Variable']=='Cl2').dropna()

        query_df =
[dfTa,dfTs,dfTr,dfTg,dfTpv,dfTo,dfIo,dfvo,dfQhc,dfSH,dfCl1,dfCl2]
        query_y = []
        query_x = []
        query_tag = []
        fig = plt.figure()
        fig.set_figheight(10)
        fig.set_figwidth(21)
        for i in range(0,len(query_df),1):
            if query_df[i].empty == False:
                query_y.append(query_df[i]['Value'])
                query_x.append(query_df[i]['Time'])
```

```
                query_tag.append(query_df[i].iloc[0,0])
        for i in range(0,len(query_tag),1):
            plt.plot(query_x[i],query_y[i])
        plt.legend(query_tag)
        plot = plt.savefig('plot.png', bbox_inches='tight')
        plt.close()
        plot_ = Image.open('./plot.png')
        plt_img = ImageTk.PhotoImage(plot_)
        lbl_plt.configure(image=plt_img)
        lbl_plt.image = plt_img
############################################################
global map_img

lines_img = ImageTk.PhotoImage(lines)
sim_bar_img = ImageTk.PhotoImage(ini_bar)

map_img = ImageTk.PhotoImage(base_map)
bar_img = ImageTk.PhotoImage(base_bar)
TA2bar_img = ImageTk.PhotoImage(TA2_bar)
WNDbar_img = ImageTk.PhotoImage(WND_bar)
CLbar_img = ImageTk.PhotoImage(CL_bar)
broken_clouds_img = ImageTk.PhotoImage(broken_clouds)
clear_sky_img = ImageTk.PhotoImage(clear_sky)
few_clouds_img = ImageTk.PhotoImage(few_clouds)
mist_img = ImageTk.PhotoImage(mist)
rain_img = ImageTk.PhotoImage(rain)
scattered_clouds_img = ImageTk.PhotoImage(scattered_clouds)
shower_rain_img = ImageTk.PhotoImage(shower_rain)
snow_img = ImageTk.PhotoImage(snow)
thunderstorm_img = ImageTk.PhotoImage(thunderstorm)
#################################################################
btn_frame1 = ctk.CTkButton(frame_left, text="Current Status", command=nav_frame1
                           , width=360, height=80, text_font=("Roboto Medium",16)
                           , corner_radius=0)
btn_frame2 = ctk.CTkButton(frame_left, text="Weather Forecast", command=nav_frame2
                           , width=360, height=80, text_font=("Roboto Medium",16)
                           , corner_radius=0)
btn_frame3 = ctk.CTkButton(frame_left, text="Historical Data", command=nav_frame3
                           , width=360, height=80, text_font=("Roboto Medium",16)
                           , corner_radius=0)
#################################################################
lbl_sensors = ctk.CTkLabel(frame_sensor, text="Sensor Data", text_font=("Roboto
Medium",18))
lbl_Ta = ctk.CTkLabel(frame_sensor, text="Ambient Temperature:",
text_font=("Roboto Medium",16))
lbl_To = ctk.CTkLabel(frame_sensor, text="Outdoor Temperature:",
text_font=("Roboto Medium",16))
lbl_Io = ctk.CTkLabel(frame_sensor, text="Solar Irradiance:", text_font=("Roboto
Medium",16))
lbl_vo = ctk.CTkLabel(frame_sensor, text="Wind Speed:", text_font=("Roboto
Medium",16))
lbl_Qhc = ctk.CTkLabel(frame_actuator, text="Heater/Cooler Power:",
text_font=("Roboto Medium",16))
lbl_SH = ctk.CTkLabel(frame_actuator, text="Shades Extension:", text_font=("Roboto
Medium",16))
```

```
lbl_Cl1 = ctk.CTkLabel(frame_actuator, text="Window 1 Opening:",
text_font=("Roboto Medium",16))
lbl_Cl2 = ctk.CTkLabel(frame_actuator, text="Window 2 Opening:",
text_font=("Roboto Medium",16))

lbl_actuators = ctk.CTkLabel(frame_actuator, text="Manipulated Variables",
text_font=("Roboto Medium",18))
ind_Ta = ctk.CTkLabel(frame_sensor, text="0.0 °C", text_font=("Roboto Medium",16))
ind_To = ctk.CTkLabel(frame_sensor, text="0.0 °C", text_font=("Roboto Medium",16))
ind_Io = ctk.CTkLabel(frame_sensor, text="0.0 W/m²", text_font=("Roboto
Medium",16))
ind_vo = ctk.CTkLabel(frame_sensor, text="0.0 m/s", text_font=("Roboto
Medium",16))
ind_Qhc = ctk.CTkLabel(frame_actuator, text="0.0 W", text_font=("Roboto
Medium",16))
ind_SH = ctk.CTkLabel(frame_actuator, text="0.0 %", text_font=("Roboto
Medium",16))
ind_Cl1 = ctk.CTkLabel(frame_actuator, text="0.0 %", text_font=("Roboto
Medium",16))
ind_Cl2 = ctk.CTkLabel(frame_actuator, text="0.0 %", text_font=("Roboto
Medium",16))

sim_lines = ctk.CTkLabel(frame_1, image=lines_img)
sim_bar = ctk.CTkLabel(frame_1, image=sim_bar_img)

