





Temperature control using the simulink PLC Coder and the IEC 61131 standard

Control de la temperatura mediante Simulink PLC Coder y el estándar IEC 61131

A. F. Barrera-Cuestas  ; M. G. Mantilla-Castañeda  ; D. A. Giral-Ramírez  ;
O. D. Montoya-Giraldo 

DOI: <https://doi.org/10.22517/23447214.24947>

Artículo de investigación científica y tecnológica

Abstract—Programmable Logic Controllers (PLC) are an essential part of automated industrial production processes since their first implementation, so understanding the IEC 61131 standard and, above all, section three defines the programming languages allowed by PLCs take relevance over time. This work describes each of the programming languages described in IEC 61131-3. Additionally, it implements an automation system based on Structured Text with a Human Machine Interface (HMI). The plant is a temperature process with a classic control system developed using Matlab tools, such as System Identification, PID Tuner, and Simulink. For the HMI, was implemented the Codesys Group industrial automation process platform. The Simulink PLC Coder toolbox allows the strengthening of the connection between the control system and the HMI. This program generates the Structured Text of a control system developed in Simulink. For the analysis of results, the control behavior compared between Simulink and the system produced in Codesys Group obtained an error of less than 0.34%.

Index Terms— Classic Control, IEC 61131-3, Simulink PLC Coder, Structured Text, Programmable Logic Controllers.

Resumen— Los Controladores Lógicos Programables (PLC) son una parte esencial de los procesos de producción industrial automatizada desde su primera implementación, por lo que la comprensión del estándar IEC 61131 y, sobre todo, la sección tres que define los lenguajes de programación que permiten que los PLC toman relevancia con el tiempo. Este trabajo describe cada uno de los lenguajes de programación descritos en el estándar IEC 61131-3. Además, implementa un sistema de automatización

basado en texto estructurado con una interfaz hombre-máquina (HMI). La planta es un proceso de temperatura con un sistema de control clásico desarrollado con herramientas de Matlab, como System Identification, PID Tuner y Simulink. Para la HMI, se implementó la plataforma de procesos de automatización industrial del Grupo Codesys. El toolbox Simulink PLC Coder permite reforzar la conexión entre el sistema de control y el HMI. Este programa genera el Texto Estructurado de un sistema de control desarrollado en Simulink. Para el análisis de resultados, el comportamiento del control comparado entre Simulink y el sistema producido en Codesys obtuvo un error inferior al 0,34%.

Palabras claves— Control Clásico, IEC 61131-3, Simulink PLC Coder, Lenguaje Estructurado, Controladores Lógicos Programables.

I. INTRODUCTION

FOR diversification, technology control systems require the incorporation of electronic, electrical, information, and advanced manufacturing technologies in the means of production for the automation and digitization of processes, which is a requirement for new models of energy business and an opportunity for government-academia (research) -industrial technology synchrony in a sector that in its transition seeks to integrate: renewable sources, direct current (DC) transport systems, energy storage, distributed generation, measurement systems, smart grids and the participation of end-users. Elements, in their structure, are associated with new energy generation and efficiency technologies, such as those targeted by Industry 4.0, which has different technological trends.

Since their creation in the 60s, Programmable Logic Controllers (PLC) have become indispensable devices when carrying out an automation process and are relevant elements in the challenges proposed by the industry. In the same way, their constant updating, not only at the hardware level but at the level of software and programming languages, allow these instruments to keep at the forefront to the point of being seen as an ideal option to face the fourth industrial revolution, where, reduction of production times, optimization of the levels of quality and resources used will allow us to enter into this revolution; and, in turn, lead the industry to focus on caring for the environment [1].

This manuscript was sent on October 11, 2021 and accepted on November 30, 2021.

A. F. Barrera-Cuestas student of the Universidad Distrital Francisco José de Caldas. Bogotá, Colombia. (e-mail: afbarrerac@correo.udistrital.edu.co).

M. G. Mantilla-Castañeda student of the Universidad Distrital Francisco José de Caldas. Bogotá, Colombia. (e-mail: mgmantillac@correo.udistrital.edu.co).

D. A. Giral-Ramírez professor of the Universidad Distrital Francisco José de Caldas. Bogotá, Colombia. (e-mail: dagiralr@udistrital.edu.co).

O. D. Montoya-Giraldo professor of the Universidad Distrital Francisco José de Caldas. Bogotá, Colombia. (e-mail: odmontoyag@udistrital.edu.co).



This article analyzes the IEC 61131-3 standard based on each of the programming languages it describes, in addition to implementing a controller through the use of industrial tools such as Simulink PLC Coder and CODESYS. The objective is to adopt a temperature control to an HMI through Structured Text. The temperature control is carried out in Simulink, employing the Toolbox Simulink PLC Coder that generates the high-level programming structure. Finally, the HMI is built and programmed in Codesys using the previously obtained Structured Text.

II. STANDART IEC 61131-3

Section of the IEC 61131 standard created by the International Electrotechnical Commission (IEC). This standard defines the five programming languages for Programmable Logic Controllers (PLC) to obtain order when carrying out an industrial automation process through these devices. The IEC standard determines the programming languages: Ladder Diagram, Function Block Diagram (FBD), Instruction List (IL), Structured Text (ST), and Sequential Function Chart (SFC) [2].

Define abbreviations and acronyms the first time they are used in the text, even after they have already been defined in the abstract. Abbreviations such as IEEE, SI, ac, and dc do not have to be defined. Abbreviations that incorporate periods should not have spaces: write “C.N.R.S.,” not “C. N. R. S.” Do not use abbreviations in the title unless they are unavoidable (for example, “IEEE” in the title of this article).

A. Ladder Diagram

Designed primarily for processing Boolean signals using power rail delimited graphical symbols that together resemble steps in a ladder logic diagram. The Ladder programming language allows, through the use of contact logic, to connect elements in series (AND) or in parallel (OR), which contribute to the flow or interruption of energy to supply an output called coil [3].

B. Function Block Diagram (FBD)

Based on the standard, today retired, IEC 60617-12, the FBD programming language defines a series of graphic elements. These elements allow the generation of a structure that from rectangular boxes with inputs, outputs, and flow statements of control they manage to carry out logical, arithmetic expressions or calls a function block so that once interconnected employing signal flow lines they can feed the outputs of the system [4].

C. Instruction List (IL)

The IL programming language, considered a low-level programming language, consists of a progression of instructions made up of modifiers and operations. These instructions aim that when carried out, they recreate a list of

instructions. Even though its rapid processing makes one of its main implementations that of control processes, the IEC has decided not to take it into account for its next update [5].

D. Structured Text (ST)

The ST language is derived from the Pascal programming language, and it is considered, by the standards, as one of the high-level languages. The ST Language can control the command flow by variables, expressions, or declarations of type selection, iterations, complex mathematical formulas, or calculations; it is easy for a programmer to apply and understand [6] [7].

E. Sequential Function Chart (SFC)

The SFC was created to divide a more complex program into small manageable units. SFC graphically describes the control flow between systems using stages and transitions to design sequential and parallel processes. The SFC language allows the visualization of the dependencies or interdependencies of the laps according to the process location. It also releases programming its units in the programming languages mentioned above [8] [9].

III. CONTROLLER WITH HMI

The proposed methodology was divided into six stages. The flow charts in Fig. 1, Fig. 2 and Fig. 3 describe the six stages.

A. Stage 1: Obtaining the transfer function in discrete time

It is necessary to establish the behavior of the system to be analyzed to obtain the transfer function. Fig. 4 presents the heating curve of the selected system; this curve was generated by laboratory tests. The minimum temperature reached was 24.93 °C, and the maximum was 78.2 °C.

With the Matlab System Identification toolbox, the heating curve of the thermal plant, and the supply voltage, the data is processed to obtain the transfer function in the continuous-time domain (1), gathering a 96.03% correspondence.

Employing the Matlab *c2d* command the transfer function is obtained in the discrete-time domain (2), using the *Tustin* discretization method. Fig. 4 represents the system heating curve and the discrete-time transfer function curve.

$$H_{(s)} = \frac{2.72s + 0.05514}{s^3 + 2.238s^2 + 9.543s + 0.05266} \quad (1)$$

$$H_{(z)} = \frac{0.1523z^3 + 0.1553z^2 - 0.1461z - 0.1492}{z^3 + 0.3798z^2 + 0.1075z + 0.5011} \quad (2)$$