#####################################################################
lbl_map = ctk.CTkLabel(frame_2, image = map_img)
lbl_bar = ctk.CTkLabel(frame_2, image = bar_img)

lbl_time1 = ctk.CTkLabel(frame_forecast, text = "Day 0")
lbl_time2 = ctk.CTkLabel(frame_forecast, text = "Day 1")
lbl_time3 = ctk.CTkLabel(frame_forecast, text = "Day 2")
lbl_time4 = ctk.CTkLabel(frame_forecast, text = "Day 3")
lbl_time5 = ctk.CTkLabel(frame_forecast, text = "Day 4")
lbl_time6 = ctk.CTkLabel(frame_forecast, text = "Day 5")
lbl_time7 = ctk.CTkLabel(frame_forecast, text = "Day 6")
lbl_time8 = ctk.CTkLabel(frame_forecast, text = "Day 7")

lbl_temp1 = ctk.CTkLabel(frame_forecast, text = "Max/Min Temperature:")
lbl_temp2 = ctk.CTkLabel(frame_forecast, text = "Max/Min Temperature:")
lbl_temp3 = ctk.CTkLabel(frame_forecast, text = "Max/Min Temperature:")
lbl_temp4 = ctk.CTkLabel(frame_forecast, text = "Max/Min Temperature:")
lbl_temp5 = ctk.CTkLabel(frame_forecast, text = "Max/Min Temperature:")
lbl_temp6 = ctk.CTkLabel(frame_forecast, text = "Max/Min Temperature:")
lbl_temp7 = ctk.CTkLabel(frame_forecast, text = "Max/Min Temperature:")
lbl_temp8 = ctk.CTkLabel(frame_forecast, text = "Max/Min Temperature:")

ind_temp1 = ctk.CTkLabel(frame_forecast, text = "0/0 °C")
ind_temp2 = ctk.CTkLabel(frame_forecast, text = "0/0 °C")
ind_temp3 = ctk.CTkLabel(frame_forecast, text = "0/0 °C")
ind_temp4 = ctk.CTkLabel(frame_forecast, text = "0/0 °C")
ind_temp5 = ctk.CTkLabel(frame_forecast, text = "0/0 °C")
ind_temp6 = ctk.CTkLabel(frame_forecast, text = "0/0 °C")
ind_temp7 = ctk.CTkLabel(frame_forecast, text = "0/0 °C")
ind_temp8 = ctk.CTkLabel(frame_forecast, text = "0/0 °C")
```

```
lbl_wind1 = ctk.CTkLabel(frame_forecast, text = "Wind Speed:")
lbl_wind2 = ctk.CTkLabel(frame_forecast, text = "Wind Speed:")
lbl_wind3 = ctk.CTkLabel(frame_forecast, text = "Wind Speed:")
lbl_wind4 = ctk.CTkLabel(frame_forecast, text = "Wind Speed:")
lbl_wind5 = ctk.CTkLabel(frame_forecast, text = "Wind Speed:")
lbl_wind6 = ctk.CTkLabel(frame_forecast, text = "Wind Speed:")
lbl_wind7 = ctk.CTkLabel(frame_forecast, text = "Wind Speed:")
lbl_wind8 = ctk.CTkLabel(frame_forecast, text = "Wind Speed:")

ind_wind1 = ctk.CTkLabel(frame_forecast, text = "0 m/s")
ind_wind2 = ctk.CTkLabel(frame_forecast, text = "0 m/s")
ind_wind3 = ctk.CTkLabel(frame_forecast, text = "0 m/s")
ind_wind4 = ctk.CTkLabel(frame_forecast, text = "0 m/s")
ind_wind5 = ctk.CTkLabel(frame_forecast, text = "0 m/s")
ind_wind6 = ctk.CTkLabel(frame_forecast, text = "0 m/s")
ind_wind7 = ctk.CTkLabel(frame_forecast, text = "0 m/s")
ind_wind8 = ctk.CTkLabel(frame_forecast, text = "0 m/s")

lbl_solar1 = ctk.CTkLabel(frame_forecast, text = "Solar Irradiance:")
lbl_solar2 = ctk.CTkLabel(frame_forecast, text = "Solar Irradiance:")
lbl_solar3 = ctk.CTkLabel(frame_forecast, text = "Solar Irradiance:")
lbl_solar4 = ctk.CTkLabel(frame_forecast, text = "Solar Irradiance:")
lbl_solar5 = ctk.CTkLabel(frame_forecast, text = "Solar Irradiance:")
lbl_solar6 = ctk.CTkLabel(frame_forecast, text = "Solar Irradiance:")
lbl_solar7 = ctk.CTkLabel(frame_forecast, text = "Solar Irradiance:")
lbl_solar8 = ctk.CTkLabel(frame_forecast, text = "Solar Irradiance:")

ind_solar1 = ctk.CTkLabel(frame_forecast, text = "0 W/m²")
ind_solar2 = ctk.CTkLabel(frame_forecast, text = "0 W/m²")
ind_solar3 = ctk.CTkLabel(frame_forecast, text = "0 W/m²")
ind_solar4 = ctk.CTkLabel(frame_forecast, text = "0 W/m²")
ind_solar5 = ctk.CTkLabel(frame_forecast, text = "0 W/m²")
ind_solar6 = ctk.CTkLabel(frame_forecast, text = "0 W/m²")
ind_solar7 = ctk.CTkLabel(frame_forecast, text = "0 W/m²")
ind_solar8 = ctk.CTkLabel(frame_forecast, text = "0 W/m²")

lbl_icon1 = ctk.CTkLabel(frame_forecast, image = clear_sky_img)
lbl_icon2 = ctk.CTkLabel(frame_forecast, image = broken_clouds_img)
lbl_icon3 = ctk.CTkLabel(frame_forecast, image = few_clouds_img)
lbl_icon4 = ctk.CTkLabel(frame_forecast, image = scattered_clouds_img)
lbl_icon5 = ctk.CTkLabel(frame_forecast, image = mist_img)
lbl_icon6 = ctk.CTkLabel(frame_forecast, image = rain_img)
lbl_icon7 = ctk.CTkLabel(frame_forecast, image = shower_rain_img)
lbl_icon8 = ctk.CTkLabel(frame_forecast, image = thunderstorm_img)

btn_TA = ctk.CTkButton(frame_2, text = "Temperature", command = TA,
                       height=80, width=200, text_font=("Roboto Medium",16))
btn_WND = ctk.CTkButton(frame_2, text = "Wind Speed", command = WND,
                        height=80, width=200, text_font=("Roboto Medium",16))
btn_CL = ctk.CTkButton(frame_2, text = "Cloudiness", command = CL,
                       height=80, width=200, text_font=("Roboto Medium",16))

btn_daily = ctk.CTkButton(frame_2, text = "Get daily forecast!",
                          command = daily_fore, height=80, width=200,
                          text_font=("Roboto Medium",16))
btn_hourly = ctk.CTkButton(frame_2, text = "Get 3-hour forecast!",
```

```
                            command = hourly_fore, height=80, width=200,
                            text_font=("Roboto Medium",16))
##############################################################################
plt_img = ImageTk.PhotoImage(base_plt)
lbl_plt = ctk.CTkLabel(frame_3, image=plt_img)

swt_Ta = ctk.CTkSwitch(frame_query, text= "Ta ", command=query_Ta,
text_font=("Roboto Medium",16),width=60, height=30)
swt_Ts = ctk.CTkSwitch(frame_query, text= "Ts ", command=query_Ts,
text_font=("Roboto Medium",16),width=60, height=30)
swt_Tr = ctk.CTkSwitch(frame_query, text= "Tr ", command=query_Tr,
text_font=("Roboto Medium",16),width=60, height=30)
swt_Tg = ctk.CTkSwitch(frame_query, text= "Tg ", command=query_Tg,
text_font=("Roboto Medium",16),width=60, height=30)
swt_Tpv = ctk.CTkSwitch(frame_query, text= "Tpv", command=query_Tpv,
text_font=("Roboto Medium",16),width=60, height=30)
swt_To = ctk.CTkSwitch(frame_query, text= "To ", command=query_To,
text_font=("Roboto Medium",16),width=60, height=30)
swt_Io = ctk.CTkSwitch(frame_query, text= "Io ", command=query_Io,
text_font=("Roboto Medium",16),width=60, height=30)
swt_vo = ctk.CTkSwitch(frame_query, text= "vo ", command=query_vo,
text_font=("Roboto Medium",16),width=60, height=30)
swt_Qhc = ctk.CTkSwitch(frame_query, text= "Qhc", command=query_Qhc,
text_font=("Roboto Medium",16),width=60, height=30)
swt_SH = ctk.CTkSwitch(frame_query, text= "SH", command=query_SH,
text_font=("Roboto Medium",16),width=60, height=30)
swt_Cl1 = ctk.CTkSwitch(frame_query, text= "Cl1", command=query_Cl1,
text_font=("Roboto Medium",16),width=60, height=30)
swt_Cl2 = ctk.CTkSwitch(frame_query, text= "Cl2", command=query_Cl2,
text_font=("Roboto Medium",16),width=60, height=30)

btn_query = ctk.CTkButton(frame_3, text="Get query!", text_font=("Roboto
Medium",16)
                            , height=80, width=200, command=get_query)

combo_query = ctk.CTkComboBox(frame_time,
                                values=['-5m','-15m', '-1h', '-3h', '-6h', '-12h',
'-24h', '-7d', '-30d'],
                                height=40,width=150, text_font=("Roboto Medium",16))

lbl_query = ctk.CTkLabel(frame_time, text="Specify time range", text_font=("Roboto
Medium",16))
##############################################################################
sim_lines.grid(column=0,row=1, rowspan=2,columnspan=2, sticky="w", padx=30,
pady=100)
#sim_bar.grid(column=0,row=3)