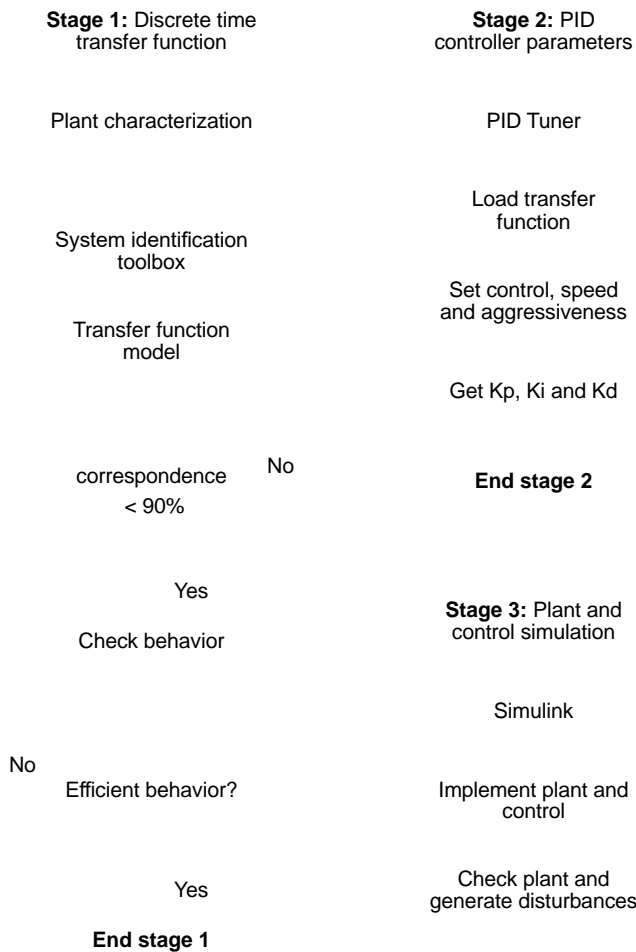


Fig. 1. Stage 1, 2 and 3 of the design and implementation of the controller with HMI. Own Work