lbl_sensors.grid(column=1,row=0,columnspan=2, pady=40)
lbl_Ta.grid(column=1,row=1, sticky="nw", padx = 60, pady=10)
ind_Ta.grid(column=2,row=1, sticky="ne", padx = 60, pady=10)
lbl_To.grid(column=1,row=2, sticky="nw", padx = 60, pady=10)
ind_To.grid(column=2,row=2, sticky="ne", padx = 60, pady=10)
lbl_Io.grid(column=1,row=3, sticky="nw", padx = 60, pady=10)
ind_Io.grid(column=2,row=3, sticky="ne", padx = 60, pady=10)
lbl_vo.grid(column=1,row=4, sticky="nw", padx = 60, pady=10)
ind_vo.grid(column=2,row=4, sticky="ne", padx = 60, pady=10)
```

```
lbl_actuators.grid(column=1,row=0, columnspan=2, pady=40)
lbl_Qhc.grid(column=1,row=1, sticky="nw", padx = 60, pady=10)
ind_Qhc.grid(column=2,row=1, sticky="ne", padx = 60, pady=10)
lbl_SH.grid(column=1,row=2, sticky="nw", padx = 60, pady=10)
ind_SH.grid(column=2,row=2, sticky="ne", padx = 60, pady=10)
lbl_Cl1.grid(column=1,row=3, sticky="nw", padx = 60, pady=10)
ind_Cl1.grid(column=2,row=3, sticky="ne", padx = 60, pady=10)
lbl_Cl2.grid(column=1,row=4, sticky="nw", padx = 60, pady=10)
ind_Cl2.grid(column=2,row=4, sticky="ne", padx = 60, pady=10)
#############################################################################
lbl_map.grid(row=0, column=2, rowspan=3, columnspan=3, padx=20)
lbl_bar.grid(row=0, column=5, rowspan=3, columnspan=2, padx=20)

lbl_time1.grid(row=4, column=0, padx=15)
lbl_time2.grid(row=4, column=1, padx=15)
lbl_time3.grid(row=4, column=2, padx=15)
lbl_time4.grid(row=4, column=3, padx=15)
lbl_time5.grid(row=4, column=4, padx=15)
lbl_time6.grid(row=4, column=5, padx=15)
lbl_time7.grid(row=4, column=6, padx=15)
lbl_time8.grid(row=4, column=7, padx=15)

lbl_icon1.grid(row=5, column=0, padx=15)
lbl_icon2.grid(row=5, column=1, padx=15)
lbl_icon3.grid(row=5, column=2, padx=15)
lbl_icon4.grid(row=5, column=3, padx=15)
lbl_icon5.grid(row=5, column=4, padx=15)
lbl_icon6.grid(row=5, column=5, padx=15)
lbl_icon7.grid(row=5, column=6, padx=15)
lbl_icon8.grid(row=5, column=7, padx=15)

lbl_temp1.grid(row=6, column=0, pady=15, padx=15)
lbl_temp2.grid(row=6, column=1, pady=15, padx=15)
lbl_temp3.grid(row=6, column=2, pady=15, padx=15)
lbl_temp4.grid(row=6, column=3, pady=15, padx=15)
lbl_temp5.grid(row=6, column=4, pady=15, padx=15)
lbl_temp6.grid(row=6, column=5, pady=15, padx=15)
lbl_temp7.grid(row=6, column=6, pady=15, padx=15)
lbl_temp8.grid(row=6, column=7, pady=15, padx=15)

ind_temp1.grid(row=7, column=0, padx=15)
ind_temp2.grid(row=7, column=1, padx=15)
ind_temp3.grid(row=7, column=2, padx=15)
ind_temp4.grid(row=7, column=3, padx=15)
ind_temp5.grid(row=7, column=4, padx=15)
ind_temp6.grid(row=7, column=5, padx=15)
ind_temp7.grid(row=7, column=6, padx=15)
ind_temp8.grid(row=7, column=7, padx=15)

lbl_wind1.grid(row=8, column=0, pady=15, padx=15)
lbl_wind2.grid(row=8, column=1, pady=15, padx=15)
lbl_wind3.grid(row=8, column=2, pady=15, padx=15)
lbl_wind4.grid(row=8, column=3, pady=15, padx=15)
lbl_wind5.grid(row=8, column=4, pady=15, padx=15)
lbl_wind6.grid(row=8, column=5, pady=15, padx=15)
```