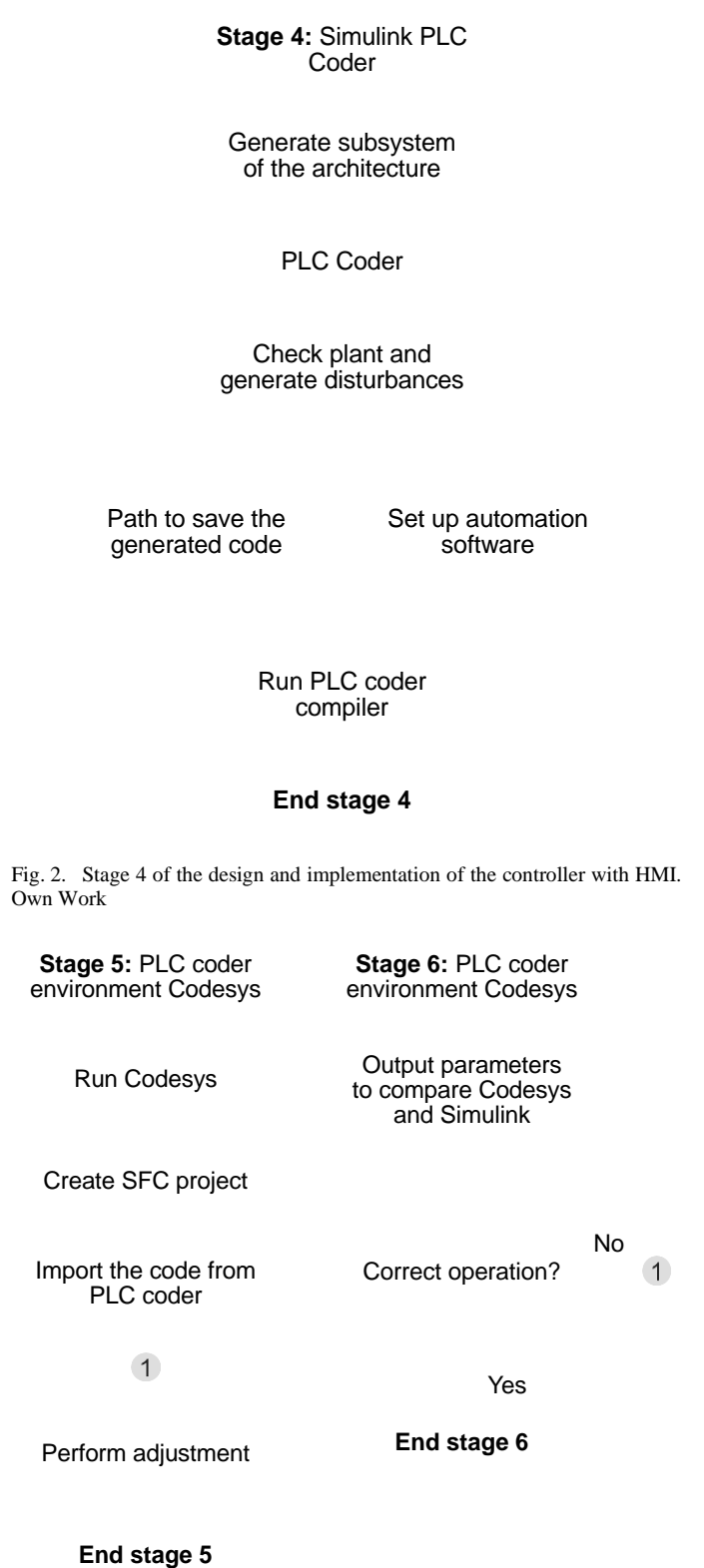


Fig. 2. Stage 4 of the design and implementation of the controller with HMI. Own Work

Fig. 3. Stage 5 and 6 of the design and implementation of the controller with HMI. Own Work

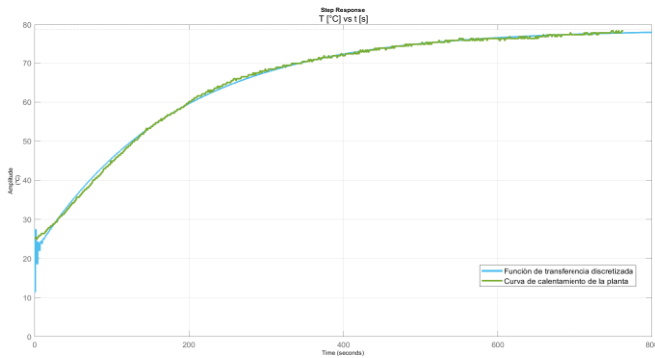


Fig. 4. Plant heating curve and discrete transfer function. Own Work

B. Stage 2: Obtaining the constants of the classical controller

Using the PID Tuner toolbox on the transfer function in the discrete-time domain, a PI controller is implemented that adjusts to the system under study. A fast response time and a robust transient behavior are selected, obtaining the constants of proportionality and integrity described in Table I and an output behavior as observed in Fig. 5.

Constant	Values
Constant of proportionality (Kp)	0.49989
Integrity constants (Ki)	0.45537

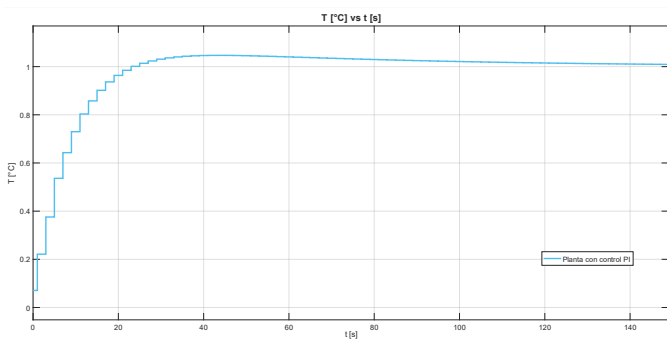


Fig. 5. The behavior of the plant with a PI controller. Own Work

C. Stage 3: Design and simulation of the plant architecture in the Simulink environment

Using the Simulink block diagram environment, the system behavior is simulated, implementing the PI controller, as seen in Fig. 6 and Fig. 7 shows the behavior of the system with and without control when applying disruptions.