```
lbl_wind7.grid(row=8, column=6, pady=15, padx=15)
lbl_wind8.grid(row=8, column=7, pady=15, padx=15)

ind_wind1.grid(row=9, column=0, padx=15)
ind_wind2.grid(row=9, column=1, padx=15)
ind_wind3.grid(row=9, column=2, padx=15)
ind_wind4.grid(row=9, column=3, padx=15)
ind_wind5.grid(row=9, column=4, padx=15)
ind_wind6.grid(row=9, column=5, padx=15)
ind_wind7.grid(row=9, column=6, padx=15)
ind_wind8.grid(row=9, column=7, padx=15)

lbl_solar1.grid(row=10, column=0, pady=15, padx=15)
lbl_solar2.grid(row=10, column=1, pady=15, padx=15)
lbl_solar3.grid(row=10, column=2, pady=15, padx=15)
lbl_solar4.grid(row=10, column=3, pady=15, padx=15)
lbl_solar5.grid(row=10, column=4, pady=15, padx=15)
lbl_solar6.grid(row=10, column=5, pady=15, padx=15)
lbl_solar7.grid(row=10, column=6, pady=15, padx=15)
lbl_solar8.grid(row=10, column=7, pady=15, padx=15)

ind_solar1.grid(row=11, column=0, padx=15)
ind_solar2.grid(row=11, column=1, padx=15)
ind_solar3.grid(row=11, column=2, padx=15)
ind_solar4.grid(row=11, column=3, padx=15)
ind_solar5.grid(row=11, column=4, padx=15)
ind_solar6.grid(row=11, column=5, padx=15)
ind_solar7.grid(row=11, column=6, padx=15)
ind_solar8.grid(row=11, column=7, padx=15)

btn_TA.grid(row=0, column=0, columnspan = 2)
btn_WND.grid(row=1, column=0, columnspan = 2)
btn_CL.grid(row=2, column=0, columnspan = 2)

btn_daily.grid(row=1, column=7)
btn_hourly.grid(row=2, column=7)

btn_frame1.grid(row=0,column=0, pady=25)
btn_frame2.grid(row=1,column=0, pady=25)
btn_frame3.grid(row=2,column=0, pady=25)

lbl_plt.grid(row=0, column=0, columnspan=5, rowspan=5,padx=15, pady=15)

swt_Ta.grid(row=0,column=0, sticky="w", columnspan=2, padx=30, pady=30)
swt_Ts.grid(row=0,column=2, sticky="w", columnspan=2, padx=30, pady=30)
swt_Tr.grid(row=0,column=4, sticky="w", columnspan=2, padx=30, pady=30)
swt_Tg.grid(row=0,column=8, sticky="w", columnspan=2, padx=30, pady=30)
swt_Tpv.grid(row=0,column=10, sticky="w", columnspan=2, padx=30, pady=30)
swt_To.grid(row=1,column=0, sticky="w", columnspan=2, padx=30, pady=30)
swt_Io.grid(row=1,column=2, sticky="w", columnspan=2, padx=30, pady=30)
swt_vo.grid(row=1,column=4, sticky="w", columnspan=2, padx=30, pady=30)
swt_Qhc.grid(row=2,column=0, sticky="w", columnspan=2, padx=30, pady=30)
swt_SH.grid(row=2,column=2, sticky="w", columnspan=2, padx=30, pady=30)
swt_Cl1.grid(row=2,column=4, sticky="w", columnspan=2, padx=30, pady=30)
swt_Cl2.grid(row=2,column=8, sticky="w", columnspan=2, padx=30, pady=30)
```

```
btn_query.grid(row=7, column=6)
lbl_query.grid(row=1, column=0, pady=15, padx=100)
combo_query.grid(row=2, column=0, pady=15,padx=15)
############################################################
def hourly():
    temp_max = []
    temp_min = []
    wind = []
    solar = []
    clouds = []
    ts = []
    dt = []
    icon_id = []
    icons = []

    if is_daily == True:
        response = requests.get(url_forecast_daily)
        hourly_format = '%A %d/%b'

        forecast = response.json()

        for i in range(0,8,1):
            temp_max.append(forecast['list'][i]['temp']['max'])
            temp_min.append(forecast['list'][i]['temp']['min'])
            wind.append(forecast['list'][i]['speed'])
            icon_id.append(forecast['list'][i]['weather'][0]['icon'])
            clouds.append(forecast['list'][i]['clouds'])
            ts.append(forecast['list'][i]['dt'])
            dt.append(datetime.fromtimestamp(ts[i]))
            solar.append(irradiance_on_plane(vnorm, h, dt[i],
 lat)*(1-(0.85*clouds[i]/100)))
            if (icon_id[i] == '01d')or(icon_id[i] == '01n'):
                icons.append(clear_sky_img)
            elif (icon_id[i] == '02d')or(icon_id[i] == '01n'):
                icons.append(few_clouds_img)
            elif (icon_id[i] == '03d')or(icon_id[i] == '02n'):
                icons.append(scattered_clouds_img)
            elif (icon_id[i] == '04d')or(icon_id[i] == '04n'):
                icons.append(broken_clouds_img)
            elif (icon_id[i] == '09d')or(icon_id[i] == '09n'):
                icons.append(shower_rain_img)
            elif (icon_id[i] == '10d')or(icon_id[i] == '10n'):
                icons.append(rain_img)
            elif (icon_id[i] == '11d')or(icon_id[i] == '11n'):
                icons.append(thunderstorm_img)
            elif (icon_id[i] == '13d')or(icon_id[i] == '13n'):
                icons.append(snow_img)
            else:
                icons.append(mist_img)
    else:
        response = requests.get(url_forecast_3h)
        hourly_format = '%H:%M %d/%b'

        forecast = response.json()

        for i in range(0,8,1):
```

```
            temp_max.append(forecast['list'][i]['main']['temp_max'])
            temp_min.append(forecast['list'][i]['main']['temp_min'])
            wind.append(forecast['list'][i]['wind']['speed'])
            icon_id.append(forecast['list'][i]['weather'][0]['icon'])
            clouds.append(forecast['list'][i]['clouds']['all'])
            ts.append(forecast['list'][i]['dt'])
            dt.append(datetime.fromtimestamp(ts[i]))
            solar.append(irradiance_on_plane(vnorm, h, dt[i], lat)*(1-
(0.85*clouds[i]/100)))
            if (icon_id[i] == '01d')or(icon_id[i] == '01n'):
                icons.append(clear_sky_img)
            elif (icon_id[i] == '02d')or(icon_id[i] == '01n'):
                icons.append(few_clouds_img)
            elif (icon_id[i] == '03d')or(icon_id[i] == '02n'):
                icons.append(scattered_clouds_img)
            elif (icon_id[i] == '04d')or(icon_id[i] == '04n'):
                icons.append(broken_clouds_img)
            elif (icon_id[i] == '09d')or(icon_id[i] == '09n'):
                icons.append(shower_rain_img)
            elif (icon_id[i] == '10d')or(icon_id[i] == '10n'):
                icons.append(rain_img)
            elif (icon_id[i] == '11d')or(icon_id[i] == '11n'):
                icons.append(thunderstorm_img)
            elif (icon_id[i] == '13d')or(icon_id[i] == '13n'):
                icons.append(snow_img)
            else:
                icons.append(mist_img)

    ind_temp1.configure(text = str(temp_max[0])+'/'+str(temp_min[0])+' ºC')
    ind_wind1.configure(text = str(wind[0])+' m/s')
    ind_solar1.configure(text = str('{:.2f}'.format(solar[0]) + ' W/m²'))
    lbl_time1.configure(text = datetime.strftime(dt[0], hourly_format))
    lbl_icon1.configure(image = icons[0])

    ind_temp2.configure(text = str(temp_max[1])+'/'+str(temp_min[1])+' ºC')
    ind_wind2.configure(text = str(wind[1])+' m/s')
    ind_solar2.configure(text = str('{:.2f}'.format(solar[1]) + ' W/m²'))
    lbl_time2.configure(text = datetime.strftime(dt[1], hourly_format))
    lbl_icon2.configure(image = icons[1])

    ind_temp3.configure(text = str(temp_max[2])+'/'+str(temp_min[2])+' ºC')
    ind_wind3.configure(text = str(wind[2])+' m/s')
    ind_solar3.configure(text = str('{:.2f}'.format(solar[2]) + ' W/m²'))
    lbl_time3.configure(text = datetime.strftime(dt[2], hourly_format))
    lbl_icon3.configure(image = icons[2])

    ind_temp4.configure(text = str(temp_max[3])+'/'+str(temp_min[3])+' ºC')
    ind_wind4.configure(text = str(wind[3])+' m/s')
    ind_solar4.configure(text = str('{:.2f}'.format(solar[3]) + ' W/m²'))
    lbl_time4.configure(text = datetime.strftime(dt[3], hourly_format))
    lbl_icon4.configure(image = icons[3])

    ind_temp5.configure(text = str(temp_max[4])+'/'+str(temp_min[4])+' ºC')
    ind_wind5.configure(text = str(wind[4])+' m/s')
    ind_solar5.configure(text = str('{:.2f}'.format(solar[4]) + ' W/m²'))
    lbl_time5.configure(text = datetime.strftime(dt[4], hourly_format))
```

```
    lbl_icon5.configure(image = icons[4])

    ind_temp6.configure(text = str(temp_max[5])+'/'+str(temp_min[5])+' °C')
    ind_wind6.configure(text = str(wind[5])+' m/s')
    ind_solar6.configure(text = str('{:.2f}'.format(solar[5]) + ' W/m²'))
    lbl_time6.configure(text = datetime.strftime(dt[5], hourly_format))
    lbl_icon6.configure(image = icons[5])

    ind_temp7.configure(text = str(temp_max[6])+'/'+str(temp_min[6])+' °C')
    ind_wind7.configure(text = str(wind[6])+' m/s')
    ind_solar7.configure(text = str('{:.2f}'.format(solar[6]) + ' W/m²'))
    lbl_time7.configure(text = datetime.strftime(dt[6], hourly_format))
    lbl_icon7.configure(image = icons[6])

    ind_temp8.configure(text = str(temp_max[7])+'/'+str(temp_min[7])+' °C')
    ind_wind8.configure(text = str(wind[7])+' m/s')
    ind_solar8.configure(text = str('{:.2f}'.format(solar[7]) + ' W/m²'))
    lbl_time8.configure(text = datetime.strftime(dt[7], hourly_format))
    lbl_icon8.configure(image = icons[7])