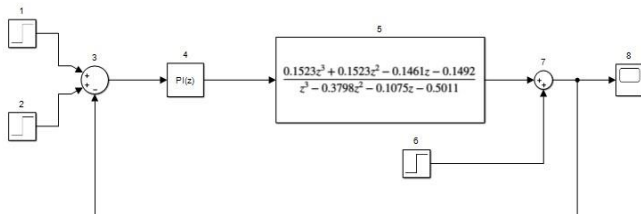


Fig. 6. System modeling in Simulink. Own Work

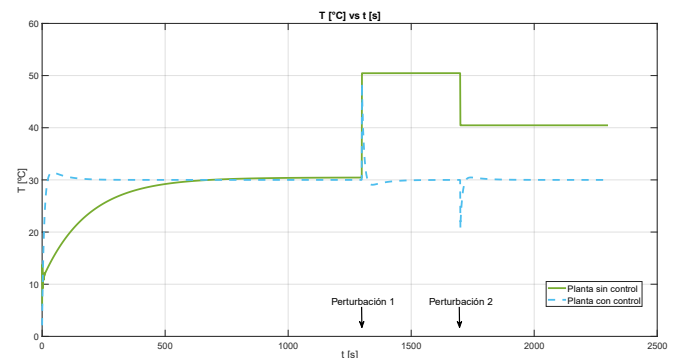


Fig. 7. Plant with PI control and without PI control with two disruptions. Own Work

D. Stage 4: Installation and configuration of the Simulink PLC Coder toolbox

The Matlab toolbox, Simulink PLC Coder, is a tool that allows the generation of programming code in ST language from Simulink models. It is possible to generate the programming code in 15 different simulators, including Codesys 3.5 [10]. Fig. 8 presents the programming code obtained through Simulink PLC Coder for the controller described in Fig. 6.

```

1 CASE ssMethodType OF
2 SS_INITIALIZE:
3     Integrator_DSTATE:=0;
4     DiscreteTransferFcn_state[0]:=0;
5     DiscreteTransferFcn_state[1]:=0;
6     DiscreteTransferFcn_state[2]:=0;
7 SS_STEP:
8     DiscreteTransferFcn_tmp:=(((0.49989*u)+Integrator_DSTATE)-(-0.3798
9     *DiscreteTransferFcn_state[0]))-(-0.1075*DiscreteTransferFcn_state[1]))-
10     (-0.5011*DiscreteTransferFcn_state[2]));
11     rtb_DiscreteTransferFcn:=(((0.1523*DiscreteTransferFcn_tmp)+(0.1523*
12     DiscreteTransferFcn_state[0]))+(-0.1461*DiscreteTransferFcn_state[1]))+
13     (-0.1492*DiscreteTransferFcn_state[2]));
14     Integrator_DSTATE:=(0.45537*u)+Integrator_DSTATE;
15     DiscreteTransferFcn_state[2]:=DiscreteTransferFcn_state[1];
16     DiscreteTransferFcn_state[1]:=DiscreteTransferFcn_state[0];
17     DiscreteTransferFcn_state[0]:=rtb_DiscreteTransferFcn;
18     Out1:=rtb_DiscreteTransferFcn+In2;
19 END_CASE
    
```

Fig. 8. Programming code generated by Simulink PLC Coder. Own Work

E. Stage 5: Adaptation of the code generated in the Codesys environment

Some modifications are required to implement the programming code generated by Simulink PLC Coder in the Codesys interface; adjustments such as, change the global variables of constant type for general global variables to modify their state once the project is carried out. Additionally, in ST is created a statement that represents the feedback of the system of Fig. 6. The Simulink PLC Coder toolbox does not have the support to generate the closed-loop so that it is necessary to make that adjustment. Fig. 9 presents the programming code obtained with closed-loop.

```

1 IF GVL.SET_POINT>=24.93 AND GVL.SET_POINT<=78.2 THEN
2   DiscreteTransferFcn_tmp:=(((0.49989*u)+Integrator_DSTATE)-(-0.3798*
3   DiscreteTransferFcn_states[0]))-(-0.1075*DiscreteTransferFcn_states[1]))-
4   (-0.5011*DiscreteTransferFcn_states[2]);
5   rtb_DiscreteTransferFcn:=(((0.1523*DiscreteTransferFcn_tmp)+(0.1523*
6   DiscreteTransferFcn_states[0]))+(-0.1461*DiscreteTransferFcn_states[1]))+
7   (-0.1492*DiscreteTransferFcn_states[2]);
8   Integrator_DSTATE:=(0.45537*u)+Integrator_DSTATE;
9   DiscreteTransferFcn_states[2]:=DiscreteTransferFcn_states[1];
10  DiscreteTransferFcn_states[1]:=DiscreteTransferFcn_states[0];
11  DiscreteTransferFcn_states[0]:=DiscreteTransferFcn_tmp;
12  Out1:=rtb_DiscreteTransferFcn+u1;
13  u:=GVL.SET_POINT-ABS(Out1);
14  Error:=-ABS(u);
15 END_IF
    
```