def READ(client):
    global Ta, To, Io, vo, Qhc, SH, Cl1, Cl2, Ts, Tr, Tg, Tpv
    global air_Ta, air_Tr, front_Tg, front_Tpv, back_Tg, back_Ts
    node_Qhc = client.get_node("ns=2;i=2")
    node_SH = client.get_node("ns=2;i=3")
    node_Cl1 = client.get_node("ns=2;i=4")
    node_Cl2 = client.get_node("ns=2;i=5")
    node_Ta = client.get_node("ns=2;i=7")
    node_Ts = client.get_node("ns=2;i=8")
    node_Tr = client.get_node("ns=2;i=9")
    node_Tg = client.get_node("ns=2;i=10")
    node_Tpv = client.get_node("ns=2;i=11")
    node_To = client.get_node("ns=2;i=15")
    node_Io = client.get_node("ns=2;i=16")
    node_vo = client.get_node("ns=2;i=17")

    try:
        client.connect()
        Qhc = node_Qhc.get_value()
        SH = node_SH.get_value()
        Cl1 = node_Cl1.get_value()
        Cl2 = node_Cl2.get_value()
        Ta = node_Ta.get_value()
        Ts = node_Ts.get_value()
        Tr = node_Tr.get_value()
        Tg = node_Tg.get_value()
        Tpv = node_Tpv.get_value()
        To = node_To.get_value()
        Io = node_Io.get_value()
        vo = node_vo.get_value()

        if Ta<10:
            air_Ta = air_Ta_5
        elif (Ta<15)and(Ta>=10):
            air_Ta = air_Ta_10
        elif (Ta<20)and(Ta>=15):
```

```
    air_Ta = air_Ta_15
elif (Ta<25)and(Ta>=20):
    air_Ta = air_Ta_20
elif (Ta<30)and(Ta>=25):
    air_Ta = air_Ta_25
elif (Ta<35)and(Ta>=30):
    air_Ta = air_Ta_30
else:
    air_Ta = air_Ta_35

if Tr<10:
    air_Tr = air_Tr_5
elif (Tr<15)and(Tr>=10):
    air_Tr = air_Tr_10
elif (Tr<20)and(Tr>=15):
    air_Tr = air_Tr_15
elif (Tr<25)and(Tr>=20):
    air_Tr = air_Tr_20
elif (Tr<30)and(Tr>=25):
    air_Tr = air_Tr_25
elif (Tr<35)and(Tr>=30):
    air_Tr = air_Tr_30
else:
    air_Tr = air_Tr_35

if Ts<10:
    back_Ts = back_Ts_5
elif (Ts<15)and(Ts>=10):
    back_Ts = back_Ts_10
elif (Ts<20)and(Ts>=15):
    back_Ts = back_Ts_15
elif (Ts<25)and(Ts>=20):
    back_Ts = back_Ts_20
elif (Ts<30)and(Ts>=25):
    back_Ts = back_Ts_25
elif (Ts<35)and(Ts>=30):
    back_Ts = back_Ts_30
else:
    back_Ts = back_Ts_35

if Tpv<10:
    front_Tpv = front_Tpv_5
elif (Tpv<15)and(Tpv>=10):
    front_Tpv = front_Tpv_10
elif (Tpv<20)and(Tpv>=15):
    front_Tpv = front_Tpv_15
elif (Tpv<25)and(Tpv>=20):
    front_Tpv = front_Tpv_20
elif (Tpv<30)and(Tpv>=25):
    front_Tpv = front_Tpv_25
elif (Tpv<35)and(Tpv>=30):
    front_Tpv = front_Tpv_30
else:
    front_Tpv = front_Tpv_35

if Tg<10:
```

```
            back_Tg = back_Tg_5
            front_Tg = front_Tg_5
        elif (Tg<15)and(Tg>=10):
            back_Tg = back_Tg_10
            front_Tg = front_Tg_10
        elif (Tg<20)and(Tg>=15):
            back_Tg = back_Tg_15
            front_Tg = front_Tg_15
        elif (Tg<25)and(Tg>=20):
            back_Tg = back_Tg_20
            front_Tg = front_Tg_20
        elif (Tg<30)and(Tg>=25):
            back_Tg = back_Tg_25
            front_Tg = front_Tg_25
        elif (Tg<35)and(Tg>=30):
            back_Tg = back_Tg_30
            front_Tg = front_Tg_30
        else:
            back_Tg = back_Tg_35
            front_Tg = front_Tg_35
    except:
        print("OPC UA client could not reach the server!")
    else:
        ind_Qhc.configure(text=str('{:.2f}'.format(Qhc))+" W")
        ind_SH.configure(text=str('{:.2f}'.format(SH*100))+" %")
        ind_Cl1.configure(text=str('{:.2f}'.format(Cl1*100))+" %")
        ind_Cl2.configure(text=str('{:.2f}'.format(Cl2*100))+" %")
        ind_Ta.configure(text=str('    '+'{:.2f}'.format(Ta))+" °C")
        ind_To.configure(text=str('    '+'{:.2f}'.format(To))+" °C")
        ind_Io.configure(text=str('{:.2f}'.format(Io))+" W/m²")
        ind_vo.configure(text=str('    '+'{:.2f}'.format(vo))+" m/s")
    finally:
        client.disconnect()

def sim():
    base = lines.copy()
    base.paste(back_Ts, (0,0), mask = back_Ts)
    base.paste(back_Tg, (0,0), mask = back_Tg)
    base.paste(air_Ta, (0,0), mask = air_Ta)
    base.paste(air_Tr, (0,0), mask = air_Tr)
    base.paste(front_Tg, (0,0), mask = front_Tg)
    base.paste(front_Tpv, (0,0), mask = front_Tpv)
    sim_img = ImageTk.PhotoImage(base)
    sim_lines.configure(image=sim_img)
    sim_lines.image = sim_img

def read_every_second():
    READ(client)
    sim()
    hourly()
    root.after(5000, read_every_second)
############################################################
read_every_second()
root.mainloop()
```

A10. Telegraf configuration code

```
# Configuration for telegraf agent
[agent]
  ## Default data collection interval for all inputs
  interval = "20s"
  ## Rounds collection interval to 'interval'
  ## ie, if interval="10s" then always collect on :00, :10, :20, etc.
  round_interval = true

  ## Telegraf will send metrics to outputs in batches of at most
  ## metric_batch_size metrics.
  ## This controls the size of writes that Telegraf sends to output plugins.
  metric_batch_size = 1000

  ## Maximum number of unwritten metrics per output.  Increasing this value
  ## allows for longer periods of output downtime without dropping metrics at the
  ## cost of higher maximum memory usage.
  metric_buffer_limit = 10000

  ## Collection jitter is used to jitter the collection by a random amount.
  ## Each plugin will sleep for a random time within jitter before collecting.
  ## This can be used to avoid many plugins querying things like sysfs at the
  ## same time, which can have a measurable effect on the system.
  collection_jitter = "0s"

  ## Default flushing interval for all outputs. Maximum flush_interval will be
  ## flush_interval + flush_jitter
  flush_interval = "10s"
  ## Jitter the flush interval by a random amount. This is primarily to avoid
  ## large write spikes for users running a large number of telegraf instances.
  ## ie, a jitter of 5s and interval 10s means flushes will happen every 10-15s
  flush_jitter = "0s"

  ## By default or when set to "0s", precision will be set to the same
  ## timestamp order as the collection interval, with the maximum being 1s.
  ##   ie, when interval = "10s", precision will be "1s"
  ##       when interval = "250ms", precision will be "1ms"
  ## Precision will NOT be used for service inputs. It is up to each individual
  ## service input to set the timestamp at the appropriate precision.
  ## Valid time units are "ns", "us" (or "µs"), "ms", "s".
  precision = ""

[[outputs.influxdb_v2]]
  ## The URLs of the InfluxDB cluster nodes.
  ##
  ## Multiple URLs can be specified for a single cluster, only ONE of the
  ## urls will be written to each interval.
  ##   ex: urls = ["https://us-west-2-1.aws.cloud2.influxdata.com"]
  urls = ["https://europe-west1-1.gcp.cloud2.influxdata.com"]

  ## Token for authentication.
  token =
"NLugSKIDycTDET3RIoMUj3ggh4gnYZlNPcd6lcWhe6GuChnQTsmFduXCNNRRhSkUPxY5Xr4fK7JKDYPjy
WWaKw=="
```

```
  ## Organization is the name of the organization you wish to write to; must
exist.
  organization = "jato11channel@gmail.com"

  ## Destination bucket to write into.
  bucket = "OPCUA"

[[inputs.opcua]]
  ## Metric name
  # name = "opcua"
  #
  ## OPC UA Endpoint URL
  endpoint = "opc.tcp://192.168.1.69:4840"
  #
  ## Maximum time allowed to establish a connect to the endpoint.
  connect_timeout = "10s"
  #
  ## Maximum time allowed for a request over the estabilished connection.
  request_timeout = "5s"

  ## Node ID configuration
  ## name              - field name to use in the output
  ## namespace         - OPC UA namespace of the node (integer value 0 thru 3)
  ## identifier_type   - OPC UA ID type (s=string, i=numeric, g=guid, b=opaque)
  ## identifier        - OPC UA ID (tag as shown in opcua browser)
  ## Example:
  ## {name="ProductUri", namespace="0", identifier_type="i", identifier="2262"}
  nodes = [
  {name="Ts", namespace="2", identifier_type="i", identifier="8"},
  {name="Tr", namespace="2", identifier_type="i", identifier="9"},
  {name="Tg", namespace="2", identifier_type="i", identifier="10"},
  {name="Tpv", namespace="2", identifier_type="i", identifier="11"},
  {name="Qhc", namespace="2", identifier_type="i", identifier="2"},
  {name="SH", namespace="2", identifier_type="i", identifier="3"},
  {name="Cl1", namespace="2", identifier_type="i", identifier="4"},
  {name="Cl2", namespace="2", identifier_type="i", identifier="5"},
  {name="To", namespace="2", identifier_type="i", identifier="15"},
  {name="Io", namespace="2", identifier_type="i", identifier="16"},
  {name="vo", namespace="2", identifier_type="i", identifier="17"},
  {name="Ta", namespace="2", identifier_type="i", identifier="7"}]
```