Fig. 9. Programming code with feedback. Own Work

F. Stage 6: Comparison of the Simulink and Codesys output curves

To determine qualitatively and quantitatively the graphs obtained in Simulink and Codesys, it was necessary to carry out three case studies to find the percentage of error between them. So, scenarios are analyzed where the behavior of each of the graphs is compared at specific points. These scenarios are:

- 1) *Initial maximum temperature:* The maximum temperature that the plant reaches during the heating process at the initial setpoint.
- 2) *Initial maximum temperature:* The maximum temperature that the plant reaches during the heating process at the initial setpoint.
- 3) *Maximum temperature in the disruption:* The maximum temperature that the plant reaches when it is applied disruption to the system.
- 4) *The maximum temperature change of the setpoint:* The maximum temperature that the plant reaches when establishing a new value of the setpoint.
- 5) *Stabilization:* Temperature registered by the plant when stabilizing at the second setpoint.
- 6) *Before the disturbance:* The indicated temperature by the plant before the disruption occurs.
- 7) *Before the setpoint changes:* The recorded temperature by the plant before the setpoint change occurs.

From (3) is found the percentage of error between behaviors.

$$e_{\%} = \frac{|T_{Simulink} - T_{CODESYS}|}{T_{Simulink}} \quad (3)$$

Where:

$T_{Simulink}$: Temperature recorded by Simulink
 $T_{Codesys}$: Temperature recorded by Codesys.

Case 1: Disruption and positive setpoint

The behavior and results obtained in Fig. 10 and Table II

relate the curves produced by Simulink and Codesys from assigning a negative setpoint and disturbance. These results are from the following parameters:

- From the beginning of the code implementation, it establishes an initial setpoint of 30°C.
- Disruption generation of 20 °C within 120 seconds of executing the code.
- Updating the setpoint to 60 °C within 200 seconds of carrying out the code.

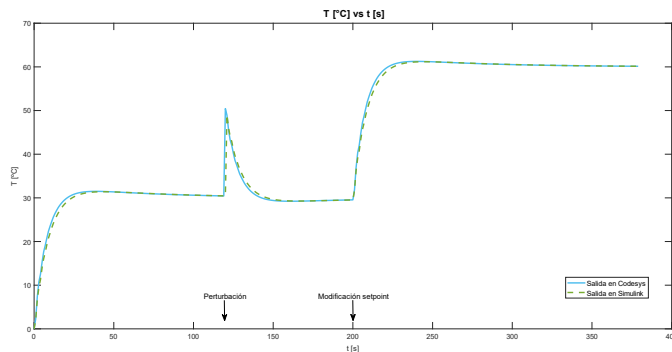


Fig. 10. The heating curve obtained from the Codesys (blue) and Simulink (green) software for positive disruption and setpoint. Own Work

TABLE II
 ERRORS BETWEEN SIMULINK AND CODESYS FOR HEATING CURVE TIMES
 DETERMINED WITH POSITIVE DISTURBANCE AND SETPOINT

Action of interest	Time [s]	T. Codesys [°C]	T. Simulink [°C]	Error [%]
Max. initial Temp.	41	31.4846	31.390	0.3010
Max. disruption Temp.	120	50.4341	49.0293	2.8652
Max change setpoint Temp.	241	61.1377	61.1775	0.1281
Stabilization	379	60.1377	60.1431	0.0089
Before the disruption	119	30.4415	30.4518	0.3382
Before the setpoint change	199	29.5427	29.5249	0.0602

Case 2: Disturbance and negative setpoint

The behavior and results obtained in Fig. 11 and Table III relate the curves produced by Simulink and Codesys, the error percentages obtained in each of them, from assigning a negative setpoint and disruption. These results are from the following parameters:

- From the beginning of the code completion, it establishes an initial setpoint of 50°C.
- Generation of a disruption of -20 °C within 120 seconds of executing the code.
- Updating the setpoint to 30 °C within 200 seconds of implementing the code.

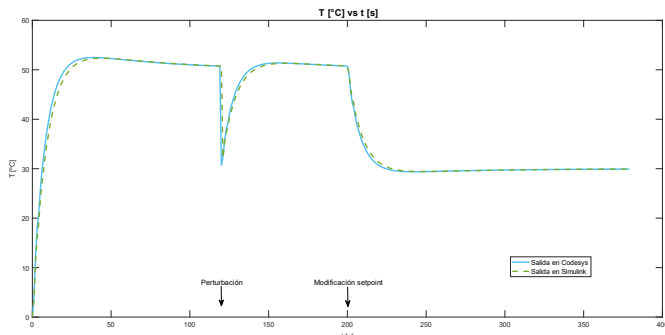


Fig. 11. The heating curve obtained from the Codesys (blue) and Simulink (green) software for a negative disruption and setpoint. Own Work

TABLE III

ERRORS BETWEEN SIMULINK AND CODESYS FOR DETERMINED HEATING CURVE TIMES WITH NEGATIVE DISTURBANCE AND SETPOINT

Action of interest	Time [s]	T. Codesys [°C]	T. Simulink [°C]	Error [%]
Max. initial Temp.	43	52.4471	52.3377	0.2090
Max. disruption Temp.	120	30.5275	32.1553	5.0623
Max change setpoint Temp.	248	29.4014	29.4275	0.08869
Stabilization	379	29.9299	29.9271	0.00935
Before the disruption	119	50.7358	50.753	0.03388
Before the setpoint change	199	50.7685	50.785	0.03248

Case 3: Positive disturbance and negative setpoint

The behavior and results recorded in Fig. 12 and Table IV relate the curves produced by Simulink and Codesys and the error percentages obtained in each of them from assigning a positive disturbance and a negative setpoint. These results are from the following parameters:

- From the beginning of the code execution, it sets down an initial setpoint of 40°C.
- Generation of a 30 °C disruption within 120 seconds after having implemented the code.
- Updating the setpoint to 28 °C within 200 seconds of having executed the code.

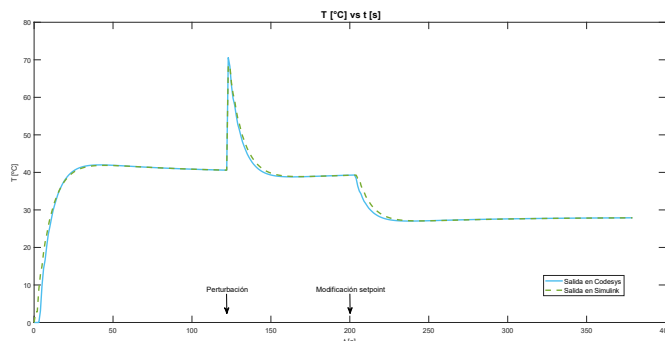


Fig. 12. The heating curve obtained from the Codesys (blue) and Simulink (green) software for positive disruption and negative setpoint. Own work

TABLE IV

ERRORS BETWEEN SIMULINK AND CODESYS FOR HEATING CURVE TIMES DETERMINED WITH POSITIVE DISTURBANCE AND NEGATIVE SETPOINT

Action of interest	Time [s]	T. Codesys [°C]	T. Simulink [°C]	Error [%]
Max. initial Temp.	44	41.9431	41.8701	0.1743
Max. disruption Temp.	120	70.5788	68.4699	3.0800
Max change setpoint Temp.	237	27.0289	27.0732	0.1636
Stabilization	379	27.9021	27.9000	0.0075
Before the disruption	119	40.5887	40.6024	0.0034
Before the setpoint change	119	39.2835	39.268	0.0395

G. Human-Machine Interface (HMI)

Employing the visualization manager of the Codesys software is possible to present the most relevant information on the plant control process. This visualization manager allows the user to observe the current status of the thermal plant; it also has the freedom to assign a new setpoint and disruption value if required. In that sense, three screens are designed to have total control of the plant. These screens are the presentation screen, control-display screen, and control-management screen.

1) *Presentation screen:* The presentation screen defines the name and title of the project, the names of the developers, and the institutional affiliation (Fig. 13).



Fig. 13. HMI splash screen layout. Own Work

2) *Control-display screen:* In the control and visualization screen, the user can identify the current behavior of the thermal plant and the disturbances that have been generated, comparing the setpoint established with the current temperature of the plant. In the same way, it displays the on or off status of the plant, the current temperature value inside it, and the error, in Celsius degrees. Finally, it has the option of

establishing a new setpoint value and generating disruptions in the system; it also allows turning the plant on or off (Fig. 14).

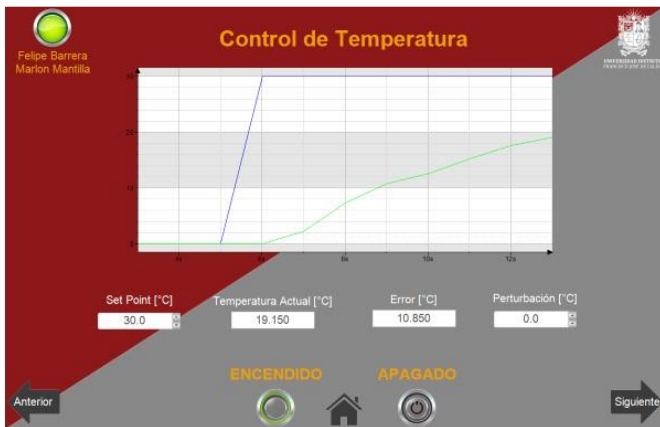


Fig. 14. Design of the HMI presentation and control screen. Own Work

3) *Control-management screen:* The control and management screen allows determining, from the activation of the indicators and a bar-shaped temperature chart, the current status of the plant from the temperature registered inside it. So that the thermal scale is divided into five ranges: very high, high, medium, low, or very low. Using two additional indicators, the user could determine if the system is stable or if a setpoint has been established outside the allowed range; and identify which does not guarantee the correct operation of the control (Fig. 15).

Finally, the user can set a new setpoint value and display the error of the current work.



Fig. 15. Design of the HMI control and management screen. Own Work

IV. GENERATED MATERIAL

Table V shows the name and the corresponding links to the videos that complement the explanation of the stages implemented.

TABLE V

ERRORS BETWEEN SIMULINK AND CODESYS FOR HEATING CURVE TIMES DETERMINED WITH POSITIVE DISTURBANCE AND NEGATIVE SETPOINT

Stage	Link
1	https://youtu.be/T6r58dViQyY
2	https://youtu.be/f072iUgtzAo
3	https://youtu.be/6WTzW24Uufk
4	https://youtu.be/WOqBgep14i0
5	https://youtu.be/giOsiJFyxpM
6	https://youtu.be/m3Bhq1Ikpio

V. CONCLUSION

From the programming languages presented in the IEC 61131-3 standard, it is possible to highlight the facilities that come with developing a project in high-level programming languages (ST and SFC). The ST has selection type and iteration type structures; the similarity between the programming syntax with languages such as C ++, Matlab or Python, makes the adaptability for a programmer who does not know the standard more enjoyable. The SFC simplifies projects where sequential, combinational, and parallel systems predominate.

Analyzing the tables, It determinates that the error of the results obtained from the simulated system in Codesys and the system simulated in Simulink is less than 0.34% in all its measurements, except for the one generated in the peak of maximum or minimum disruption.

REFERENCES

- [1] A. Instrumentación, ¿Hacia dode camina el futuro de los controladores lógicos programables (PLCs)?, 2020.
- [2] International Electrotechnical Commission, IEC 61131-3: Programmable Logic Controllers, 2013.
- [3] J. Alvarino, Método de programación para PLC's basado en el estándar IEC 61131-3 – caso de estudio proceso de elaboración de pan. Universidad de la Salle, 2016.
- [4] F. Angerer, H. Prähofer, R. Ramler and F. Grillenberger, Point to analysis of IEC 61313-3 programs: Implementation and application, 2013.
- [5] K. John and M. Tiegelkamp, IEC 61131-3: Programming Industrial Automation Systems, 2010.
- [6] L. Huang, W. Liu and Z. Liu, Algorithm of Transformation from PLC Ladder Diagram to Structed Text, 2009.
- [7] L. Brito, J. Almeida, J. Pecorelli, and P. Sousa, Simulation of Structured Text Language for PLC Programming, 2015.
- [8] R. Bilbao, and A. Mantilla, Diseño e Implementación de Automatismos Lógicos Secuenciales en SFC para el PLC S7-200 y S7-Graph para el PLC S7-300, 2011.
- [9] R. Jaspe and A. Mosquera, Grafcet Aplicado al diseño de Automatismos con PLC S7-200, 2007.
- [10] The MathWorks, Inc, Simulink PLC Coder Generación de diagramas de contactos (ladder) y texto estructurado IEC 61131-3 para PLCs y PACs, 2021.